データベースシステム 99−23
（1994. 7. 21）

多重階層化機構を持つオブジェクト指向データモデル

堀 宣男　吉川正俊　植村俊亮

奈良先端科学技術大学院大学 情報科学研究科

オブジェクト指向データベースにおけるデータモデル ASKA を考案した。このモデルは、クラス階層を主体にしたスキーマを持つが、同時にテンプレート階層を導入することで、データオブジェクトの配置を機能的、かつ効率的にした。また、今までのデータモデルではあまり重視されなかったスキーマ表示に関しても考慮されている。Obase モデルの特徴をふまえた上で、枝属性やスキーマ進化機構を ASKA モデルに採り入れ、全体を通して Obase モデルと比較させながらこのモデルの概念を述べる。

An Object-Oriented Data Model with Multiple Hierarchy Dimensions

Nobuo Hori　Masatoshi Yoshikawa　Shunsuke Uemura

Graduate School of Information Science
Nara Institute of Science and Technology

8916-5 Takayama, Ikoma, Nara 630-01, Japan

We developed an object-oriented data model ASKA. This model supports schemas based on class hierarchies and introduces template hierarchies. Therefore, it arranges data objects functionally and efficiently. We consider the schema expression of the model, which has not been paid much attentions to in other data models. After careful consideration of characteristics of the Obase model, we adopt the edge attribute and the functionality of schema evolution in the Obase model into ASKA model. We describe this model in comparison with the Obase model throughout the paper.

1 はじめに

共通するデータや手続きをひとまとめにして表現する一方で、オブジェクトに特有なデータや手続きは下位のクラスで定義できるという理由から、オブジェクト指向データモデルでは、クラスベースの階層が必要である [7] ことが多い。提案する ASKA モデルでは、クラス階層を主体としてスキーマを構築する。

Obase モデル [4, 5, 10] では、スキーマ進化への対応からクラスとインスタンスの枠を取り払い、継承は動的なもののみを認めたインスタンスベース階層主体のモデルであった。また、枝属性によりオブジェクトの多重化を試みていた。

ASKA モデルでは、クラスベース階層のままで Obase モデルにおける特徴を引き継ぐ方向を目指す。従って、スキーマ進化に対する何らかの機構を整える必要がある。また、Obase モデルの枝属性によって実現された機能に関しても、ビュー機構を用いずに実現する。

提案する ASKA モデルの特徴として以下のことが挙げられる。

- インスタンスベース階層を持った Obase モデルの特徴である枝属性やスキーマ進化機構を、クラスベース階層のままで実現する。
- 継承機構において、特に多重継承では、文献 [6] による優先度をつけるだけで属性の衝突を回避できるとは考えず、属性を 4 つの種類に分け、詳細に分析した。
- ASKA モデルは O_2 モデルにおけるクラス階層と型制約と類似しているが、属性を付加しないものをテンプレートとして、その階層を構築している。
- テンプレート階層の導入により、スキーマを 3 次元的に表現することで、データを機能的、かつ効率的に配置できる。
- スキーマ表示が空間的になるが、実際の設計スキーマと利用者に表示されるスキーマを写像するインタフェイスを作成し、利用者本位の表示機構を達成することを目指している。

以下、2節ではクラスの生成とテンプレートの概念について、3節ではスキーマおよびインスタンスの定義、4節ではテンプレート階層、5節ではオブジェクトの多面的な見方、6節ではスキーマ進化への対応を取り上げ、最後に7節で今後の課題・方針を述べる。

2 クラス生成とテンプレートの概念

この節では、is–a 関係におけるサブクラス生成について、そして ASKA モデルで新たに提案したテンプレートの概念について述べる。

2.1 is–a 関係におけるクラス生成

一般にクラス階層は is–a 関係を元にして構築されるが、Obase モデル [4, 5, 10] では、スキーマ進化の対応からモデルはクラスとインスタンスの区別をなくし、全てインスタンスベースとしてオブジェクトの階層を構成していた。インスタンスベース階層は、スキーマ進化において威力を発揮する。それは、全てのデータオブジェクトを階層構造に含み、あるデータ構造に属性を付加しても、そのデータオブジェクトに対するスキーマ進化を気にせずにそのまま階層構造の中へ組入れできることにある。しかし、多数のデータオブジェクトを抱えるデータベース構成において階層が複雑、かつ巨大になる場合、スキーマ表示の問題が起こる。あるクラス的な役割を担ったオブジェクトのサブオブジェクトとしてインスタンス的な役割を持ったオブジェクトが多数ある場合、スキーマ表示、つまり、データオブジェクトの位置付けを視覚的に得るのは困難である。

ASKA モデルではスキーマ表示機構を強く意識する立場から、インスタンスベース階層では困難を伴うので、クラスベース階層を採用した。一般にサブクラスとスーパークラス間は is–a 関係である。具体的には、スーパークラスの全属性を継承し、特殊化のためにさらにいくつかの属性を付加したものがサブクラスといえる。しかし、is–a 関係は見方によって幾通りにも存在するものであり、必ずしも継承機構を効率的に運用する立場においては優れているとはいえない [3, 11] ので、利用者や設計者に恣意的にクラス階層が構築される。このスーパー・サブクラス関係にさらに型の制約を導入する。

2.2 テンプレート概念の導入

前節では、スーパークラスより属性を追加したものがサブクラスの条件であった。サブタイプであり属性が追加されないものをサブクラスとする方法 [1] もある。サブクラスは一般にスーパークラスを特化したものであるが、属性を追加しないものは性質上その度合が弱く、スーパークラスの種類と見なせる。その種類をここではテンプレートと呼ぶ。

一般に、クラスの属性に関して定義域のより小さい属性値をとることでテンプレートが作成される。テンプレート導入の利点は、クラス階層の作成過程にある。サブクラス生成法に頼るクラス階層は、属性が追加されないクラスが非常に多いと思われるが、属性が追加したクラスと追加されないクラスとのクラス階層上の配置の違いが明確にならない。クラス階層におけるクラスを効率良く配置しようとする場合、種類のような関係であるテンプレートをクラス階層上におくよりは、クラスの種類としてクラス階層とは直行した別の階層 (4節後述) 上に置くべきではないかと考えた。

もう一つの利点は、クラス階層のみを許した場合のクラスの組合せ爆発を防ぐことである。クラス階層のみであれば、属性の追加や属性値の定義域の変更をするだけでクラスが増えていき、その数が巨大になるとクラス階層はかなりの複雑さを持つことになり、スキーマ表示の問題とも絡んで、収拾がつかなくなる恐れがある。それ故、テンプレートを導入することで、クラス階層上に属性を付加しないクラス群をテンプレート階層上におく。

これにより、クラスの見易さのみならず、クラス配置の効率が先の利点と合わさってかなり向上できる。

3 クラス階層のみからなる ASKA モデル

本節では、ASKA の基本的な部分として、テンプレートの概念を含まないスキーマおよびインスタンスの形式的な定義とその説明を行なう。

3.1 複合値とオブジェクト

クラスとは、データの抽象を表したものである。つまり、データに共通の属性、振舞いを定義するものであり、インスタンス生成のための型版の役割をする。クラスは、データベースにおけるデータとはなり得ないものとする。

データオブジェクト (インスタンス) の生成は、ある一つのクラスを指定することで行なわれる。オブジェクトは、一般にクラスを具象化したものであり、データとみなせる。データオブジェクトは必ずある一つのクラスから生成したものであり、そのクラスに属さなければならない。あるクラスで定義された属性は、クラスに属しているオブジェクトを制約することになる。

クラスを表現する型として原子型、組型、集合型などを許す場合もあるが、ASKA では、組型のみを考える。従って、オブジェクトは以下に定義するようにオブジェクト識別子と組構造を持つ複合値と考える。

定義 3.1: $a_1, \ldots, a_n$ を属性名、$v_1, \ldots, v_n$ を原子値またはオブジェクト識別子とする (ただし、$n \geq 1$) とき、$[a_1 : v_1, \ldots, a_n : v_n]$ を**複合値**と呼ぶ。

オブジェクトは オブジェクト識別子と複合値の対である。オブジェクト識別子に対応する複合値は高々一つ存在し、また存在する場合はそれをオブジェクト識別子の値と呼ぶ。□

$\mathcal{O}$をオブジェクトの可算無限集合とする。以後、混乱が生じない場合はオブジェクトとオブジェクト識別子という用語を区別なく使用することにする。従ってオブジェクト識別子の値のことをオブジェクトの値と呼ぶこともある。

3.2 型

定義 3.2: 型は基本型、組型、集合型から成り、以下のように再帰的に定義される。

1. integer, boolean, string, real の 4 つの**基本型**は型である。
2. $a_1, \ldots, a_n$ を属性名、$d_1, \ldots, d_n$ を原子値またはオブジェクト識別子または基本型またはクラス名とする (ただし、$n \geq 1$) とき、$[a_1 : d_1, \ldots, a_n : d_n]$ は**組型**と呼ばれる型とする。
3. t を型とするとき、$\{t\}$ は**集合型**と呼ばれる型とする。

□

定義 3.2の組型において、d_iは属性 a_iの定義域を表す。ある組型のある定義域が基本型 integer であれば、その組型に所属する複合値はその属性値としてどのような整数値でもとり得ることを意味する。また、定義域が Person のようにクラスである属性を持つ組型の場合、その組型に所属する複合値は、Person クラスのインスタンスをその属性値として持つことを意味する。定義域として基本型やクラス以外にも原子値やオブジェクト識別子を指定できる。例えば、組型

```
[age:integer, name:string,
 married: yes, child:{Person}]
```

の場合、married 属性の定義域は型ではなく特定の原子値である yes となっている。このように属性の定義域として型ではなく特定値を宣言することができる。この型に所属する複合値の married 属性の値はもちろん yes である。また、定義域として集合型も認められる。上述の型に所属する複合値の child 属性の値は Person クラスのインスタンスの集合である。

3.3 スキーマ

定義 3.3: ASKA のスキーマ Sは、次のような四つ組 $(\mathcal{C}, \sigma, \mu, ISA)$ である。

1. $\mathcal{C}$ はクラス名の有限集合である。
2. σ は、$\mathcal{C}$から 組型への写像である。
3. μ は、(C, a) から 多重継承モード $\mathcal{M}$ = { 同値, 選択, 再定義, 相違 } への部分写像である。ただし $C \in \mathcal{C}$, a は $\sigma(C)$ に現れる属性名とする。
4. ISA は、$\mathcal{C}$ 上の二項関係でありクラス**階層**と呼ばれる。ただし $(\mathcal{C}, ISA)$ は、非巡回有向グラフである。

□

この定義のようにあるスキーマにおいてクラスはある組型へ写像される。たとえば、あるスキーマにおいてクラス MarriedPerson は前述の組型

```
[age:integer, name:string,
 married: yes, child:{Person}]
```

に写像される。

あるスキーマ Sにおいて、$(C_i, C_j) \in ISA$ なる二つのクラスがあったとき、C_i は (スキーマ Sにおいて) C_j のサブクラスであると言い、また C_j は (スキーマ Sにおいて) C_i のスーパクラスであると言う。

3.4 合法的なスキーマ

スキーマのクラス階層は型に関する制約を満足しなければならない。そこで次に型同士の包摂の概念を定義する。ASKA では組型の定義域として原子値やオブジェクト識別子も許すため、ここで定義する包摂の概念は、これらをも含む一般的なものとなっている。3.2節の説明で、複合値が型に「所属する」という用語を厳密に定義することなく使用したが、包摂の概念は、複合値の型への所属の概念も含んでいる。

定義 3.4: $S(\mathcal{C}, \sigma, \mu, ISA)$ をスキーマとする。また、t_1, t_2 を原子値またはオブジェクト識別子または基本型またはクラス名とする。t_1, t_2 が以下のいずれかの条件を満足するならば、t_1 はスキーマ S のもとで t_2 を包摂すると言い、それを $t_1 \sqsupseteq t_2$ と表記する。

1. $t_1 = t_2$
2. 原子値 t_2 の型が基本型 t_1 である。
3. クラス t_1, t_2 に対して、$t_2\ ISA^*\ t_1$ が成立する。(ただし、ISA^* は ISA の反射推移的閉包とする。)
4. t_2 が複合値 $[a_1 : v_1, \ldots, a_n : v_n]$、または、オブジェクト t_2 の値が 複合値 $[a_1 : v_1, \ldots, a_n : v_n]$ であり、t_1 が組型 $[a_1 : d_1, \ldots, a_n : d_n]$ の場合。ただしここで $1 \leq i \leq n$ に対し、v_i は d_i に包摂される。
5. t_1 は組型 $[a_1 : d_{11}, \ldots, a_n : d_{1n}]$、$t_2$ は組型 $[a_1 : d_{21}, \ldots, a_n : d_{2m}]$ であり、$1 \leq m \leq n$ が成立し、かつ $1 \leq i \leq m$ なる i について $d_{1i} \sqsupseteq d_{2i}$ が成立する。
6. t_1, t_2 は、それぞれ集合型 $\{t_{1x}\}, \{t_{2y}\}$ であり、$t_{1x} \sqsupseteq t_{2y}$ が成立する。

これらのうち 2, 4の場合は、t_2 は t_1 に所属するとも言う。□

定義 3.5: スキーマ $S(\mathcal{C}, \sigma, \mu, ISA)$ は、$(C_i, C_j) \in ISA$ なる任意の二つのクラス C_i, C_j が次のいずれの条件も満足するならば合法的なスキーマであるという。

1. $\sigma(C_j) \sqsupseteq \sigma(C_i)$
2. $\sigma(C_i)$ を組型 $[a_1 : d_{11}, \ldots, a_n : d_{1n}]$、$\sigma(C_j)$ を組型 $[a_1 : d_{21}, \ldots, a_n : d_{2m}]$ とするならば、$n > m$

□

以後、ASKA では合法的なスキーマのみを考慮する。合法的なスキーマでは、スーパ/サブクラス関係で考えると、スーパークラスの全属性を継承または再定義したサブクラスの属性値は、必ずスーパークラスの属性値の定義域に包含されなければならないことになる。原子値やオブジェクト識別子などを定義域として指定する属性に関しては再定義を行なえず、サブクラスはその値を踏襲するのみである。

サブクラスの条件として、スーパークラス型のサブタイプでり、さらに属性がスーパークラスのものより追加されなければならないというかなり強い制約を導入した。

3.5 継承機構

3.5.1 静的継承

ASKA モデルではクラスベース階層を構築することから、サブクラス生成時にスーパークラスの属性を継承させる。もちろん、静的継承された属性値はスーパークラスの属性値の型制約を被る。

3.5.2 動的継承

Obase モデル [4, 5, 10] ではオブジェクト生成時には全くスーパーオブジェクトから属性が継承されない。質問時に明示的に継承を宣言することで属性の動的継承がなされた。ASKA モデルでは静的に属性を継承するため、動的継承は再定義属性の場合におけるスーパークラスからの継承 (再定義する前の属性の継承) が必要な場合であるが、スーパクラスの属性の定義域の方がサブクラスのそれよりも「広い」ためこのような必要性はあまりないと考え、これを認めない。

3.5.3 多重継承

ASKA モデルでは複数のクラスをスーパークラスとするサブクラスの宣言を認める多重継承の概念を採用している。多重継承で問題になるのは、属性の衝突、つまり、同じ属性名の属性が複数継承した場合である。従って、多重継承をするクラスに関して、属性の継承の種類を明確にしておく必要がある。

定義 3.3における多重継承モードは、あるクラスがクラス階層において多重継承を行なう場合に、競合する属性はスーパクラスからどのような継承を行なうかを指定するモードである。4 種類の多重継承モードを順に説明する。

1. **同値** ‥衝突する属性の意味が全く同じで、かつ同じ定義域を持つ時に定義される。
2. **選択** ‥衝突する属性のいずれかを選択する。但し、選択した属性値の定義域は、全衝突属性値の定義域の共通集合に含まれなければならない。つまり、選択属性の値は共通集合のサブタイプにならなければならない。
3. **再定義** ‥属性値を変更する。型制約として、再定義された値の定義域は全衝突属性値の定義域の共通集合に包含されなければならない。
4. **相違** ‥衝突する属性が属性名が同じでも意味が異なる場合に用いる。従って、衝突属性の数だけ異なった属性が存在することになる。属性 age の場合で示すと、age 属性は削除する (実際は隠蔽している)。衝突する属性の数だけ属性が付加されることになるので、属性名としてはスーパークラス名を語尾につけることにする。age の場合では、ageClass1, ageClass2, ageClass3, ... となる。
   この新しく生成された属性は、スーパークラスの age 属性にリンクしている。多面的な見方 (5節後述) はこのリンクをたどることで実現される。

これら 1〜4 の属性の種類の情報は属性自身に持たせる。2 か 3 の情報を持ちたい場合、衝突属性値の共通集合が空であればその情報の属性が持てないことになり、この多重継承は実現できない。

3.6 インスタンス

定義 3.6: ASKA のスキーマ $S(\mathcal{C}, \sigma, \mu, \mathcal{ISA})$ が与えられたとき、S のインスタンスは、次の条件をすべて満足するような $\mathcal{C}$ から $\mathcal{P}^{fin}(\mathcal{O})$ への[1]写像 $Extent$ である。

[1] $\mathcal{P}^{fin}(X)$ は、集合 X のすべての有限部分集合の族を表すものとする。

1. for $\forall C_i, \forall C_j \in \mathcal{C}$:
   if $i \neq j$ then $Extent(C_i) \cap Extent(C_j) = \emptyset$
2. すべての $C \in \mathcal{C}$ および $o \in Extent(C)$ について o は C に所属する。

□

4 テンプレート階層の導入によるスキーマの多重階層化

テンプレートを実現することは質問処理に影響を与える。テンプレートはクラスの種類であると2.2節で述べたが、要は修飾句を持ったクラスと考え、質問時の条件文にテンプレートを使用することで、質問の簡素化がはかれることになる。従って、質問をより効率良くしようとすれば、テンプレートもある程度の数になり、系列化も必要となる。この節では、テンプレート階層について述べる。

4.1 テンプレート階層

一般にテンプレートは1つのクラスに多数できる。また、テンプレートは属性の種類毎にでき、一つのみならず複数の属性の組み合わせによるものも可能である。従って、テンプレートに対して何らかの系列化を行なうために、テンプレート階層を導入する。テンプレートの中で、一方のテンプレートのサブタイプになっているものが存在すると、テンプレート階層ではそのサブタイプになっているテンプレートをサブテンプレートとして扱う。

$Template A(a_1 = b_{11}, a_2 = b_{12}, \ldots, a_n = b_{1n})$
$Template B(a_1 = b_{21}, a_2 = b_{22}, \ldots, a_n = b_{2n})$
(ただし、$a_1, a_2, \ldots$ は属性、$b_{11}, b_{12}, \ldots$ は定義域であり、それは型や特定値を含む)
である時、$b_{11} \sqsupseteq b_{21}, b_{12} \sqsupseteq b_{22}, \ldots, b_{1n} \sqsupseteq b_{2n}$
であるならば、テンプレート B はテンプレート A のサブテンプレートという。

いずれのテンプレートも属しているクラスのサブテンプレートとしてテンプレート階層を構築する。クラス階層との違いは属性が追加されないものだけの階層構造であることである。テンプレート A はクラスでもあり得る。

テンプレートは属性値の制限を行なうような質問に対して一種の索引機構を提供する。例えば図1において、Person クラスであり、性別が男性である人を検索する時、sex 属性に関するテンプレート階層に男性を表すテンプレートが存在すると、そのテンプレートに属するインスタンスを返すことになる。また、クラスとテンプレートを組み合わせることで結合に近い操作を行なう上から、ビュー機構ともみなせる。これは、図1において男子である NaistStudent を検索する場合、Person クラスの sex 属性に関するテンプレート階層の Male テンプレートに属するインスタンスと、Student クラスの school 属性に関するテンプレート階層上の NaistStudent テンプレートに属するインスタンスの共通集合を返すことになる。仮想テンプレート MaleNaistStudent を作成することでビューにもなる。このビューに属するインスタンスは、NaistStudent と Male テンプレートの両方に属していて、インスタンスに対して属性値の定義域制約内の更新操作も可能である。

このように、クラス階層に属するインスタンスはテンプレート条件に合う限り、そのテンプレートのインスタンスにもなっている。すなわち、インスタンスは一つのクラスとゼロ以上のテンプレートに属することを意味する。

属性に関するテンプレート階層で、複数の属性に関してのテンプレートももちろん定義できる。図1では school 属性に関するテンプレート階層上の MaleNaistStudent がこれに当たる。

このクラス・テンプレート階層は、スキーマ階層の多重階層化に行きつく。大量のデータを抱えるデータベースシステムでは、いかにスキーマを整理統合し、利用者に使い易い環境を提供するかが重要である。従って、クラスにクラス階層とは異なった平面上でテンプレート階層を付け加えることでスキーマを3次元的に表現でき、データオブジェクトをかなり機能的に配置できる。スキーマ階層自体3次元であり、利用者がその複雑さを意識せずに、つまり、スキーマ表示の問題が無いようにデータベースを扱えるために、利用者にはクラス階層を提示し、質問によってテンプレートを自動的にシステムが構築する方法をとる。

4.2 テンプレートにおける継承

この場合の継承はクラスの継承と同じように認められるので、テンプレート作成はある1つのテンプレート(クラス)を指定すると、テンプレート階層の中に組み込まれる。

また、複数のテンプレートをスーパーテンプレートとする多重継承もクラス間と同様に認められる。3.5.3節で述べた属性の衝突は存在するが、4の情報を持つことはない。これは、ある1つのクラスから派生したテンプレート階層の中での多重継承であるので、衝突する属性の意味が異なるということはあり得ないと考えるからである。

4.3 インスタンスのテンプレート化

Student クラス (name = string, age = integer, sex = boolean, school = Univ) にインスタンス NaistStudent(name = string, age = integer, sex = boolean, school = "Naist") があるとする。これはデータオブジェクトである限り、インスタンスを生成できない。例えば、インスタンス堀 (name = "hori", age = 25, school = "Naist") を作成したい場合、Student クラスに属するインスタンスとして生成しなければならない。Obase モデル [4, 5, 10] において、オブジェクト NaistStudent が Student オブジェクトのサブオブジェクトであれば、堀

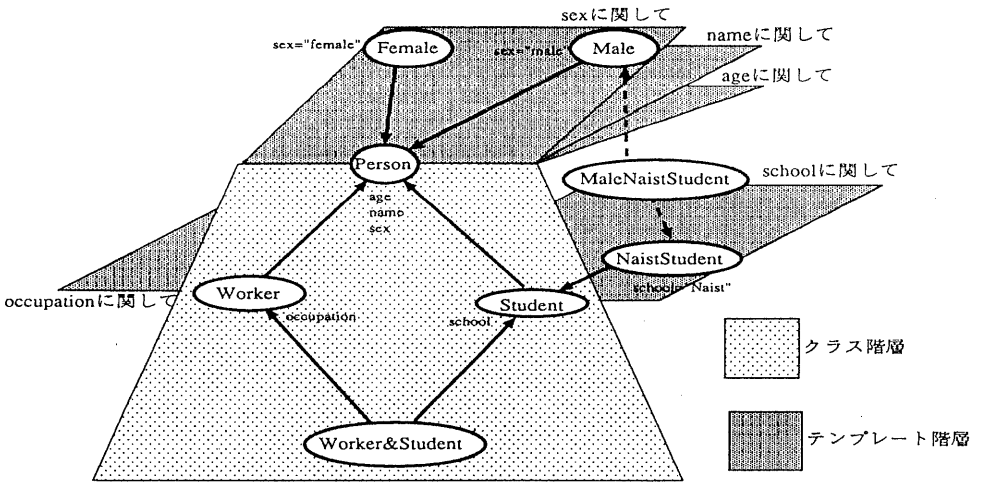

図 1: ASKA モデルにおけるクラス・テンプレート階層

オブジェクトをサブオブジェクトとして生成でき、動的継承機構を利用できるので、堀オブジェクトの宣言においてschool属性を定義する必要がなかった。そこで、あるインスタンスにクラス的な役割を持たせたい場合の対処として、インスタンスのテンプレート化を考える。

インスタンス NaistStudent をテンプレート化すると、Student クラスのテンプレート NaistStudent となる。この場合、属性の追加がなく、Student クラスで school 属性が NaistStudent クラスで "Naist" という具体的なオブジェクト値になった。テンプレート階層では、NaistStudent は Student のサブテンプレートである。テンプレート NaistStudent は元々Student クラスのインスタンスであるので、サブテンプレートとしてのサブタイプ条件を満たしている。

NaistStudet インスタンスはテンプレート化するまでは属性値として型を持っていた。このように、インスタンスが型の属性値を持つことは少なからず存在するので、この場合の Student クラスは抽象クラスの役割を持ち、基本的にはインスタンスを生成するよりは、サブクラスの生成を目的とする。抽象クラスとは、インスタンスを生成しないクラスであり、そのサブクラスの共通の属性やメソッドをまとめあげて保持するものである。クラス階層においての抽象クラスの存在は、細部へのこだわりをなくし、下位の具象クラス群を大づかみすることを可能にし、そのことが認知的経済性を高める [8]。また、属性やメソッドの再利用が促進され継承効率も上がることが多い。

5 オブジェクトの多面的な見方

実世界をモデリングする際に一つの実体が複数の側面を持つ、すなわち、環境が変わればその実体の見え方が変わることをオブジェクトの多面的な見方と呼ぶことにする。

この節では同じ例 (図 2, 3) を用いて、まず Obase モデルにおける枝属性を紹介し、その多面的な見方の特徴を引き継ぎ、その問題を克服した機構を ASKA モデルに採用する。

5.1 Obase モデルにおける枝属性

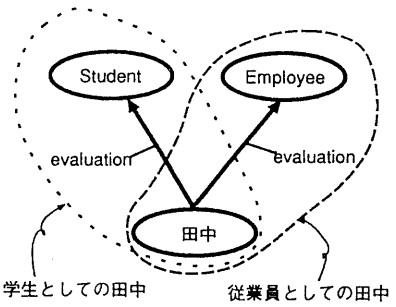

図 2: Obase モデルにおける枝属性

Obase モデル [5] において、この多面的な見方を支援する機構は枝属性であった。例として、Student と Employee オブジェクトのサブオブジェクトとして田中オブジェクトを定義する場合を考える (図 2参照)。Student と Employee オブジェクトの属性は重複がない限り田中オブジェクトに動的に継承される。成績を表す属性 evaluation が両方のオブジェクトにある場合、静的継承であると属性の衝突が起こるが、質問時に動的に継承する場合は、継承されるルート情報さえ示せば属性の衝突は起

こらない場合が多いので、ここでは問題として取り上げないことにする。

枝属性の概念が発生したのは、例えば、田中オブジェクトに対して evaluation 属性をつける場合、Student と Employee オブジェクトからの 2 つの見方が存在し、かつ Student や Employee オブジェクト、田中オブジェクトのどれにも属性をつけるのが適当でない状況であったからである。従って、Student と田中オブジェクトの間の枝に属性 evaluation を存在させ、同様に Employee オブジェクトとの間にもその属性を存在させることにより、この問題に対処した。

この枝属性の特徴としては、田中オブジェクトの Student としての見方や Employee としての見方を検索できることである。オブジェクト指向の概念から考えると、全てのものはオブジェクトとしなければならない。従って、枝属性のような属性のみ単独で存在することは概念的に問題がある。枝属性へのアクセスは質問時の動的継承機構により、継承された属性として枝属性が扱われてはじめて可能となる。

枝属性では多面的な見方を支援するあまり、行きつくところ全オブジェクト階層に枝属性が付けられることになる。これでは、オブジェクト自身が属性を持たず、全て枝属性のみで成り立つことにもなり、オブジェクトとしての存在意義が薄れる。ASKA モデルではこのような問題点を意識し、多面的な見方を支援する機構を構築した。

5.2 ASKA モデルにおける多面的な見方

Student かつ Employee である田中オブジェクトを生成しようとすると、Student と Employee クラスをスーパークラスとする Student&Employee クラスを作成し、そのインスタンスとして田中オブジェクトを生成する。この多重継承によって evaluation 属性の衝突が起こるが、3.5.3 節によりこの属性は 4 の情報を持つことになる。つまり、evaluation 属性は属性名は同じでも意味が異なるものであるので、Student&Employee クラスでは evaluation 属性を削除し、evaluationStudent と evaluationEmployee 属性を新たに生成する。

多面的な見方を支援するため、Student&Employee クラスのインスタンスである田中オブジェクトを射影する。1 や 2、3 の情報を持った衝突属性の場合、いずれの属性値の定義域も全衝突属性の共通集合内にあるので、属性はスーパークラスからの見方の場合でもそのまま存在できる。

次に、4 の情報を持っている属性が存在する場合の対処方法である。“ evaluation クラス名 ” 属性は、元のクラスの evaluation 属性へリンクしている。Student としての見方を実現しようとすると、新しく生成された 4 の情報を持つ evaluationStudent 属性は、Student クラスの evaluation 属性にリンクしているので、属性名は evaluation に変更され、属性値は evaluationStudent 属性の値と同じである (図 3参照)。

これにより、あるオブジェクトは属しているクラスのクラス階層を遡ることで、いかなる多面的な見方にも対応できる。反対に、サブクラスへ下っていけば、多面的な見方の考え方では対処できない。これは移動 (migration) の問題に結び付くからである (6.2.2参照)。

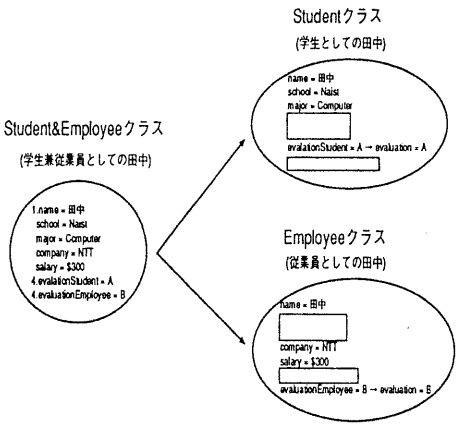

図 3: ASKA モデルにおける多面的な見方の実現

5.3 多面的な見方における属性値の更新

多面的な見方を支援した場合、インスタンスの射影の更新は主に属性値の更新である。もちろん、これは型制約の範囲内で行なわれる。

射影の実現は異なったオブジェクト識別子を持ったオブジェクトを生成するのではなく、同じオブジェクト識別子を持ったインスタンスにマスクをしただけである。つまり、必要でない属性を隠している操作なので、その射影されたインスタンスに対する属性値更新は、元のインスタンスの更新を意味することになる。4 の情報を持った属性の場合も、操作はやや複雑になるが同様に実現できる。

6 スキーマ進化への対応

6.1 スキーマ進化の問題

クラスベース階層においては、スキーマ進化に対応した何らかの方針が必要である。スキーマ進化の問題とは、あるクラスにさらに属性を加えたものを作成しようとすると、新たにその属性を含んだクラスを定義しなければならなくなり、スキーマ進化を多く抱えるデータベースでは、クラス階層の変更が頻繁に起こることである。従って、クラス階層の組変えに伴うデータベースの再コンパイルが必要になり、処理が繁雑しかも負荷が増大することから、Obase モデル [4] ではインスタンスベースになった経緯がある。

この問題に対しては、自動クラス生成機構を採り入れた概念を導入することでスキーマ進化への対応はある程度システムに任せ、この問題の軽減をはかる。

6.2 スキーマ進化機構

図2の例で示したObaseモデルは、スキーマ進化を気にせずにStudentとEmployeeのサブオブジェクトとして田中オブジェクトが定義できるところに特徴があった。しかし、提案したモデルにおいては多重継承を行なうクラスを生成してからでないとオブジェクトを生成できない。スキーマ進化を頻繁に行なうデータベースではこれは大きな問題となる。そこで、まずStudent&Employeeクラスを定義せずに学生兼従業員の田中オブジェクトを生成する場合、Obaseにおける場合と同様に、StudentとEmployeeの両方のクラスの特徴を持ったオブジェクトとして田中オブジェクトを定義する。オブジェクトの宣言に複数のスーパークラスを指定した場合は、システムは自動的にStudentとEmployeeクラスをスーパークラスとするStudent&Employeeクラスを作成し、そのクラスに田中オブジェクトを生成させる方法をとる。

6.2.1 インスタンスのテンプレート化の場合

インスタンスのテンプレート化においては、4.3節の例ではインスタンスNaistStudentから堀インスタンスを宣言すると、堀インスタンスはNaistStudentテンプレートに属する。この例では明示的にテンプレート化を行なったが、この場合は暗黙的にテンプレートを作成し、宣言されたインスタンスをそのテンプレートのインスタンスにさせる。

6.2.2 移動の場合

オブジェクトの役割が変化し、属性の追加や削除が行なわれることは設計などの分野ではよくあることである[9]。ASKAモデルにおいてはこの移動機構を許す。属性が追加される場合は、オブジェクトが生成されたクラスにその追加属性を加えたサブクラスが生成され、オブジェクトはそのインスタンスとなる。もし、そのようなサブクラスが存在していれば、生成は必要がなくなる。削除も同様に、スーパークラスを作成する。

この機構により再コンパイルを行なうシステム側の負荷は変わらないが、利用者にとっては多重継承を行なう繁雑さやクラスとインスタンスの違いをあまり意識せずに、自動的にクラス作成やクラス階層の組替えが可能となる。

7 おわりに

今回の研究では、スキーマ表示を重視するデータモデルを考案し、データオブジェクトの配置の効率に特に力を入れた。また、Obaseモデルの特徴である枝属性の機能を踏襲し、オブジェクトの多面的な見方を実現した。クラスベース階層のままで、スキーマ進化への対応も行なった。

今後の課題としては、インスタンス階層の実現を考えている。これは利用者にはほとんど負担をかけないように行ないたい。インスタンス階層を導入するのは時間の概念を採り入れたデータの取り扱いを考慮するためである。文献[2]では、インスタンスの属性が変更される度にインスタンスをコピーし、インスタンスの履歴を管理し、ポインタでそれらをリンクすることで、時間軸を採り入れた質問ができた。

ASKAモデルは多面的な見方を必要とするデータが多くあるものや、テンプレートが作成する環境が必要であるものには最適である。このモデルの実用例を考えていく上では、クラスとテンプレートがデータ並に増えないようにする機構が必要であると思われる。

謝辞: 貴重なご議論を頂いた石川佳治助手を始めとする植村研究室の諸氏に感謝致します。

参考文献

[1] François Bancilhon, Claude Delobel, and Paris Kanellakis. *Building An Object-Oriented Database System (The Story of O_2)*. Morgan Kaufmann, 1992.

[2] Wolfgang Käfer and Harald Schöning. Realizing a Temporal Complex-Object Data Model. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pp. 266–275, 1992.

[3] W. Lalonde and J. Pugh. Subclassing ≠ Subtyping ≠ Is-a. *Journal of Object-Oriented Programming*, Vol. 3, No. 5, pp. 57–62, 1991.

[4] Obaseコンソーシアム. Obaseプロジェクト第二回研究成果報告書. 千里国際事業財団, 1993.

[5] Obaseコンソーシアム. 第3回Obaseシンポジウム資料集. オブジェクト指向データベース研究プロジェクト公開成果報告書, Mar 1994.

[6] Edward Sciore. Object Specialization. *ACM Transaction on Information Systems*, Vol. 7, No. 2, pp. 103–122, Apr 1989.

[7] 宇田川佳久. オブジェクト指向データベース入門. ソフト・リサーチ・センター, 1992.

[8] 青木淳. オブジェクト指向システム分析設計入門. ソフト・リサーチ・センター, 1993.

[9] 蛭井潤. オブジェクト指向データベースのビュー機構について. 文献調査報告書, 1993.

[10] 吉川正俊, 田中克己, 上善恒夫, 田中康暁, 蛭井潤, 堀田光次郎. ObaseSQL:拡張経路式と継承演算子を持つオブジェクトベース言語. In *Proceedings of Advanced Database Symposium '93*, pp. 63–72, 1993.

[11] 田渕敦之, 堀宣男, 中野秀男. オブジェクト指向におけるクラス階層に関する考察. ソフトウェアシンポジウム論文集. ソフトウェア技術者協会, 1993.