The Structure and Management of Object Data

7C−2

Yao Zhuojun
University of Tokyo

Takashi HAMADA
National Center for Science Information Systems

1 Introduction

For an object-oriented database(OODB) system, the object data management is an important issue because there are two types of data, i.e., attributes and methods in OODB, rather than only one type of data(attributes) in traditional database. How to manage object data, particularly, how to manage methods is an issue of developing OODB systems.

The methods in OODB have two feature. One is *passive*, which is treated like ordinary data. Another is *active*, i.e, the methods are used to compute the attributes. Around this issue in this paper, we, first, point out it's possible to describe three relations among objects with the link structure, from which some benefits can be gotten. We, then, propose an approach to describe object behaviors when being treated using FSM(*Finite State Machine*) mechanism, and indicate the relations among objects can be transferred to the relation of *requesting* and *providing* of service. With this approach, the *active feature* of methods can be reflected.

2 The Link Structure

In OODB, depending on the semantic of the relation among objects or classes which objects belong to, there are three basic relations(see Fig.1). The first is reference, e.g., the attribute *manager* of the object generated from the class **Group** refers to the object generated from the class **Person**. The Second is aggregation, which also is reference relation. But for aggregation an object may refer to some objects which may belong to different classes, e.g., the attribute *documents* of `project` object refers to three object-lists(*contacts, plans, manuals*). Above two relations can be simply described with *link structure*. The third is inheritance. In this section we present an approach to solve how to use *link structure* to describe the relation between two objects that their classes have the inheritance relation.

If two classes have inheritance relation, there is a part of data which is the object of the ancestor in all of the children's objects. For example, the class **Author** inherits from the class **Person**, so in `author` objects there are data of `person` objects(Fig.2(a)). The part of data may be taken out from the `author` objects, and preserved independently using *link structure*(see Fig.2 (b)). The *link structure* is very simple, just one pointer is used. But we can get some benefits from it.

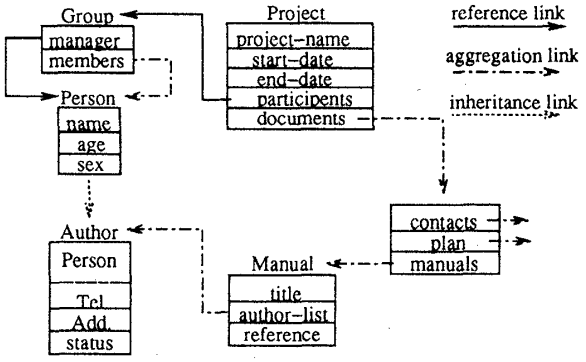

Figure 1: An Example of Classes' Hierarchy

- To reduce duplicate data. For example, if a person is an author of a manual, the `person` object data is a part of the `author` object data. Concurrently, the same person may be a member of a `group` object. For example, in the case of general(Fig.2(a)), double of the some `person-j` object data is preserved. But in the case of using *link structure*(Fig.2(b)), just one is preserved. The `group-k` object and the `author-i` object use a pointer to link the `person-j` object, respectively.
- When updating the `person-j` object, in the case of Fig.2(a) for the consistency, two object data(to indecate the same person) must be modified simultaneously, but in the case of Fig.2(b), just one object data is need to be modified.

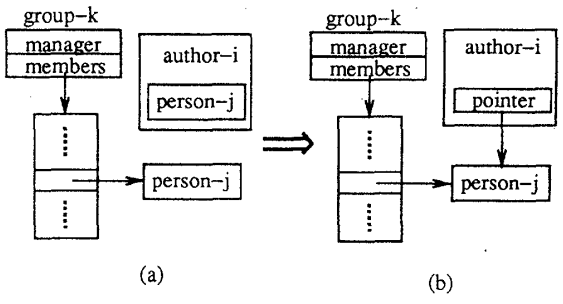

Figure 2: Transformation of data Structure

Of course, in the case of Fig.2(b) it is need to record

The Structure and Management of Object Data
Yao Zhuojun, University of Tokyo.
Takashi HAMADA, National Center for Science Information Systems.

the information of the *meaning of link*, .i.e. which is about reference or about inheritance.

3 Objects Implementation

Although the attributes of an object can be simply preserved because it can be described with integer, string and so on, the methods cannot because methods in OODB have two aspects, i.e., entry descriptions and executive functions. We use a method on which the entry descriptions of methods are preserved like the attributes and the executive functions of methods are organized into a function library, which are dynamically loaded and linked when used. For this constitution in this section, we discuss how to implement object data after retrieved, particularly execute methods.

For OODB, the searched targets are objects. For general consideration, the result of searching can be described as an object-set like the following.

$$Retrieve = \{O | O \in SearchDom\}$$

If the searching is successful, in accordance with object-oriented concept, the retrieved objects should be treated using the methods defined in the classes by designer of the OODB one by one(see Fig.3).

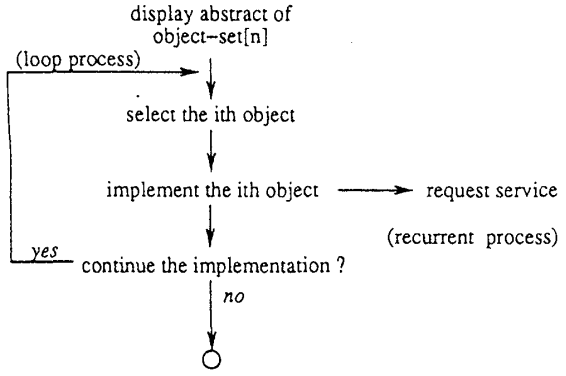

Figure 3: Object Implementation Process

An object in implemented has many states. The behaviors of an object(regarded as small system) are described by state transition[1]. So we can define these states an FSM as the following :

$$FSM = (O_s,\ E,\ M,\ S)$$

- O_s is a set of the object-states which can be viewed or not to users. For example, while a user is treating a document object, "showing title", "changing page", and "referring other documents" etc. may be regarded as the different states. Of course, the number of states of an object is finite.

- E is a set of events. When treating an object, the user, perhaps, has some requests, e.g., "let me see the next page", "tell me about the authors" and so on. These requests may be regarded as the external events when the object is implemented, and told to system using mouse to select items on screen for example.

- M is a set of methods mapping from Cartesian product $O_s \times E$ to the O_s. For every event from the user's requests, there are some methods in the relevant class to implement it, that is, $\forall e_i \in E \longleftrightarrow \{m_{ij} \mid m_{ij} : O_s \times E \longrightarrow O_s; j = 1, \cdots\}$.

- S is the first state of the object when it is treated, say start state($S \in O_s$).

All of the events in the E and the methods in the M are defined by the designer of the OODB. After a request from the user is inputed as an *event*, the system action which transfer current object's state to next is, really, to call a method prepared in the class in advance. This is regarded as *requesting and providing relation of service* between two states.

If the *requesting* is from an object to other, there are two manners of requesting.

Explicit requesting. For this manner, the accessing functions provided by the system is used to access the requested object directly, then the relevant method of the object is called to get service.

Implicit requesting. The requesting side saves current state of oneself, transfers controlling right to the system. Then like ordinary searching, the system searches requested object from OODB and the FSM of the requested side is started. When the requested object implementation is over, the system retrieves the previous state of the requesting object.

Besides, There are two forms of treating the *requesting* of service from an object to other, following the pointer between the two objects. One may be seen navigation another non-navigation. They are relevant to *implicit requesting* and *explicit requesting*, respectively.

4 Conclusions

An object is a real-world entity. This is not only regard as a concept of the object-oriented model, but also persists in implementation. Above we discussed how to realize it using *link structure*, *FSM mechanism* and the relation of *requesting-providing* of service.

References

[1] Derek Coleman, Fiona Hayes, and Stephen Bear, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 18, No. 1, January 1992, pp 9-18