データベースシステム 90－3
（1992. 9. 11）

データベース管理システム「沙羅」の機能拡張

石丸 知之　植村 俊亮
東京農工大学 工学部 電子情報工学科

われわれは、オブジェクト指向データベース管理システム「沙羅」を試作中である。その沙羅のデータモデルの拡張について述べる。
最初の拡張は、実世界の存在に対するオブジェクトの表現を複数使用するために異なったクラスのオブジェクトをひとつのオブジェクトに結び付ける機構を取り入れた。次は、継承の種類のひとつの制約継承をデータモデルに取り入れた。これは、スーパークラスのインスタンスのうちで条件を満たすものをインスタンスとしてもつクラスを作成する機能である。属性の参照の種類として、挿入属性に、自動（AUTOMATIC）、保留属性には、永続（MANDATORY）と固定（FIXED）を導入した。

Data Model Enhancement of Object-Oriented Database Management System Sarah

Tomoyuki Ishimaru　Synsuke Uemura

Faculty of Technology, Tokyo University of Agriculture and Technology

There is no standard data model for object-oriented database system. And it is known that a simple object-oriented data model is not sufficient to support multi media data and complex data. We investigate data model of Sara; an object-oriented database management system under development at the Tokyo University of Agriculture and Technology. Our extensions of data model are multi-class instance, restricted class and constraint of attribute. Also we describe implementation strategy.

1. はじめに

文書処理、ＣＡＤプログラム、マルチメディア処理、そしてハイパーテキストで代表される最近の応用プログラムは、高度なデータ管理を実現している。

これまでのデータベースシステムは、対象とした応用プログラムが主に事務処理であったため、このような最近の応用や環境にふさわしいものとは言えない。

われわれは、高度な応用プログラムの基盤となるような、マルチメディアの情報処理のためのデータベース管理システム「沙羅」を作成中である[1]。われわれが実際に作成する理由は、アイデアを確認するために変更できるシステムが必要であること、実際に試作することでデータモデルやシステム構成の上での問題が明らかになることを期待しているからである。

沙羅の最初の設計（第１版）では、基本と考えられる機能だけを取り入れた。その設計の途中や実現で、モデリング能力の不足が発見できた。沙羅の第２版では、この経験に基づき、データモデルを強化する。本稿では、沙羅の第１版のデータモデルとデータモデルの拡張について述べる。

2. データモデル

沙羅のデータモデルは、オブジェクト指向データモデル[2]に分類できる。以下では、沙羅の第１版のデータモデルについて説明する。

データモデルの設計においても、必要最小限の機能だけを取り入れることにしたため、他のデータモデルと比較して際だった特徴はない。

2.1 オブジェクト

沙羅の中の「オブジェクト（object）」は、現実の世界の実体（entity）に対応する。実体は沙羅の中ですべてオブジェクトとして存在する。このオブジェクトは、沙羅が生成するオブジェクト識別子（object identifier）、内部データを表現する属性（property）、そしてオブジェクトに関係した手続きであるメソッド（method）からなる。

2.2 クラス

属性の宣言とメソッドの定義が同じオブジェクトの集合を「クラス（class）」と呼ぶオブジェクトで管理する。

クラスは、属性の構造が同じでメソッドも同じオブジェクトの定義を保持する機能と、属するインスタンスの全体を管理する機能をもつ。

2.3 継承

クラスは、指定したクラスの属性の宣言やメソッドの定義をそのまま利用する機能をもつ。いわゆる「継承（inheritance）」である。沙羅では多重継承を許している。

2.4 検索

オブジェクトは、そのオブジェクトのクラスから検索できる。検索では、条件を設定し、その条件を満たすインスタンスだけを選択することができる。検索条件を満たしたインスタンスは、指定によって、新しいクラスまたは「集合（set）」オブジェクトの要素とすることができる。

集合オブジェクトは、異なるオブジェクトの集まりを管理する機能をもつ。しかし、クラスと異なり、オブジェクトの振舞いや内部構造を定義する機能はない。

検索は、単独クラスの、またはそのクラスをスーパークラスとするクラス階層を対称とすることができる。

2.5 システム定義クラス

沙羅の中には、「システム定義クラス」と呼ぶクラスがいくつか前もって定義してある。このクラスは、オブジェクトの表現のための基本的なオブジェクト、オブジェクト管理のためのオブジェクトの定義が記述してある。システム定義クラスを書き換えることはできない。

3. データモデルの追加機能

第１版の沙羅のデータモデルでは以下のような問題がある。

(1) 一つの実体に対する異なる表現をうまく取り扱えない。一つの実体も視点によって、複数の見え方がある。
(2) 継承の種類の一つの「制約（constraint）」がうまく扱えない。
(3) 参照が単純すぎる。Ｃのポインタと同じレベルである。

沙羅の第２版でのこれらの問題を解消するためにデータモデルを拡張した。以下では問題点と沙羅での解決方法について述べる。

3.1 マルチクラスインスタンス

通常の、オブジェクト指向データモデルでは、インスタンスはただ一つのクラスに属する。

オブジェクトは、実世界の存在のデータベース内での表現である。増永[3]が主張するように、実世界の存在をデータベースに写像する方法は、唯一ではない。たとえば、実世界の人物の表現には、写真のようなイ

メージによるものと、履歴書のような文字データによるものが考えられる。これらのデータは、一つの実世界の実体の異なった表現である（図 1 参照）。

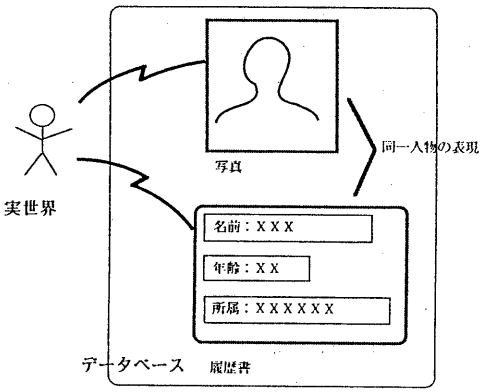

図 1. 実世界の複数の表現

オブジェクトの存在について一貫性をとらなければならないこともある。たとえば、その人物の情報がデータベースから取り除かれるときには、イメージによる表現と文字による表現を同時にデータベースから取り除くようなことである。

このように実世界の存在に対するオブジェクトの表現を複数使用するために、沙羅では異なったクラスのオブジェクトを一つのオブジェクトに結び付ける機構を導入した。これを「マルチクラスインスタンス」と呼ぶ。一つに結び付けられたオブジェクトは参照にクラスを引数にもち、そのクラスによって参照する。このクラスによって見える範囲を「視野」と呼ぶ（図 2 参照）。

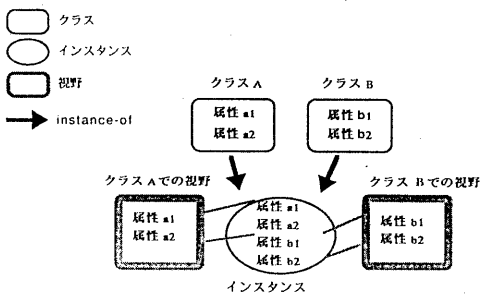

図 2. マルチクラスインスタンス

たとえば、イメージによる表現と文字による表現の二つのインスタンスを一つに結び付けたインスタンスは、参照時の指定によって、クラス「写真」が視野となったり、クラス「学生」が視野になったりする（図 3 参照）。

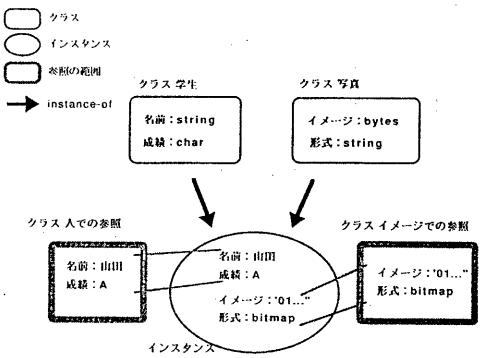

図 3. 人物の情報の表現

視野は、関係型データベースのビューとは異なる。ビューは、スキーマの部分として定義されるのに対して、視野は結び付けられたインスタンスの見える範囲を設定しているからである。

このようなインスタンスは、属するクラスが保持する属性をすべてもつ。ただし、それらクラスが共通のクラスから属性を継承してるときには、インスタンスでは、一つの属性を共有する。つまり、視野に共通の部分が存在する。（図 4 参照）

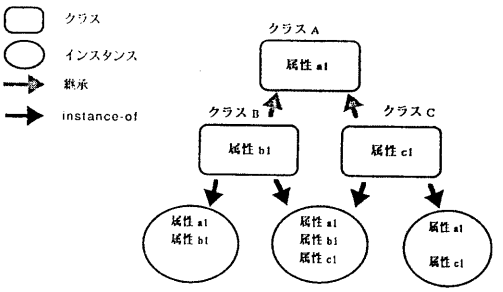

図 4. クラス階層と視野

マルチクラスインタンスは、多重継承と異なっている。多重継承では、クラス定義で複数のクラスの性質を引用している。また、必要に応じてスーパークラスの属性やメソッドを利用した新しい性質を定義できる。

それに対してマルチクラスインスタンスでは、インタンスの構造を一つのオブジェクト内に格納するだけで、新しく性質や意味を追加することはできない。しかし、クラス階層の操作したりや新たなクラスの作成したりせずに、インスタンスの表現を拡張することができる。

以下では、沙羅でのインスタンスの操作の例を示す。

まず、インスタンスをほかのクラスに属するために、インスタンスの識別子を引数として、新しいクラスのインスタンスを作成する。

aPerson = STUDENT.new	aPerson をクラス「学生」として作成する。
aPerson.degree = 'A'	aPerson の成績を設定する
aPerson = PHOTO.new_with(aPerson)	aPerson をクラス「写真」のインスタンスとする。
aPerson.image = file("#3902")	aPerson にイメージデータを設定する

インスタンスの視野を、特定のクラスに設定するときには、そのクラスの識別子を引数として渡す。

((PHOTO)aPerson).image	aPerson を写真として参照する
((PERSON)aPerson).degree	aPerson を人物として参照する
delete(aPerson)	aPerson の存在をデータベースから消去する

メソッドも、属性と同様に、インスタンスが属した順にクラスを検索する。また、特定のクラスからのインスタンスを視るときには、指定したクラスに属しているとして、メソッドを検索する。

インスタンスが複数のクラスに属すると、性質や意味の異なるオブジェクトを一つのインスタンスにまとめ、視野によって選択することができる。反面、参照（呼び出し）している属性（またはメソッド）を定義しているクラスは、インスタンスの作成方法で動的に決まるので、視野が有効かどうかは動的に決まる。たとえば、クラス「写真」のインスタンスはクラス「人物」のインスタンスと結び付けられたり、クラス「家」のインスタンスに結び付けられる。結び付けられたインスタンスによって属性やメソッドが異なる。そのため、インスタンスのクラスを指定するアクセス方法では、インスタンスの属するクラスがあらかじめわかっている必要があり、オブジェクト指向の特徴である「動的結合」が利用できなくなる。

3.2 制約継承

オブジェクト指向データベースで提案されている継承の方式は、以下のように 4 種の継承に分類できる。[3]

(1)代用（substitution）継承

「もし、型 t のオブジェクトに対して、型 t' のオブジェクトに対してより多くの操作を実行できるなら、型 t は型 t' を継承しているという。」

(2)包含（inclusion）継承

「型 t のすべてのオブジェクトが型 t' のオブジェクトであるとき、型 t は型 t' の部分型という」

(3)特殊化（specialization）継承

「型 t のオブジェクトが型 t' のオブジェクトでもあり、しかもより詳細な情報をもっている場合、型 t は型 t' の部分型であり、特殊化の継承を行っているという。」

(4)制約（constraint）継承

「ある型 t' の要素で、ある制約条件を満たすオブジェクトのみからなる型 t を t' の部分型という。」

これらの継承の方式の最初の 3 つは、オブジェクト指向データベースでは、これまで自然に取り入れられている。

代用継承はクラス t' をスーパークラスとするクラス t にメソッドを追加することで、包含継承はサブクラスが属性を追加することで、また、特殊化継承は、サブクラスがメソッドと属性を追加することで表現できる。

残った制約継承を沙羅のデータモデルに取り入れる。そのために、スーパークラスのインスタンスのうち条件を満たすものをインスタンスとしてもつクラスを設定できるようにする。このようなクラスを「制約クラス」と呼ぶ。

制約クラスは、指定したスーパークラスと比較して、メソッドや属性の追加や変更はない。しかし、設定された条件によって、スーパークラスのインスタンスの集合の部分集合をインスタンスとする（図 5 参照）。

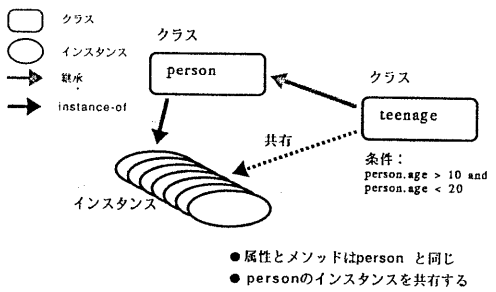

図 5. 制約クラスとインスタンス

制約クラスのインスタンスは、スーパークラスと制約クラスの複数のクラスに所属する。これは、3.1 で述べたマルチクラスインスタンスの機能を内部的に利用している。

制約クラスは、通常のサブクラスと異なって、スーパークラスをただ一つしかもつことができない。さらに、制約クラスに対しするサブクラスは制約継承のみが許される。これらの制限は、制約継承だけがインスタンスの集合を継承するという点で、ほかの継承の方式と性質が異なるためである。

制約クラスの例を以下に示す。

例：

クラス person のサブクラスとしてクラス teenage を設定できる。クラス teenage の定義では、クラス person への条件のみを記述する。

```
defclass teenage
   super-class person
   instance-constraint-condition
        age > 10 and age < 20
endclass
```

制約継承のインスタンスの指定は、上記のような (a) 条件を指定する方法と、(b) インスタンスを選択する手続きを記述する方法を用意する。

これまでの沙羅のデータモデルでは、あるクラスのインスタンスの集合の部分集合を設定するために、新たにインスタンスを生成する必要があった。そのとき、元のクラスのインスタンスの状態が変化したときに、他方のクラスのインスタンスに反映する操作を利用者が用意する必要があり、データの一貫性がとるのがむずかしい。今回の拡張の方式では、インスタンスをスーパークラスと共有するのでインスタンスの一貫性は保証される。

制約クラスに対してインスタンスを生成すると、そのインスタンスは、自動的にスーパークラスとのインスタンスにもとなる。実際の処理は、制約クラスにインスタンスを作成すると、その制約クラスに課された条件を満たしているか検査した後に、スーパークラスのインスタンスになる。条件を満たしていないならば、インスタンスは生成されない。

同様に、制約クラスのインスタンスの値の変更でも制約の検査が必要になる。

制約クラスを削除した場合、インスタンスはスーパークラスのインスタンスとして残る。（この仕掛けは、「4. クラスの変更」で述べる。）

3.3 属性

オブジェクト指向データベースでの属性の指定は、属性となる オブジェクトのクラスの指定や inverse attributes のような指定しかできなかった。

一方、CODASYL 方式のデータベースでの親子属性の種類を以下のように分類している。[4]

(1) 挿入属性

(a) 手動　MANUAL

操作によって特定のレコードと親子関係が成立する。

(b) 自動　AUTOMATIC

自動的にあるレコードと親子関係が成立する。

(2) 保留属性

(a) 一時　OPTIONAL

レコードの意味は他のレコードと親子関係があるかどうかとは関係がない。

(b) 永続　MANDATORY

レコードの意味は他のレコードと独立であるが、どれかのレコードと親子関係がなければならない。

(c) 固定　FIXED

レコードの意味は、他のレコードと親子関係があるときに意味をもつ。

CODASYL 方式と同様に、オブジェクト指向データベースでも、このような属性の意味による代入の制御が必要である。人物のデータを格納した人事データベースをオブジェクト指向データベースでマルチメディア化した場合も、データ管理の基本はこれまでのものと変わらない。たとえば、写真データはある人物に結び付けられているときに意味をもつ（保留属性が固定）というような情報が必要である。このような機能はデータベース管理システムで用意すべきである。

属性に関する制約を沙羅のデータモデルに追加した。CODASYL 方式の属性の制約は親子構造に関したものであったのに対し、オブジェクト指向データベースではオブジェクトを構成する要素としての属性に関する制約となる。

つまり、あるクラスのインスタンスを作成したときに、そのインスタンスが属性となるインスタンスを設定する機能や、あるクラスのインスタンスはそのインスタンスを属性としてもつインスタンスの生存期間に依存する機能をもつという機能を提供する。

このような属性の制約は、inverse attribute とは異なっている。inverse attribute は、二つのインスンスでの参照の方式を、片方から参照しているときには必ず他方からも参照するように指定するものである。

それに対して、属性の制約は、インスタンスを作成したときの参照の作成の制約や、参照の解消の制約を指定している。したがって、たとえば、保留属性が固定のときの参照は、inverse attribute であっても、そうでなくてもかまわない。

属性の制約は、データベース操作言語のレベルでは、クラスの属性への宣言であるが、指定された属性のアクセスの直前または、直後に起動されるメソッドで実現している。

データベース操作言語「沙羅双樹」の水準では、クラスの宣言の属性に、AUTOMATIC、MANDATORY、FIXED を記述する。これらの参照の属性は、参照側と被参照側の両方のクラスに定義する。参照側は属性に対する付加的な指定として記述し、被参照側はインスタンスに対する付加的な指定として記述する。

4. クラスの変更

クラスは継承によって階層構造を構成する。継承によって、あるクラスの属性とメソッドの定義を利用することができる。

継承は、また、インスタンスの集合の包含関係も表わす。言い替えると、スーパークラスのインスタンスの集合は、そのクラスのインスタンスの集合と、すべてのサブクラスのインスタンスの集合の和集合となる。

たとえば、クラス「学生」のサブクラスに、クラス「テニスクラブ学生」やクラス「情報工学科学生」を作成する。このときクラス「学生」は、クラス「テニスクラブ学生」とクラス「情報工学科学生」を含んだインスタンス全体も表現している。そして、特定のクラブや学科としての学生について検索するときは、サブクラスを利用し、学生全体を検索するときはスーパークラスを使用する（図 6 参照）。

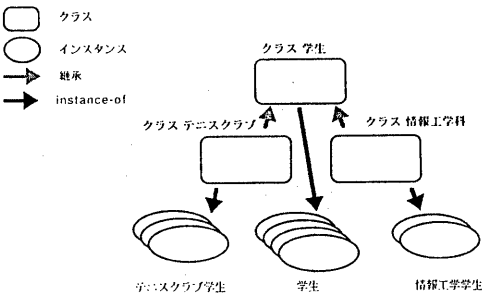

図 6. クラス階層と検索

クラス階層でインスタンスの扱いを決めているとも言える。つまり、ひとりの学生に対して、クラブ員として、学科の学生として、そして大学の学生としての複数の視点があり、その視点ごとに対応したクラスを使用する。

このようにオブジェクト指向データベースでは、クラス階層でも情報を表現しているので、クラスの変更でも、できる限り元のデータベースを保存しなければならない。

オブジェクト指向データベースでのクラスの変更は、クラス階層の変更とクラス定義の変更の２種類が考えられる。

クラス階層の変更は、クラスの削除、クラスの階層内での位置の変更である。

クラス定義の変更は、クラスが定義しているインスタンスの形状（属性とメソッド）が変わることである。

4.1 クラスの削除

あるクラスをデータベースより削除するときに、そのクラスのインスタンスをすべてデータベースより削除すると、スーパークラスからの視点のデータもデータベースから削除することになる。つまり、テニスクラブが廃部になっても学生は大学に在籍し、学生として検索できなければならない。

これは、クラスの表現している情報がそのクラスで局所的でないことが原因である。そのため、クラスの削除は容易ではない。

まず、削除するクラスのインスタンスの扱いについて考える。解決案として、インスタンスがあるクラスは削除しない方法が考えられる。しかし、クラス階層の整理などでクラスを削除する必要が発生したときに、この方法では、利用者がインスタンスの移動をしなけ

ればならない。そのため、根本的な解決策ではない。

インスタンスをデータベース内に残すためには、削除するクラスのインスタンスをどこかのクラスのインスタンスとして再登録する必要がある。検索の経路からスーパークラスにインスタンスを移動するのが妥当である。もし、スーパークラスが一つしかないときは、削除するクラスで定義している属性をインスタンスから取り除きながら、スーパークラスのインスタンスとすればよい（図 7 参照）。

削除するクラスがスーパクラスを複数もつときには、単純ではない。このときには、(a) スーパークラスのどれかに属するインスタンスとするか、(b) インスタンスを複製してすべてのスーパークラスのインスタンスとするか、(c) マルチクラスインスタンスとしてスーパークラスのインスタンスとするかしなければならない（図 8 参照）。

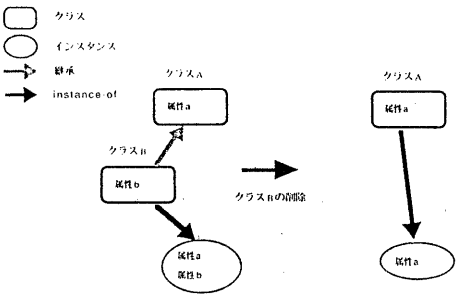

図 7. クラス削除によるインスタンスの移動

方法 (a) のインスタンスを選択された一つのスーパークラスのインスタンスとすると、これまでの検索ではどのクラスからでも参照できたインスタンスが、選択されたクラスでは参照できるが、選択されなかったクラスでは参照できなくなる。これは、最初のデータベースをできるだけ保存する目標に反する。

方法 (b) のインスタンスの複製を作成して、すべてのスーパークラスのインスタンスになるようにする方法が考えられる。しかし、この方法では唯一の存在であったインスタンスが複数のインスタンスとなってしまい、好ましくない。たとえば、複数のクラスに属する人物（留学生で情報処理学科の学生）の属するクラス「テニスクラブ」がなくなったときに、インスタンスは 2 人の人物データ（留学生と情報処理学科学生）に分解されてしまう。

方法 (c) の、削除されたクラスのインスタンスは、マルチクラスインスタンスの機能を使用して、すべてのスーパークラスに属するただ一つのインスタンスとする方法が考えられる。これは、データモデル上の複数のクラスに属するインスタンスとする。これの方法ではデータベースの元の意味を保存している。

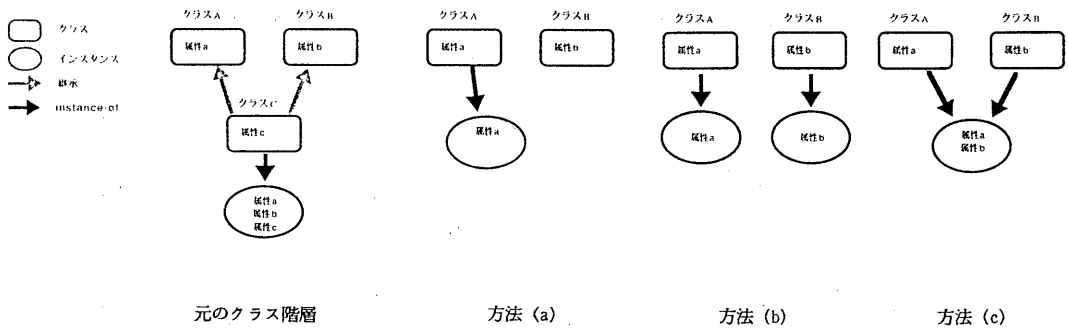

図 8. 多重継承とクラス削除

次に、クラスがサブクラスをもつときのサブクラスの扱いについて考える。サブクラスからは、削除対象となるクラスは、さらにそのスーパークラスの定義と削除対象のクラスの定義の和として見える。そのなかから、削除対象のクラスを削除することは、削除対象のクラスのスーパークラスをサブクラスのスーパークラスとすることになる。

4.2 クラス定義の変更

クラス定義が更新されたときには、それに従ってインタンスを変更しなければならない。

クラス定義の変更は、以下の場合に分類できる。

(a) 属性の追加

属性を追加するとインスタンスの構造が変わる。追加された属性が、初期値を必要とする属性は、初期値をインスタンス参照時に設定する。

(b) 属性の削除

クラスの属性の定義を削除すると、インスタンスの構造が変わる。また、削除された属性を、メソッドで参照しているかもしれない。インスタンスを操作するメソッド内を調べるには、クラス継承をたどり、そのクラスで有効なメソッドを実行する必要がある。しかし、メソッドで参照している属性が動的に決まるようなときには、調べることはできない。

(c) メソッドの追加

メソッドを追加しても、インスタンスの構造を変更しないので問題はない。

(d) メソッドの削除

これも、インスタンスの構造を変更しないので問題はない。

インスタンスをすべて同時に更新すると、膨大な時間がかかる。クラス定義の変更に対して、必要になるまでインスタンスの変更を遅延する。

このようにインスタンスの構造の変更を遅延させると、インスタンスの構造が一様でない状態で処理を継続しなければならない。

4.3 クラス階層での位置の変更

クラス階層でのクラスの位置の変更は、対象のクラスのスーパークラスの変更となる。インスタンスからみると、クラスの定義はスーパークラスの定義とクラスの定義の和であるので、クラスの位置の変更はクラスの定義の変更となる。

4.4 実現方式

沙羅では、データベース管理システムのクラスの変更の影響を少なくする方法として、版番号を利用した遅延処理を採用した[5]。

準備として、クラスには版番号を、インスタンスには、版番号と、属性名と属性値の組を格納しておく。インスタンスに、属性名まで含めるのは、インスタンスの構造が、クラスの構造と一致しなくても、処理ができるようにするためである。

基本的な手順は、以下のようになる。

(a) インスタンスを生成したときは、対応するクラスと同じ版番号をインスタンスに設定する。
(b) クラスが変更されたならば、クラスの版番号を更新する。
(c) インスタンスへのアクセスのときに、版番号を調べ、変更があったときに、インスタンスの構造を更新する。この更新は、属性の削除と、新規属性の設定である。

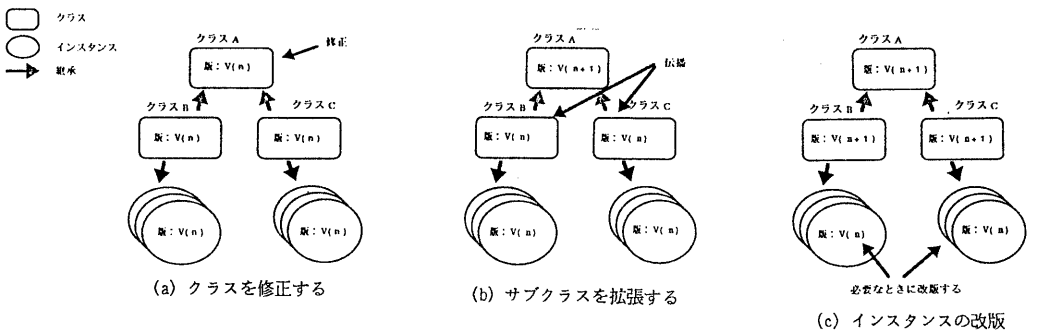

図 9. クラスの変更

属性の削除は、インスタンスの参照には影響を出さないので、その処理はバックグラウンドのガーベージコレクタで回収することもできる。

クラスを変更したときには、そのクラスのインスタンスだけではなく、サブクラスそしてサブクラスのインスタンスにも影響がでる。

クラスの定義の変更をサブクラスに伝播するために、サブクラスの版番号をクラス定義が変更されたときに同時に変更する。これによって、そのサブクラスの定義が変わっていないが、継承によって影響を受けたことが記録され、インスタンスの更新が必要であることがわかる。この手順を図 9 に示した。

クラスはインスタンスをアクセスする度に参照するので、クラスの更新を遅延すると、クラスに対するアクセスごとにスーパークラスとの版の確認をする必要があり、オーバーヘッドが大きくなる。一般に、クラスの数はインスタンスの数に比べて小さいので、クラスの変更だけならばオーバーヘッドは小さいと期待している。

5. おわりに

沙羅の第２版の設計について述べてきた。現在、この設計に基づいて、実装を進めている。実装は、仮想機械方式を採用した。しかし、まだ、データベース管理システムと呼ぶためには機能が不足している。これらの設計と、実装も順に進めたい。

［参考文献］

[1] 石丸他, "データベース管理システム「沙羅」の設計と試作", 電情信学会研究報告, DE92-1, 1992

[2] Masunaga, Y., "Design Issues of OMEGA: An Object-Oriented Multimedia Database Management System", J. IPS Japan, 14, 1, (1991 Jan), 60-74

[3] Atkinson, M., at el, "The Object-Oriented Database System Manifesto", Deductive and Object-Oriented Database, North-Holland, 1990, 223-240

[4] 植村、"データベースシステムの基礎", 1979, オーム社

[5] Tan, L., T. Katayama, "Meta Operations for Type Management in Object-Oriented Databases", Deductive and Object-Oriented Databases, North-Holland, 1990, 241-258