データベースシステム　97－7
（１９９４．　３．　１８）

# 不定属性集合と動的継承機構を持つオブジェクトベースのための質問処理

田中 康暁　　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

Obase に基づくオブジェクトベースにおける、二次記憶上のファイル配置の最適化と索引を用いた高速処理について考察する。Obase モデルは、スキーマとインスタンスの区別をしない、質問文に無名属性変数や継承付き航行演算子を用いた経路式を使用できる、などの特徴を持つため、従来のデータモデルとは異なったクラスタリングと索引が必要になる。本論文では Obase モデルに適用できる論理クラスタリングと索引を提案し、その得失について述べる。

# Query Processing for an Objectbase with Unfixed Attribute Set and Dynamic Inheritance Mechanism

Yasuaki Tanaka　　Masatoshi Yoshikawa　　Shunsuke Uemura

Graduate School of Information Science
Nara Institute of Science and Technology

We investigate the file organization issues for an object model which is based on the Obase model. The Obase model does not distinguish schema and instance. Also, the language developed for the model, ObaseSQL, has unnamed attribute variables and inheritance operators. Hence, it is necessary to employ new clustering and indexing techniques which take into account the features of the objectbase model. In this paper, we will present the logical clustering and indexing techniques for the Obase data model.

# 1 まえがき

現在、新しい概念に基づくオブジェクトデータベースであるObaseデータベースシステム [1] の設計と実装を行っている。Obaseデータベースの操作言語であるObaseSQL言語は、従来のデータベース操作に用いられる経路式に、継承付き航行演算子および正規表現 (無名属性変数や閉包演算) を導入し、機能を拡張している。これらの機能を用いた質問処理に対する最適化手法として、従来の経路式には経路式索引などの方法が提案されているが、過去の論文などには上記に述べたの拡張に対する最適化については触れられていない。

無名属性変数、継承付き経路式を扱う場合の問題点を以下に示す。

- 無名属性変数には任意の値が入り得るため、検索範囲が大きくなる。
- 閉包演算では、その閉包部分に何個のオブジェクトが入るか決まっていない。
- 継承付き演算を行った検索結果の場合、更新すべきデータが必ずしも検索されたオブジェクト上にある訳ではない。
- 多重継承を持つ経路の場合、どちらからの継承かを明示する必要がある。

これに対し本論文では、Obaseデータベースの質問処理の最適化を目的として、無名属性変数や継承付き経路式に対応した索引を構築し、索引を通して実行することにより検索、更新等の質問処理を高速化する。これに関連して、格納されたデータへのアクセス処理を高速化するための記憶媒体上の論理的なファイル配置 (論理クラスタリング) についても述べる。

# 2 Obaseデータモデル

Obaseデータモデルは、マルチメディアデータの柔軟な表現を目的として開発され、各オブジェクトはサブオブジェクト集合およびプロパティ列の二つの構造的構成子を持ち、型、クラスやインスタンスオブジェクトを区別せずに統一的に扱うという特徴がある [1]。

Obase オブジェクトは、オブジェクト識別子 (oid)、サブオブジェクト集合、プロパティ列の三つの構成子から構成される。

図 1に自動車 (auto) と人間 (person) に関するObaseデータモデルの例を示す。

このモデルでは、autoのサブオブジェクトとして3種類の自動車a1、a2、a3と、a2にオプション (カーオーディオ、自動車電話) を付けたa21、a22の計5台が定義されている。またpersonのサブオブジェクトとしてp1、p2の二人の人間オブジェクトが定義され、p1がa2とa3の2台、p2がa3の自動車を所有している。図 1にこのモデルのスキーマ表現を示す。

どの自動車オブジェクトも、参照経路を辿ると、ドア (door) という名前の、オブジェクトを参照するプロパティを持つ。ところが、例えばa1のドアの色 (color) を参照したい場合、doorの指すオブジェクト d1にはcolorの記述がない。このモデルでは、colorが下位継承可能なプロパティであり、a1.b1.d1の経路でドアd1が参照されるので、経路中のa1とb1に指定されたcolorのどちらかの値が継承される。ここでは経路上でd1により近い位置にあるb1.colorの値'red'がd1に継承されると規定する。同様にa2はb2.colorの値がd2に継承され'red'が、a3はd3に直接記述されているcolorの値'red'がそれぞれドアの色として参照される。

またa21、a22はa2のサブオブジェクトとして定義されているため、a2のドアの色'red'が継承され、a21、a22のドアの色も'red'になる。

# 3 Obase質問言語

Obaseデータモデルの質問は、ObaseSQLによって行う。ObaseSQLの構文はSQLに倣い、検索文については目的オブジェクト、変数の範囲、検索条件式の指定をそれぞれSELECT句、FROM句、WHERE句で行う。

従来のSQL文と異なり、ObaseSQLではFROM句における変数の範囲の指定は必ずしも必要でなく、FROM句での範囲指定なしに用いられた変数は、全オブジェクトを指定範囲と見做す。WHERE句における検索条件式は、Obase代数式、ObaseSQL文、メソッド式などの表記が可能であるが、

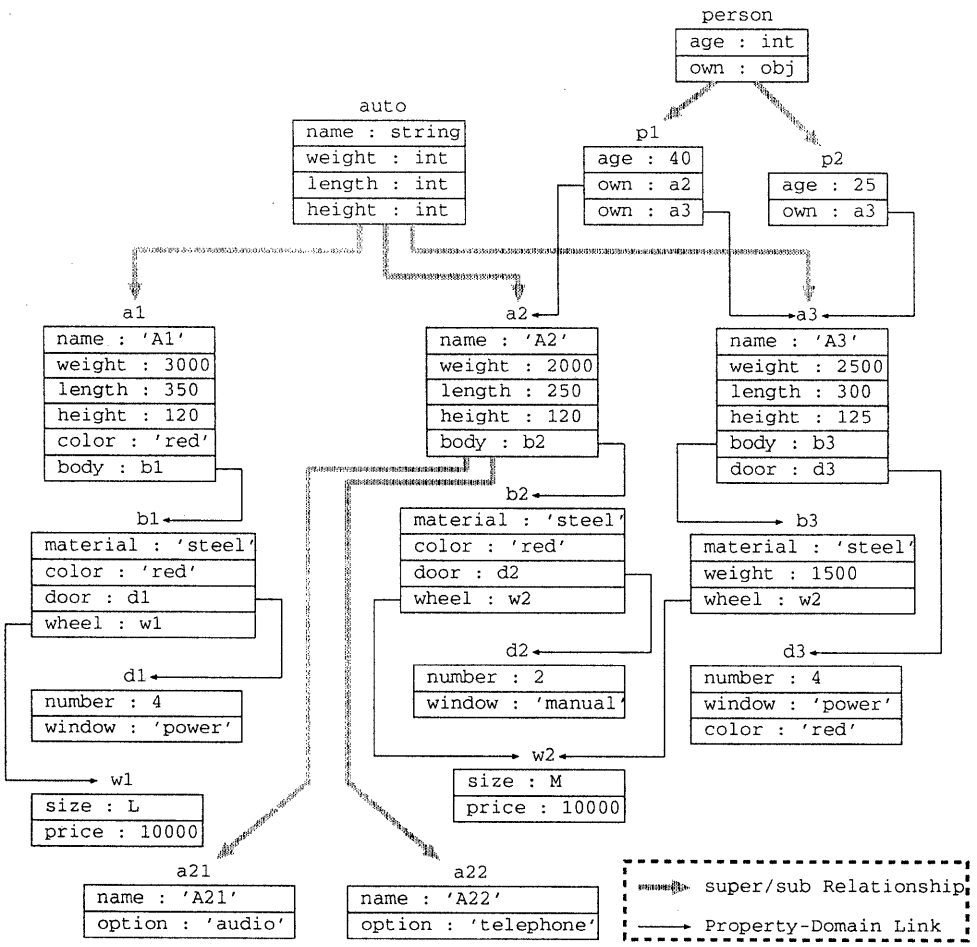


図 1: auto データモデルのスキーマ図

次節以降ではオブジェクト中にあらかじめ与えられているメソッド式を使った条件文のみを仮定し、メソッド式に対する質問処理の最適化を考えることにする。

## 4 Obase のファイル構成

データベースシステムの記憶構造を考える上で、二次記憶上に格納されたデータに対するアクセス処理を高速かつ効率的に行うことが重要になる。従来のデータベースシステムでは、データの集まりが比較的単純で規則的であり、データ構造を二次記憶上での格納に適したレコードに展開するのは容易であったが、一般にオブジェクトベースシステムでは、内部に他のオブジェクトへの参照を含む複合オブジェクトなどがあり、このような複雑な構造を二次記憶上に展開する論理クラスタリングが必要になる。

従来のオブジェクトベースシステムと Obase システムを比較した場合、従来のシステムでは、

1. 各オブジェクトにクラスが定義されており、各クラス内では規則的なデータ構造を持つ。
2. スキーマにより各クラスの階層構造が厳密に定義され、データベースのインスタンスとして super/sub の階層が規定されている。

という特徴がある。これに対しObaseモデルは、オブジェクトにクラス型の概念がなく、各オブジェクトに対して型を自由に決められる。このため論理クラスタリングを行う場合、

1. 規則的なデータ構造を持たないので、構造単位の正規化ができない。
2. クラスが定義されないのでサブオブジェクト階層の構造をインスタンスとして記憶できないため、サブオブジェクト階層情報も明示できる論理クラスタリングが必要になる。

などの問題が生じる。

このため、Obaseモデルでは、オブジェクト同士の関連として参照構造とサブオブジェクト階層の二つの構造を持つ論理クラスタリングを行う必要がある。

またクラスタリングにより展開されたデータに対し、内容検索処理を高速に処理するためには、アクセス情報を付加した索引技術も不可欠である。

Obaseに適用する論理クラスタリングが上のような特徴を持つため、索引についてもObaseに適した構成を考えなければならない。

# 5 ファイル配置技術

## 5.1 従来のクラスタリング技術

従来のオブジェクトデータベースは、クラスやスキーマが厳密に定義されており、各クラス内のオブジェクトの構造は固定されている。このようなモデルに対するオブジェクトの論理クラスタリングの方法は、基本的には直接クラスタリングと正規化クラスタリングに分けられ、どちらもさらに幾つかの方法に分けられる [2, 3]。以下に主な論理クラスタリングの方法について説明する。

**a 直接クラスタリング**

内部に構造を持ったオブジェクトを、その構造通りに展開して直接格納する方法。あるプロパティ値がオブジェクトであれば、そのオブジェクトの内容も一緒に記述するというように、データベースの階層構造を入れ子形式で記述する。

実際のシステムでは、上記の方法をそのまま適用するのではなく、スキーマ構造を適当な大きさに分割し、各部分について直接クラスタリングを行うという手法が用いられる。以下に$O_2$システムにおいて用いられる例 [4] を紹介する。

**a′ 配置木クラスタリング**

$O_2$のクラスタリング法は、一つのファイルに格納する単位を自由に指定できる。どのオブジェクトをどのファイルに格納するかという指示は、クラス毎に配置木と呼ばれる概念スキーマを表す部分木を作ることで行う。またオブジェクトの共有を複製によらず、オブジェクト識別子によるポインタで行う。配置木の最適化は、想定したメソッドによるデータ読み出しコスト (ファイルアクセス回数、メモリにロードされるデータ量) が最小となるように決定される。

以後の議論では、この論理クラスタリングを便宜上、配置木クラスタリングと呼ぶことにする。

**b 正規化クラスタリング**

オブジェクトに適当な正規化を行い、原子値や識別子を用いた規則的なデータ構造を単位として、ファイルに格納する方法。データは内部に入れ子を持たない平坦なレコードの集まりとして管理できる。正規化の単位により、以下の三つに分けられる。

**b-1 NSM(N-ary クラスタリング)**

オブジェクトをクラスで分類し、クラス毎にファイルに格納する。あるクラスがN個の属性を持てば、そのクラスのファイルはN個の値を持つデータの連続で記述される。

**b-2 DSM(分解クラスタリング)**

オブジェクトを属性値単位まで分解し、オブジェクト識別子と属性値との組で別々のファイルに格納する。

**b-3 P-DSM(部分分解クラスタリング)**

DSMとNSMの中間に位置付けられるクラスタリングで、同時に参照されることの多い属性値は一つのファイルに格納する方法である。

## 5.2 Obaseへの適用

Obaseデータモデルは参照構造とサブオブジェクト階層の二種類の構造を持つ。従来のオブジェ

クトベースシステムの論理クラスタリングは、参照構造を展開したものである。

Obase モデルの参照構造には、オブジェクトが規則性を持たないために NSM 型の正規化クラスタリングは適用できない。Obase データモデルに直接適用できるものは、直接 (配置木) クラスタリングと、DSM 型 (分解) クラスタリングである。

NSM 型の正規化クラスタリングの代わりに同等なグループ分けをできるファイル単位として、以下のものが考えられる。

- 内部に同じプロパティ名を持つオブジェクトを一つのファイルに格納する。
- プロパティ値がオブジェクトを参照する場合、そのプロパティ名が同じオブジェクトは一つのファイルに格納する。

この論文では、このグループ分けを適用した論理クラスタリングを中間クラスタリングと呼ぶことにする。

参照構造がプロパティとオブジェクトの関連を表すのに対し、サブオブジェクト階層はオブジェクト同士の関連を表す。このサブオブジェクト階層情報を論理クラスタリングで実現する方法として、以下の五つが考えられる。

1. オブジェクト間のサブオブジェクト階層を記述したファイルを追加する。
2. サブオブジェクト階層を持つオブジェクトに、subobjects という名前のサブオブジェクト集合を指すプロパティを加え、サブオブジェクトへのポインタを持たせる。
3. 同一のオブジェクトをスーパオブジェクトとするものを、一つのファイルにまとめる。
4. スーパオブジェクトの構造中にサブオブジェクトを入れ子に展開して記述する。
5. スーパオブジェクトから継承されるプロパティ値をすべてのサブオブジェクトに複製して記入する。

上記の方法をそれぞれ (1) ファイル追加型、(2) サブオブジェクト集合型、(3) グループ化型、(4) 統合型、(5) 継承値記入型と呼ぶことにする。

1 のファイル追加型は、追加するファイルの形式は元のファイルと独立に決定できるが、今回は追加元のファイル形式との整合を考慮し、同様のクラスタリング方法で追加することとする。

3 のグループ化型の場合、ファイルにまとめる単位として、単純にあるオブジェクトがサブオブジェクトとして指すものだけを集める方法と、さらに再帰的にそのサブオブジェクトが指すものまですべてを集める方法とが考えられる。後者の場合、オブジェクト同士のサブオブジェクト階層が曖昧になるので、ファイル追加型等の他の方法を併用する。

5 の継承値記入型は、更新操作を考慮すればどのオブジェクトから継承された値かという継承元の情報も必要になり、後述の例のように入れ子構造で記述する必要がある。従って、統合型および継承値記入型は、入れ子構造が可能な直接/配置木クラスタリングでのみ可能である。

これまでの議論を表にまとめると、以下のようになる。

| subobject 階層の記入方法 | クラスタリング | | | |
|---|---|---|---|---|
| | 直接 | 配置木 | 中間 | 分解 |
| ファイル追加型 | ○ | ○ | ○ | ○ |
| subobject 集合型 | ○ | ○ | ○ | ○ |
| グループ化型 | ○ | ○ | ○ | × |
| 統合型 | ○ | ○ | × | × |
| 継承値記入型 | ○ | ○ | × | × |

表中、○は構築可能な論理クラスタリングを、×は不可能なクラスタリングを表す。

ここで、論理クラスタリングの例として図 1の auto データモデルを考え、上記のクラスタリングを行なった例を示す。

この例では [ ] で囲まれた要素を一つの構造単位とする。なお、各オブジェクトの最初の要素には、オブジェクト識別子 (oid) を記入することにする。

**a ファイル追加型**

図 1のスキーマ図からサブオブジェクト階層の枝を除いたグラフに対して論理クラスタリングを行い、それにサブオブジェクト階層を記述したファイルを追加することですべての情報を記録する。追加ファイルの情報としては、スーパオブジェクトとサブオブジェクトの対を記述すれば十分である。

オブジェクト a1、a2、a3 を auto のサブオブジェクトと定義するための追加ファイルは以下のようになる。

(1) 配置木クラスタリング型

```
super/sub ::=
  [ auto の OID,
      [ a1 の OID,
        a2 の OID,
        a3 の OID
      ]
  ]
```

(2) 中間クラスタリング型

```
super/sub ::=
  [ auto の OID, a1 の OID ]
  [ auto の OID, a2 の OID ]
  [ auto の OID, a3 の OID ]
```

例 (1) はサブオブジェクト階層を直接クラスタリングの記法で表現し、(2) はスーパオブジェクトとサブオブジェクトの組をオブジェクトとして表現している。この方法は、参照構造とサブオブジェクト階層とを別々のファイルに記録できるという特徴がある。

**b サブオブジェクト集合型**

サブオブジェクト階層を持つオブジェクトに、サブオブジェクト集合 subobjects というプロパティを挿入してサブオブジェクト階層情報を記述する。例えば auto オブジェクトの場合、プロパティ subobjects には {a1,a2,a3} という集合値が記入される。

またサブオブジェクト集合の記述法に、Join Index [5] と呼ぶ方法がある。これは Join Index という新しいオブジェクトを作り、そこに集合値を記述するものである。この方法はサブオブジェクト階層を多く用いる経路式の検索に有利である。

Join Index を用いた場合の例を、以下に示す。

```
autofile ::=
  [ oid : auto の OID,
    subobjects : #201,...
  ]

subobjects ::=
  [ oid : #201,
    sub : a1 の OID,
    sub : a2 の OID,
    sub : a3 の OID ]
```

**c グループ化型**

auto データモデルの場合、

```
auto/ ::= [[a1][a2][a3]]
a2/ ::= [[a21][a22]]
person/ ::= [[p1][p2]]
```

の三つのファイルを作ることができる。ここで [a1] 等で表現したものはオブジェクトを表し、その内部は直接/配置木/中間クラスタリング等により既にデータ配置が実行されているものとする。またオブジェクト a2 が auto のサブオブジェクトなので、初めの二つを結合して

```
autofile ::= [[a1][a2][a3][a21][a22]]
```

としてもよい。

なお、この方法を中間クラスタリングに適用した場合、上記のファイル構成をオブジェクトのグループ分けの一環とみなすことができる。

**d 統合型**

サブオブジェクト階層を他の構造と同様に入れ子構造で記憶させ、一つのファイルにまとめる。前述のように直接あるいは配置木クラスタリングでのみ可能な方法である。

この方法では、あるオブジェクトに対するスーパオブジェクトの値は、その入れ子より外側にある値を参照すればよい。なお、この方法は、オブジェクトの階層が多くなるとデータ長が長くなり易く、また入れ子構造が参照オブジェクトの場合とサブオブジェクトの場合との区別を付ける機構が必要になる。さらにトップダウン式の記述のため、多重継承を持つオブジェクトに対するアクセスの場合、アクセス回数が多くなる、などの問題がある。

**e 継承値記入型**

直接クラスタリングされたファイル内で、サブオブジェクト階層で上位オブジェクトから継承されるプロパティを持つサブオブジェクトすべてにその継承値を記入する。

この方法には、継承された値が元々どのオブジェクトの値かを記録する機構や、オブジェクト自身が持つ値と継承値とを区別する機構が必要であり、また二つ以上の枝を経た継承が行われる場合、サブオブジェクト階層の経路情報を記入できない、階

層が多くなるとプロパティ値の複製が増大するため更新には不利、などの問題がある。

上記のように、統合型、継承値記入型は問題点が多いため、あまり実用的な方法とはいえない。今回はこの二つの方法はObaseモデルのクラスタリングには適さないと見て、以後の議論では考えない。

## 5.3 オブジェクト操作を考慮したクラスタリングの選択

参照構造の展開方法の内、直接クラスタリングはファイルが大きくなり易く、またファイル構成を最適化できないなどの欠点があり、また分解クラスタリングはObaseで用いられる経路式による航行検索には適さない。

従って、Obaseモデルに適した論理クラスタリングは、配置木または中間クラスタリングにファイル追加型、サブオブジェクト集合型、グループ化型のいずれかを適用したものとし、これらの中から用途に応じて選ぶものとする。

適用するクラスタリング法を選択するための手がかりとしては、データベース上でどのような検索操作および更新操作が実行されるかを見る。すなわち、検索操作において複合オブジェクトを検索する場合、検索の対象がオブジェクト全体を指すかあるいはある要素だけが必要かにより、前者であれば配置木クラスタリング、後者であれば中間クラスタリングの方が有利である。

また更新操作を考えた場合、データベースの更新操作には

1. プロパティ値(原子値)の変更
2. プロパティ値(オブジェクト)の変更
3. プロパティ値(原子値)の挿入、削除
4. プロパティ(参照関連)の挿入、削除
5. サブオブジェクト階層の挿入、削除
6. オブジェクトの挿入、削除

というレベルが有り、どのレベルの更新が多いかにより、最適なクラスタリングも異なる。原子値の変更や挿入、削除は、参照構造やサブオブジェクト階層には影響しないが、それ以外の操作はデータモデルのスキーマ自体が変化するため、配置木クラスタリングでは配置木そのものを作り直す必要が生じる。従って、配置木クラスタリングは、オブジェクトレベルでの更新を行うようなデータベースでは実用的でない。

Obaseのスキーマには図1のように参照構造とサブオブジェクト階層の二種類の枝が存在するため、この二つの相互関係を考える必要がある。

配置木クラスタリングを適用する場合、配置木をどのように決めるかが問題となる。検索操作だけを考えた場合、データベースに対する操作が既に与えられたメソッドを適用するだけであれば、メソッド実行に適した配置を選べばよい。

最も単純な方法としては、二つの関連を切り離して全く独立に配置木を決定し、両方の配置木を二次記憶上に格納する方法が考えられる。この場合、同じオブジェクトが両方の配置木に含まれる可能性があるため、重複するオブジェクトはoidのみを記述することにする。例として5.1節で述べた$O_2$のコスト最小アルゴリズムを用いて、次の6節に示したメソッドDoorColorに対する最適配置木を作ると、以下のようなクラスタリングとなる。

```
autofile ::=
  [ oid : #001,
    name : string,
    weight : int,
    length : int,
    height : int
  ]
  [ oid : #002,
    name : A1,
    weight : 3000,
    length : 350,
    height : 120,
    color : 'red',
     [ body.oid : #011,
       body.color : 'red',
       ...
     ]
  ]
  ...

personfile ::=
  [ oid : #501,
    age : int,
    own : obj
  ]
  ...

super/sub ::=
  [ oid : #001,
      [ oid : #002,
        oid : #003,
          [ oid : #005,
            oid : #006
          ]
        oid : #004
      ]
  ]
  [ oid : #501,
      [ oid : #502,
        oid : #503
      ]
  ]
```

配置木を切り分ける枝は参照、super/sub のどちらかに統一し、分割した枝情報(参照枝またはサブオブジェクト階層)を一括して記録する。

## 6 索引機構

内部に構造化されたデータを持つ複合オブジェクトに対する従来の索引として、入れ子索引、経路索引、マルチ索引の三つが提案されている [6, 3]。

議論を一般化するため、検索条件式を記述する経路式はメソッドで定義されており、索引は既に与えられたメソッドの処理を高速化ために構築するものとする。

Obase モデルの経路式の特徴は、継承付き演算と無名属性変数を含むことである。従って、この二つの特徴を持った経路式に対する索引の構成法を考えればよい。

それぞれの特徴に対する問題点を以下に示す。

1. 継承付き演算子を含む経路式の場合、目的のプロパティ値が経路上のどのオブジェクトにあるかが一意に決まらない。

2. 経路式の中に無名属性変数が挿入されている場合、そこに入る経路のパターンは固定されていない。

例として、auto データベース(図 1)を対象とし、継承付き演算と無名属性変数の両方を含んだ「ドアが赤色である自動車」を検索する質問を考える。質問文を以下に示す。

```
SELECT $x
FROM auto/ $x
WHERE $x.DoorColor = 'red'
```

メソッド DoorColor は、

```
update (auto/).DoorColor =
            (self(._.)*..door.color)
```

で定義される。

サブオブジェクト集合の索引は、前節で述べたクラスタリングの中にその情報が含まれているため、新しく索引を導入しなくても、例えばファイル追加型クラスタリングの場合、追加部分の super/sub ファイルを読み込むことでサブオブジェクト集合を求めることができる。

この質問文の場合、具体的には

- 多重継承が存在する場合、例えば今回の例ではある door オブジェクトを複数のオブジェクトが参照する場合、その参照経路によってメソッドの答が異なる可能性がある。

- メソッドの評価結果を更新する場合、どのオブジェクトのプロパティを書き換えればよいかを求める必要がある。

などの問題を生じる。

そこで、メソッド DoorColor の索引を作成する場合、継承付き演算と無名属性変数とを考慮すると以下のような特徴を満たす必要がある。

- 任意の引数$x に対し、door.color の値との対応を付ける。

- color の値がどのオブジェクトから継承されたかという情報が明記されている。

- 多重継承が存在するモデルでは、$x から door までの経路の情報が記録されていなければならない。

これらの要件を満たす索引として、以下の構造が考えられる。

1. メソッドの引数となるオブジェクトと、目的のプロパティ値を持つオブジェクトとの組で索引に記入する。

2. 経路索引を用いて、メソッドの検索式に対応する経路式全体を記述する。

1 を継承元併記型、2 を経路式型索引と呼ぶことにする。メソッド DoorColor に対して、これらの索引を適用した例を示す。

### a 継承元併記型

color がどのオブジェクトから継承されたかを明記するため、入れ子索引の要素を、継承元のオブジェクトと引数のオブジェクトとの組にして記録する。

また、auto/(_..)*door という経路で、経路中に color の値を持たないものも存在し、そのような経路式に対して color の値を挿入する更新操作や、逆に既に存在する color の値を消す更新操作も起こり得る。この場合、索引中の要素を挿入/削除する

のは手間がかかるため、索引に color を持たない項目$y = - を設ける。color を持たない経路式としては、経路式中の color プロパティに値が記入されていない場合と、経路式中のオブジェクトに color プロパティが定義されていない場合とがあるため、後者の場合継承元オブジェクトを⊥の記号で表す。これにより、更新で color プロパティが挿入/削除された場合、索引上は項目の移動で対応できる。

なお、この方法は経路情報を含まないため、多重継承が存在しないという条件の下でのみ使用できる。

索引 1:$x.DoorColor[$y]

| $y | ($x, $y の値を持つオブジェクト) |
|---|---|
| 'red' | (a1,b1),(a2,b2),(a3,d3) |
| 'green' | (a4,a4) |
| - | (a5,d5),(a6,⊥) |

なおオブジェクト a1 にある color='red' は、door オブジェクト d1 に継承されないため、返り値 (a1,a1) はこの索引には記入されていない。

また更新操作として、例えば a1 から d1 への経路の一部が削除された場合には、上の索引を修正して該当する項目を削除しなければならないが、この索引では経路情報を持たないため、更新された内容から修正すべき項目を特定することは困難である。このようにこの索引は、5.3節で述べた更新操作のレベルの内、1、3 には対応できるが、2、4、5、6 には対応できない。従ってこの索引は、原子値の変更という更新操作しか行われないデータベースにのみ適用できる。

**b 経路索引型**

この索引は、経路式の途中のオブジェクトも含めて記憶したものである。この場合、経路式中に継承元のオブジェクトも含まれるため、そこに印を付けることで更新時にどのオブジェクトにアクセスすればよいかも記述できる。記入する経路式の長さは、メソッド評価時に必要な最小限の長さとする。

また、経路式の color プロパティに値が記入されていない場合を$y = - 、経路式中に color プロパティが定義されていない場合を$y = ⊥として記入する。

索引 2:auto/(_..)*door.color[$y]

| $y | 経路式 |
|---|---|
| 'red' | a1.body[b1].door[d1]<br>a2.body[b2].door[d2]<br>a3.door[d3] |
| 'green' | a4.body[b4].door[d4] |
| - | a5.body[b5].door[d5] |
| ⊥ | a6.body[b6].door[d6] |

下線は color を持つオブジェクト。

なお、上の索引では door プロパティを持たない経路式は格納されないため、5.3 節の 4 のレベルに相当する更新、例えば color プロパティを持ち、door プロパティを持たないオブジェクト A があり、その A が door プロパティを持つオブジェクト B への参照構造が挿入された場合を考えると、A の color の値が B へ継承されるため、経路式 A.B は索引中に挿入されなければならない。しかし、この索引には A を含む経路式は記入されていないため、索引の更新が困難になる。

このように、上記の索引を単独で用いた場合、5.3節 4、5、6 のレベルの更新に対応できない。これに対応するためには、上の索引の経路式以外で color プロパティを含む経路式の索引を作り、この索引と組で記録しておく。

索引 2′:

| color | 経路式 |
|---|---|
| 'green' | a7.body[b7] |
| - | a8 |

索引 2′に含まれるオブジェクトから、索引 2 の項目⊥中の経路式にあるオブジェクトへの参照構造が挿入された場合、新たに door への color の継承が発生した可能性があるので、この索引中の二つの経路式から更新により生まれた経路式を生成し、door.color を持つならば索引 2 の該当項目に挿入する。関連の削除の場合は、逆の操作を行えばよい。

Obase モデルの場合、同じオブジェクトを指す経路式であっても、経路式が異なればオブジェクトの中身が異なることがある。このため、上記の索引中に格納された経路式は、分割して記述することができない。従って従来のマルチ索引のような索引は Obase モデルに適用できない。

## 7 クラスタリングと索引の組み合わせ

議論を一般化するため、索引の構築はあらかじめデータベースのオブジェクト中にメソッドとして定義されている検索式に対して行うものとし、それ以外の検索は、ファイルを直接読み込むことで行われるとする。

すべてのメソッドに対する索引が構築されていれば、論理クラスタリング法の選択は、まずどのような更新が行われるかどうかで決められる。参照関連やサブオブジェクト階層の挿入や削除が任意に行われるデータベースには配置木クラスタリングは不利であり、中間クラスタリング以上に分解した構造にしなければならない。

また質問に対してクラスタリングを最適化する場合、使用される質問文がオブジェクトのどのレベル (オブジェクト全体/単独のオブジェクト/ オブジェクトの一部) までを呼び出すか、あるいはメソッドを用いない経路式の質問文にどのような経路式が現れるかによってそのクラスタリングの方法が決められる。

例えば、6節に示した質問文だけを考えれば、複合オブジェクト全体を呼び出す必要があるため、auto のサブオブジェクトについて参照構造をまとめた配置木クラスタリングを用いた方が有利である。さらに質問文中の経路式に参照構造の航行演算 (組航行演算子) とサブオブジェクト階層の航行 (集合航行演算子) のどちらが多いかによって、最適な配置木が異なる。

また前節までに示したように、サブオブジェクト階層の挿入、削除など、データベースのスキーマ構造を変更する更新が起こるモデルでは、中間クラスタリングを用いる。中間クラスタリングは、長い経路式を用いた質問には不利であるので、経路索引を充実することでこれに対処しなければならない。

## 8 あとがき

本論文では Obase データモデルの特徴である無名属性変数、継承付き航行演算の二つを考え、これらに適したファイル配置および索引機構を述べた。

Obase モデルはスキーマ構造を変更できるため、スキーマ構造を直接展開するような論理クラスタリングは適当でなく、本文中に述べた中間クラスタリングのように、一旦オブジェクト単位に分解して再構築する方法が適していると考えられる。また索引も経路索引のように、経路式との対応を記述したものが適していると考えられる。

## 参考文献


[1] 吉川正俊, 田中克己, 上善恒雄, 田中康暁, 蛭井潤, 堀田光治郎. 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 *Proc. of Advanced Database System Symposium '93*, pages 63–72, Dec. 1993.

[2] Patrick Valduriez, Setrag Khoshafian, and George Copeland. "Implementation Techniques of Complex Objects". In *Proc. of the 12th Int. Conf. on Very Large Data Bases*, Aug. 1986.

[3] 加藤和彦. オブジェクト指向データベースシステムの記憶構造. 情報処理, 32(5), May 1991.

[4] François Bancilhon, Claude Delobel, and Paris Kanellakis, editors. *"Building an Object-Oriented Database System: The Story of O2"*. Morgan Kaufmann, 1992.

[5] Patrick Valduriez. "Join Indices". *ACM Trans. on Database Systems*, 12(2), June 1987.

[6] Elisa Bertino and Won Kim. "Indexing Techniques for Queries on Nested Objects". *IEEE Trans. Knowledge and Data Engineering*, 1(2), June 1989.