オブジェクト指向データベースにおける継承機構に基づいたビューについて

6H−1

田島 敬史　加藤 和彦　益田 隆司
東京大学 理学部 情報科学科

1 はじめに

近年データベースの利用分野が広がるにつれ,より複雑なデータ構造を扱うことができるデータベースシステムが求められるようになっている.これらの要求への対応策としてオブジェクト指向データベースシステム(以下,OODBと略す)が提案されている.そのような複雑なデータ構造を扱うOODBでは,唯一の複雑なスキーマが全ユーザーに強制され,また,スキーマの変更に対しての柔軟性が欠ける点が欠点として指摘されている.これらへの解決策として,OODBへの仮想クラスの機能の導入が研究されている[1, 4].仮想クラスは従来の関係データベースにおけるビューにあたるもので,実在のクラス(ベース・クラス)に対する問い合わせに基づいて定義されるクラスである.しかし従来の仮想クラスのアプローチでは,仮想クラスに対してはインスタンスの生成が出来ないなど,仮想クラスが通常のクラスと同等には扱えない.これは,従来の仮想クラスでは情報の流れがベース・クラスから仮想クラスへの一方向であり,ベース・クラスと仮想クラスが対等ではないためといえる.

本研究では,通常のクラスと全く同等に扱える仮想クラスの定義を可能とするために,従来の継承の機構を変更・拡張することによって,仮想クラスの機能を継承機構の概念と統合した形で導入したデータ・モデルについて提案する.また,このモデルが,ユーザー毎の外部スキーマの提供や,クラスの複数のヴァージョンの管理や,クラス階層の動的変更に対しても有用であることについても述べる.

2 柔軟性のある継承機構に基づくデータモデル

本章では当論文で提案する柔軟な継承機構に基づくデータモデルについて述べる.このデータモデルでは,インスタンスやメソッドが他のクラスから動的に決定される従来の仮想クラスの機能を,従来より拡張された継承機構によって実現する.

2.1 インスタンスの内部構造の定義の継承を行なわないクラス定義

このモデルでは,クラス間の継承関係をクラスの定義とは独立に定義できるようにするために,インスタンスの内部構造の定義の継承を廃止する.そのために,従来のインスタンス変数を,内部変数と概念変数の二層に分離する.(図1参照).内部変数が物理的に実在する変数を表すものであるのに対して,概念変数は,より上位の階層として仮想的に定義されるものであり,ユーザーには変数のように見えるが,実際には,読み出し,書き込みが行なわれた時にそれぞれ起動される二つのコードによって実現されている.内部変数をアクセスできるのは,この概念変数を定義するためのコード中のみであり,メソッドからは,概念変数へのアクセスを通してのみオブジェクトにアクセスできる.内部変数の定義はそのクラスで生成されるオブジェクトの内部構造を決定するものであり,概念変数は,他クラスで生成されそのクラスのインスタンスとして継承されるオブジェクトが持たなければいけない変数を定義するものである.内部変数の定義は継承されないため,クラスは継承関係とは全く独立に,そのクラスのインスタンスの内部構造を決定し,インスタンスを生成できる.よって,継承関係が変更されても過去に生成されたインスタンスの内部構造の変更が必要になることはない.ただし,クラスに継承関係が定義された際は,継承されるメソッドの実行を保証するために,サブクラスにおいて,最低限スーパークラスで定義されている概念変数が定義されなければならない.概念変数とメソッドは,クラスが定義された後も自由に追加できるが,内部変数は変更できない.

以下にSmalltalk風の構文を用いてクラスEmployeeを定義した例を示す.

```
System newClass: #Employee
      internalVariables: #(nm, sal).
Employee defineConseptualVariables:
        #(name, [^nm], [s|nm:=s]
        salary, [^sal], [i|sal:=i]).
Employee defineMethod: 'addSalary: i'
                  as: [salary := salary + i].
```

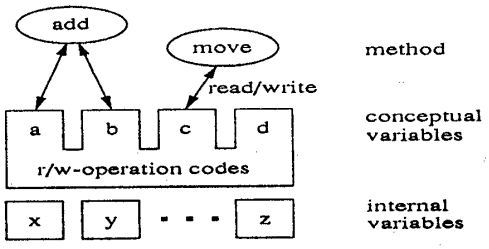

図1: 内部変数,概念変数,メソッドの関係

2.2 拡張された継承機構

このモデルでは,インスタンス生成後も任意のクラス間にスーパークラス,サブクラスとしての継承関係を定義できる.スーパークラス,サブクラスの間では以下のような従来通りの継承が行なわれる.

- 内部変数,概念変数については継承は行なわない
- メソッドについては,スーパークラスからサブクラスへの継承が行なわれる
- インスタンスについてはサブクラスからスーパークラスへの継承が行なわれる

また,これらの従来通りの継承に加えて,従来の仮想クラスの射影と選択の機能に当たる,次の二つの逆向きの継承も行なわせるように記述できる.

- サブクラスのメソッドのうち,ある概念変数に(直接,または間接的に)依存していないものを,スーパークラスに継承させる.また,このような継承はあるクラスが定義された後にそのクラスののスーパークラスを追加できるようなモデルにおいては,有用であることが[3]で述べられている.
- スーパークラスのインスタンスのうち,与えられた条件式を真とするものを,サブクラスに継承させる.またこの時,スーパークラスにはなくサブクラスのみが要求する概念変数を実現するコードを,継承関係の定義中に記述することが出来る.

以下に,継承関係の定義の例を示す.一つ目の定義は,従来通りの継承関係を定義している.二つ目の定義は,サブクラスEmployeeで定義されたメソッドのうち,概念変数salaryに依存しないメソッドをスーパークラスのPublicEmployeeにも継承させる定義である.後者は,スーパークラスEmployeeのインスタンスのうち,メソッドsinceの返り値が1991のものをサブクラスNewfaceに継承させるものである.

```
System newEdgeFrom: #Person to: #Employee.
System newEdgeFrom: #PublicEmployee
                to: #Employee
      inheritMethodWithout: #(#salary).
System newEdgeFrom: #Employee to: #Newface
      inheritInstance: [:i| ^i since = 1991].
```

これらの例は,射影や選択を表す仮想クラスに当たるクラスを,継承機構の枠組の中で記述した例になっている.この方法の利点は,従来のアプローチでは,情報の流れがベース・クラスから仮想クラスへの一方向であったのに対し,クラス間のインスタンスやメソッドの共有関係が双方向に詳細に定義出来るため,全てのクラスが対等に扱えることである.このため,インスタンスの生成などが全てのクラスに対し区別なく可能となる.

Views Based on an Inheritance Mechanism in an Object-Oriented Database System, by Keishi TAJIMA, Kazuhiko KATO, and Takashi MASUDA (Dept. of Information Science, Univ. of Tokyo)

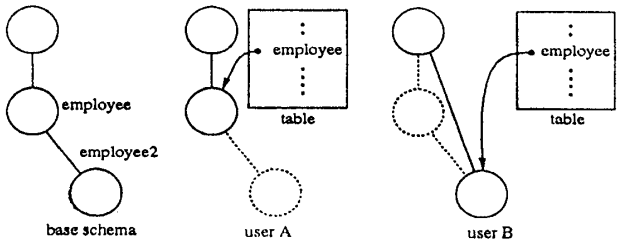

図 2: ベース・スキーマおよび,各ユーザーに見えるスキーマ

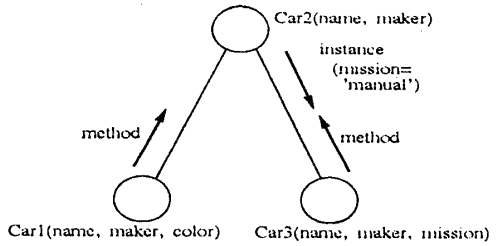

図 3: クラス Car の複数のヴァージョンのクラス階層による管理

3 変更・拡張された継承機構を用いた応用例

本章では,前章で述べた拡張された継承機能が,ユーザー毎に異なるスキーマの提供や,クラスの複数のヴァージョンの管理や,クラス階層の動的な変更にも有用であることについて説明する.

3.1 ユーザー毎に異なる外部スキーマの提供

前述の拡張された継承機構と,スキーマの一部を特定のユーザーに隠蔽する機構を使って外部スキーマを実現する方法について説明する.この論文では各ユーザー毎に異なる外部スキーマを提供する方法として,ベース・スキーマとして全てのユーザーのスキーマを包含するものをつくり,ユーザー毎にその一部のクラスを隠すという方法をとる.この場合も,従来の外部スキーマのアプローチでは,情報の流れがベース・スキーマから外部スキーマへの一方向であったのに対し,外部スキーマ中のクラスとベース・スキーマ中のクラスの間のインスタンスやメソッドの共有関係がベーススキーマ中で双方向に定義出来るため全てのクラスが対等に扱える.

ユーザーに対するスキーマの一部の隠蔽は,次のようにして行なう.まず,クラス名と実際のクラスへのポインタの対応を示す表をユーザー毎に用意し,表にないクラスはそのユーザーには隠蔽される.クラス階層は,各ユーザーには隠されたクラスを除き,隠されたクラスのスーパークラスとサブクラスが直接継承関係を持った形で見せられる.オブジェクトの所属するクラスは,そのオブジェクトのOID を持ち,かつ隠されていないクラスのうち最も下位の物となるが,そのようなクラスがない場合には,そのオブジェクトには一切のメソッドが適用できないため,実質的にそのオブジェクトはユーザーがら隠蔽される.

次の定義は,前節で定義したクラス Employee に対し選択を行なった結果に相当するクラス Employee2 を定義している.ここで,ユーザー A,B にそれぞれ一方のクラスのみを見せることで,各ユーザーには,図 2の実線で表されるような階層が見えるようになる.ここでは,ユーザー B が定義したメソッドが全て Employee にも上向きに継承され,ユーザー A にも共有されるように定義されている.

```
System newClass: #Employee2
       instanceVariables: #(nm, sal).
Employee2 defineConseptualVariables:
          #(name, [^nm], [s|nm:=s]
            salary, [^sal], [i|sal:=i]).
System newEdgesFrom: #Employee to: #Employee2
       inheritInstance: [:i| i sal > 300,000]
       inheritMethodsWithout: #().
```

3.2 クラスのヴァージョン管理

拡張された継承関係はクラスの複数のヴァージョンの管理に応用することが出来る.例として,クラス Car に 3 つのヴァージョンがあり,最初のヴァージョン Car1 では name,maker,color の 4 つの概念変数が定義されていたが,次のヴァージョン Car2 では,color が削除され,さらに次のヴァージョンでは mission が追加されたとする.このような場合,当論文の提案する拡張された継承機構を使って下のような継承を定義することにより,ヴァージョン間のメソッドやインスタンスの共有関係を詳細に管理できる.(図 3参照).この例では,Car1,Car3 のインスタンスを全て Car2 でも共有し,また,Car1 のメソッドのうち color に依存しないものと Car3 のメソッドのうち mission に依存しないものは,Car2 に共有され,Car1 のインスタンスは概念変数 mission の値を規定値の 'manual' として,Car3 に共有されるものとして定義している.

```
System newEdgesFrom: Car2 to: Car1
       inheritMethodsWithout: #(color).
System newEdgesFrom: Car2 to: Car3
       inheritInstance: [:i| ^true]
       withConseptualVariables:
          #(mission, [^'manual'], [])
       inheritMethodsWithout: #(mission).
```

3.3 継承関係の変更

このデータモデルでは,内部変数が継承されずクラスが継承関係とは独立にインスタンスの内部構造を決定できるため,クラス間の継承関係をインスタンス生成後に変更することが出来る.クラス Point,Circle が次のように定義されているとする.

```
System newClass: #Point
       internalVariables: #(x, y).
Point defineConseptualVariables:
        #(x, [^x], [:a| x:=a]
          y, [^y], [:a| y:=a]).
System newClass: #Circle
       internalVarables: #(cntr, r).
Circle defineConceptualVariables:
         #(center, [^cntr], [:p| x:=p]
           radius, [^r], [:l| r:=l]).
```

次の記述は,クラス Circle を全く物理的構造の異るクラス Point のサブクラスとする新たな継承関係を定義することで,クラス Circle のインスタンスをクラス Point のインスタンスとしても扱えるようにしている例である.この時,Point で定義されている概念変数 x と y についてはクラス Circle の中で再定義を行なっている.

```
System newEdgeFrom: #Point to: #Circle.
Circle defineAttributes: #(x,[^cntr x],[:a| cntr x: a]
                           y,[^cntr y],[:a| cntr y: a]).
```

4 結論

当論文では,OODB において,継承機構と統合された枠組でビューの機能を実現するモデルについて提案した.また,このモデルでは「スーパークラスのインスタンスの集合ははサブクラスのインスタンスの集合を包含し,サブクラスのメソッドの集合はスーパークラスのメソッドの集合を包含する」という,OODB のスキーマに必要な IS-A 関係は保ちつつ,インスタンス変数の継承の廃止や,逆向きの継承の機能などによって,より柔軟なクラス階層の記述が可能になっているため,前述のような例の他にも様々な,有用な記述が可能である.今後の研究としては,リレーショナル・データベースにおける結合演算にあたるような,オブジェクトを新たに自動的に生成するビュー [1, 2] の記述をサポートすることが考えられる.

謝辞

日頃,議論して頂く ICOT NDB-WG の皆様に感謝致します.

参考文献

[1] S. Abiteboul and A. Bonner. Objects and views. In *Proceedings of the ACM SIGMOD*, pages 238–247. ACM, June 1991.

[2] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli – a polymorphic language with static type inference. In *Proceedings of the ACM SIGMOD conference*, pages 46–57, Portland, Oregon, May – June 1989.

[3] M. Schrefl and E. J. Neuhold. Object class definition by generalization using upward inheritance. In *Proceedings Fourth International Conference on Data Engineering*, volume 4, pages 4–13. IEEE, IEEE Computer Society Press, Feb. 1988.

[4] K. Tanaka, M. Yoshikawa, and K. Ishihara. Schema virtualiztion in object-oriented databases. In *Proceedings of Fourth International Conference on Data Engineering*, volume 4, pages 23–30. IEEE, IEEE Computer Society Press, Feb. 1988.