データベース・システム 63－5
（1988. 1. 18）

オブジェクト指向データモデルMORE

津田 和幸、 山本 研策、 平川 正人、 田中 稔、 市川 忠男

広島大学工学部

データベースシステムのマルチメディア化、高機能化の要求を受けて、オブジェクト指向の概念に基づく新しいデータモデルの提案が盛んに行なわれている。本論文では、MOREと呼ぶオブジェクト指向データモデルを提案する。MOREでは、クラス間で通信されるメッセージに応じてアグリゲーション階層が動的に決定されるようになっており、データベーススキーマの多重化の実現、ならびにユーザごとに個人のスキーマを規定することができる。

MORE: An Object-Oriented Data Model

K. TSUDA, K. YAMAMOTO, M. HIRAKAWA, M. TANAKA, and T. ICHIKAWA
Information Systems, Faculty of Engineering, Hiroshima University
Shitami, Saijo-cho, Higashi-Hiroshima 724, Japan

There are several researchs on new data models which are based on object-oriented concept for the realization of mutimedia databases and sophisticated database systems. This paper describes an object-oriented data model called MORE. In MORE, an aggregation hierarchy is changed dynamically by passing messages between classes. This mechanism supports the definition of multiple database schema and the creation of a user-specific schema.

１．はじめに

データベース技術は関係モデルを核として飛躍的な発展をとげてきた。しかしながらその反面、関係モデルのデータ表現能力の低さのために、従来のシステムでは、拡大しつつあるデータベースアプリケーションに対して満足のいくサービスを提供することが困難になってきている。特に、複雑な対象物の取り扱いが要求されるデータベースのアプリケーション、例えばＣＡＤ、ＣＡＩ、オフィス情報システム、地理情報システムにおいては、図形・音声などといった新しいメディアの使用が不可欠であり、また、これらのデータ間に存在する複雑な関係を的確に表現できる高次データ表現能力が要求される。

これに対して、近年、データの意味表現についての研究が行なわれており、汎化/専化、集約といった概念をモデル内に自然に表現できる意味論的データモデルが多く提案されてきている[1]-[3]。一方、Smalltalkを用いてデータベースを記述しようとするGemStone[4],[5]の試みに端を発したオブジェクト指向データモデルの研究がある[6]。意味論的データモデルならびにオブジェクト指向データモデルは互いに強く関連しあっており、現在ではそれらをまとめてオブジェクト指向データモデルと考えることが多い。以下において、オブジェクト指向データモデルと言うときはこの呼び方に従うものとする。

オブジェクト指向データモデルは、関係モデルなどの従来のデータモデルのようにレコード単位で情報を扱うのではなく、現実世界に存在する全ての物をオブジェクトとして表現・管理する。オブジェクトは複雑なデータの階層を表現できる。これによって、データベースの操作ならびに管理が統一的に行なえる。

本稿では、オブジェクト指向データモデルＭＯＲＥを提案する。ＭＯＲＥは、主に、マルチメディアデータの取り扱いならびにデータベースシステム上でのアプリケーションシステム開発の容易性を実現する目的で設計されている。基本的にはオブジェクト指向モデルの概念を踏襲しており、オブジェクト、クラス階層、インヘリタンスなどの概念に基づいてデータベースの記述が行なわれる。さらにＭＯＲＥでは、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クラスタイプが新たに提案されている。

２章では、これからのデータベースシステムに要求される機能についてまとめるとともに、提案するＭＯＲＥデータモデルの概要について説明する。３章では、ＭＯＲＥのクラス記述ならびにモデル上のオペレーションについて述べ、４章では、ＭＯＲＥ独自の機能について詳述する。

２．ＭＯＲＥの概要

本章では、我々が目指しているデータベースシステムについて明らかにするとともに、そのようなシステムの実現に向けて現在研究を行なっているデータモデルＭＯＲＥの概略について述べる。

２．１　データベースシステムへの要件

これからのデータベースシステムに要求される機能として以下の４つを挙げることができる。

(1)マルチメディア対応
(2)アプリケーションシステム管理
(3)カスタム化
(4)システム拡張性

(1)については、単に文字、数値、図形、音声などといった複数のメディアのデータが取り扱えるというだけでなく、それらのデータの持つ理論的な構造や意味が明確に表現でき、なおかつそれらを統一的に管理できることが要求される。さらに、既存のデータベースを変更することなしに新しいメディアの追加を行なうことができてはじめて、マルチメディア対応が実現できると考えている。

(2)は、各アプリケーションシステムの管理をデータベースシステムの下でおこない、データベースシステムに備えられている諸機能をアプリケーションシステムに解放しようとするものである。データとアプリケーションを統括的に管理し、データだけでなくアプリケーションシステムの諸機能をアプリケーション間で共有できるようにする。これによって、アプリケーションシステム作成に伴う負荷を軽減することができる。

(3)は、ユーザごとにデータベース環境やデータベーススキーマの作成を支援する機能である。これによって、各ユーザは自分の作業環境を定義しその上で自分の所望するスキーマに沿ってデータベースを操作することができる。

(4)は、ハードウェア/ソフトウェアの進歩に伴って、データベースシステムの諸機能の追加、変更が容易に行なえることを要求している。

上に述べたような機能を備えているデータベースシステムを作成するにあたっては、データモデルはデータベース記述に用いられるだけでなく、それ自体がデータと共にアプリケーションに提供されなければならない。次節では、そのような機能を備えたデータモデルとして、我々の提案するＭＯＲＥについて概説する。

２．２　ＭＯＲＥ

ＭＯＲＥデータモデルはオブジェクト指向データモデルの１つである。一般のオブジェクト指向データモデルと同様、ＭＯＲＥでは、既に存在しているオブジェクトのアグリゲーション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たオブジェクトを複合オブジェクトと呼んでいる。これに対して、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んでいる。ＭＯＲＥでは、複合オブジェクトの表現あたって属性、関係を用いず、すべてアグリゲーション階層だけで行なう。

オブジェクトが持つ共通の性質は、すべてクラスにおいて管理される。ただし、ＭＯＲＥにおいてはクラスは共通性質を持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。クラス記述にはオブジェクトのアグリゲーション階層の情報、さらに階層構造上の制約、操作などの定義がなされる。また、クラス記述にはクラス間の汎化/専化を実現するためのクラス間関係が定義されるようになっており、これによってクラス階層が形成される。データベーススキーマはこれら二つの階層、すなわち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。データベース操作にあたっては、こうしたデータ自身とそれに適応できる操作がひとまとまりとなって定義される。このような情報のカプセル化はさまざまなメディアのデータの取り扱いを規定するのに都合がよい。

ＭＯＲＥでは、上で述べたような従来のオブジェクト指向データモデルに共通する概念に付け加え、次のような拡張を行なっている。

まず、オブジェクトに関しては、データだけでなくファンクションやプロシージャなどもオブジェクトとしてとらえ、データベースシステムの管理対象としている。これによって、データベースシステムの拡張性[7]-[9]やアプリケーション管理といった機能を実現する。

次に、クラス記述で定められたオブジェクトのアグリゲーション階層を唯一不変のもとせず、それをクラス間のメッセージ通信によって動的に変更できるものとしている。これによって、１つのオブジェクトに複数の表現を持たせることができ、データベースに対する操作の局面に応じて適切なオブジェクト表現を選択できるようになる。また、同じ仕組みを用いてオブジェクトのカスタム化を支援することができる。詳しくは4.2で述べる。

加えて、クラス定義の容易化、並びに他のモデル表現を通してＭＯＲＥで記述されたデータベースを操作するために、クラスのタイプを定義するクラスタイプという考えを導入している。クラスタイプはSmalltalkのメタクラスに類似している。しかし、メタクラスのようにクラスに対して１つのメタクラスが定義され、クラス変数の初期値などを行なうのではなく、類似する構造を持つクラスの構造的制約を与え、これによってメソッドの生成などを行なう。これについては4.3で述べる。

また、全てのオブジェクトに対して、オブジェクトの変更の遷移を管理するバージョン管理のための機構を備えている。

３．クラス定義

３．１　クラス記述

クラスは共通性質を持ったオブジェクトの集合であり、クラスに属する全てのオブジェクトに対して共通に施すことのできる操作を与えている。

クラス記述は以下に示す項目から成っており、システムが表示する入力テンプレートの各項目を埋めることでクラスの定義を行なう。

CLASS

定義するクラスの名前と、このクラスの注釈を記述する。

CLASSTYPE

クラスのクラスタイプを定義する。CLASSTYPEの役割については4.3で述べる。デフォルト値は'OBJECT'である。

INTERCLASS CONNECTION

既に定義されているクラスと、ここで定義するクラスとの関係を集合関係によって記述する。許される関係には、 'SUBSET-of'、'SUPERSET-of'、 'INTERSECTION-of'、および'UNION-of'がある。すべてのクラスは'Universe'のサブセットとして取り扱われる。

ここに記述された関係によってクラス階層が形成される。この階層に従って4.1で述べる継承が定義され、オブジェクトスコープが決定される。

CONSTRAINT

クラスに属するオブジェクトの満たすべき制約を、STRUCTURAL-CONSTRAINTS, OBJECT-CONSTRAINTS, MESSAGE-CONSTRAINTSの三つで記述する。

STRUCTURAL-CONSTRAINTS

このクラスに属するオブジェクトの構造に関する制約である。すなわち、クラスのもつアグリゲーション階層を記述するもので、'reference-name, domain, order'形式をとる。

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…オブジェクトが存在すべきクラスの名前またはオブジェクトの集合。

order…オブジェクトとして取りうる個数に関する制約(上限値)で、'1'あるいは'N'のいずれかをとる。必ず何かのオブジェクトを持つことを要求するNON-NULL制約の場合は、'1'あるいは'N'の後ろに'+'を付けて表現する。

OBJECT-CONSTRAINTS

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。

MESSAGE-CONSTRAINTS

クラスに属するオブジェクトの満たすべき制約条件を、オブジェクトに与えたメッセージに対するアクションによって記述する。

METHOD

クラスあるいはクラスに属するオブジェクトに対する操作を定義する。メソッドには、daemon、structural、operationalの三種類がある。daemonメソッドは他のオブジェクトが変更されたり、自分自身が変更されたときに起動される。すなわちトリガー／アラータ機能を実現するものであり、'IF reference-name/object IS-MODIFIED action'の形式で記述される。structuralメソッドは、送られてきたメッセージのパターンに従って、このクラスの現在のアグリゲーション構造を決定するように働く。このメソッドは'%message-pattern action'の形式で定義される。operationalメソッドはクラスあるいはクラス内のオブジェクトを操作するためのメソッドであり、'!message-pattern action'の形式で記述される。

actionは'destination message/operation argments;'の繰り返しで記述される。destinationとargumentsは、特定のオブジェクトあるいはクラスを直接記述するか、またはreference nameを用いて、あるいはactionから返されるオブジェクトを用いて定義する。message/operationには、destinationのクラスに定義された先頭に'%'の付いているstructuralメソッドまたは先頭に'!'の付いたoperationalメソッドであるか、あるいは次節で説明するモデル上のオペレーションが記述されなくてはならない。オペレーションを用いる場合には'#'をオペレーション名の先頭に付ける。

文書を対象としたクラス記述の例を図1に示す。また、そのグラフィカルな表現を図2に示す。図中では、楕円がクラスを、円がオブジェクトを表している。またアグリゲーション階層を矢印の付いた線で表現している。

```
CLASS
    NAME:          Document
    DESCRIPTION:   Multimedia documents which are
                   created by using of document editor.
INTERCLASS_CONNECTION
    SUBSET_of  Universe;
    SUPERSET_of  Paper;
    SUPERSET_of  Report;
CONSTRAINT
    STRUCTURAL_CONSTRAINTS:
        date,          Date,          1;
        title,         String,        1+;
        kind,          Doc.-kind,     1;
        header,        Header,        1+;
        body,          Body,          1+;
        ...
METHOD
    !title:        title !;
    !kind:         kind !;
    IF body IS-MODIFIED date !today;
    ...
```

(a) クラス 'Document'

```
CLASS
    NAME:          Paper
    DESCRIPTION:   Multimedia documents formatted as a
                   paper for a document editor.
INTERCLASS_CONNECTION
    SUBSET_of  Document;
CONSTRAINT
    STRUCTURAL_CONSTRAINTS:
        date,          Date,          1;
        title,         String,        1+;
        kind,          Doc.-kind ,    1+;
        journal-name,  Journal,       1;
        header,        Header,        1+;
        body,          Body,          1+;
        footer,        Footer,        1;
        ...
    OBJECT_CONSTRAINTS:
        kind = "paper";
        ...
METHOD
    %contents:     header %contents; body %contents;
    %reference:    header %refer; journal-name %;
    ...
```

(b) クラス 'Paper'

図1 クラス記述の例

３．２　オペレーション

オブジェクトおよびクラスに対するオペレーションには以下のものがある。

クラスに対しては、生成、削除、および変更の操作が許されている。オブジェクトが存在しているクラスを削除する場合には、そのクラスに属するオブジェクトはそのクラスのスーパーセットであるクラスに移される。削除されるクラスに定義されていたオブジェクトの性質は、そのクラスのサブセットであるクラスに移される。

オブジェクトに対しては、生成、削除、変更、移動、複写が定義されている。

これらのオペレーションは、メソッドのアクションに記述でき、ユーザから与えられた１つのメッセージによって、データベース中のさまざまなオブジェクトやクラスに対して操作を行なうことができる。

４．ＭＯＲＥの特徴

本章では、ＭＯＲＥに新たに導入した概念・機能について、例を用いながら詳細に述べる。

４．１　クラス間関係

3.1で示したように、すべてのクラスには必ずその他のクラスとの集合的関係が規定されている。この関係によって、オブジェクトの性質のインヘリタンスを実現し、クラスでのオブジェクトスコープを決定する。

インヘリタンスならびにオブジェクトスコープに関して、次のルールを定義する。

C1⊃C2のとき
o2∈C2ならば　　o2∈C1,
　　$Re_{C1}(oi)$,
　　$Me_{C1}(oi)$

C3 = C1∪C2のとき
o3∈C3ならば　　$Re'_{C1}(o3)$, $Re'_{C2}(o3)$,
　　$Me_{C1}(o3)$, $Me_{C2}(o3)$

ここで、oiはオブジェクトを、Cjはクラスを表している。また、オブジェクトoiがクラスCjで定められた制約を満足することを$Re_{Cj}(oi)$と表している。オブジェクトoiに対してクラスCjに記述されているメッセージが適応可能であるならば、$Me_{Cj}(oi)$と表している。但し、Re'はルーズリリストリクションを表しており、STRUCTURAL-CONSTRAINTSのNON-NULL制約を排除したものである。

例えば、図２において、オブジェクト'p-a'、'p-b'、'c'、及び'r-d'はすべてクラス'Document'に属しており、そのクラスからそれら全てのオブジェクトを観測することができる。このとき、クラス'Document'に定義されている制約およびメッセージはそのサブクラスである'Report'および'Paper'に継承されるため、それらのすべてのクラスでは、'Document'クラスにおける制約を満たしておかなければならず、また'Document'クラスに記述されているメッセージのすべてを受理することができる。

一方、あるオブジェクトから考えると、そのオブジェクトは複数のクラスに属しているので、オブジェクトの扱いは現在注目しているクラスの記述に応

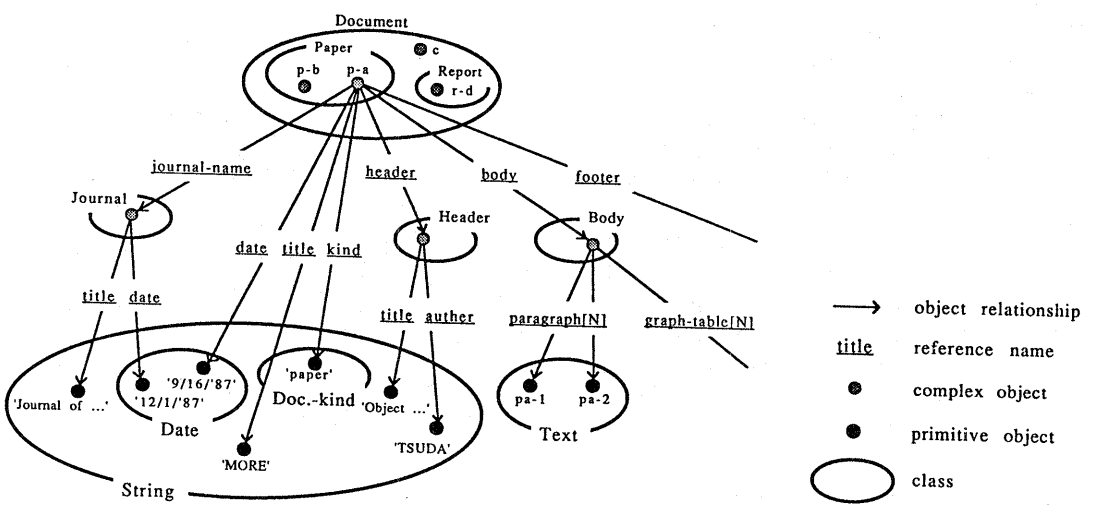


図２　図１のグラフィカル表現

じて行なわれる。つまり、ユーザが注目しているクラスが異なれば、たとえ同じオブジェクトであってもユーザが得る情報は異なったものとなる。例えば、図2におけるオブジェクト'p-b'をクラス'Document'あるいは'Paper'において取り扱う場合、オブジェクトの操作は、図1で定義したそれぞれのクラスのアグリゲーション構造に応じてしか許されない。すなわち、'Document'クラスに注目しているさいは、'journal-name'とか'footer'に対する操作は行なえない。

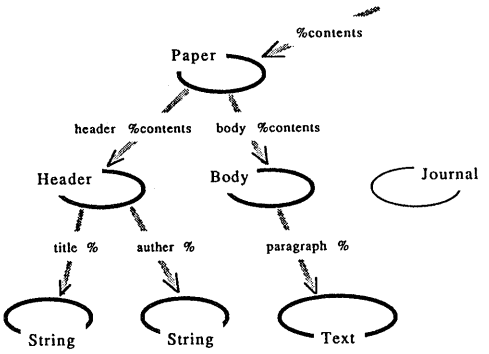
さらに、ＭＯＲＥでは各クラスのアグリゲーション構造が動的に決定されるため、より複雑な表現の記述/取り扱いが可能となる。これについては次節で述べる。

４．２　メッセージ評価とアグリゲーション階層

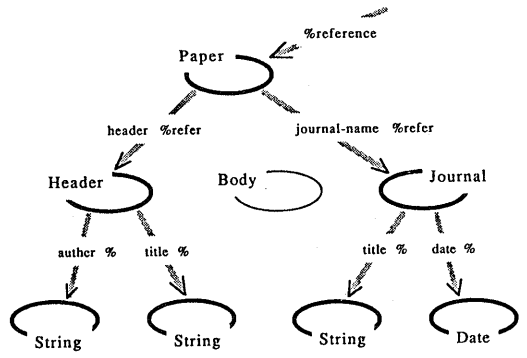
ＭＯＲＥの最大の特徴は、固定されたスキーマに従ってオブジェクトを操作をするのではなく、対象とするスキーマを動的に決定したうえで操作を行なえることである。すなわち、ＭＯＲＥではオブジェクトが複数のビューを持つことを許している。これは、例えば、円錐という物体を円として取り扱ったり（真上からの眺め）、三角形として取り扱ったり（側面からの眺め）することができることを意味している。このようなしくみを取り入れるのは、単にデータベーススキーマを記述し定義されたスキーマに基づいてデータの操作を行なうデータベースシステムを作成するためのデータモデルとしてＭＯＲＥを用意するのではなく、ユーザにモデルそのもの、すなわちアプリケーション特有のデータ、関連性、およびそれらの操作環境までも提供し、そのうえでアプリケーションシステムの作成を行なわせるようなデータベースシステムを実現するためのモデルとして用いるからである。

スキーマの決定はクラス記述のstructuralメソッドによって実現される。この仕組みを、図１および図２に示した文書を表現するクラスの例を用いて説明する。

ある論文は、例えばそのまま１つの文書として取り扱いたい場合と、１つの参考文献として引用したい場合とがある。前者の場合、論文の全体の構成を表現するようなアグリゲーション階層が必要であるが、後者の場合には論文の題目や著者といった項目を引用するアグリゲーション階層だけでよい。ＭＯＲＥではこうした場合、メッセージを直接、あるいはオブジェクトを通じて間接的に論文のクラスに送り、対応するアグリゲーション構造を決定したうえでオブジェクトを参照することで上述の機能を実現する。アグリゲーション階層決定の過程の例を図３に示す。図１のクラス記述に定義されているように、'Paper'というクラスはメッセージ'%contents'と'%reference'を受け取ることができる。'%contents'を受けたとき、対応するアクションとしてクラス'Paper'はそのアクション部に記述されている'header %contents; body %contents;'を実行し、headerおよびbodyで参照されるクラスにそれぞれのメッセージを送る。'%contents'を受けたクラス'Header'は'title %; author %;'を実行する。これによって図３(ａ)に示すようにメッセージが伝搬されメッセージを受けたアクティブなクラス（図中の太線のクラス）がクラス'Paper'がメッセージ'%contents'を受けた時の操作できるアグリゲーション階層を形成する。一方、メッセージ'%reference'に対しては、メッセージの通信がクラス間で図３(ｂ)のように行なわれ、それに従ったアグリゲーション階層が形成される。



（ａ）　メッセージ '%contents'



（ｂ）　メッセージ '%reference'

図３　動的アグリゲーション階層決定の例

さらに、このメッセージ評価メカニズムを用いて、個人用のデータベースの外部スキーマを定義することができる。外部スキーマの定義は各クラスのアグリゲーション階層の一部を見えなくしたり、アグリゲーション階層のつながりを変更することによって行なう。このような機能を実現するために、structuralメソッドの送り先オブジェクトの指定を次のように拡張している。

［１］アグリゲーション階層中の任意のノードを、自分自身のクラスからのreference-nameを用いたパスによって指定する。

［２］クラス記述のSTRUCTURAL-CONSTRAINTSのorder制約が１であれば、'～'を付けた参照名によって、クラスの参照関係を逆にたどっていくことができる。

４．３　クラスタイプ

クラスタイプはデータベーススキーマを表現するために導入したものではなく、クラス設計の簡素化を支援するためのものである。例えば、家族とか会社の部門を表すクラスを考えてみる。これらのクラスの定義にあたっては、クラスを構成するメンバーの集合と、オブジェクトを他と区別するための住所、電話、部名、内線番号といったいわゆる属性の役割をはたすオブジェクトとが必要である。この例のように、構造が類似しているクラスでは、それらの間には共通な基本操作系があるはずである。前述の例においては、例えば個々のメンバーをアクセスするためのものや、全ての属性を表示するものなどが考えられる。このように複数のクラスに存在する構造の類似性を基に、それらのクラスに共通に用いることのできる操作を生成するためのものがクラスタイプである。

クラスタイプはSmalltalkのメタクラスと類似している。しかしながら、メタクラスでは、クラスに対して１つのメタクラスが定義され初期化などの操作を受け持っているのに対して、クラスタイプにおいてはクラスタイプとクラスとは一対多の対応を示し、クラスにおけるアグリゲーション階層の定義を制約するとともにそれらのクラスに共通に有効な操作系を生成する。

図４にクラスタイプの宣言例を示す。これは前述の家族と会社の部門の例に用いるクラスを生成するためのクラスタイプである。例えばクラスタイプ'WITH-G-A'を宣言するクラスではGROUP型でorder制約が'１'であるクラスを１つと、ATTRIBUTE型でorder制約が'１'であるクラスを複数持つ構造でなければならないことがこのクラスタイプで定義されている。

```
CLASSTYPE
    NAME:          WITH-G-A
    DESCRIPTION:   For a class which has a grouped object
                   and several attributive objects.
    STRUCTURE:     G:        GROUP        1,   1;
                   A:        ATTRIBUTE    1,   N;
    METHOD:        !GET-E:   G !ELEMENTS;
                   !GET-A:   A !; .
```

```
CLASSTYPE
    NAME:          GROUP
    DESCRIPTION:   For representing a grouped object which
                   is a set of objects in a certain class.
    STRUCTURE:     GO:       UNIV,        N,   1;
    METHOD:        !ELEMENTS: GO !;
                   !I-TH-E:  GO[$1] !; .
```

```
CLASSTYPE
    NAME:          ATTRIBUTE
    DESCRIPTION:   For an attributive object which is used
                   for specifying the characteristics.
    RULES:         UNIV; .
```

図４　クラスタイプ宣言の例

```
CLASS
    NAME:          Family
    DESCRIPTIONS:
    CLASSTYPE:     WITH-G-A
INTERCLASS_CONNECTION
    SUBSET_of Universe;
CONSTRAINT
    STRUCTURAL_CONSTRAINTS:
        G:  $G,   $GROUP,      1;
        A:  $A,   $ATTRIBUTE,  1;
                  ...
        A:  $A,   $ATTRIBUTE,  1;
METHOD
        GET-E:    $G !ELEMENT;
        GET-A:    $A !; .
```

（ａ）　変換された入力テンプレート

```
CLASS
    NAME:          Family
    DESCRIPTIONS:
    CLASSTYPE:     WITH-G-A
INTERCLASS_CONNECTION
    SUBSET_of Universe;
CONSTRAINT
    STRUCTURAL_CONSTRAINTS:
        G:  f-set,    F_set,    1;
        A:  address,  String,   1;
        A:  phone,    String,   1;
METHOD
        GET-E:    f-set !ELEMENT;
        GET-A:    address !;  phone !; .
```

（ｂ）　生成されたメソッド

図５　図４を用いたクラス記述の例

次に、このクラスタイプを用いてクラス'Family'を定義する手順を説明する。'Family'のクラス記述を定義するさい、まずCLASSTYPEの項目において'WITH-G-A'を宣言する。このときシステムの提供するクラス記述用の入力テンプレートがクラスタイプ'WITH-G-A'の記述によって図５(ａ)のように変更される。図において'＄'の付いた項目はそれを展開することが要求される項目である。ここで、例えば＄Ｇに'f-set'を、＄Ａにそれぞれ'address'と'phone'を代入してやると、図５(ｂ)に示される'!GET-E'という家族の構成員を取り出すメソッドと、'!GET-A'という家族の住所および電話番号を取り出すメソッドが生成される。

５．終りに

本稿では、現在我々が開発をしているオブジェクト指向データモデルＭＯＲＥについて述べた。一般のオブジェクト指向データモデルの性質に加えてメッセージ通信による操作スキーマの動的決定機構、クラスタイプの導入などを行なっている。

現在、このモデルに基づく実験システムを開発中である[10]。今後は、分散データベースシステム開発の足掛りとして、本稿で提案したクラスタイプを用いて、任意のデータモデルの間でのモデル変換について研究を行なう予定である。具体的にはターゲットモデルのコンポーネント(例えばＥ－Ｒモデルでは事象、関連、属性ならびに、それに対する操作)をクラスタイプを用いて記述することによって、ターゲットモデルのデータベースをＭＯＲＥデータベースで表現し、逆に、ターゲットモデルの表現を通してＭＯＲＥデータベースを操作することを考えている。

参考文献

[1] M. Hammer and D. Mcleod, "Database description with SDM: A Sementic Database Model," ACM TODS, Vol. 6, No. 3, pp. 351-386, Sep. 1981.

[2] D. Shipman, "The Functional Data Model and Data Language DAPLEX," ACM TODS, Vol. 6, No. 1, pp. 140-173, 1981.

[3] T. B. Stee, Jr. and R. Meersman (eds.), Database Semantics(DS-1), North-Holland, Amsterdam, 1985.

[4] G. Copeland and D. Maier, "Making Smalltalk a Database System," ACM SIGMOD '84, pp. 316-325, 1984.

[5] D. Maier, J. Stein, A. Otis, and A. Purdy, "Development of an Object-Oriented DBMS," ACM OOPSLA '86, pp.472-482, 1986.

[6] K. Dittrich and U. Dayal (eds.), International Workshop on Object-Oriented Database Systems, 1986.

[7] D. S. Batory and M. Mannino, "Extensible Database Systems," (Pannel) ACM SIGMOD '86, pp. 187-190, 1986.

[8] M. J. Carey and D. J. DeWitte, "The Architecture of the EXODUS Extensible DBMS," International Workshop on Object-Oriented Database Systems, pp. 52-65, 1986.

[9] M. Stonebraker and L. A. Rowe, "The Design of POSTGRES," ACM SIGMOD '86, pp. 340-355, 1986.

[10] 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.