データベース・システム 82－7

（1991. 3. 11）

オブジェクト指向データベースシステムにおけるスキーマ変更支援

吉高 淳夫、 平川 正人、 田中 稔、 市川 忠男

広島大学工学部

オブジェクト指向データベースシステムにおけるスキーマ変更とそれに伴うデータベースの変更支援について述べる。データモデルとしては本研究室で提案しているオブジェクト指向データモデルＭＯＲＥを前提としている。データベーススキーマはクラス間のスーパ／サブ関係を表現するクラス階層とオブジェクトの集約を表現するアグリゲーション階層から成り立っている。クラス／アグリゲーション階層に対する変更を分類し、それらの変更がスキーマの記述やメソッド内のメッセージ送出に関する記述に与える影響について考察した。また、画面上でスキーマが視覚的に変更できるようなユーザインタフェースを提案している。

Schema Modification Support in an Object-Oriented Database System

A.Yoshitaka, M.Hirakawa, M.Tanaka, and T.Ichikawa
Information Systems, Faculty of Engineering, Hiroshima University
Kagamiyama-cho, Higashi-Hiroshima 724, Japan

Schema modification and corresponding database modification support in an object-oriented database system are described. Here we describe them based on the MORE object-oriented data model which has been proposed in our laboratory. Database schema consists of two hierarchies: the one is a class hierarchy which represents the relationships among classes, and another is an aggregation hierarchy which represents the structure of objects. This paper classifies the modification for the above-mentioned hierarchies and describes the modification effect on schema description including method specification. A user interface which can modify a schema on a screen in terms of a graph representing hierarchies is also proposed.

1．はじめに

関係データベースシステムに代って最近ではオブジェクト指向データベースシステムが注目されている。オブジェクト指向データベースシステムでは現実世界に存在する物をオブジェクトとして表現し管理しており、複雑な構造を持つオブジェクトを柔軟に表現できる反面、スキーマの定義が複雑になってしまうといった面がある。

データベースを構築し、それを運用していく場合、データの保守が一つのテーマとなる。データベースシステムの運用中に対象領域の変化が生じた場合、既存のスキーマを変更して新しい要求に対応できるようにする必要がある。スキーマを変更すると、それまでに作成されたデータベースはそのままでは使えなくなってしまう。そこで旧スキーマを新スキーマに変更するのにともなって、既存のオブジェクトデータベースを新しいスキーマの記述に従って再構成することが必要となる。

現在までにORION[1],GemStone[2],Iris[3],O_2[4]などのオブジェクト指向データベースシステムが提案されている。これらの中でスキーマの変更について詳しい考察がおこなわれているものは、ORION[6]と、GemStone[5]である。他のシステムではスキーマ変更については考察していなかったり、非常に単純な変更のパターンにしか対応していない。スキーマの概念については各々のシステムで考えられているデータモデルに違いがあるため、スキーマ変更の考え方も、自ずと各々のシステム／データモデルに依存したものになっている。

筆者らの研究室ではオブジェクト指向データモデルMORE[7]に基づくデータベースシステム[8]を開発している。本研究では、スキーマ変更及びそれに伴うデータベースの変更の支援について提案する。また実験システムについても述べる。ここで提案するシステムでは、クラス間のスーパ・サブ関係を表わすクラス階層、オブジェクトの集約の階層を表わすアグリゲーション階層をグラフィカルに表示し、これらのグラフをユーザが画面上で操作することによってデータベーススキーマの変更を視覚的に行なうことができる。クラスあるいはアグリゲーション階層を変更した場合、既に定義されているメソッドも影響を受け、記述の一部を書き換える必要があるが、提案するシステムでは、クラス／アグリゲーション階層を変更したことによって引き起こされるメソッドの記述への影響をシステム側で自動的に修正する。

以後、2章でオブジェクト指向データモデルＭＯＲＥについて簡単に説明し、3章ではクラス階層の変更とその影響、4章でアグリゲーション階層の変更とその影響について考える。5章でメソッドの変更について考察した後、6章でその他の変更についてまとめ、7章でインプリメンテーションについて述べ、8章で本研究によって得られる成果と今後の課題について述べる。

2．オブジェクト指向データモデルＭＯＲＥ

本章では、本研究室で既に提案しているオブジェクト指向データモデルＭＯＲＥについて簡単に述べる。

2．1 ＭＯＲＥの概要

MOREでは、一般のオブジェクト指向データモデルと同様に、既に存在しているオブジェクトの集約操作によって新しいオブジェクトを定義することができる。オブジェクトの集約操作によって定義されるオブジェクトを複合オブジェクトと呼び、具体的にデータベース内に値が定義されているオブジェクトをプリミティブオブジェクトと呼ぶ。MOREでは複合オブジェクトの表現はアグリゲーション階層で行なっている。

またMOREにおいては、メッセージによるスキーマの動的決定機構、ならびにクラス定義を容易化するためのクラスタイプを新たに導入している。特に前者の機構によって、固定されたスキーマにしたがってオブジェクトを操作するのではなく、対象とするスキーマを動的に決定した上でオブジェクトの操作を行なうことができる。すなわち、MOREではオブジェクトが複数のビューをもつことを許しており、これにより、個々のユーザの要求に沿った視点からデータベースを操作することができる。

スキーマのグラフィカルな表現を図1に示す。ここで、オブジェクトの集約関係はオブジェクト間のリンクで表現されている。オブジェクトがもつ共通の性質はクラスにおいて管理される。データベース内の全てのクラスの最上位のクラスとして、クラス'Universe'が存在するものとしている。ここでは、'Universe'クラスはデータベースを操作する場面ではユーザにはみえないものとして考えているので、'Universe'クラスの直接のサブクラス（図1におけるElectric_Appliance,VCR_Spec,Dimension,Real等のクラス）をここではクラス階層におけるrootのクラスと考える。クラス間のスーパ・サブ関係を表わしたものがクラス階層である。MOREにおいてクラスは共通な性質をもつオブジェクトの集合と考えており、その性質はクラス記述に持たされる。データベーススキーマはアグリゲーション階層とクラス階層によって記述される。

2．2 クラス記述

クラス記述の例を図2に示す。クラス記述は以下に示す項目からなる。

CLASS

NAME:定義するクラスの名前

INTERCLASS-CONNENCTION

既に定義されているクラスと、定義しようとしているクラスとの関係を記述する。受理可能な関係には、'SUPERSET-OF','SUBSET-OF','INTERSECTION-OF','UNION-OF'がある。

CONSTRAINT

クラスに属するオブジェクトの満たすべき制約を記述する。

STRUCTURAL-CONSTRAINTS

クラスのもつアグリゲーション階層を記述する。reference-name, domain, orderの3つ組で記述する。

reference-name:アグリゲーション階層を構成するオブジェクトを参照するための名前

domain　　:上記のオブジェクトが存在すべきクラス

order　　:オブジェクトとしてとり得る個数に関する制約で、1あるいはNをとる

OBJECT-CONSTRAINT

クラスに属するオブジェクトの満たすべき制約条件を、アグリゲーション階層内のオブジェクトの値によって記述する。

METHOD

クラスに属するオブジェクトに対する操作を定義する。メソッドとしてはdaemon, structural, operationalの3種類のメソッドがある。

daemon method:オブジェクトが更新された時に起動されるメソッド

structural method:スキーマを動的に変更するためのメソッド

operational method:クラスあるいはオブジェクトに対する操作を行うもの

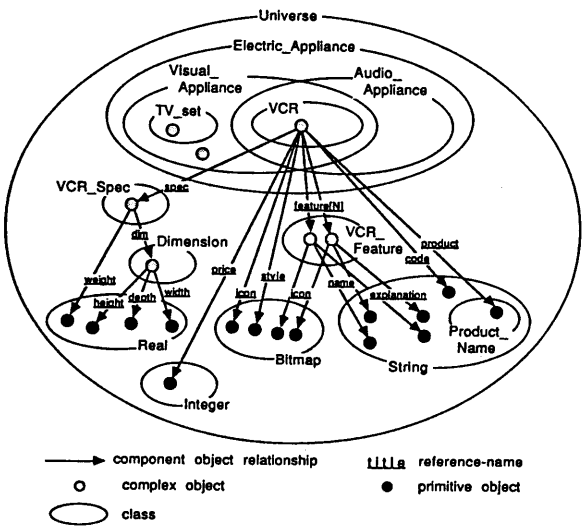

図1　スキーマのグラフィカル表現

```
CLASS
        NAME:           Electric_Appliance
        DESCRIPTION:    Catalog for Electric products.
INTERCLASS-CONNECTION:
                        SUBSET_of Universe;
STRUCTURAL-CONSTRAINT:
                        code,           String,         1+;
                        product,        Product_Name,   1;
                        price,          Integer,        1;
METHOD
                        IF Self IS_MODIFIED             #display Self.
                        %price:                         name %; price %.
                        ....
```

(a) class Electric_Appliance

```
CLASS
        NAME:           TV_set
        DESCRIPTION:    Catalog for TVs.
INTERCLASS-CONNECTION:
                        SUBSET_of Visual_Appliance;
STRUCTURAL-CONSTRAINT:
                        spec,           TV_Spec,        1;
                        style,          Bitmap,         1;
                        feature,        TV_Feature,     N;
METHOD
                        %spec:          name %; spec %spec;
                                        feature %name.
```

(b) class TV_set

図2　クラス記述の例

3. クラス階層の変更

データベーススキーマはクラス階層とアグリゲーション階層から成り立つ。ここでは、クラス階層を変更した場合にどの様な影響があるかをクラス記述（以下では単に記述という）のレベルで考える。

3.1 クラス階層の変更

ここでのクラス階層の変更は、検索対象となるオブジェクトの属するクラスの階層（すなわち、図1におけるElectric_Applianceクラスをrootとするクラス階層）を考える。あるクラスAに対して行なうことのできる変更操作として、次に挙げる5つとその組合せを考える。以下に操作の種類とその意味について述べる。

(3.1.1)クラスの追加

(3.1.1a)スーパクラスの追加

クラスAにスーパクラスBを追加する。この場合、Aは既に定義されている（検索対象となるオブジェクトを管理するクラス階層上の）任意のクラスであり、Bは新たに生成され、Aのスーパクラスとして定義されることを意味する。

(3.1.1b)サブクラスの追加

クラスAにサブクラスBを追加する。この場合、Aは既に定義されている（検索対象となるクラス階層上の）任意のクラスであり、Bは新たに生成され、Aのサブクラスとして定義されることを意味する。

(3.1.2)クラスの削除

これは（検索対象となるクラス階層上の）任意

のクラスを消去する操作である。サブクラスをもたないクラスが消去された場合は他のクラスには影響はない。サブクラスを持つクラスAが消去された場合は、Aのサブクラスは Aのスーパクラスのサブクラスとして定義されるものと考える。なおクラス階層のrootにあたるクラスは消去できないものとする。

(3.1.3)クラスの分割

この操作はクラス階層上で葉にあたるクラスに対してのみ行えるものとする。この場合、クラスAはクラスBとCに分割される。B,Cの記述は基本的にAの記述をコピーする事で生成される。Aに属していたオブジェクトはユーザにより与えられたインスタンス変数に関する条件を満たすかどうかでBとCに振分けられる。

(3.1.4)2つのクラスの結合

この操作は、共通の親をもつ2つのクラスを結合する操作と考える。

(3.1.5)スーパ／サブ関係の追加／削除

既に定義されているクラスの間にスーパ・サブ関係を新しく定義したり、既に定義されているスーパ・サブ関係を削除したりする。ここでは、クラスAに対して、既に定義されているクラスを新たにスーパクラスとする操作、あるいはAのスーパクラスとして定義されているクラスとのスーパ・サブ関係を取消す操作と考える。

(3.1.5a)スーパ・サブ関係の追加

追加操作を行った結果、クラス階層がある制約を満たさなくなるような場合は、スーパ・サブ関係は追加できないものとする。この制約については後述する。

(3.1.5b)スーパ・サブ関係の削除

クラスAのスーパクラスが一つしか存在しない場合は削除操作は行えないものとする。

3.2 クラス階層の変更とその正当性

ここでは、3.1で分類した変更が行なわれた場合も、変更されたクラス階層を正当なものにすることについて述べる。ここで、クラス階層の正当性を次の様に定義する。

<クラス階層の正当性>

クラスをノード、クラス間のスーパ・サブ関係をリンクととらえる。また、リンクの向きをスーパクラスからサブクラスへ向かうものとする。ここで、以下の条件を満たす場合、そのクラス階層は構造的に正しいとする。

1．ノードaからbへのpathが存在するとき、

①aからbへ（直接）接続されているリンクが1本存在する。(3.2.1)

②aから（b以外の）1つ以上のノードを経由してbに至るpathがn本（n≧1）存在する。(3.2.2)

①,②のどちらか一方が全てのノードの組（a, b）について成り立つ

2．任意のノードaについて、aから出発しaに到達するようなpathは存在しない。(3.2.3)

上記の様にクラス階層の正当性を定義した。次に3.1で分類した各々の変更を施した場合とその影響についてのべる。(3.1.1a),(3.1.1b),(3.1.3),(3.1.5b)の変更を施した場合、クラス階層の正当性が保たれることは明白である。そこで以下ではその他の変更操作を施した場合のクラス階層に対する影響と正当性を保ための制約について述べる。

(3.1.2)クラスの削除

サブクラスを持つクラスAが消去された場合、Aのサブクラスは Aのスーパクラスのサブクラスとして定義されるので、クラス削除操作を施すと、Aのスーパクラスと Aのサブクラスとの間に(3.2.1)と(3.2.2)の両方のpathができることが有り得る。この場合、(3.2.2)のパスが存在することから、Aのスーパクラスで定義されているインスタンス変数、メソッドはAのサブクラスに継承されることになる。従って(3.2.1)のpath（リンク）は存在する意味がない。よって、クラスの削除操作を行なったことにより(3.2.1),(3.2.2)のpathが両方存在することになる場合は、(3.2.2)のpathは消去する。

(3.1.4)2つのクラスの結合

結合操作を行なった場合、結合された2つのクラスとそれらの（共通の）スーパクラスとを結ぶリンクは1本とする。また、結合後のクラスからそのサブクラスへのpathについては、(3.2.1),(3.2.2)の両方のpathが存在することが有り得る。この場合も(3.1.2)のときと同様に、結合操作を行なったことにより(3.2.1),(3.2.2)の両方のpathが存在することになる場合は、(3.2.2)のpathは消去する。

(3.1.5a)スーパ／サブ関係の追加

スーパ／サブ関係を追加したことにより2つのクラスの間に(3.2.1)と(3.2.2)の両方のpathが存在することになる場合は、(3.2.2)のpathは消去する。

3.3 クラス階層の変更に伴う影響

ここでは、クラス階層を変更した場合の影響をクラス記述のレベルで考える。また、クラス階層の変更の影響をうけるクラスに属するオブジェクトの振舞いについても述べる。クラス記述の中で定義されているメソッドの振舞いについては5章で考察する。

(3.1.1)クラスの追加

(3.1.1a)スーパクラスの追加

クラスAにスーパクラスBを追加する場合、Aの記述の'INTERCLASS-CONNECTION'の項に'SUBSET-O

F B'が追加され、新たに生成されたクラスBの記述には'SUPERSET-OF A'が追加される。

(3.1.1b)サブクラスの追加

クラスAにサブクラスBを追加する場合、Aの記述の'INTERCLASS-CONNECTION'の項に'SUPERSET-OF B'が追加され、新たに生成されたBの記述の'INTERCLASS-CONNECTION'の項に'SUBSET-OF A'が追加される。

(3.1.2)クラスの削除

消去されるクラスAのスーパクラスBの記述の'INTERCLASS-CONNECTION'の'SUPERSET-OF A'の定義が消去される。Aがサブクラス（Cとする）をもつ場合には、Aのスーパクラスには'SUPERSET-OF C'が追加され、'SUBSET-OF B'がクラスCに追加される。消去されるクラスAの'STRUCTURAL-CONSTRAINT'の定義はクラスCに移される。Aで定義されていたオブジェクトは消去される。

(3.1.3)クラス階層上でleafにあたるクラスの分割

クラスAをBとCに分割する場合、Aの記述をコピーする事により、BとCの記述を生成する。名前の重複を避けるために、各々の名前の後にsuffixが付け加えられる。

(3.1.4)２つのクラスの結合

クラスAとBを結合してCとする場合、AとBにおける'INTERCLASS-CONNECTION','STRUCTURAL-CONSTRAINT'の定義がマージされ、新たにCの記述として定義される。結合前にA，Bで定義されていたオブジェクトは結合後にはCのオブジェクトとなる。'OBJECT-CONSTRAINT'やクラス名はAまたはBで定義されていたものを引き継ぐか、あるいは変更時に新たに定義される。

(3.1.5)スーパ・サブ関係の追加／削除

(3.1.5a)スーパ・サブ関係の追加

AをBのスーパクラスとすると、'INTERCLASS-CONNECTION'の項には、Aに'SUPERSET-OF B'、Bには'SUBSET-OF A'が追加される。

(3.1.5b)スーパ・サブ関係の削除

AをBのスーパクラスとするようなスーパ／サブ関係を削除しようとすると、'INTERCLASS-CONNECTION'の項の、Aにおける'SUPERSET-OF B'、Bにおける'SUBSET-OF A'が削除される。また、Bに属するオブジェクトの、Aからの継承により定義されていたインスタンス変数の値は消去される。

4．アグリゲーション階層の変更

本章ではアグリゲーション階層の変更について分類し、変更をした場合の影響について考察する。

4.1 アグリゲーション階層の変更

以下に操作の種類とその影響について示す。

(4.1.1)オブジェクトの追加・定義

(4.1.1a)primitive objectの追加

既に定義されている複合オブジェクトの下にprimitive object aを追加する。aの親に相当するcomplex objectは１つしか指定できないとする。aの属すべきクラスが定義されていない場合は新しいクラスが定義される。

(4.1.1b)complex objectの定義

共通の親bをもつ複数のオブジェクト$c_1,\ldots,c_n$の集約として新しいオブジェクトaを定義することを考える。このとき、$c_1,\ldots,c_n$の親はa、aの親はbとなる。(4.1.1a)と同様に、aの属すべきクラスが定義されていない場合は新しいクラスが定義される。

(4.1.2)オブジェクトの削除・消去

(4.1.2a)primitive objectの削除

複合オブジェクトの子として定義されているprimitive object aを削除する。この場合、aの属していたクラスの記述は消去されない。

(4.1.2b)complex objectの消去（アグリゲーション階層におけるroot以外のobject）

complex object aの親をb、aの子をcとすると、aの消去にともなってcはbの子として新たに定義されるものとする。

(4.1.3)complex objectの分割（アグリゲーション階層におけるroot以外のオブジェクト）

complex object aが$b_1,\ldots,b_{m-1},b_m,\ldots,b_n$のn個のオブジェクトの集約として定義されている時に、aをa'とa''の２つに分け、a'を$b_1,\ldots,b_{m-1}$の集約として、a''を$b_m,\ldots,b_n$の集約として定義するような操作である。このときa''の属するクラスとして適当なものが存在しない場合新たに生成される。

(4.1.4)complex objectの結合（但し両オブジェクトの親は共通）

共通の親bをもつcomplex object a'，a''を１つのcomplex object aとする操作である。この場合、a'が$b_1,\ldots,b_{m-1}$、a''が$b_m,\ldots,b_n$の集約として定義されていたとすると、aは$b_1,\ldots,b_n$の集約として定義される。

4.2 アグリゲーション階層の変更とその正当性

クラス階層についてと同様に、アグリゲーション階層の正当性を規定し、それを満たすように変更操作は行なわれる。アグリゲーション階層の正当性を次の様に定義する。

〈アグリゲーション階層の正当性〉

以下の２つの条件を満たす場合、そのアグリゲーション階層は構造的に正しいとする。

1．　1つ以下のprimitive/complex objectの集約として定義されるcomplexobjectが存在しない。(4.2.1)

2．　各々のprimitive/complex objectはただ1つの親をもつ(4.2.2)

4.1で分類した変更操作を施した場合、(4.1.2a)以外の操作では(4.2.1),(4.2.2)の条件を満たすことは明らかである。(4.1.2a)の操作を行った場合は、(4.2.1)の条件を満たさなくなることがある。そこで、(4.1.2a)の操作をしたことによって(4.2.1)を満たさなくなる場合は、システム側で自動的にアグリゲーション階層を修正し、(4.2.1),(4.2.2)を満たすようなものにする。この自動修正については後述する。

4.3 アグリゲーション階層の変更がクラス記述に及ぼす影響

アグリゲーション階層の変更は以下に挙げる6通りの操作及びその組合せで行われる。なお、アグリゲーション階層の変更は、その階層を定義したクラスに対してのみ行なえるものとする。なお、アグリゲーション階層の変更によるメソッドへの影響については5章で考察する。

(4.1.1)オブジェクトの追加・定義

(4.1.1a)primitive objectの追加

追加しようとするprimitive object aの親にあたるオブジェクトの属するクラスの記述の'STRUCTURAL-CONSTRAINT'の項にaの定義を追加する。domainの指定で定義されていないクラスが記述された場合は、そのクラスを生成する。

(4.1.1b)complex objectの定義

共通の親bをもつ複数のオブジェクト$c_1,\ldots,c_n$の集約として新しいオブジェクトaを定義する。このとき$c_1,\ldots,c_n$の親はa、aの親はbとなる。このとき、bの属するクラスの記述のうち、'STRUCTURAL-CONSTRAINT'の$c_1,\ldots,c_n$の記述がaの記述にコピーされ、そしてbにおける$c_1,\ldots,c_n$の記述はaの定義に書き換えられる。

(4.1.2)オブジェクトの削除・消去

(4.1.2a)primitive objectの削除

削除しようとするprimitive object aの親にあたるオブジェクトの属するクラスの記述で'STRUCTURAL-CONSTRAINT'の項におけるaの属するクラスに関する記述を削除する。

(4.1.2b)complex objectの消去（root以外のオブジェクト）

complex object aの属するクラスに定義されていた'STRUCTURAL-CONSTRAINT'の記述がaの親の属するクラスの記述とされ、aの定義が削除される。

(4.1.3)complex objectの分割（root以外のオブジェクト）

complex object aが$b_1,\ldots,b_{m-1},b_m,\ldots,b_n$のn個のオブジェクトの集約として定義されているとすると、aをa'とa''の2つに分け、a'を$b_1,\ldots,b_{m-1}$の集約として、a''を$b_m,\ldots,b_n$の集約として定義するような操作である。このときa',a''の属するクラスが新たに生成され、aの親の属するクラスの'STRUCTURAL-CONSTRAINT'の記述にa',a''の属するクラスに関する記述が追加されaの属するクラスに関する記述に置き換えられる。aのクラスの'STRUCTURAL-CONSTRAINT'の記述の$b_1,\ldots,b_{m-1}$の定義がa'の定義としてaからコピーされ、そして$b_m,\ldots,b_n$の定義がa''のクラスのものとして定義される。

(4.1.4)complex objectの結合（但し結合される2つのオブジェクトの親は共通）

complex object a'の記述とa''の記述がマージされaの記述となる。a',a''の親の属するクラスのa',a''に関する'STRUCTURAL-CONSTRAINT'の記述は削除され、aの記述に置き換えられる。

5．メソッドの変更

5.1 メソッドに関する変更支援

メソッドに関する変更に関しては、新しいメソッドの定義、既に定義されているメソッドの削除がまず考えられる。その他に、クラス階層／アグリゲーション階層の変更の影響によってメソッドの記述の一部を変更する場合がありうる。

MOREにおけるメソッドの記述は基本的に、

<message>:<action>

から成り立つ。<message>はメソッドを起動する際に受け取るべきメッセージ名であり、<action>はそのメッセージを受け取った時に実行される制御文の集まりである。MOREでは、メソッドの<action>部の記述において他のオブジェクトにメッセージを送る場合は、メッセージの送り先はreference-nameを使って記述している。メソッド中のreference-nameの記述の変更についてはクラス／アグリゲーション階層の変更を参照して、変更の影響により必要となる修正をシステム側で自動的に行うことも考える。本研究におけるメソッドに関する変更を分類すると以下のようになる。

1．メソッドの定義
2．メソッドの削除
3．クラス階層の変更に伴うメソッド記述の変更
4．アグリゲーション階層の変更に伴うメソッド記述の変更

1，2はユーザが陽に指定する操作であり、具体的にはメソッド名の指定をすることで行なう。3，4はクラス/アグリゲーション階層を変更するとシステム側で自動的

に行われ、その結果をユーザは見ることができる。

5．2 クラス階層の変更に伴うメソッドの変更

ここでは、クラス階層を変更した場合のメソッドの振舞いについて考える。クラス階層に関する変更のうち、(3.1.2),(3.1.3),(3.1.4)の変更が行なわれた場合、関係するクラスに定義されているメソッドが影響を受ける。以下に、それぞれの変更を行なった場合の影響を簡単にまとめる。

(3.1.2)の変更をおこなった場合

消去されるクラスに定義されていた記述はそのサブクラスに移される。

(3.1.3)の変更をおこなった場合

A'のクラス記述の'METHOD'の項にはAの記述がコピーされ、suffixが各名前に付けられる。

(3.1.4)の変更をおこなった場合

2つのクラスで定義されているメソッドはどちらもAに属するメソッドとなる。

5．3 アグリゲーション階層の変更に伴うメソッド記述の変更

オブジェクトは複数のオブジェクトの集約として定義されている。メソッドの実行時には上位のオブジェクト（被集約オブジェクト）から下位のオブジェクト（上位のオブジェクトの構成要素）へメッセージが送られることがある。アグリゲーション階層を変更した場合、その構造の変化に対応してメッセージ送出の記述の部分を書き換えてやる必要がある。そのような場合に対する変更をシステム側で支援することを考える。

メソッド内のメッセージ送出に関する記述の変更支援が必要になるような変更操作を挙げると、(4.1.1b),(4.1.2a),(4.1.2b),(4.1.3),(4.1.4)がある。それぞれの操作を行った場合の影響を以下に示す。a，bはそれぞれ4．1での説明に対応している。

(4.1.1b)の変更をおこなった場合

・定義されるオブジェクト(a)に関する変更

temporaryなメッセージ名（bにおいて定義されたもの）に対する動作として、bで定義されていた$c_1,\ldots,c_n$に対するactionを定義する。（bでの定義からコピーする）

・定義されるオブジェクトの親(b)に関する変更

$c_1,\ldots,c_n$に関する記述を削除してaへコピーして、それをbからaへのactinの記述（送出するメッセージ名の記述）と置き換える。

(4.1.2a)の変更をおこなった場合

・削除されるオブジェクトaの親(b)に関する変更

bにおける定義でaに関する記述を削除する。

(4.1.2b)の変更をおこなった場合

・削除されるオブジェクトaの親(b)に関する変更

aにおける$c_1,\ldots,c_n$に関する記述をbへ移し、aへのactionの記述と置き換える。

(4.1.3)の変更をおこなった場合

・aの分割により生成されたオブジェクト(a')に関する変更

$b_1,\ldots,b_{m-1}$に関する記述をaで定義されていた記述からコピーし、actionの起動用のメッセージ名を生成する。

・aの分割により生成されたオブジェクト(a'')に関する変更

$b_m,\ldots,b_n$に関する記述をaで定義されていた記述からコピーし、actionの起動用のメッセージ名を生成する。

・分割されたオブジェクトの親(b)に関する変更

a'，a''に対するactionの記述の追加をし、aに対するactionの記述を削除する。

(4.1.4)の変更をおこなった場合

・結合されて生成されるオブジェクト(a)に関する変更

a'における$b_1,\ldots,b_{m-1}$の記述とa''における$b_m,\ldots,b_n$に関するactionの記述と結合する。

・結合されるオブジェクトの親(b)に関する変更

a'，a''に対するactionの記述を削除し、aに対するactionの記述を生成する。

(3.1.2),(3.1.3),(3.1.4)の変更が行われた場合、そのアグリゲーション階層が定義されているクラスの他に、そのサブクラスで定義されているメソッドについても同様に変更操作を施す必要がある。

6．その他の変更

・'STRUCTURAL-CONSTRAINT'における'order'の変更

1からNへの変更は許すが、その逆は許さないものとする。

・名前の変更

名前の変更としてはクラス名、reference名、メッセージ名、メソッド名の変更が考えられる。これらの名前の変更が影響する範囲を図3に示す。

7．インプリメンテーション

7．1 ユーザインタフェース

データベーススキーマの変更にあたって具体的には、画面上にグラフィカルに表示されたクラス階層／アグリゲーション階層に対して、マウスを用いて画面上でその構造を変更していくことでおこなう。ユーザは画面上で階層の変更の状況を確認しながらスキーマを変更することができる。

操作画面を図4に示す。操作画面は大きく分けてクラス階層ウィンドウ(CHW)とアグリゲーション階層ウィンドウ(AHW)から構成される。メインウィンドウのメニューには、クラス／アグリゲーション階層のチェック、データベースの書き換えなどの項目がある。

Effect / Modification	CLASS NAME:	INTERCLASS-CONNECTION	STRUCTURAL -CONSTRAINT	METHOD
Class name	X	X	X	-
Reference name	-	-	X	X
Message	-	-	-	X
Method name	-	-	-	X

X:Modification required
- :No effect

図３　名前の変更と影響のおよぶ範囲

クラス階層を変更する場合はまずCHWを選択する。次に、変更したいクラスを表わしている楕円上でマウスをクリックすることで、そのクラスに対して施すことのできる操作を表わすメニューが図４のように表示される。変更を行いたい操作を選択すると、それに応じて画面上に表示されているクラス階層のグラフが変更され、クラス名や参照名などの入力が必要な場合は入力を促すダイアログボックスが画面上に現れる。

アグリゲーション階層を変更する場合は、はじめにAHWを選択する。そのクラスで定義されているアグリゲーション階層を表わすリンクは実線で表示され、スーパクラスから継承されているものは点線で示される。変更操作は実線で表わされている部分についてのみ可能である。変更を行なう場合は、クラス階層の変更の場合と同様に、変更を施したいオブジェクトを表わしている円の上にマウスカーソルを移動し選択すれば、そのオブジェクトに対して施すことのできる操作の一覧が現れる（図５）。希望の操作を選択すると、それに応じてアグリゲーション階層を表わすグラフが変化する。

７．２　クラス階層／アグリゲーション階層のチェック

ここでは、(3.1.1)-(3.1.5),(4.1.1)-(4.1.4)の各々の操作及び６章で述べた変更をスキーマ変更に対するプリミティブな操作とする。既存のスキーマに対する変更はプリミティブな操作の組合せにより行われるものとする。データベースの変換を行う前に、変更されたスキーマが構造的に正しいものかどうかをチェックする必要がある。システムは以下の点についてチェックを行うことができる。

(1)クラス階層（変更時にチェック）

・(3.1.2)の変更を施した結果、消去されたクラスのスーパクラスとサブクラスとの間に(3.2.1)と(3.2.2)のリンクが共に存在する場合は、(3.2.1)のリンクを消去する。

・(3.1.4)の変更においては、共通の（直接な）スーパクラスをもつ２つのクラス同志の場合しか許さない。また、２つのクラスの結合の結果、(3.2.1)と(3.2.2)のリンクが共に存在するようなクラスの組が存在する場合は、(3.2.1)のリンクを消去する。

・(3.1.5a)の変更において、関係を追加された２つのクラスの間に(3.2.1),(3.2.2)の２つのリンクができる

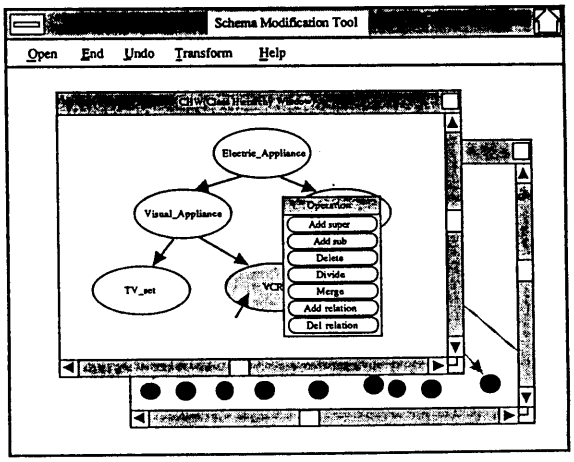

図４　クラス階層の変更画面

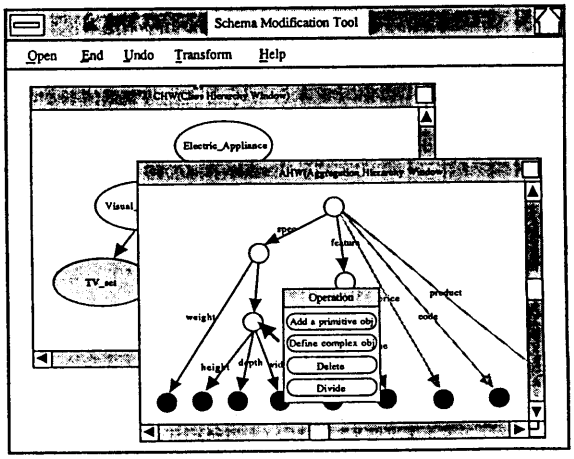

図５　アグリゲーション階層の変更画面

場合は、その操作は無効とする。

・(3.1.5b)の変更において、サブクラスのもつスーパクラスへのリンクが１つしかない場合は、この変更はできない。

(2)アグリゲーション階層（変更終了時にチェック）

・(3.2.1)-(3.2.4)の変更操作を行った結果、１つのオブジェクトの集約により定義されている複合オブジェクトができた場合、そのオブジェクトを消去する。子をもたない複合オブジェクトがある場合、それも消去する。

(3)名前の変更（変更時にチェック）

名前を変更した場合、既に定義されているものと重複する場合は再入力を促す。

7.3 データベース変換

MOREに基づくデータベースシステムの構成を図６に示す。オブジェクトデータベースはスキーマデータベースとストレージオブジェクトデータベースから成り立つ。これらのデータベースへのアクセスの際にプラットホームの役割をするのがオブジェクトマネージャである。オブジェクトスペースはユーザのデータベース操作環境であり、本研究で提案するシステムはそのうちの１つである。

スキーマ変更ツールはDisplay Manager(DM), Interface Manager(IM), Transaction Manager(TM)及びWorking Schema(WS), Original Schema(OS), Modification Descriptor(MD)から成り立つ。DMはクラス／アグリゲーション階層の表示などに関する処理を行ない、IMはユーザからの入力や変更時のメニューの管理をしている。TMはオブジェクトマネージャ（以下OMと略す）と通信してスキーマデータベースからスキーマに関する記述を引き出したり、変更時／後の階層チェックをおこなう。

変更するスキーマを指定すると、TMはOMに要求してスキーマデータベースからスキーマの記述を取り出してOSおよびWSに書込む。ユーザによる変更操作があると、TMは変更の内容をMDに書込み、そしてWSにあるスキーマの記述（クラス記述）を変更する。スキーマに対する変更操作が終了すると、スキーマ情報を保持しているスキーマデータベースを書き換え、そしてオブジェクトの情報を保持しているオブジェクトデータベースを書き換える。

7.3.1 スキーマデータベース変換

CHW, AHWに表示されている階層はクラス記述の形でWSに保持されており、それを内部表現になおした後、スキーマデータベースに蓄えられる。

7.3.2 オブジェクトデータベース変換

変更されたスキーマにそってオブジェクトデータベースを再構成する際に問題となるのが、旧スキーマに基づいて定義されたオブジェクトをどのように新しいスキーマ上に反映するかということである。TMがOMを通じて得たスキーマの記述をOS, WSに書込む時に、primitive objectの属するクラスへのreference-nameにはユニークなラベルを付けておく。プリミティブな操作一つ一つが行われるごとにWSのスキーマ記述は変更されるが、ラベル内容の変更はおこなわれない。スキーマ変更に関するプリミティブな操作が繰り返された後、OSとWSのラベルのマッチングをとることによって、プリミティブオブジェクトを対応づけることができ、その対応を参照してTMはWSに記述されている新しいスキーマに沿ったオブジェクトを生成し、OMを通じてデータベースに蓄える。

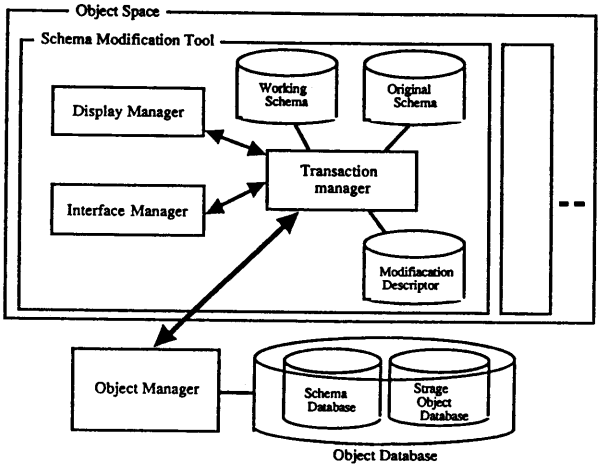

図６　システム構成

8. おわりに

オブジェクト指向データモデルMOREに基づくデータベースシステムにおいて、スキーマ変更に伴うデータベースの変更の支援と実現の手法について述べた。

各々のクラスに注目し、そこでの定義を修正するという観点からスキーマの変更を考えるのではなく、クラス間の相互関係や複合オブジェクトを構成しているオブジェクトの階層構造に対して変更操作を施すことを提案している。クラス階層やアグリゲーション階層のグラフィカル表示とあわせて、ユーザとの親和性に優れたスキーマ変更がおこなえる。さらに、スキーマの変更によって引き起こされるメソッド記述の変更をシステム側でサポートすることでユーザの負担を軽減している。

他のシステムとの比較をした場合、GemStoneでは、考えられているデータモデルが比較的簡単なものであるので、スキーマの変更に関する支援は比較的簡単に実現で

きる。しかしながら、クラス階層／アグリゲーション階層を考えた場合、変更の能力が低いといえる。ORIONでは多岐に渡る変更が考慮されているが、それは、valueの型やリンクの型の種類が多いことや名前の重複が起こった場合のoverrideを考慮していることに起因している。両者とも、個々のクラスに注目し、それに関する変更という観点からスキーマの変更を考えて操作の分類をおこなっているので、オブジェクトの構造が複雑になった場合、ユーザにとってスキーマ変更が考えにくいものとなってしまう。

本研究では、スキーマの変更とクラスタイプとの関連については述べていない。しかしながらスキーマ定義／変更に係わるユーザの負担を更に軽減していくために考察が必要である。また、スキーマのバージョン管理などについての考察が今後の課題として挙げられる。

本システムはパーソナルコンピュータ上でMS-Windows3を用いて構築中である。

参考文献

[1]W. Kim, N. Ballou, H. T. Chou, J. F. Garza, D. Woelk, "Features of the ORION Object-Oriented Database System," in Object-Oriented Concepts, Databases, and Applications, ACM press, pp251-282, 1989.

[2]R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, M. Williams, "The GemStone Data Management System," in Object-Oriented Concepts, Databases, and Applications, ACM press, pp283-308, 1989.

[3]K. Wilkinson, P. Lyngbaek, W. Hasan, "The Iris Architecture and Implementation, "Transantions on Knowledge and Data Engineering, Vol. 2, Num. 1, pp63-75, Mar. 1990.

[4]O. Deux et al., "The Story of O_2, "Transantions on Knowledge and Data Engineering, Vol. 2, Num. 1, pp91-108, Mar. 1990.

[5]D. J. Penny and J. Stein, "Class Modification in the GemStone Object-Oriented DBMS, "Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, September 1987.

[6]J. Banerjee, W. Kim, H. J. Kim, H. F. Korth, "Semantics and Implementation of Schema Evolution in Object-Oriented Databases, "Proceedings of the ACM SIGMOND Conference, 1987.

[7]津田, 山本, 平川, 田中, 市川, "オブジェクト指向データモデルＭＯＲＥ, "情報処理データベースシステム研究会報告 88-63-(5), 1988

[8]津田, 山本, 平川, 田中, 市川, "ＭＯＲＥに基づくオブジェクト指向データベースシステム, "情報処理データベースシステム研究会報告 88-63-(6), 1988