Title	Generalized Object Oriented Data Model for Multi-Media Data
Author(s)	KAMBAYASHI, Yahiko; ARIKAWA, Masatoshi
Citation	数理解析研究所講究録 (1988), 655: 52-63
Issue Date	1988-04
URL	http://hdl.handle.net/2433/100514
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
第655巻 1988年52-63

Generalized Object Oriented Data Model for Multi-Media Data

マルチメディアデータのための オブジェクト指向データモデルの一般化

Yahiko KAMBAYASHI and Masatoshi ARIKAWA
上林弥彦　　有川正俊
Dept. of Computer Science and Communication Eng.
Kyushu University
九州大学工学部
Fukuoka 812, Japan

Abstract

An object oriented data model is considered to be a suitable model for multi-media databases. Since object oriented concept was first introduced for programming languages, we need to modify the concept in order to handle databases. In this paper, we will discuss representation of various relationships and inheritance of the model. In conventional object oriented model only one relationship "IS-A" is considered and inheritance of values from parents to children (top-down) is used. In order to handle more than one relationship in multi-media data, we have introduced multiple-relationship object hierarchy. As in databases inheritance from children to parents (bottom-up) is also important, general version of inheritance is also discussed, both top-down and bottom-up. Concepts introduced in this papers are considered to be useful to realize various views in multi-media databases.

1. Introduction

Recently there have been a lot of efforts to realize multi-media databases. One promising data model for such databases is the object oriented data model. To realize a multi-media database system, we need to handle problems which do not appear in conventional database systems. Among these problems, we have a

problem of expressing complicated relationships in data as well as a problem of handling large amount of data. In the object oriented approach, usually one relationship IS-A is used, which may not be enough to handle complex data. Inheritance is a useful property to reduce the amount of data as well as to express data relationships economically. We will discuss how to express more than one relationship in one hierarchy and various kinds of inheritance in this paper.

In database model classes are selected as proper units of data processing. Depending on the application different sets of classes are required. Each class has instances. In order to handle relationships among instances, it is better to handle instances as classes. For some applications union of classes can be regarded as a class. To handle such class abstraction together with hierarchies among objects in the same abstraction level, we have introduced a concept of multiple-relationship object (class) hierarchy. In this hierarchy more than one relationship satisfying conflict-free property, is permitted.

One important property of object hierarchies is inheritance of values and methods. In conventional object oriented model inheritance from a parent to its children is considered. We can generalize the inheritance such that each child inherits a different value, by computing the value using the parent value and child's key values. Besides inheritance from parents to children (top-down), bottom-up inheritance is also useful in databases. General definition of bottom-up inheritance is also given.

In Section 2 basic definitions on object oriented models are given. Multiple-relationship object hierarchies and generalized inheritance are discussed in Sections 3 and 4, respectively.

2. Object Oriented Models

Database systems have been attempting to increase their power by associating more functionality with the data. In particular, object oriented databases may

incorporate notions of type, data abstraction, and inheritance. These idea have been studied extensively in the programming language domain.

Smalltalk is a general purpose programming language, and its basic components are as follows.

(1) classes

(2) instances

(3) class / instance variables

(4) class / instance methods

(5) IS-A relationships

Each class is an object to describe a set of instances of the same type. The description of class C contains definitions of instance variables, class variables, instance methods, and class methods. The method is a procedure describing how to perform one of object's operations, and is shared by all the instances of a class and owned by the class.

Between two classes, the IS-A relationships can be defined. A hierarchy on the set of classes is defined by IS-A relationships. Meaning of IS-A is shown in Example 1. If class C_1 IS-A class C_2, then the variables and methods of class C_2 is automatically inherited to the class C_1. Basically, Smalltalk program is a collection of message expressions, each of which consists of an object (receiver) followed by an appropriate message. According to the message sent to a receiver object, an appropriate method of the receiver or the inherited method is executed.

The notation of class is different in database and language communities. In the language view, a class means a data type, that is the definition of the behavior and structure of a set of objects as the above. In the database view, a class is used to refer to the set of objects themselves rather than a definition. A classes is used as a primary mean of grouping objects and searching for objects. Thus database systems should support these mechanisms for grouping objects into collections automatically. These mechanisms will reduce the burden of users.

In order to simplify discussion we will omit inheritance of methods, and only inheritance attribute values are considered in this paper.

We assume that there are n attributes, A_1, A_2, ..., A_n. Each object O_i is defined by an ordered set of these attribute values. If the value is not defined we use # to denote the null value. The value of attribute A_j of object O_i is expressed by $v_i(A_j)$. Partial ordered set p_i defines a hierarchy on the object set. Here p_i need not to be IS-A relationship.

We permit a set of values as an attribute value in order to handle inheritance of various kinds. Especially multiple inheritance from more than one parent object (see Example 1) or from more than one child object (see Definition 7) is important.

[Definition 1] Object hierarchy is defined by a graph $H_i=(V_i, E_i, p_i)$, where V_i is a set of vertices each of which corresponds to an object, E_i is a set of edges defined by partial order p_i. There is edge e_{jk} from vertex O_j to vertex O_k if and only if $O_j p_i O_k$ is satisfied. In this case O_k is called a parent of O_j, and O_j is called a child of O_k.

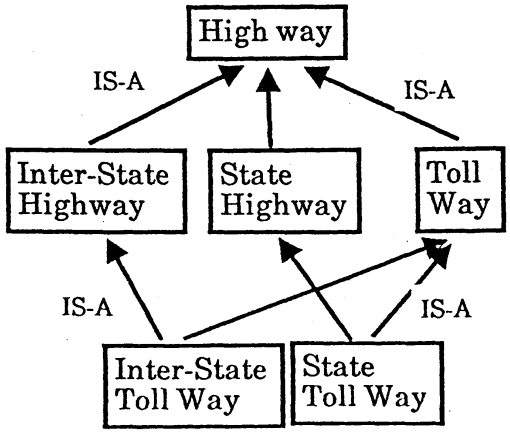
[Definition 2] Assume that there are m child objects O_{j1}, O_{j2}, ... , O_{jm} for a parent object O_k. We define inheritance on attribute A as follows.

Values of A in O_{ji} ($i=1, ..., m$) are determined identically

by the value of A in O_k.

A minimum set of attributes which uniquely determines the object is called a key. This definition is similar to the key for relational databases with universal relations.

How to express data by an object hierarchy is shown in the following examples.



[Fig.1 An example of an object hierarchy]

[Example 1] Fig.1 shows an example of such hierarchy. Here we assume that there is only one attribute and p_i is 'IS-A' relationship. Some examples are as follows.

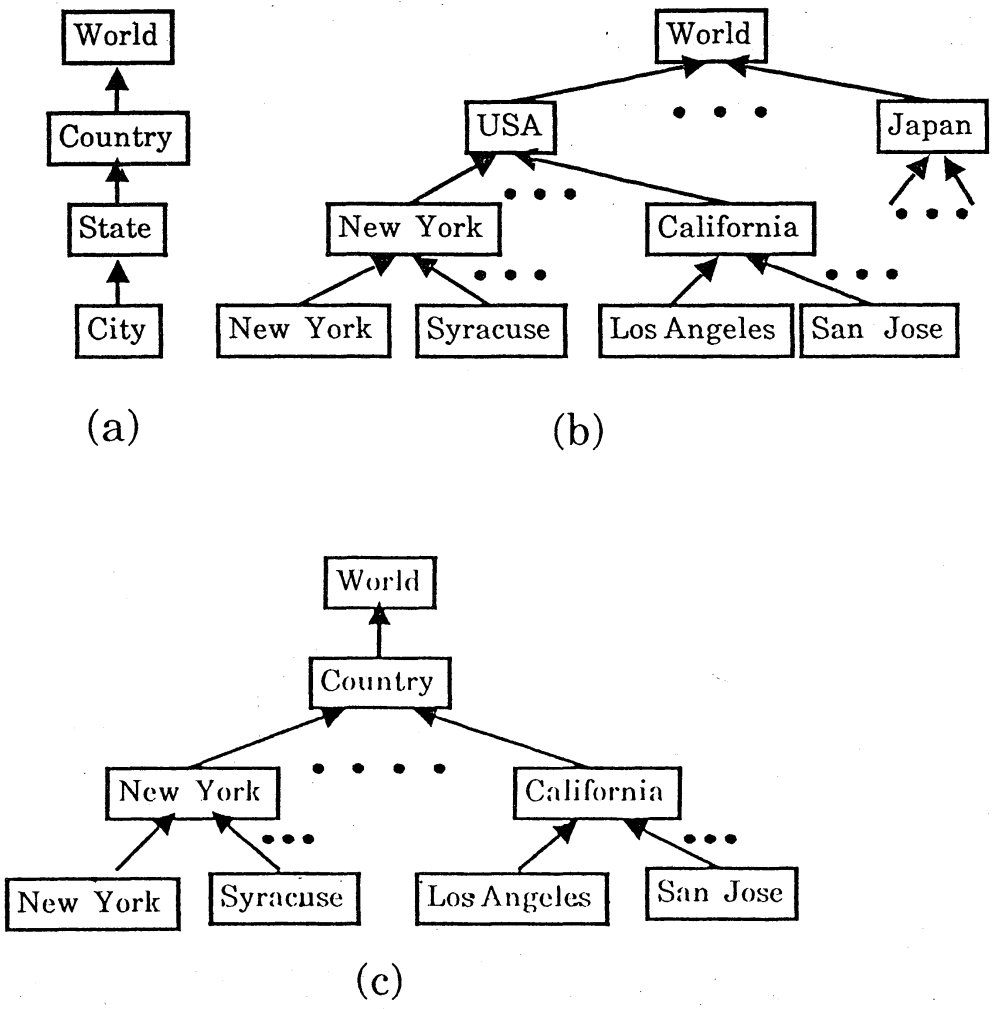
Inter-State Highway IS-A Highway.

State Highway IS-A Highway.

Toll way IS-A Highway.

For each class there are instance values. For example, Highway 101 is an instance of Inter-State Highway. Depending on applications, the set of instances of class "Inter-State Highway" can include the set of instances of class "Inter-State Tollway", or can exclude it.

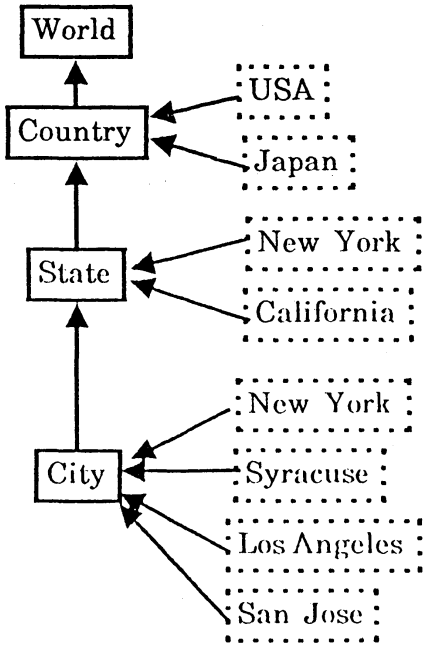
For example if the maximum speed of Highway 70 miles/hour, it should be satisfied by all highways. We only need to specify the values at the top value and the value is inherited to others. Inter-State Tollway has some properties (for example, the supporter is the government) of Inter-State Highway and some properties (for example, we must pay for the use) of Tollway, such a case is called multiple inheritances.



[Fig.2 Selection of objects]

[Example 2] There are many ways in order to express the same set of data depending on the selection of objects. Fig.2 shows such an example. If we select "world", "country", "state" and "city" as objects, the hierarchy shown in Fig.2 (a) is obtained. We can use smaller units as objects as shown in Fig.2 (b). Depending on the applications, a proper hierarchy is selected. Furthermore, we may need mixture of the both as shown in Fig.2 (c). In this case it is assumed that detailed treatment on states and cities are required. Another method of handling such

case is shown in Fig.3, which is commonly used. For objects "state" and "city" we have values called instances of the objects. A problem of this approach is that the relationships on instances ("California" consists of "Los Angeles", "San Jose", etc.) are not expressed explicitly.



[Fig. 3 Objects with instances]

3. Object Hierarchies with Multiple Partial Orders

In the object oriented data model usually only one kind of partial order is used. For database systems we can define hierarchy with multiple partial orders. In this section we will discuss a hierarchy with more than one partial order. Such a hierarchy is expected to be useful to support various kinds of database views.

[Definition 3] For partial order p, the closure p^* is defined as follows.

(1) If $O_j p O_k$ then $O_j p^* O_k$.

(2) If $O_j p^* O_k$ and $O_k p^* O_h$ then $O_j p^* O_h$.

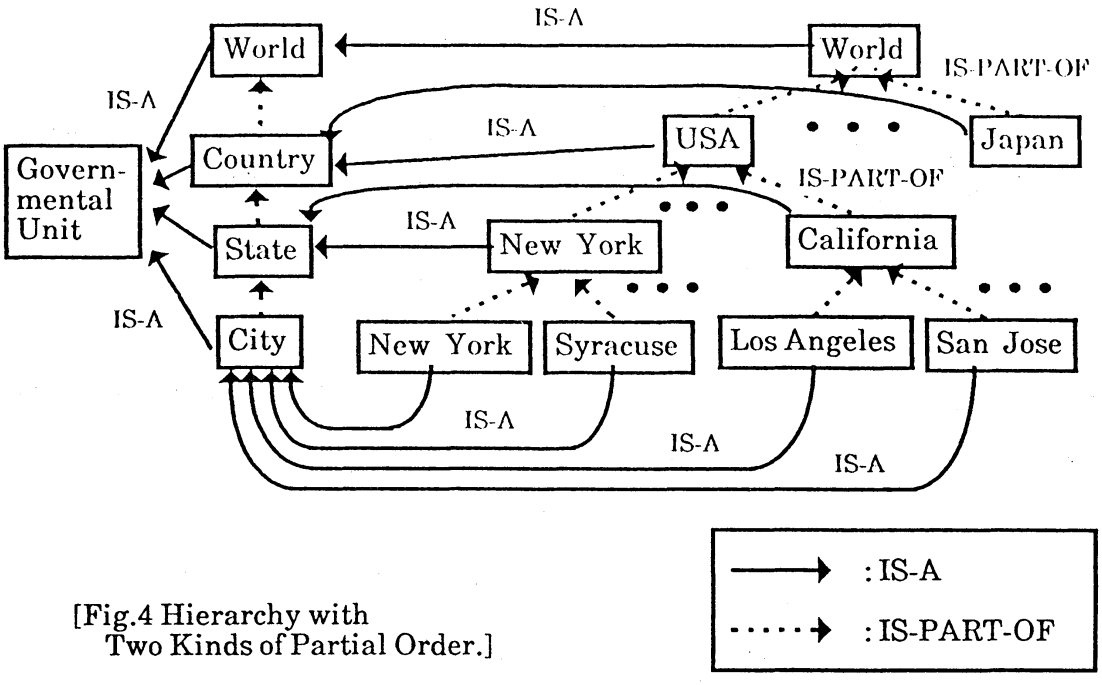
(3) Repeat (2) until no further application is possible.

[Definition 4] Two partial orders p_i and p_j are said to be conflict-free if p_i^* does not contain $O_k p_i^* O_h$ such that $O_h p_j^* O_k$. A set P of partial orders is said to be conflict-free if it is conflict-free for any pair of partial orders in P.

In one hierarchy we may need to use more than one kind of partial order. Thus we define multiple-relationship object hierarchy as follows.

[Definition 5] Multiple-relationship object hierarchy is defined by a graph $M_i=(V_i, E_i, P_i)$, where V_i is a set of vertices each of which corresponds to an object, E_i is a set of edges defined by at least one partial order contained in conflict-free set P_i of partial orders. The name of the partial order is used as a label of each edge.

Examples with two kinds of partial orders are shown as follows.



[Fig.4 Hierarchy with Two Kinds of Partial Order.]

[Example 3] Fig.4 shows a hierarchy with two kinds of partial orders. Two hierarchies in Fig.2 (a) and (b) are combined by IS-A relationships. As discussed in Section 2, we have a lot of views for one hierarchy due to the difference of defining objects. By using Fig.4 we can realize various views by just taking a subgraph. Furthermore, various kinds of inheritance can be also expressed, such as inheritance in IS-A relationships, inheritance in IS-PART-OF relationships of Fig.2 (a) and inheritance in IS-PART-OF relationships in Fig.2 (b).

Multiple-relationship object hierarchy is necessary when we need to distinguish partial orders. For example in $M_i=(V_i, E_i, \{p_1, p_2\})$ we can distinguish p_1 and p_2, while in $H_i=(V_i, E_i, p_1 \cup p_2)$ p_1 and p_2 cannot be distinguished. In the both cases p_1 and p_2 are required to be conflict-free. Here $p=p_1 \cup p_2$ is defined as follows.

$O_i p O_j$ if $O_i p_1 O_j$ or $O_i p_2 O_j$.

4. Generalization of Inheritance

In conventional object oriented models for programming only one kind of inheritance (from parents to children) is considered. In database models aggregation which can be regarded as inheritance from children to parents is also important. We will generalize such inheritance in this section. The following notations to define general inheritance.

①$v_i(A)$ is the value of attribute A in object O_i.

②S_1 and S_2 are the predetermined subsets of $\{A_1, A_2, ..., A_n\}$.

③$v_j(S_i)$ corresponds to a vector $(v_j(A_{i1}), v_j(A_{i2}), ..., v_j(A_{ik}))$
when $S_i=\{A_{i1}, A_{i2}, ..., A_{ik}\}$.

④f is a predetermined function to calculate a value.

⑤The domain and the range of function f can be set values satisfying the following property.

$$f(a \cup b, x) = f(a,x) \cup f(b,x)$$

where a and b are values (or set values) for some attribute and x shows other values. Thus f for set values is defined by f for simple domain values by applying the above equation recursively.

[Definition 6] A general version of inheritance from parents to children in object hierarchy determined by partial order p is as follows.

Type D (down): The value of attribute A_k in a child object is determined by the values of attributes in it and its parent object.

If $O_i p O_j$ then $\{f(v_j(S_1), v_i(S_2))\} \subseteq v_i(A_k)$, $\quad A_k \notin S_2$

Set inclusion is used since both f and $v_i(A_k)$ can be set values.

The above definition is very general. There are many useful special cases. For example, the conventional inheritance is defined as follows, which is a special case of Type D.

Type D1: If $O_i p O_j$ then $\{v_j(A_k)\} \subseteq v_i(A_k)$

The value of A_k in parent object O_j determines the value of A_k in its child object O_i. $v_i(A_k)$ is a set value since O_i can have more than one parent in the hierarchy determined by p.

A simple generalization of Type D1 is that $v_i(A_k)$ can be computed from $v_j(A_k)$, that is

Type D2: If $O_i p O_j$ then $\{f(v_j(A_k))\} \subseteq v_i(A_k)$.

Furthermore we can use value of more than one attribute.

Type D3 If $O_i p O_j$ then $\{f(v_j(S_1))\} \subseteq v_i(A_k)$

In these cases, the value of $f(v_j(A_k))$ or $f(v_j(S_1))$ is identical for any child object of O_j. If we want to change the value for different children, we need to add some attribute values of each child object. Thus the definition of Type D is obtained.

[Definition 7] A general version of inheritance from children to parents in object hierarchy determined by a partial order p is as follows.

Type U (up): The value of attribute A_k in a parent object is determined by values of attribute in it and its child object(s).

If $O_{i_1} p O_j$, $O_{i_2} p O_j$, ..., $O_{i_m} p O_j$ (O_j has m children O_{i_1}, ..., O_{i_m}),

then $\{f(v_1(S_1), v_{i_1}(S_2), v_{i_2}(S_2), ..., v_{i_m}(S_2))\} \subseteq v_j(\Lambda)$, $A_k \notin S_1$.

Type U is a new kind of inheritance introduced in this paper. A simple case is as follows.

Type U1: If $O_{i_1} p O_j$, $O_{i_2} p O_j$, ..., $O_{i_m} p O_j$ (O_j has m children O_{i_1}, ..., O_{i_m}),

then $\{f(v_{i_1}(\Lambda), v_{i_2}(\Lambda), ..., v_{i_m}(\Lambda))\} \subseteq v_j(\Lambda)$.

Two typical cases of type U1 are given as follows.

(1) f is an addition operation.

(2) f is a set union function.

Examples of these two cases are as follows.

[Example 4] Let O_i corresponds to a state and its child objects correspond to the counties of the state. If the value of attribute A is population, we can compute the value for O_j by summarizing all values of its child objects.

[Example 5] Let O_j and its child objects correspond to a state and its counties as shown in Example 4. If the value of A shows factories in the area, then the factories in a state is the union of factory sets in its counties.

5. Concluding Remarks

In this paper we have introduced two new concepts; multiple-relationship object hierarchy and generalized inheritance. These concepts are especially important for multi-media database systems in order to support various views. Applications to such systems are currently investigated.

References

[1] A.Goldberg and D.Robson, "Smalltalk-80 : The Languages and Its Implementation," Addition-Wesley, 1983.

[2] B. Toby, "Issues in the Design of Object-Oriented Database Programming Language," In OOPSLA'87, Conference Proceedings, SIGPLAN, October 1987.

[3] D., Maier, et al, "Development of an Object-Oriented DBMS," In OOPSLA'86, Conference Proceedings, SIGPLAN, September 1986.

[4] J. Banerjee, H. Chou, J. Grarze, W. Kim, D. Woelk, "Data Model Issues for Object-Oriented Applications," ACM Transactions on Office Information Systems, April 1987.

[5] Smith, J. and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems, vol. 2, no. 2, 1977.