논문 03-01-16

A Unified Data Model for Conceptual Data Modeling

개념적 데이타 모델링을 위한 통합 데이타 모델

羅 然 默*
Yun-mook Nah*

Abstract

In this paper, a conceptual data model, called the UDM(Unified Data Model), to efficiently represent database structures related with object technology and complex structured data, is proposed. This model integrates major features of modern data models, such as E-R model, Semantic Object Model, and UML, especially from the viewpoint of database design. This model is basically a simplified, but extended version of the Object-Relationship Model, which was proposed to model complex structures of temporal-spatial multimedia data.

This model incorporates some of the important semantic and structural information of modern database applications and it is designed to support all of the major logical database models, including relational, object-relational, object-oriented, and (semi-)structured databases. A special diagrammatic technique, called the UDD(Unified Data Diagram), is introduced as a tool for database design. Also, possible ways to derive logical views of data from this unified data model are presented. The proposed model can be utilized as a convenient and practical tool for conceptual database designs.

Keywords: Unified Data Model, Unified Data Diagram, conceptual data model, database design, data modeling

요 약

본 논문에서는 객체 기술과 복잡한 구조적 데이터와 관련된 데이터베이스 구조를 효율적으로 표현하기 위한 통합 데이타 모델(UDM)이라고 하는 개념 데이터 모델을 제안한다. 이 모델은 E-R 모델, 의미 객체 모델, UML 등의 데이타 모델의 주요 기능을 데이타베이스 설계 관점에서 통합한 것이다. 이 모델은 시공간 멀티미디어 데이타의 복잡한 구조를 모델링하기 위해 제안된 객체-관계 모델을 단순화시키고 일부 기능을 확장시킨 버전이다.

이 모델은 현대적인 데이터베이스 응용의 주요한 의미적, 구조적인 정보의 표현을 지원하며, 관계, 객체-관계, 객체-지향, (반-)구조 데이터베이스 등 주요 논리적 데이터베이스 모델을 지원하도록 고안되었다. 데이터베이스 설계를 위한 도구로 통합 데이타 다이아그램(UDD)을 제안하며, 이 통합 데이타 모델로부터 논리적 뷰를 유도하기 위한 방법도 제안한다. 제안된 모델은 다양한 개념적 데이터베이스 설계를 위한 편리하고 실용적인 도구로 활용될 수 있다.

*檀國大學校 電氣電子컴퓨터工學部
School of Electrical, Electronics, and Computer Engineering, Dankook University

※ 이 연구는 2002학년도 단국대학교 대학연구비의 지원으로 연구되었음.
接受日:2003年 5月 23日, 修正完了日:2003年 22月 10日

I. Introduction

The conceptual modeling of data has been an important issue in conceptual database design. In database world, the most popular conceptual modeling tool is the E-R(Entity-Relationship) model[1,2] and an alternative is the Semantic Object Model[3], which has a smaller following. In object-oriented software engineering, the UML(Unified Modeling Language) is well known as unanimous notation for object-oriented software design[4].

These models have their own strengths and weaknesses. The E-R model is well suited for relational database design, but its capability to represent object-relational database and object-oriented database structures has been challenged. The Semantic Object Model, which is considered to be richer and easier than the E-R model by some researchers, provides a more natural view of application data and represents set values and complex object structures having nested attributes, but its capability to represent object database structures is not well studied. The UML model, which unifies major features of some prominent methods, such as Booch, Jacobson's OOSE, and Rumbaugh's OMT(Object Modeling Technique), is the powerful method for object-oriented software analysis and design, but it lacks in important database design concepts of object-relational databases and object-oriented databases, such as OID(Object IDentifier), extent, key, and complex data structures. The modeling power of major existing data models is summarized in Table 1.

To efficiently represent database structures related with object technology and complex structured data, such as XML and SGML, we need a new conceptual data model, which is powerful than the E-R model, includes main features of modern modeling methods, and well customized to the database needs. This paper presents a unified conceptual data model, called the UDM(Unified Data Model), which has most of the advantages of the above models. As we begin our unification, we established three goals for our work:

- to create a conceptual data model which is a superset of existing conceptual models
- to create a conceptual data model that can support all the major logical database models, including relational, object-relational, object-oriented,

Table 1. Comparison of representative conceptual data modeling tools

	E-R	SOM	UML	XML DTD
entity/class	yes	yes	yes	yes
weak entity	yes	version object	aggregation	not applicable
attribute	yes	yes	yes	yes
set-value	no	yes(0,1,N)	yes	not applicable
nested attribute	no	yes	no	not applicable
binary HAS-A	yes	yes	yes	not applicable
relationship cardinality	yes(0,1,N,M,P)	yes(0,1,N)	yes(0, 1, *)	yes(*, +, ?)
n-ary HAS-A	yes	no	no	not applicable
IS-A	yes	yes	yes	not applicable
aggregation	no	no	composition, aggregation	sequence, choice
best suitable	R DB	H/N/R DB	OOP, OODB	XML DB

and (semi-)structured databases

• to minimize the number of symbols and simplify the shape of symbols, while allowing users to still use the well known previous modeling symbols

We have previously proposed a data model, the Object-Relationship model, aiming to represent temporal-spatially structured multimedia data[5,6]. This model is quite powerful in representing multimedia information structures, such as temporal sequence and time synchronization, but a little bit complex. We tries to adopt the modeling capability of complex data structures from the Object-Relationship model, while eliminating less useful constructs and simplifying some complex constructs.

This paper is organized as follows. Section 2 briefly explains the evolution of conceptual data models. In section 3, we propose the unified data model which describes real world in terms of information objects and relationships. Its diagrammatic technique, called the UDD(Unified Data Diagram), is described with examples in section 4. Section 5 presents how to derive logical views of data for relational, object-relational, object-oriented, and (semi-)structured databases from this unified data model. Finally, section 6 concludes the paper.

II. Evolution of Data Model

In database terms, formatted data means traditional data, such as numeric and character data, and unformatted data means new kinds of data, such as text, graphics, image, audio, animation, video, spatial data, time series data, and document data. We might again distinguish structured data, such as document, multimedia document, multimedia information, and other complex structures, from semi-structured data, such as XML data, SGML data, and web page data.

Data models can be divided into conceptual data models and logical data models. Conceptual data models describe database structures in our minds by using high level information units, such as entities and relationships. The E-R model, semantic network model, semantic object model, object-oriented model, and object-relationship model can be classified as conceptual data models. Logical data models describe information structure for database end-users and application programmers by using well known information units, such as records, relations and classes. The hierarchical model, network model, relational model, object-oriented model, object-relational model are well known examples of logical data models. The object-oriented model can be regarded as both conceptual and logical data model, because its conceptual database structures and logical database structures are almost the same.

The major characteristics of each conceptual data models are as follows:

• **entity-relationship model(1976)**: This model describes real world by using entities and relationships. Specially, it focuses on how to represent association relationships with their relationship cardinality, such as 1:1, 1:N, N:M.

• **object-oriented model(late 80s):** A new relationship, generalization, is emphasized to deal with inheritance of structural and behavioral properties. According to the principle of ADT(Abstract Data Types), this model tries to represent both data structures and possible operations on them.

• **complex object model(late 80s-early 90s):** This model focuses on nested tuple structures and set values.

• **semantic-object model(1988):** This model describes database structures as a collection of semantic objects, which itself is a named collection of attributes that sufficiently describes a distinct identity. When compared to the E-R model terms, a semantic object can be regarded as an entity with normal attributes(so called simple attributes), set value attributes(so called multi-value attributes),

nested attributes(so called group attributes), and all its related relationships(so called object attributes).

• **(semi-)structured data model(late 90s)**: Emphasis is put on hierarchical data structures.

From the above short survey of data models, we can speculate that data models are evolving toward the direction of representing more relationships and more complex structures for unformatted and (semi-)structured data.

According to the evolution of conceptual data models and new needs of modern applications, logical data models are also changing to adopt object technology and structuring capability.

III. Unified Data Model

Database consists of information objects, which are independent information units, and their relationships. We will use the traditional modeling terms, such as entities, attributes, types, and relationships. But, in our unified model and in our modern age of object technology, the distinction between them become less clear, because entities(classes) can be used as types and attributes can be extended to classes, and relationships are represented as attributes in later stages of database design. Therefore, this situation results in lots of confusions to the E-R and relational model-oriented designers. Nevertheless, we still need such distinction among them in the initial design stage like conceptual data modeling. One main aim of this paper is to make such confusions among major design terms clear.

3.1 Information Objects

An information object(or shortly object) is something that can be identified, such as an entity of the E-R model, a class of object-oriented model, and an element of XML/SGML document models. In this paper, we will use the terms, such as information object, object, entity, class, and element, interchangeably. Examples of information objects are PERSON, EMPLOYEE, PROFESSOR, DEPARTMENT, ARTICLE, and PAPER. The existence of some information objects can be dependent on other information objects.

In relational database age, entities are clearly distinguished from data types, or domains. But, in this object age, a user-defined information object(actually, the set of occurrences of that information object) can be usually used as data types of other information object's attributes. In this paper, we will regard the representative system data types, such as Integer, Decimal, VARCHAR, etc, as basic types(BTs); we will regard information objects(actually, their extensions), such as PERSON, ADDRESS, NAME, etc, as user-defined types(UDTs), if they are intended to be used as data types of other information object's attributes.

3.2 Attributes

Information objects have attributes or properties that describe the information object's characteristics. Examples of attributes are Name of Person, Salary of Employee, and Title of Article. Each attribute can be assigned an identifying role, such as key, identifier, unique identifier, object identifier(OID), and user-generated object identifier.

But, in this object age, attributes can be extended as independent types or classes. In such cases, entity-attribute relationships become inter-entity relationships.

3.3 Relationships

Information objects are containers of independent information(i.e. information islands), while their relationships are bridges to cross over among these information islands. There are three kinds of relationships which take important role in database structuring and information extraction.

• **association(HAS-A):** Information objects can be associated with one another. If an information object, E1, is associated with another information object, E2, we denote this bidirectional relationship as E1<h>E2. (This notation can cover only binary

case. For more than binary cases, we have to use diagrammatic symbols.) An example of this relationship is "EMPLOYEE HAS-A relationship with DEPARTMENT".

• **generalization(IS-A):** An information object, E2, can inherit structures, relationships, and behaviors of its parent information object, E1. We denote this unidirectional relationship as E1<|E2, which can be read as "E2 IS-A E1". If E2 inherits from E1, we call E1 as supertype or superclass and E2 as subtype or subclass. An example is "EMPLOYEE IS-A PERSON".

• **aggregation(PART-OF):** An information object, E, consists of other information objects, P1, P2, ..., Pn. An example is "PAPER consists of HEAD and BODY" or "HEAD and BODY are PART-OF PAPER". In the unified data model, entity-attribute relationships are regarded as a special case of aggregation.

A relationship, that is not classified as generalization or aggregation relationship, can be regarded as an association relationship. Also, we can classify the above relationships as unidirectional relationsips(IS-A, PART-OF) and bidirectional relationsips(HAS-A). Aggregation relationships can be further classified as follows.

• **simple aggregation(tuple aggregation):** An information object, E, simply consists of other information objects, P1, P2, ..., Pn. We denote this as E<p>(a1:P1, a2:P2, ..., an:Pn), where ai means attribute names of E. Entity-attribute relationships are regarded as a special case of aggregation, where each Pi is basic type BTi, i.e., E<p>(a1:BT1, a2:BT2, ..., an:BTn), which has been modeled simply in the E-R diagram as E<p>(a1, a2, ..., an).

• **sequence aggregation:** An information object, E, consists of a sequence of other information objects, P1, P2, ..., Pn. We denote this as E<p><a1:P1, a2:P2, ..., an:Pn>, where ai means attribute names of E.

• **parallel aggregation:** An information object, E, consists of a parallel occurrence of other information objects, P1, P2, ..., Pn. We denote this as E<p>[a1:P1, a2:P2, ..., an:Pn], where ai means attribute names of E.

• **choice aggregation:** An information object, E, consists of one of alternative information objects, P1, P2, ..., Pn. We denote this as E<p>(P1|P2|...|Pn).

The introduction of sequence, parallel, and choice aggregations are to represent complex structures of modern data, such as multimedia documents and XML/SGML documents.

As we will see later, one important thing to note is that these relationships are finally represented as attributes or used to add additional attributes, thus changing the numbers of attributes in the later stages of database design, such as logical or physical database design.

3.4 Changes of Attribute Domains to Information Objects

Attributes(actually, the domains or data types of those attributes) can be modeled as independent information objects, by means of object technology. For example, the information object EMPLOYEE<p>(Name:CHAR, Address:VARCHAR), shortly EMPLOYEE<p>(Name, Address), can be extended to EMPLOYEE<p>(Name:NAME, Address:ADDRESS), where NAME and ADDRESS are UDTs. In such case, the entity-attribute relationship between the entity EMPLOYEE and attributes Name and Address, changes to the entity-entity aggregation relationship between entity EMPLOYEE and two part entities, NAME and ADDRESS.

3.5 Cardinality

In the E-R model, the only relationship that allows multiple value cardinality is HAS-A association relationship. In the unified data model, the multiple value cardinality can be applied to both attributes and inter-entity relationships.

• **attribute cardinality:** This cardinality represents the allowed occurrences of attribute values in the given entity-attribute relationship. By using this attribute cardinality, we can represent set value attributes and ordered set value attributes.
• **relationship cardinality:** This cardinality represents the allowed occurrences of participating entities in the given inter-entity relationship.

We recommend the form 'min.max' with the symbols, 0, 1, N, but users can use well known existing symbols, such as 0, 1, N(multiple), ?(0 or 1), *(0 or more), +(1 or more), alternatively. In representing cardinalities, we denote set value as 'min.max', while ordered set value as '<min.max>'. Typical examples are:

• 0.1: The minimum cardinality is 0 and the maximum cardinality is 1. In UML and XML style, this is usually written as '?'.
• **0.N:** The minimum cardinality is 0 and the maximum cardinality is N. In UML style, this is usually written as '*' or '0..*'. In XML, this is simply written as '*'.
• **<0.N>:** This means an ordered set with minimum cardinality 0 and maximum cardinality N.

In a unidirectional relationship, cardinality can be assigned onto at most one side. In a bidirectional relationship, cardinality can be assigned onto both sides. For example, in the case EMPLOYEE<p>(Name, Phone), the attribute Name can be assigned the cardinality '1.1' and the attribute Phone can be assigned the cardinality '1.N'. We can denote this by using subscripts as EMPLOYEE<p>(Name1.1, Phone1.N). In the bidirectional case of EMPLOYEE <h> DEPARTMENT, the entity EMPLOYEE can be assigned the cardinality '0.N' and the entity DEPARTMENT can be assigned the cardinality '1.1'. We can denote this by using subscripts as EMPLOYEE0.N<h>1.1DEPARTMENT.

3.6 Comparison to the E-R model terms

Table 2 compares the major conceptual modeling terms of the unified data model with those of the E-R model. The distinction among the terms, such as entity, attribute, domain, and data type become less clear in the unified data model.

Table 2. Comparison of major modeling terms

E-R model	Unified Data Model
entity, in general	information objects, such as entity, class and element
entity type	information object type
entity set	information object extension
entity (occurrence)	information object (instance)
attribute	attribute
value set, domain	value set of basic types or extension of information objects
data type	BTs, such as NUMBER, DECIMAL, CHAR, etc, and UDTs, which are names of information objects
entity-attribute relationship	regarded as entity-to-BT aggregation relationship and can be extended to entity-entity aggregation relationship
entity-entity relationship	entity-entity relationship
relationship cardinality	attribute cardinality and relationship cardinality

IV. Unified Data Diagram

4.1 Symbols

Figure 1 shows the diagrammatic symbols of the unified data model. The major symbols are adopted from the E-R diagram and the UML diagram, according to our design principles. Each information object type is represented using a rectangular box, as in the E-R and UML. The double-sided rectangle means weak entity, whose existence is dependent on another entity. Each rectangle can have one or more sections inside, each of which can represent information object name, attribute list, method list, and constraint list, just as in UML.

Attributes are represented as ellipses or small circles. A black small circle represents a key attribute. Alternatively, all attributes can be listed in textual form within the attribute list section of information object rectangle.

Large non-square diamond, triangle and small square diamond each represents association(HAS-A), generalization(IS-A), and aggregation(PART-OF) relationship. Small square diamonds, representing aggregation, are further classified as normal diamond, diamond with 'arrow symbol' inside, diamond with 'equal symbol' inside, and diamond with 'bar symbol' inside, meaning simple aggregation, sequence aggregation, parallel aggregation, and choice aggregation, respectively.

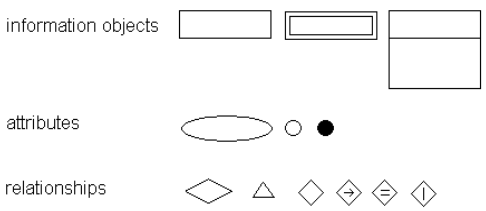

Figure 1. Symbols of unified data diagram

On the relationship arc linking related entities, name labels can be written. For a bidirectional relationship, HAS-A, name labels can be assigned on both side of the arcs. But, in case of unidirectional relationship, PART-OF, name labels can be assigned on one side only. These name labels can be regarded as attribute names of appropriate entities and also used as attribute names of tables or classes in logical database design stage.

4.2 Representation of Information Objects

In this section, we introduce a diagrammatic technique for exhibiting information objects, which represent entities, classes, or elements.

4.2.1 Information Objects

Figure 2 shows an entity EMPLOYEE, which contains attributes, such as Eno, Name and Salary. The first diagram is in the traditional E-R diagram style; the second is in the E-R diagram style of [3]; and the third is in the semantic object diagram or UML class diagram style.

Figure 2. Alternatives of entity representation

As described earlier, entity-attribute relationship can be regarded as entity-entity aggregation relationship as shown in Figure 3, which is another alternative for the diagrams in Figure 2, if we treat basic types, such as NUMBER, CHAR, and DECIMAL, as entities. In this figure, the name labels beside the relationship arcs represent attribute names, while the numeric labels represent minimum and maximum cardinality of each attributes. Using this kind of notation, we can easily extend domains of each attributes by replacing the name of each leaf boxes as names of other user-defined information objects.

Figure 3. Entity-attribute relationship represented as entity-entity relationship

4.2.2 Information Objects with Set Values

In the unified data diagram, we can easily represent set value attributes by using the numeric labels with maximum cardinality value greater than 1. In Figure 4, the Phone attribute of the EMPLOYEE entity can have multiple values.

Figure 4. Alternative representation of entities with set values

4.2.3 Weak Entities

The notation for weak entities, which are existentially dependent on parent entities, is similar with the E-R diagram as shown in the left side diagram of Figure 5. They also can be represented as parts of other entities by using aggregation relationships as shown in the right side diagram of Figure 5. The meaning of existential dependency can be added to the right side diagram, by changing the rectangle of DEPENDENT entity to a double-sided rectangle. The relationship between EMPLOYEE entity and DEPENDENT entity can be treated as association or aggregation, depending on the data modeler's decision, but both of them usually result in the same logical representation.

Figure 5. Alternatives of weak entity representation

4.2.4 Nested Attributes

In object-oriented models and complex object models, it is usual to have nested attributes. Figure 6 shows various alternatives to represent the nested attribute Address of the entity EMPLOYEE. The first diagram is in the semantic object diagram style; the second one represents it by using an entity-entity aggregation relationship between the entity EMPLOYEE and the anonymous entity with (EMPLOYEE's) attribute name Address; and the third one is similar with the second, except that unnested attributes are shown by using small circles.

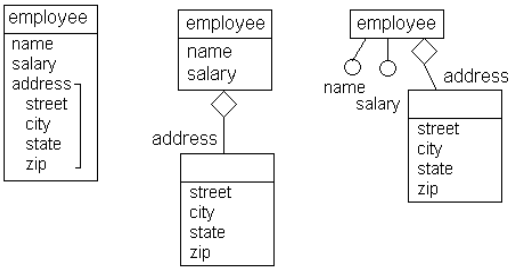

Figure 6. Alternatives of nested attribute representation

4.3 Representation of Relationships

In this section, we introduce a diagrammatic technique for exhibiting relationships between information objects by using well known examples.

4.3.1 Association and Generalization

Figure 7 shows a university schema example described in [10] for IBM Universal Database. Two basic entities are PERSON and DEPT(department). University peoples come in various flavors, such as EMP(employee) and STUDENT. Within employees, there are again various flavors, such as PROF(professors)

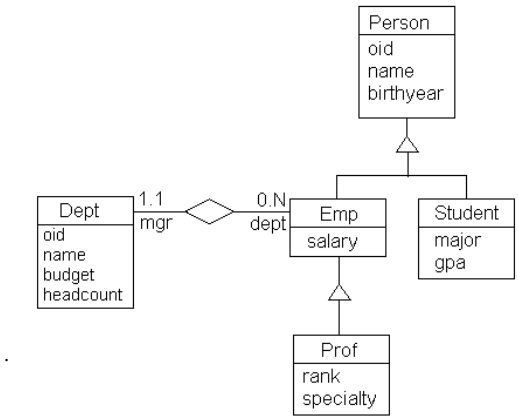

Figure 7. An example of association and generalization relationships

4.3.2 Aggregation

Figure 8 shows an example schema used in the manual of Informix Universal Server, which allows user-defined data types to be used as domains. The entity EMPLOYEE has four attributes, Eid, Age, Ename and Eaddr, where the domains of Ename and Eaddr are user-defined types NAME and ADDRESS. The entity ADDRESS has again four attributes Street, City, State, and Zip, where the domain of Zip is user-defined type ZIP.

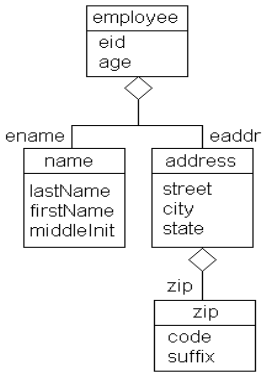

Figure 8. An example of aggregation relationships

4.3.3 Modeling of Structured Data through Advanced Aggregation

We can utilize aggregation relationships to represent complex structured data, such as multimedia documents and XML/SGML documents. Figure 9 shows an example schema which corresponds to the following DTD explained in [8].

<!ELEMENT article (author+, title, year?, (shortversion|longversion))>
<!ATTLIST article type CDATA>
<!ELEMENT author (firstname?, lastname)>

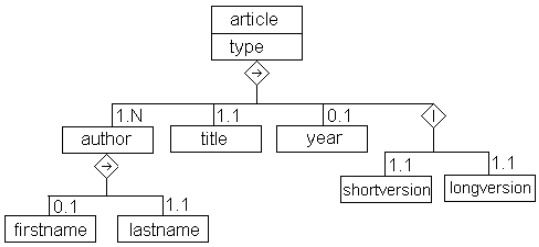

Figure 9. An example of sequence and choice aggregation relationships

4.3.4 Comprehensive Example

Figure 10 shows a combind schema, which extends Figure 7 to include information object PAPER, which is modeled as structured data using sequence aggregation and ordered set cardinality.

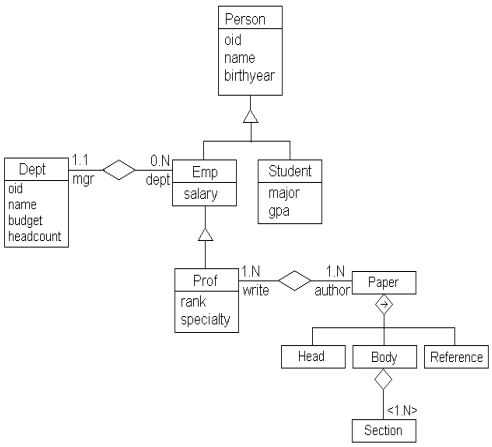

Figure 10. More comprehensive example

V. Derivation of Logical Data Models

This section describes how to transform conceptual data structures described in the unified data model into logical database designs, supported by popular logical data models, such as relational, object-relational and object-oriented data models, and XML/SGML databases.

5.1 Derivation of Relational Model

The transformation rules for the relational model are relatively well known. Each set values and each nested attributes, as well as each entities, are transformed into a relation, with inclusion of foreign key attribute(s) to dependent tables. Each HAS-A relationship is transformed into foreign key attribute(s) or an independent intersection relation, according to their relationship cardinality(1:1, 1:N, N:M). There are two or more design alternatives in representing IS-A relationships. Two of the possible methods are horizontal partitioning and vertical partitioning using one table for each entity as explained in [10]. For PART-OF relationships, all detail aggregation semantics are ignored. These relationships can be transformed by using the same method for 1:N HAS-A relationship.

5.2 Derivation of Object-Relational Model

The transformation methods are a little bit different according to the target object-relational platforms, because the features of each platforms are different. The major object features, such as OIDs, inheritance of type/table hierarchy, methods, user-defined data types, nesting of structured types, collection-valued attributes, and path expressions, are not fully covered by most of the existing object-relational DBMSs. For example, IBM UDB V5.2 supports user-generated OIDs and inheritance of table/view hierarchy, but does not support nesting of structured types and collection-valued attributes. Oracle 8 supports pointers(OIDs) by REF attributes, and collection types, such as array types and table types, but does not provide inheritance of table/view hierarchy. The simplest guideline for object-relational cases is to use the transformation rule of relational cases for certain features that are not directly supported by the target object-relational platforms.

Therefore, each set values and each nested attributes can be directly transformed into set-valued attributes and nested structured attributes on some platforms, or each of them will be transformed into a relation similar to the relational cases.

Each HAS-A relationships is directly represented as OID attribute(s) referencing related objects or an independent relation containing both OIDs referencing related tables(objects), or simply transformed into foreign key attribute(s) or an independent intersection relation, also according to their relationship cardinality. One of such detail transformation guideline for HAS-A relationships with cardinality 1:1, 1:N, N:M is proposed, on the platform directly supporting OIDs(REF attributes) and collection values[11].

There can be much more design alternatives in representing IS-A relationships, if they are transformed by following the transformation rules of relational case, because we can use OID attribute(s) instead of foreign key attribute(s) in vertical partitioning. But, if the system directly supports inheritance of type/table hierarchy, we don't have to worry about the alternatives of relational structures. Users and programmers can assume that the logical representation of a table hierarchy is in the form of one table for each entity in the IS-A hierarchy, but each table would contain all of the columns including inherited attributes. In such cases, the physical implementation structures are determined by the system. For example, IBM UDB represents one table hierarchy as a physical table, which is a union of the columns required to store rows of any subtables.

For PART-OF relationships, some detail semantics of tuple aggregation and sequence aggregation can be represented by utilizing OIDs, nesting of structured types, and collection-valued or

array-valued attributes.

5.3 Derivation of Object-Oriented Model

The mapping from conceptual structures to logical structures in object-oriented model is relatively simple and straightforward, because the object-oriented platform usually support important object features, such as OIDs, inheritance of class hierarchy, collection and set values. Each set values is directly represented as set-valued attributes. Each nested attributes can be directly represented(e.g., on O2) or represented as an independent class, with addition of a relationship or inclusion of an OID attribute to the parent class.

Each HAS-A relationship is directly represented, but usually bidirectionally. In the ODMG 3.0 standard, each relationship is denoted by using the keyword relationship with unique relationship name instead of OID attribute, in both of the related classes. Each IS-A relationship is directly supported by the system, resulting in one class per each entity in IS-A hierarchies. Each PART-OF relationship can be represented by using the same method for HAS-A relationships. In some systems, the detail meaning of aggregation, such as existential dependency and exclusive/shared ownership, is directly supported.

5.4 Derivation of XML/SGML Databases

It seems that the association and generalization relationships are less important or not useful in XML/SGML-related document databases. But, attribute or relationship cardinalities and various aggregations are deeply related with this platform. The mapping procedure is also simple and straightforward. Each information object is represented as an ELEMENT tag. The PART-OF relationship cardinalities, such as 0.1, 1.N, 0.N, are transformed into cardinality symbols ?, +, *, respectively. A sequence aggregation is represented by using XML sequence notation with comma(,) symbol and enclosing parenthesis. A choice aggregation is represented by using XML logical-OR notation with bar(|) symbol and enclosing parenthesis.

VI. Conclusion

We have proposed a unified conceptual data model by integrating major features of representative conceptual data models, such as the E-R model, the Semantic Object Model, and the Unified Modeling Language and enhancing some features. We can represent every conceptual database structures in terms of information objects and their relationships. Information objects, or just simply objects, represent entities, classes, and elements. Their relationships are association, generalization, and aggregation, where aggregation relationships are further refined to represent more complex PART-OF semantics of modern database applications. We have clarified how attributes, attribute-entity relationships, and attribute cardinalities are extended to entities, relationship cardinalities, and entity-entity relationships.

A special diagrammatic technique is also introduced for conceptual database designs. We also presented the transformation methods of conceptual structures described in the unified data model into popular logical structures. We strongly believe that this unified conceptual data model can be utilized in the conceptual database design stage for various logical database platforms, such as relational, object-relational, object-oriented, and (semi-)structured databases.

References

[1] Peter Chen, "The Entity-Relationship Model - Toward a Unified View of Data," TODS, 1(1),

pp.9-36, March 1976.

[2] David M. Kroenke, The Entity-Relationship Model(Chapter 3), Database Processing(8th Ed.), Prentice Hall, pp.51-78, 2002.

[3] C. Batini, S. Ceri, and S.B. Navathe, Conceptual Database Design: An Entity Relationship Approach, Benjamin Cummings, August 1991.

[4] David M. Kroenke, The Semantic Object Model(Chapter 4), Database Processing(8th Ed.), Prentice Hall, pp.79-117, 2002.

[5] Booch, Jacobson, and Rumbaugh, The Unified Modeling Language User Guide, Addison Wesley, 1999.

[6] Yunmook Nah and Sukho Lee, "Object-Relationship Model for Conceptual Modeling of Multimedia Data," Advanced Database Research and Development Series, Vol.3, World Scientific, pp.125-132, 1992.

[7] Yunmook Nah and Sukho Lee, "Two-level Modeling Schemes for Temporal-Spatial Multimedia Data Representation," in Proc. of Int'l Conf. on Database and Expert Systems Applications(DEXA), Springer-Verlag, Spain, Valencia, pp.102-107, Sept. 1992.

[8] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu, "A Query Language for XML," www.research.att.com/~mff/files/final.html.

[9] Jayavel Shanmugasundaram, et al., "A General Technique for Querying XML Documents using a Relational Database System," SIGMOD Record, 30(3), pp.20-26, Sept, 2001.

[10] Michael Carey, et al., "O-O, What Have They Done to DB2?," in Proc. VLDB, pp.542-553, 1999.

[11] Christian Soutou, "Modeling relationships in object-relational databases," DKE, 36(1), pp.79-107, Jan. 2001.

저 자 소 개

羅 然 默 (正會員)

1964년 1월 19일생.
1986년 서울대학교 컴퓨터공학과 공학사, 1988년 서울대학교 컴퓨터공학과 공학석사, 1993년 서울대학교 컴퓨터공학과 공학박사. 1991년 IBM T. J. Watson 연구소 객원연구원, 2001년~2002년 University of California, Irvine 객원교수, 1993년~현재 단국대학교 전기전자컴퓨터공학부 부교수.
주관심 분야 : 데이타베이스, 데이타 모델링, 멀티미디어 정보 검색, 이동 객체 데이터베이스.