データベースシステム 91−3
(1992. 11. 5)

オブジェクト指向データベースにおけるスキーマ・バージョニングの実現法について

田島 敬史　　加藤 和彦　　益田 隆司
東京大学理学部情報科学科

OODB のスキーマ設計は非常に自由度が高く，またどのようにスキーマを設計すべきかという指針となる方法論も確立されていない．そのため，一つのデータベースに対して複数のスキーマのバージョンを管理するスキーマバージョニングの機能は重要である．我々はスキーマバージョニングの機能を，各バージョン間でのデータ共有を管理するための双方向継承機構と，ユーザー毎に異なったスキーマを提供するためのスキーマの一部隠蔽の機能によっ実現する方法に関する研究を進めている．この論文では，まず双方向継承機構とスキーマ一部隠蔽の機能を使ったスキーマバージョニングの概要について説明し，これらの機能を実現するための方法について述べる．

Implementation of Schema Versioning in an Object-oriented Database System

Keishi Tajima　　Kazuhiko Kato　　Takashi Masuda
Department of Information Science, Faculty of Science, the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 Japan

Schema versioning, a function to manage multiple versions of a schema, is important for object-oriented database systems (OODBSs) because methodology to design schemata of OODBs has not been established while many alternatives exist when designing schemata of OODBs. We are developing the system which realizes schema versioning using two mechanisms, the extended bi-directional inheritance mechanism, which manages the sharing of data among multiple schema versions, and schema hiding, which provides each users with different schemata. This paper explains the schema versioning using these two mechanisms and describes how these twe mechanisms can be implemented.

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)が注目されている．

OODB のスキーマ設計は，必要なクラスを定義し，それらのクラスの間の IS-A 関係を表現する IS-A 階層を構築することによって行なわれる．しかし， OODB のスキーマ設計は自由度が高く，またどのようにスキーマを設計すべきかという指針となる方法論も確立されていない．そのため，一度スキーマを定義した後に，クラスを追加・削除して IS-A 階層の構成を変えたり，クラスの属性を追加・削除したりする必要が生じることが多い．一つのデータベースにこのような複数のスキーマのバージョンを定義する場合，それらのバージョンの間でのインスタンスやメソッドの共有を実現する機構が必要となる．

このようなスキーマの複数のバージョンを管理する機能(スキーマバージョニングの機能)については，スキーマとスキーマ中のデータを含めたデータベース全体の新しいバージョンを生成する方法 [2] や，クラス毎にの複数のバージョンを管理する方法 [3, 1] などが提案されている．しかし，これらの研究では前述のような，異なるバージョンのスキーマ間での柔軟なデータの共有を行なうための機構が不十分である．

我々は，このようなスキーマバージョン間でのデータ共有を，従来の継承機構の拡張にあたる双方向のデータ共有機構により行なう方法に関する研究を進めている [4]．我々の提案しているアプローチでは，各スキーマバージョン中の全てのクラスを含むユニバーサルスキーマと呼ばれるものを定義し，このユニバーサルスキーマ中で各クラス間に，拡張された継承機構を使った双方向のデータ共有関係を定義する事によって，スキーマバージョン間のデータ共有を行なう．また各スキーマバージョンは，バージョンの数だけ個別にクラス階層を生成するのではなくユニバーサルスキーマの一部を隠蔽することによって実現する．

この論文では，この双方向継承機構を，メソッドサーチパス，インスタンスサーチパスと呼ばれるリストを静的に決定することで効率良く実現し，またスキーマの一部隠蔽についても，これらのパスについて各ユーザー毎に異なるものを使用することで実現する方法について述べる．以下，まず2章で我々のアプローチによるスキーマバージョニングの概要を説明し，次に 3章で双方向継承機構とスキーマ隠蔽について，より形式的な定義を行なう．さらに， 4章でこれらの機能を実現する方法について詳しく述べる．

2 スキーマバージョニングの概要

従来のオブジェクト指向データモデルは，クラス間のメソッドやインスタンスの共有の機能として IS-A 階層に基づく継承機構を持っていた．本研究では，互いに異なるスキーマバージョンに所属するクラス間でのメソッドやインスタンスの共有機構も，継承機構と統合した形で導入する．そしてユニバーサルスキーマと呼ばれる IS-A 階層を定義し，この中に各スキーマバージョン中に現れる全てのクラスを位置付け，それらの間の共有関係を継承機構を使って定義することで，バージョン間のデータの共有を管理する．

しかし，従来の継承機構では，メソッドは常にスーパークラスからサブクラスへ(インスタンスはその逆へ)と一方向にのみ継承されるため，あるクラス間で互いに自分のメソッド(またはインスタンス)を共有するような共有関係は定義できない．また，従来の継承機構では共有関係が定義されたクラス間では全てのメソッド(インスタンス)が継承されるため，あるクラスのメソッドやインスタンスの一部のみを他のクラスと共有するような共有関係も定義できない．そこで，我々は従来の継承機構を IS-A 階層のセマンティクスを壊さない範囲で拡張することで，これらの相互かつ部分的なデータの共有を行なう．

また上のようにユニバーサルスキーマを定義した場合，各ユーザーが見る IS-A 階層は，ユニバーサルスキーマ中に定義されているクラス及びクラス間の共有関係の一部を取り出したものとなる．そこで，ユーザーに見せられる各バージョンは，個別に IS-A 階層を定義するのではなく，各ユーザー毎にユニバーサルスキーマの一部を隠蔽し，特定の部分のみを見せることによって実現する．

この章では，以上のような方法によるスキーマバージョニングの概略を，具体的な例を使って述べる．

2.1 双方向継承機構

例として，スキーマの新しいバージョンが作られると共にクラス Car の属性が変更された場合の例を示す．まず最初のバージョンのスキーマ(バージョン 1 とする)の中では Car(1) はクラス Object のサブ

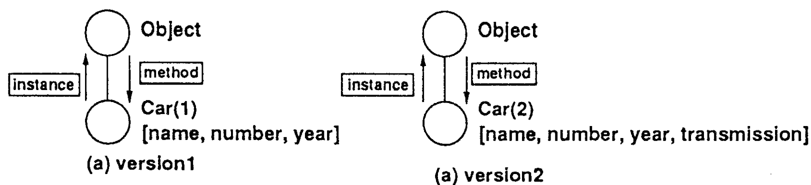

図 1: スキーマ：バージョン 1 とバージョン 2

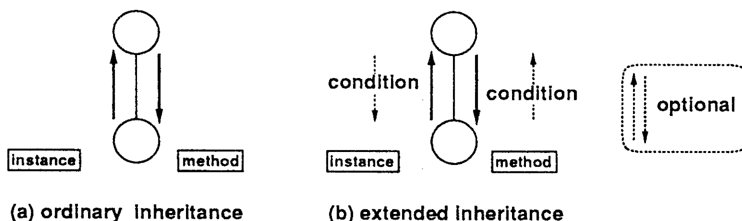

図 2: 従来の継承機構と拡張された継承機構

クラスとして name, number, year の三つの属性が定義されていたとする．次のスキーマのヴァージョン (バージョン 2 とする) の中では，属性 transmission が追加され Car(2) は name, number, year, transmission の四つの属性によって定義されたとする．(図 1 (a), (b) 参照).

この場合，二つのスキーマのバージョン間で次のようなデータの共有が必要になることが考えられる．

- Car(1) に生成されたインスタンスは全て，Car(2) のインスタンスとして共有する．
- Car(2) に生成されたインスタンスのうち，条件 year < 1975 を満たすものについては，transmission の値を既定値の 'manual' として，Car(1) のインスタンスとして共有する．
- Car(1) に対し定義されたメソッドは全て，Car(2) のメソッドとして共有する．
- Car(2) に対し定義されたメソッドのうち，Car(1) にも定義されている属性 name, number, year だけに依存しているものについては，Car(1) のメソッドとして共有する．

このような異なるスキーマバージョン間のデータの共有を管理するために，従来の継承機構を拡張する．まず従来の継承機構は次のようなものであった (図 2(a) 参照).

- スーパークラスのメソッドは全てサブクラスにも共有させる．
- サブクラスのインスタンスは全てスーパークラスにも共有させる．

しかし，これでは上の例のように二つのクラス間で相互にデータを共有したり，部分的に選択して共有したりする共有関係は定義できない．そこで，これを次のように拡張する (図 2(b) 参照).

- 従来通りの 2 つの継承は，常に行なう．
- オプションとして，サブクラスのメソッドのうちスーパークラスにも定義されている属性にのみ (直接，または間接的に) 依存しているものは，スーパークラスにも共有させるように指定できる．
- 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は，サブクラスにも共有させる．この時，スーパークラスにはなくサブクラスにのみ定義されている属性がある場合は，各インスタンスに対してのその属性の既定値の定義を，継承関係の定義とともに定義することが出来る．

この結果，ユニバーサルスキーマ中に次のような継承関係を定義することによって上のような共有関係を定義できる．(図 3参照).

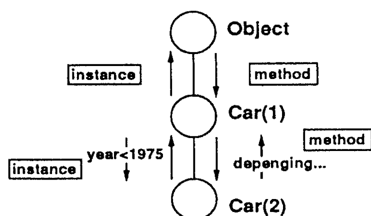

図 3: ユニバーサルスキーマ

- Car(1) を Object のサブクラス, Car(2) を Car(1) のサブクラスとする.
- オプションである Car(2) から Car(1) へのメソッドの上向きの継承を行なうように指定する.
- (year < 1975) を条件とするインスタンスの下向きの継承を行なわせ, その際に属性 transmission の既定値を 'manual' とする.

2.2 ユニバーサルスキーマの一部隠蔽

前節で説明した例で, ユニバーサルスキーマは二つのスキーマバージョンのユニオンに当たるものになっている. そこで, 各スキーマバージョンはユニバーサルスキーマの一部を隠蔽することによって実現できる. バージョン 1 のスキーマについては次のような隠蔽を行なえばよい.

- Car(2) の存在を隠蔽する
- Car(2) のインスタンスのスーパークラスへの継承についてはそのまま行なわせる
- Car(2) のメソッドの上向きの継承はそのまま行なわせる

また, バージョン 2 については次のようになる

- Car(1) の存在を隠蔽する
- Car(1) のインスタンスの下向きの部分的継承は行なうが, サブクラスへの継承は行なわない
- Car(1) のメソッドのサブクラスへの継承はそのまま行なわせる

インスタンスについて上のように定義することで, Car(1) のインスタンスの内, Car(2) に継承されたもの以外は Object にも継承されない. よって, バージョン 2 では Car(1) のインスタンスの一部分のみを共有し, 他の部分については隠蔽することが出来る.

3 形式的な定義

この章では, 2章で説明した双方向継承機構とスキーマの一部隠蔽機構について, より形式的な定義を行ない, 実装の上での詳細な仕様を定める.

3.1 双方向継承機構

2章で述べた双方向継承機構は, 従来の継承機構を IS-A 階層の持つセマンティクスを壊さない範囲で拡張したものになっている. IS-A 階層のセマンティクスとは次のようなものであると考える. $\mathcal{S}$ をスキーマ中の全てのクラスの集合, $\succ$ を $\mathcal{S}$ 上に定義された IS-A 関係を表す半順序, $C \in \mathcal{S}$ に対して Dictionary(C), Extension(C) をそれぞれ C のインスタンスに適応されるメソッドの集合, C に所属するインスタンスの集合とし, name() をメソッドからそのメソッドの名前への写像を表す関数とする. この時, $C_1 \in \mathcal{S}$,

$C_2 \in \mathcal{S}$ 間に C_1 をスーパークラス，C_2 をサブクラスとする IS-A 関係 $C_1 \succ C_2$ が定義されているとすると，IS-A セマンティクスとは次の二つが成り立つことを意味するものである．

$$\begin{aligned}\mathbf{Protocol}(C_1) &\subseteq \mathbf{Protocol}(C_2)\\ \mathbf{Extension}(C_1) &\supseteq \mathbf{Extension}(C_2)\end{aligned}$$

ただし，上の式で

$$\mathbf{Protocol}(C) \equiv \{n \mid \exists m \in \mathbf{Dictionary}(C) \,.\, \mathbf{name}(m) = n\}$$

とする．このセマンティクスを保証するため，従来の IS-A 階層では，次の式で表現されるようなメソッド及びインスタンスの継承 (共有) を行なっていた．

$$\begin{aligned}&\mathbf{Dictionary}(C) \equiv \mathbf{InheritedDct}(C) \cup \mathbf{ImDct}(C) -\\ &\quad \{m \in \mathbf{InheritedDct}(C) \mid \exists m' \in \mathbf{ImDct}(C) \,.\, \mathbf{name}(m') = \mathbf{name}(m)\}\\ &\mathbf{Extension}(C) \equiv \mathbf{InheritedExt}(C) \cup \mathbf{ImExt}(C)\end{aligned}$$

ただし，上の式で，

$$\begin{aligned}\mathbf{SupC}(C) &\equiv \{C' \in \mathcal{S} \mid C' \succ C\}\\ \mathbf{SubC}(C) &\equiv \{C' \in \mathcal{S} \mid C \succ C'\}\\ \mathbf{ImSupC}(C) &\equiv \{C' \in \mathcal{S} \mid C' \text{は}\ \mathbf{SupC}(C)\ \text{の極小元}\}\\ \mathbf{ImSubC}(C) &\equiv \{C' \in \mathcal{S} \mid C' \text{は}\ \mathbf{SubC}(C)\ \text{の極大元}\}\\ \mathbf{ImDct}(C) &\equiv \{C \text{に対して定義されたメソッド}\}\\ \mathbf{ImExt}(C) &\equiv \{C \text{によって生成されたインスタンス}\}\\ \mathbf{InheritedDct}(C) &\equiv \bigcup_{C' \in \mathbf{ImSupC}(C)} \mathbf{Dictionary}(C')\\ \mathbf{InheritedExt}(C) &\equiv \bigcup_{C' \in \mathbf{ImSubC}(C)} \mathbf{Extension}(C')\end{aligned}$$

とする．最初の二式で右辺第一項が継承に対応し，一つ目の式の第三項はメソッドをローカルに再定義した場合に対応する．そしてオブジェクト o にメッセージが送られた時は，集合 $\{C \in \mathcal{S} | o \in \mathbf{Extension}(C)\}$ の最小元となるクラスを $C_{min}(o)$ として，$\mathbf{Dictionary}(C_{min}(o))$ から実行すべきメソッドが選ばれる．

この従来の継承は，IS-A 階層のセマンティクスを保つための最低限の必要条件にあたる．しかしこれでは，メソッドは常に半順序に従って大きい方から小さい方へ (インスタンスはその逆) とその全てが継承されるため，あるクラス間で互いに自分のメソッド (あるいはインスタンス) の一部を共有させるような共有関係は記述不可能である．そこで本研究では，IS-A セマンティクスは保ったまま，双方向の共有が可能となるように，従来の継承機構を次のように拡張する．まず，$C_1 \in \mathbf{ImSupC}(C_2)$ という関係にあるような C_1, C_2 の各組合せに対して，メソッドの上向きの継承を行なうかどうかを表す論理値 $\mathbf{MtdUpwd}(C_1, C_2)$，インスタンスの下向きの継承の際の条件式を表す，オブジェクトから論理値への任意の関数 $\mathbf{IstDnwd}(C_1, C_2)$ の二つが定義されているものとする．そして，前述の継承を表す項を次のように拡張する．

$$\begin{aligned}&\mathbf{InheritedDct}(C) \equiv \bigcup_{C' \in \mathbf{ImSupC}(C)} \mathbf{Dictionary}(C') \cup\\ &\quad \bigcup_{C' \in \{C'' \in \mathbf{ImSubC}(C) \mid \mathbf{MtdUpwd}(C, C'') = true\}}\\ &\quad \{m \in \mathbf{Dictionary}(C') \mid m \text{は}\ C\ \text{に定義されている属性のみに依存}\}\\ &\mathbf{InheritedExt}(C) \equiv \bigcup_{C' \in \mathbf{ImSubC}(C)} \mathbf{Extension}(C') \cup\\ &\quad \bigcup_{C' \in \mathbf{ImSupC}(C)} \{i \in \mathbf{Extension}(C') \mid \mathbf{IstDnwd}(C', C)(i) = true\}\end{aligned}$$

そして，オブジェクト o にメッセージが送られた際には，クラス $C_{min}(o)$ を集合 $\{C \in \mathcal{S} | o \in \mathbf{Extension}(C)\}$ の最小元として，$\mathbf{Dictionary}(C_{min}(o))$ から該当するものを探すものとする．ただし，このように

InheritedDct(C) と InheritedExt(C) の定義を変更した場合，前述の Dictionary(C)， Extension(C) の右辺を展開すると Dictionary(C)， Extension(C) がまた現れることになり，式の解が不定になる．そこで， Dictionary(C)， Extension(C) はこの式を満たす解の内最小の物として定義する．また，異なる継承のルートから複数の同じ名前のメソッドが継承される場合は，そのようなメソッド全てについて，そのクラスでローカルに再定義しなければいけないものとする．

ここで問題となるのは，一つのクラスから複数のサブクラスにインスタンスの下向きの継承を行なった場合，あるオブジェクト o に対して集合 $\{C \in \mathcal{S} | o \in \mathbf{Extension}(C)\}$ が最小元を持たず，複数の極小元を持つ場合がある．このような場合は，そのような継承関係を定義する際に，各サブクラスに対しどの順にメソッドを探索するかを指定するものとする．

もう一つの問題点としては，例えばある二つのクラス間で，互いにインスタンスを部分的にのみ共有したい場合，ユニバーサルスキーマ中に，それら両方のスーパークラスとなるようなクラスを新たに作る必要が生じる．このようなクラスを自動的に生成する問題に関しては，[1] で述べられている．本研究では，このような場合はスキーマの管理者がそのようなクラスを定義するものとする．

3.2 ユニバーサルスキーマの一部隠蔽

各ユーザーに対してスキーマの一部を隠蔽する場合，あるクラスの存在だけでなく，そのクラスのインスタンスやメソッドなども全て隠蔽したい場合と，そのクラスの存在は隠蔽するが，そのクラスのインスタンス (またはメソッド) は，そのクラスとインスタンス (またはメソッド) を共有している別のクラスのインスタンス (メソッド) として見せたい場合とがある．そこで，ユーザーに対しスキーマのどの部分を隠蔽するかについては，各ユーザー毎に次の三つの集合を定義することで指定する．

$\mathcal{S}'$ — そのユーザーが見ることのできるクラスの集合．ただし， $\mathcal{S}' \subseteq \mathcal{S}$

$\mathcal{S}_{mtd}$ — そのユーザーがメソッドを継承できるクラスの集合．ただし， $\mathcal{S}' \subseteq \mathcal{S}_{mtd} \subseteq \mathcal{S}$

$\mathcal{S}_{ist}$ — そのユーザーがインスタンスを継承できるクラスの集合．ただし， $\mathcal{S}' \subseteq \mathcal{S}_{ist} \subseteq \mathcal{S}$

この時，各ユーザーに見えるスキーマを次のように定義する．まず， IS-A 階層は前述の $\succ$ に基づいて $\mathcal{S}'$ 上に自然に定義される半順序として定義する．そして，ユーザに見える各クラスのメソッド辞書と外延を表す Dictionary$'$()， Extension$'$() については，前述の継承を表す式を変更し以下のように定義する．

$$
\begin{aligned}
&\mathbf{InheritedDct}(C) \equiv \bigcup_{C' \in \mathbf{ImSupC}_{mtd}(C)} \mathbf{Dictionary}(C') \cup \\
&\quad \bigcup_{C' \in \{C'' \in \mathbf{ImSubC}(C) \mid \mathbf{MtdUpwd}(C, C'') = true\}} \\
&\quad \{m \in \mathbf{Dictionary}(C') \mid m \text{は} C \text{に定義されている概念属性のみに依存}\} \\
&\mathbf{InheritedExt}(C) \equiv \bigcup_{C' \in \mathbf{ImSubC}_{ist}(C)} \mathbf{Extension}(C') \cup \\
&\quad \bigcup_{C' \in \mathbf{ImSupC}(C)} \{i \in \mathbf{Extension}(C') \mid \mathbf{IstDnwd}(C', C)(i) = true\}
\end{aligned}
$$

ただし，上の式で

$$
\begin{aligned}
\succ_{mtd} &\equiv \succ \text{に基づいて} \mathcal{S}_{mtd} \text{上に自然に定義される半順序} \\
\succ_{ist} &\equiv \succ \text{に基づいて} \mathcal{S}_{ist} \text{上に自然に定義される半順序} \\
\mathbf{SupC}_{mtd}(C) &\equiv \{C' \in \mathcal{S}_{mtd} \mid C' \succ_{mtd} C\} \\
\mathbf{SubC}_{ist}(C) &\equiv \{C' \in \mathcal{S}_{ist} \mid C \succ_{ist} C'\} \\
\mathbf{ImSupC}_{mtd}(C) &\equiv \{C' \in \mathcal{S} \mid C' \text{は} \mathbf{SupC}_{mtd}(C) \text{の極小元}\} \\
\mathbf{ImSubC}_{ist}(C) &\equiv \{C' \in \mathcal{S} \mid C' \text{は} \mathbf{SubC}_{ist}(C) \text{の極大元}\}
\end{aligned}
$$

とする．つまり，メソッドの継承については，上向きのへの継承は前述の通りに行なわれるが，下向きの継承は $\succ_{mtd}$ に基づいて行なわれ，逆にインスタンスの継承については，下向きの継承は前述の通りだが，

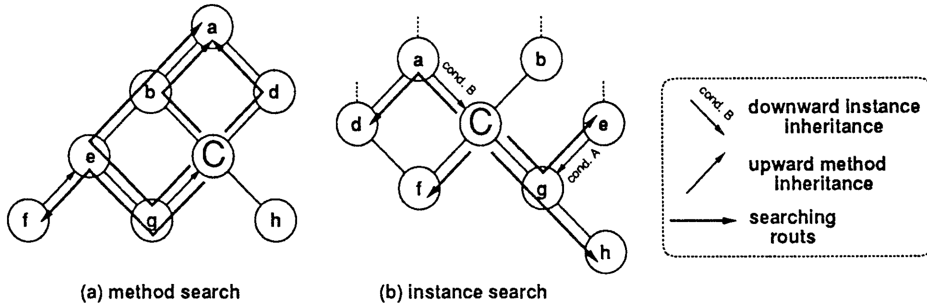

図 4: 双方向継承を含むメソッドサーチ，インスタンスサーチの例

上向きの継承は $\succ_{ist}$ に基づいて行なわれる．これは，$\mathcal{S}_{mtd}$ に含まれていないクラスに定義されたメソッドは，特に上向きの継承が行なわれたもの以外はどのクラスの Dictionary′() にも現れず，$\mathcal{S}_{ist}$ に含まれていないクラスで生成されたインスタンスは，特に下向きの継承が行なわれたもの以外はどのクラスの Extension′() にも現れないことを意味している．よって，あるクラスのメソッドやインスタンスの一部だけを拡張した継承機構を使って他のクラスに継承させ，元のクラスは隠すという形で，そのクラスの一部のインスタンスやメソッドだけを見せるということが可能になる．

オブジェクトについては，どのクラスの Extension() にも現れなくても，他のオブジェクトから OID によって参照されている場合がある．これに対しては，次のようにしてオブジェクトをユーザーから隠蔽する．オブジェクト o にメッセージが送られた時は，集合 $\{C \in \mathcal{S}'|o \in \text{Extension}'(C)\}$ の最小元となるクラスを $C_{min}(o)$ とし，前述の通りメソッドの探索を行なう．しかし，ここで $\{C \in \mathcal{S}'|o \in \text{Extension}'(C)\} = \emptyset$ である場合はそのオブジェクトに対しては一切のメソッドの呼び出しを禁止する．これによって，このユーザーはそれらのオブジェクトのポインタを持つことはできるが一切のアクセスはできない．よって，これらのオブジェクトは実質的にユーザーから隠蔽される．

4 実現

この章では，双方向継承機構とスキーマの一部隠蔽の機能について，3章で定義された仕様に基づいて実装するための方法について述べる．

この論文では，次のような方法で双方向継承を実現する．双方向継承を含む形で定義された Dictionary(C) や
Extension(C) を計算するために，どのクラスをどのような順でサーチすれば良いかは静的に決定できる．そこで，これらのパスを各クラス毎にあらかじめ求めておき，それぞれメソッドサーチパステーブル，インスタンスサーチパステーブルと呼ばれるテーブルに登録しておく．そして，Dictionary(C) や Extension(C) を求める必要が生じた場合は，これらのパスに沿ってメソッドやインスタンスのサーチを行なって Dictionary(C) や Extension(C) を求める．また，このパスは各ユーザーに対しユニバーサルスキーマのどの部分が隠蔽されているかに依存するため，このテーブルを各ユーザー毎に作成する．

しかし，オブジェクトの条件付きの下向きの継承が行なわれている場合，オブジェクトにメッセージが送られた際にどのクラスのオブジェクトとしてメソッドサーチパスを行なうかは，実行時にこの条件式を評価して決定する必要がある．この問題についてもメソッドサーチパスの中に特殊なメソッドを挿入しておくことで，必要な処理を行なわせる．以下，これらのパスやテーブルの作成手順について詳しく述べる．

4.1 Dictionary(C) と Extension(C) の計算

オブジェクトにメッセージが送られた場合，起動されるメソッドを決定するために，あるクラスについて Dictionary(C) を求める必要がある．3章で述べたように，双方向継承の機能を加えた形で Dictionary(C)

を定義した式は，単純に展開して解くことは出来ない．しかし，求めるべき「この式を満たす最小の解」は，サイクルに入らないように循環検出を行ないながら継承が行なわれる全てのルートを辿ってメソッドを集めることで求められる．例えば図4(a)のようなクラス階層が定義されている場合，**Dictionary**(C)は図4(a)に示されたパスに沿って順にメソッドを調べる事で求められる．

以下に，**Dictionary**(C)を求める関数をPascal風の構文で記述したものを示す．この関数では現在辿っているルート上に出てきたクラスを変数searchingPathに記録してサイクルに入らないようにチェックしながら，各クラスのスーパークラス及びメソッドの上向きの継承が定義されているサブクラスについてsearchMtdを再帰的に呼び出している．そして，スーパークラスについては全てのメソッドを，メソッドの上向きの継承が定義されているサブクラスについては条件に合うものだけを集合に加えている．

```
function Dictionary(C : class):set of method;
begin
  Dictionary(C) := searchMtd(C, emptyset())
end;
function searchMtd(C: class; searchingPath: set of class):set of method;
  var result: set of method;
begin
  result := ImDct(C);
  add(searchingPath, C); {add C to the currently searching path}
  for C' ∈ ImSupC_mtd(C) do
    if C' ∉ searchingPath then
      result := unionMtd(result, searchMtd(C', searchingPath));
  for C' ∈ ImSubC(C) do
    if (C' ∉ searchingPath) and (MtdUpwd(C, C') = true) then
      result := unionMtd(result, selectMtd(C, searchMtd(C', searchingPath)))
end;
function unionMtd(a, b: set of method):set of method;
begin
  {This makes union of a and b with excluding methods in b that have same names as any method in a}
end;
function selectMtd(c: class; aSet: set of method):set of method;
begin
  {select methods that depend on only attributes defined in C}
end;
```

Extension(C)についてもほぼ同様のアルゴリズムで求めることが出来る．データベースに対して検索が行なわれた場合，あるクラスの**Extension**(C)を求める必要が生じる．この場合も前述の式を直接展開して解くことは出来ないが，循環検出を行ないながら必要なクラスをサーチし，各クラスのインスタンスの全て，または条件を満たす一部を加えていくことで，**Extension**(C)を求めることができる．例えば図4(b)のようなクラス階層が定義されている場合，**Dictionary**(C)は図4(b)に示されたパスに沿って順にインスタンスを集める事で求められる．

4.2 双方向継承のためのサーチパスの静的な決定

オブジェクトにメッセージが送られる度に上の**Dictionary**(C)のアルゴリズムを実行するのは効率上問題がある．しかし，4.1の**Dictionary**(C)のアルゴリズムからわかるようにS_{mtd}とCが与えられれば，**Dictionary**(C)を求めるにはどのクラスをどの順序でサーチすればよいか，またどのクラスからは全てのメソッドを，どのクラスからは一部のメソッドのみを継承すべきかということは静的に決定できる．

そこで実際のシステムの実装では，各クラスについて，あらかじめそのクラスのメソッドサーチパスと呼ばれる物を生成しておく．メソッドサーチパスの内容は，サーチすべき各クラスの**ImDct**(C)と，継承の対象がその辞書全体か，一部かを表す論理値の，対のリストである．またこの時，継承の対象が辞書全体となるルートを優先してサーチするようにすると，後から別のルートで同じクラスに行きつく場合，そ

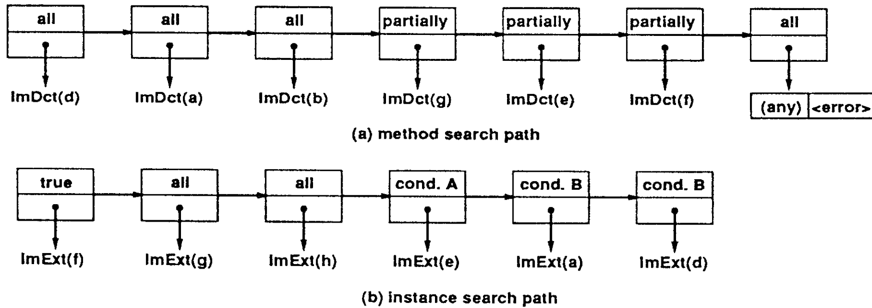

図 5: メソッドサーチパス，インスタンスサーチパスの例

の部分のサーチを省くことが出来る．例えば，図 4(a) の例の場合，図 5(a) のようなメソッドサーチパスがつくられる．また，メソッドサーチがパスの終りまで来てしまった時のために，パスの最後に全てのメッセージにマッチし，エラーを発生するメソッドをつけておく．

同様に Extension(C) についてもインスタンスサーチパスと呼ばれる物を各クラスについて静的に求めておく．インスタンスサーチパスの内容は，サーチすべき各クラスの ImExt(C) と，継承されるインスタンスが満たすべき条件式である．また，この場合条件の少ないルートを優先してサーチするようにすると，後から別のルートで同じクラスに行きつく場合，その際の条件が先にサーチする際の条件のスーパーセットになっていれば，その部分のサーチを省くことが出来る．例えば，図 4(b) の例の場合，図 5(b) のようなインスタンスサーチパスがつくられる．

4.3 スキーマの一部隠蔽

各ユーザー毎のスキーマの一部隠蔽の機構については，ユーザー毎に異なるメソッドサーチパスとインスタンスサーチパスを使用することで実現する．そのために，各ユーザー毎にメソッドサーチパステーブル，インスタンスサーチパステーブルと呼ばれるテーブルを持たせ，そのユーザーが Dictionary(C), Extension(C) を求める際に用いるべき，メソッドサーチパス，インスタンスサーチパスを登録しておく．メソッドサーチパステーブルの各クラス C のエントリの内容は次のようになる．

- $C \in S'$ の場合 — そのユーザーの S_{mtd} に基づいて決定された，Dictionary(C) を求めるためのメソッドサーチパスを登録する．
- $C \notin S', C \in S_{ist}$ の場合 — この場合，このクラスはこのユーザーからは隠蔽されているため，このクラスのインスタンスに対する $C_{min}(o)$ はそのスーパークラスのいずれかになる．よって，$C_{min}(o)$ になるクラスのメソッドサーチパスを登録する．
- $C \notin S', C \notin S_{ist}$ の場合 — この場合，このクラスのインスタンスはこのユーザーには隠蔽される．よって，全てのメッセージにマッチしエラーを発生するメソッドのみの辞書が登録される．

その結果，あるユーザーがあるオブジェクト o にメッセージが送った際の処理は次のようになる．

1. そのオブジェクトを生成したクラスを調べる．（各オブジェクトはそのオブジェクトを生成したクラスへのポインタを持っている．）
2. ユーザーのメソッドサーチパステーブルからそのクラスのエントリを探す．
3. そのエントリに登録されているメソッドサーチパスに沿ってメソッドサーチを行ない，名前のマッチするものを見つけたらそれを起動する．

4.4 $C_{min}(o)$ の実行時における決定

前節までで述べられた部分については，全て静的に決定することができる．しかしインスタンスの下向きの継承を行なう場合，インスタンスの所属するクラスを実行時に決定しなければならない場合がある．今クラス C と $C'(\in \mathbf{ImSubC}(c))$ の間に，条件式 $\mathbf{IstDnwd}(C,C')$ に基づくインスタンスの下向きの継承が定義されていて，クラス C で生成されたオブジェクト o に対してメッセージが送られたとする．この時，次の三通りの場合がある．

1. $C' \in \mathcal{S}'$ の場合 — この場合，$\mathbf{IstDnwd}(C,C')$ を評価して true となった場合は C' が $C_{min}(o)$ となるので，C' のメソッドサーチパスを用い，false となった場合は C が $C_{min}(o)$ となるので，C のメソッドサーチパスを用いる必要がある．
2. $C' \in \mathcal{S}_{ist}$ かつ $C \notin \mathcal{S}_{ist}$ の場合 — 上に同じ．
3. その他の場合 — C と C' のメソッドサーチパスは同じ内容になるため，どちらが $C_{min}(o)$ になるか判定する必要はない．

この問題に対しては，次のような方法で対応する．C，C' が上の 1，2 のいづれかに当てはまる場合には，C のメソッドサーチパスの先頭に，全てのメソッドにマッチし，$\mathbf{IstDnwd}(C,C')$ を評価するメソッドを置く．このメソッドは，$\mathbf{IstDnwd}(C,C')$ を評価し結果が true ならば C' のメソッドサーチパスに対してメソッドサーチを続け，false ならば C のメソッドサーチパスのさらに先へメソッドサーチを続ける．このような処理を行なうメソッドをメソッドサーチパスに埋め込むことで，オブジェクトにメッセージが送られた際の処理を全ての場合に共通とし，処理を単純にすることができる．

5 結論

本論文では，OODB において，スキーマバージョニングを実現するための双方向継承機構とスキーマの一部隠蔽機構について，まずその概要を述べ，さらにその実装方法について説明した．現在，本論文で説明した方法による実装のプロトタイプとして，一次記憶上の Smlltalk を改造する形での実装を行なっている．今後の研究課題としては，スキーマの新しいバージョンを定義した際に，そのバージョンを含むユニバーサルスキーマを自動的に生成するためのアルゴリズムの開発がある．また，[5] で述べられているような枠組と，この論文で述べた枠組とを統合する問題も，今後の課題として挙げられる．

参考文献

[1] Hyoung-Joo Kim and H. F. Korth. Schema versions and views in object-oriented databases. In *Proceedings of an International Conference*, pp. 277–284. IPSJ, 1990.

[2] Won Kim and Hong-Tai Chou. Versions of schema for object-oriented databases. In F. Bancilhon and D. J. DeWitt, ed., *Proceedings of the 14th International Conference on VLDB*, pp. 148–159. 1988.

[3] A. H. Skarra and S. B. Zdonik. The management of changing types in an object-oriented database. In N. Meyrowitz, ed., *OOPSLA*, pp. 483–495. ACM, 1986.

[4] 田島敬史, 加藤和彦, 益田隆司. 三階層構造を持つオブジェクト指向データベースとその継 承機構について. 田中克己, 西尾章治郎 (編), オブジェクト指向コンピューティング '92, レクチャーノート / ソフトウェア科学シリーズ. 近代科学社, 1992.

[5] 加藤和彦, 萩谷昌己, 千葉茂, 綾塚祐二, 益田隆司. オブジェクト指向データベースを用いたグラフィカルユー ザインターフェースシステムについて. 投稿中