解　説

1. オブジェクト指向データベースシステムの基本

1.3 オブジェクト指向データベース管理システムのアーキテクチャ†

牧之内　顕文††

1. はじめに

本稿ではオブジェクト指向データベース管理システム (OODBMS) の具体例を取り上げ，アーキテクチャに焦点を当てて紹介する．具体論に入る前に，データベース管理システム (DBMS) のアーキテクチャの変遷をデータベース (DB) 応用と計算機環境との変遷から眺めてみよう．

CODASYL 型とそれに対抗する形で出現した関係データベース (RDB) も当時の応用分野と計算機環境に影響されて生まれた．応用としては人事管理，在庫管理などの社内システム，あるいは，銀行の勘定システムや座席予約システムなどの社会システムが考えられた．これら事務データ応用の特徴は(1)大量のデータに，(2)少量の計算と，(3)比較的高い頻度のトランザクションである．一つのトランザクションは，(a)少量のデータを検索し，(b)検索したデータに対して単純な計算を施し，(c)その結果変更されたデータが再度データベースに格納される，といったきわめて単純な操作系列から構成される．しかし，単位時間当たりのトランザクション数は社会システムでは数万から十数万にのぼった．

一方，当時の計算機環境は共用メインフレームであり，しかも主メモリは（特に，個々のセッションが利用できる物理空間はもとより仮想空間までも）きわめて小さいものであった．したがって，できるだけ少ないメモリ量（ワーキングセット）で効率よく DB 操作を行うことが要求された．このため DBMS は小さいバッファでも動作可能でありかつバッファの内容の頻繁な変更（これは当然 DB への書き込みがともなう）が可能なシステム作りがなされた．逆に言えば，上記事務データ処理はこのようなシステムアーキテクチャで十分間に合う応用であった．実際，一つのトランザクションは長い計算を必要としないため，処理対象データを永く主記憶上に滞留させる必要がない．

その後データベース応用が事務データ処理分野以外に広がり始めた．そのような新分野が従来分野と異なるのは次の三点である．

(1)複雑な構造のデータを扱う必要がある，(2)マルチメディア化，(3)複雑な計算，これらは互いに密接に関連している部分もあるがそれぞれ独立した要求でもある．この3点セットを要求する分野として CAD があげられよう．

CAD などでは科学技術上のシミュレーション計算が必須である．これら計算は一般に CPU 時間を食う．上記三つの要求を満足するには，DBMS は関連しあう多量のデータを一度に検索し，それを主記憶上に永く滞留させることを可能としなければならない．さもないと計算の途中必要になるデータをその都度ディスクから読まねばならない．それでは時間が掛かり過ぎ実効的な計算はできない．したがって，大容量の主記憶をふんだんに使うことを想定したアーキテクチャを取らざるをえない．一般に CAD では一回当たりの計算時間は比較的長いから前後の I/O 時間が長くても容認される．大量のデータを読み込み，それをプログラミング言語で規定されるデータ構造に変換し，その上で処理を施す．処理後，変更されたデータをファイルに格納する．そのとき，データ変換が必要である．このような状況で要請されるのは主記憶中のデータ構造とファイル中のデータ構造との不一致をなくすることであろう．

安価なワークステーションの出現がこのようなアーキテクチャ実現を可能とした．

† Architectures of the Object-Oriented Database Management Systems by Akifumi MAKINOUCHI (Depertment of Computer Science and Communication Engineering, Kyushu University).

†† 九州大学工学部情報工学科

2. OODBMS 開発のアプローチ

以下，2. では OODBMS のアーキテクチャを分類し概説する．お急ぎの方はこの章だけを読むだけで大概のことは分かる．3. 以下では 2. で分類したアーキテクチャの代表例をあげ，説明を加える．

2.1 オブジェクト指向プログラミング言語 (OOPL) の処理方式

オブジェクト指向 DBMS について最初に言及した論文は "Making Smalltalk a Database System"[2] であるとされている．その名が示すとおり，Smalltalk に触発されてできた論文であろう．ここで Smalltalk プロセッサのアーキテクチャを知っておくのも OODBMS の原点を理解する上で役に立つであろう．

Smalltalk-80 は Storage Manager, Interpreter, Primitive Subroutines からなる[3]．Storage Manager の機能は（1）クラスの fetch,（2）オブジェクトのフィールドの fetch と store,（3）新しいオブジェクトの創生，及び（4）空き空間の収集と管理である．図-1 と 2 にオブジェクトの格納空間と構造を示す．図には示されていないが，オブジェクトポインタは間接ポインタで，Storage Manager が管理するテーブルのエントリを指す．そのエントリにはオブジェクトの実際の格納場所の番地が入っている．また，オブジェクトのクラスやそのフィールドもまたオブジェクトであり，それらはオブジェクトからポインタで指される．図-1 の格納空間がディスク上の空間であれば Smalltalk は OODBMS になる．

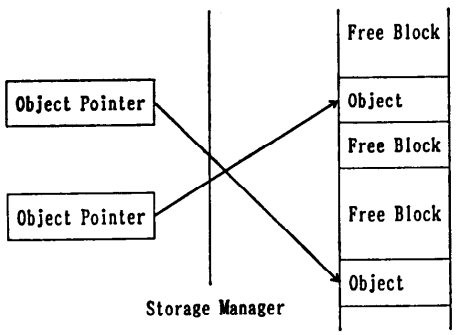

図-1 Smalltalk-80 のオブジェクトと記憶域

(Length) 4
(Class description) Triangle
(First vertex)
(Second vertex)
(Third vertex)

図-2 Smalltalk の典型的なオブジェクト記述例（文献 3) より）

2.2 OOPL 拡張派

OODBMS 開発のアプローチの一つは OOPL を拡張して永続データ (persistent data) を扱えるようにすることである．OOPL としては，Smalltalk や C++ が存在するのでとりあえずそれらの言語を利用しようという試みがある．前者では GemStone[4]，後者では O++[15] が代表的である．また，人工知能プロジェクトに関係して開発されたシステムに ORION[6] がある．これは基礎になる言語として LISP が使われている．最近は C++ 派が優勢なようで PC++[16] などはごく最近の例である．

このアプローチの特徴は，（1）オブジェクト識別子 (OID) はポインタであり，それがオブジェクトを一意に指示する，（2）オブジェクトの検索はポインタをたどることによって順次行う（巡航検索）．

Smalltalk ではすべてがポインタで指されるためポインタという概念が陽には出てこないが C++ を基盤にした言語ではオブジェクト参照に必要である．この種のポインタは（a）オブジェクトへの直接アクセスに利用される点でプログラミング言語のポインタと似ているが（b）永続オブジェクトの生存中変化しない点でそれとは異なる．システムの効率を重視する立場からは（a）の直接参照の効率化が重要である．

プログラムがオブジェクトを操作するときにはオブジェクトは主記憶上に読み込まれ，それと同時にポインタは主記憶上（仮想空間）の番地に変換されねばならない．このプログラム（通常，メソッドと呼ばれる）が動作し続けているあいだは，当該オブジェクト（データ）はメモリ中に留まっていなければならない．したがって OID から仮想空間上の番地への変換操作が容易で，かつ，一度変換したら再度の変換をせずともその番地を使ってオブジェクトにアクセスできる機構を用意することが非常に重要なことになる．

OODBMS の多くは CAD を当初の応用分野と

している．CAD で受け入れられるための重要な要件はやはり性能[5]である．性能を上げる一つの手は互いに関連するオブジェクト群を一度に主記憶上に読み込むことにある．このためオブジェクト群を二次記憶上でクラスタリングする機能が要求される．クラスタ内のオブジェクトの集合は一度に主記憶に読み込まれる．通常，このクラスタはサイズが大きいが，ワークステーションで利用可能な主記憶には収まる．主記憶上に置かれたオブジェクトは互いに OID で関連づけられているが適当なときにそれら OID は主記憶上の番地に変えられる．このようにしてプログラムは最初の参照以外は番地をたどってデータにアクセスできるため，処理効率は全体としてよくなる．

2.3 RDB 拡張派

このアプローチの代表は POSTGRES[7),17),18)]である．また，関係モデルではないがそれに近い関数データモデル[20)]にオブジェクト指向パラダイムの要素を取り入れた IRIS[19)] もこの派に分類したい．この両者の特徴は問い合わせ言語における SQL (POSTGRES では QUEL) の継承である．これはまた，OOPL 拡張派と違って「計算完備 (Computational completeness)」をそれほど重要視していない．逆に言えば，エンドユーザ重視路線を歩む．

実際，Stonebraker らは第三世代データベースシステム宣言[10)]で「第三世代 DBMS は第二世代 DBMS を包含すべきである」と言っている．言葉を代えて言えば「非手続き型アクセス」と「データ独立性」を損なうべきでないと主張している．

アーキテクチャの観点からみると既存の RDBMS を中核にして新しい機能を追加した実現法を採っている．たとえば，「POSTGRES は従来の (a fairly conventional) パーサ，問い合わせ最適化，実行エンジンを含む[7)]」と述べている．

IRIS アーキテクチャの基本的枠組みは RDB のそれである．その Storage Manager と呼ばれる「従来の関係型記憶サブシステムである HP-SQL の Storage Manager」は System R の RSS[11)]にきわめてよく似ている[19)]．

POSTGRES と IRIS 共に非手続き型問い合わせ言語の処理に重点が置かれているため，アーキテクチャは従来の RDBMS のそれと似かよっている．しかし POSTGRES の関数やオペレータの導入，IRIS の外部関数の存在が従来アーキテクチャにみられない特徴を生み出している．

RDB 拡張派のアプローチを徹底するとオブジェクト指向 RDB とも言える概念に行き着く[12)]．これは最も現実的な方法で RDBMS を OODBMS に変換するものである．DB を格納管理する DBMS としては既存の RDBMS を使い，それを OOPL で定義されるオブジェクトの格納庫として利用する．OOPL の特徴は OID による巡航操作であるからこれを高速に行う機構をいかに上手に作るかが全体システムのアーキテクチャの工夫になる．これは OOPL 拡張アプローチと同様に主記憶上に一度に展開されたデータを巡行アクセスすることで解決する．

2.4 新しい DBPL 指向派

第三のアプローチは前二者の和を取るものである．すなわち，「計算完備」を重要視する立場からメソッドの記述言語としてたとえばCなどの汎用プログラミング言語を基盤とした新しいデータベースプログラミング言語 (DBPL) あるいは Persistent Programming Language (PPL) を用意する．一方，非手続き型問い合わせ言語 (QL) も提供する．その QL はもちろん DBPL 中で利用可能である．このアプローチの代表例として O_2[13)] と Jasmine[14)] とがある．O_2 の DBPL CO_2 と Jasmine の Jasmine/C とは言語構文設計思想が似ている．両言語ともCの構文慣例にできるだけ似せることによりCプログラマの学習負担をできるだけ小さくしようとしている．また，QL についても SQL の慣例に従い，RDB プログラマへの配慮を行う．

3. OOPL 拡張派—O++[15)] と PC++[16)]—

以下では OOPL 拡張派の代表である O++ と PC++ とを紹介する．

3.1 O++

O++ は ODE (Object Database and Environment) と呼ばれる DBMS の DBPL である．DBMS のデータベースはこの言語で定義，検索，操作される．O++ は端的にいって C++ に永続性を導入した言語である．その際の代表的設計基準は，

(1) 永続性はタイプに直交であるべきである．すなわち，永続性はインスタンスの性質であ

り，タイプのそれではない．いかなるタイプのオブジェクトも揮発性（volatile）あるいは永続記憶に割りつけることが可能であるべきだ．

（2） 永続オブジェクトの割り付けや操作は揮発オブジェクトのそれと同様にできなくてはいけない．

（3） 言語の変更は最小限に抑える．

図-3 にクラス item と stockitem の例をあげる．O++ では永続オブジェクトはポインタ（OID）によって参照される．これらのポインタはヒープあるいはスタックに割り付けうるが，それによって指されるオブジェクトは永続記憶に割り付けられる．'pnew'，'pdelete' によって永続オブジェクトの割り付けと削除がなされる．これは揮発オブジェクトの創生，削除操作 'new'，'delete' に対応するものである．したがって，

```
persistent stocitem *psip;
.....
psip=pnew stocitem (initial value);
```

で psip は揮発変数で永続オブジェクト stockitem へのポインタ値が入っている．代入式により永続オブジェクトの内容を揮発オブジェクトに写すこととその逆も可能である．

永続ポインタと揮発ポインタはおのおの永続，揮発オブジェクトしか指せない．両タイプのオブジェクトを参照するためのポインタが必要になる．そのために 'dual pointer' を導入する．dual pointer が永続オブジェクトを指しているのか揮発オブジェクトを指しているのかは実行時に判定される．

```
class item  {
  Name nm;
  double wt; /* in kg */
public:
  item(Name xname, double xwt);
  Name name();
  double weight lbs();
  double weight kg();
} ;

item::item(Name xname, double xwt)
 { nm = xname;
   wt = xwt;
}
Name item::name()
 { return nm;
}
double item::weight kg()
 { return wt;
}
double item::weight lbs()
 { return (wt*2.205);
}
```

図-3 O++ のクラス定義例（文献 15）より）

O++ のクラスは永続オブジェクトの入れ物ではない．したがって，倉庫としての何かが必要になる．クラスタがそのために導入された．一つのクラスタには同一タイプの永続オブジェクトが格納される．O++ はまた，マルチセット（重複要素を許す集合）を備えている．クラスタ及びセット内のオブジェクトを検索・操作するのに高水準 for-loop がある．たとえば，employee と dept とがクラスタあるいはセットとすると（DB 解説でよく使われる例と同じ属性を想像してください），

```
for e in employee, d in dept
  suchthat (e->dno==d->dno)
print ("%s %s n", e->name, d->name);
```

で結合が可能ということになる．

3.2 PC++ (Persistent C++)[16]

これは「C++ の継承機能を用いて C++ を拡張し永続オブジェクトをサポートできるようにした」ものである．PC++ は「オブジェクト指向プログラミングインタフェース，OID 生成，オブジェクト記憶領域の効率的管理，検索，競合管理さらに障害回復」の機能を提供する．PC++ はクライアント-サーバモデルに基づいており，応用ワークスペースマネージャとデータベースサーバとからなる．**図-4** にアーキテクチャを示す．

PC++ データベースは C のヒープ領域の拡張である．オブジェクトは OID で参照されるが索引が提供されないため必ず根オブジェクトからたどらねばならない．作業領域マネージャの構造を**図-5** に示す．persistent heap が用意される．これ

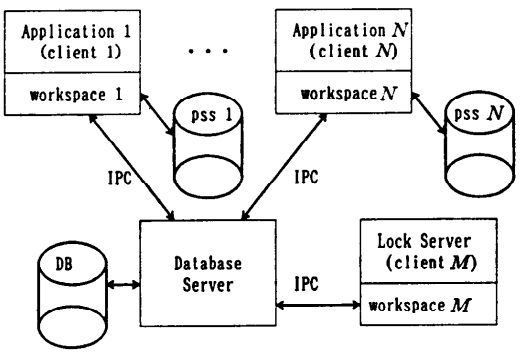

図-4 PC++ のシステムアーキテクチャ（文献 16）より）

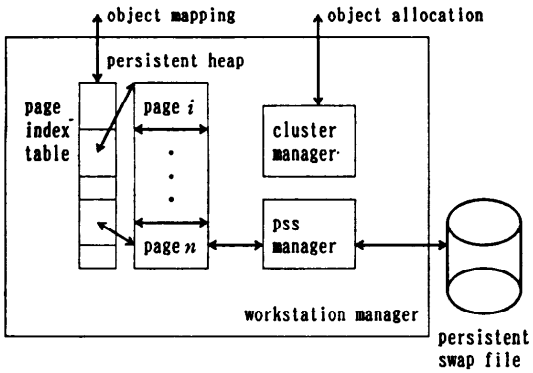

図-5 PC++ の応用作業域管理構造（文献 16）より）

は persistent swap file のバッファで複数のページからなる．ページサイズは 4K バイトである．各ページにはオブジェクト索引テーブルがあり，そのエントリがページ内のオブジェクトを指す．したがって，OID はページ番号とページ内索引のエントリ番号からなる．オペレータ '->' が OID からオブジェクトへの関数であり，OID が与えられると該当ページを persistent heap に探し，なければそこに読み込み，所望のオブジェクトの番地を返す．

page index table はページ番号と heap space 上のページへのポインタとの対をエントリとするハッシュテーブルである．cluster manager は各クラスタ内の領域管理とオブジェクトのクラス内割り付けを行う．

長いトランザクションが終了すると persistent heap と persistent swap file の各ページが検査され，変更されたオブジェクトはデータベースに反映される．

図-6 はデータベースサーバの構成である．base file には最新のページが格納される．version files には前の版のページとの差分が格納される．ページの各版はリンクされており，古いページが要求されると最新版からたどって古いページが再構成される．

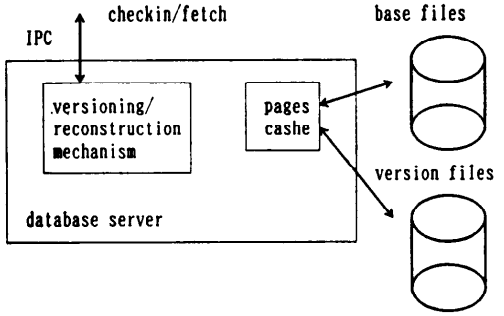

図-6 Database server 構造

4. RDB 拡張派—POSTGRES[7),17),18)] と IRIS[8),19)]—

この派に属するのは POSTGRES と IRIS である．これらについてはほかでもよく紹介されているのでごく簡単に述べるにとどめる．

4.1 POSTGRES

POSTGRES をアーキテクチャからみた場合の特徴は次の三つである．

（1） 新しいアクセスメソッドをユーザが定義できる．基本的には一つのアクセスメソッドは 13 個の通常関数の集まりであり，ユーザはこれら関数を使って応用に合ったアクセスメソッドを書く．

（2） Fast Path の存在．これはユーザが parser, optimizer, executer, アクセスメソッド，バッファ管理及びその他のユーティリティを直接呼び出せ，ユーザ定義の関数をパラメータ検査なしに直接呼び出せる機能である．

（3） write-ahead log なし，no-over write 変更が可能な storage system.

4.2 IRIS

IRIS データモデルと問合せコマンドの祖先は関数データモデル[20)]にある．しかし，IRIS コマンドは内部で関係代数に翻訳されて実行されるから，その意味で IRIS の計算モデルは関係代数である．**図-7** に IRIS システムの構成図を示す．IQIS Storage Manager は System R の RSS に近い構造・機能を有する．

IRIS の核は**図-8** に示される構造をもつ．Query Translator は IRIS の関数表現を F-tree と呼

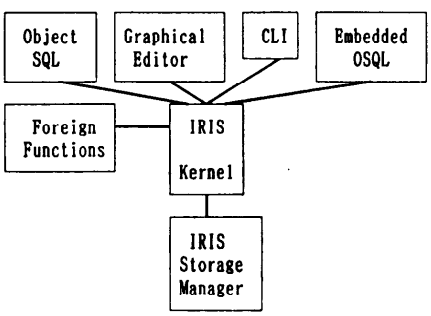

図-7 IRIS のシステムアーキテクチャ（文献 8）より）

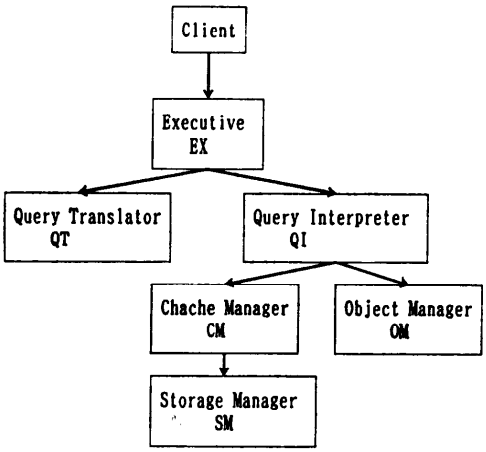

図-8 IRIS の核の構造（文献8）より）

ばれる木構造に変換する．F-tree は R-tree と呼ばれる関係代数の木構造に変換され QI により評価される．この木のノードとして foreign function ノードが存在する．これが評価されると対応する外部関数が起動され実行される．

POSTGRES と IRIS は基本的には関係データベースシステムである．両者とも属性の継承を許す．また，汎用プログラミング言語で書かれた関数を問い合わせ言語の中から呼び出すことを可能としている．しかし，元々のデータベース言語が関係型言語であるため，「計算完備」であるとは言い難い．

5. OODBMS＝OOPL＋RDBMS 派

この派の特徴は既存の OOPL と RDBMS の簡潔な結合である（ネットワーク型 DBMS はそれ自身すでに OODBMS であるという説[21]がある）．たまたま筆者の目にとまったシステムは 'An Object-Oriented Relational Database'[12] である．このシステムはエンジニアリング DB 応用のために開発された．RDBMS はデータの永続化，競合管理，トランザクションとプログラミング言語インタフェースを提供する．RDBMS の問い合わせ機能は重要でない．なぜなら複雑なデータへのアクセスは OOPL フロントエンドが行うからである（すなわち，巡航アクセスだけでよい）．

OOPL のクラスは RDBMS のテーブルに写像される．オブジェクトはテーブルの行に対応付けられる．ID が生成され，それらがオブジェクトの主キーとして利用される．ある種の関連 (relationship) はまたテーブルで実現される．

図-9 はこのシステムのコンパイル時アーキテクチャである．**図-10** が実行時システム構成である．

アーキテクチャの鍵はデータベースチェックアウトとチェックインである．DB の一部にロックし，それを RAM に転送する．OOPL で書かれた応用プログラムは RAM 上に展開されたデータを操作する．操作が終了したらそれを DB に戻す（チェックイン）．この期間は一般に長い．

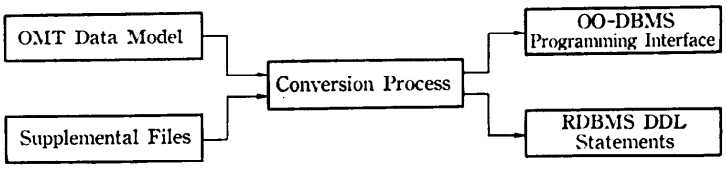

図-9 オブジェクト指向関係データベースシステムのコンパイル時フロー（文献 12）より）

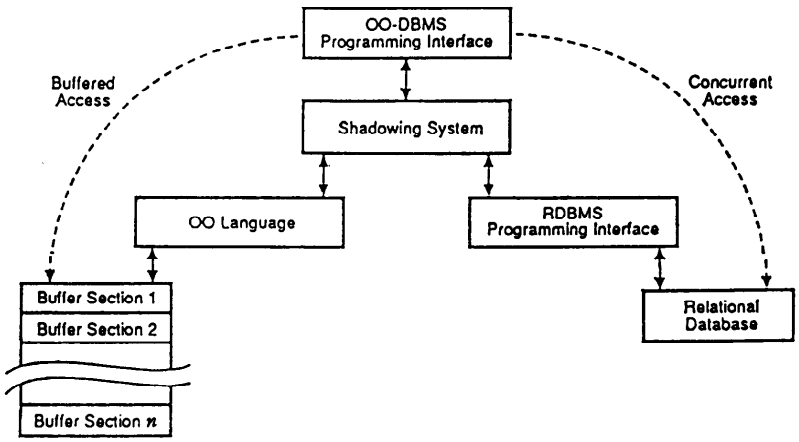

図-10 オブジェクト指向関係データベースシステムの実行時アーキテクチャ（文献 12）より）

6. 独自 DBPL—O_2[13] と Jasmine[14]—

ここでは O_2 と Jasmine とを取り上げる．二者ともに「計算完備」と非手続き的問い合わせ機能の両者を重要視している点で OOPL 拡張派とも RDB 拡張派とも趣を異にする．もちろん，'OODBMS＝OOPL＋RDBMS' 派とも異なる．独自派には OOPL 派と違い，基盤となる OOPL は存在しない．独自のオブジェクト指向 DBPL を提案している．O_2 の DBPL の一つは CO_2 で，Jasmine のそれは Jasmine/C である．共にCを基盤にしているところは同じである．

6.1 O_2

O_2 が提供する型構成子はタップル，セット，及びリストである．これらと O_2 基本タイプとを使って複雑なタイプを定義する．O_2 のオブジェクトは識別子と値との対で定義される．O_2 ではオブジェクトや値に名前をつけることができる．これらの名前はいわばグローバル変数名であり，それらには値やオブジェクトを代入できる．永続オブジェクトは(1)名前付けられているか，(2)ほかの永続オブジェクトや値の部分であるかである．

O_2 の DBPL は CO_2 と $BasicO_2$ である．それぞれCと Basic を基盤とした言語で，O_2 の永続データを操作できる．CO_2 はセットやリストを操作するための繰り返し機構を備えている．

O_2 のユーザがアドホックな問い合わせをするのに CO_2 のサブセットである SQL ふう問い合わせ言語を使う．

O_2 システムは三層からなる．スキーマ管理(SM)，オブジェクト管理 (OM) と Wisconsin Storage System (WiSS)[22] である．スキーマ管理はクラスやメソッドやグローバルな名前などの創生・検索・更新・削除を行う．オブジェクト管理は複合オブジェクトや複合値とメッセージの受渡しを扱う．WiSS はレコード構造の順ファイル，非構造化ファイル，長いデータの三種類の永続データ編成法を提供する．WiSS は OS のファイルシステム外に作られ，独自のバッファリング機能を有する．O_2 はサーバ・クライアント構成で動く．クライアント側では応用プログラム，SM と OM とが一つのプロセスを構成する．サーバ側ではその鏡映プロセスが動く．それは SM, OM と WiSS から構成される．WiSS によって管理されるロックテーブルとバッファはすべてのプロセスにより共有される．OM のコードと OM のオブジェクトメモリもまた共用である．

O_2 のオブジェクトは一つの WiSS レコードに格納され，そのOIDはレコード識別子 (RID) である．この RID はレコードを直接アクセスする番地であり，レコードの移動により変化する．このため O_2 では OID の不変性を保証するために System R などで使われた「回送方式」を利用する．図-11 にその例を図示する．オブジェクトは通常クライアント側で創生される．創生されたオブジェクトには臨時の OID が割り当てられる．コミット時に，そのオブジェクトを格納するレコードがサーバ側に作られ，その RID が当該オブジェクトの正式の OID となる．

タップルはページ内の一つのレコードとして実現される．ページを越えるタップルは長データ項目として異なるページ中のタップルの集まりとして実現される．リストは B-tree 様の順序木で実現される．セットは要素が OID であるオブジェクトである．大きなサイズのセットを表現するのに B-tree 索引を用いる．

O_2 のバッファは ORION[16] と同様二重バッファ方式を採用している．ページバッファは WiSS のそれを使い，オブジェクトバッファが O_2 のバッファである．ページバッファ中のオブジェクトはディスク形式であり，オブジェクトメモリ中のそれはメモリ形式である．したがって，O_2 のバッファ管理は OID から主記憶番地への変換を司る．オブジェクトが主記憶中にあるときは，それを OID でアクセス可能にするため，オブジェクトテーブルが必要になる．このテーブルは OID のハッシュテーブルである．

6.2 Jasmine

Jasmine はオブジェクト形式がフレーム[23]であることがほかの OODB と異なる点である．この

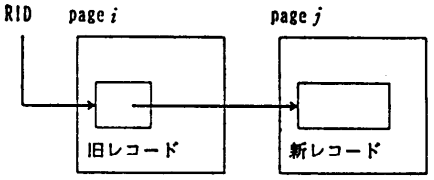

図-11 回送方式

結果，Jasmine ではデーモン機能で DB 一貫性を保持するのが容易にできる．

Jasmine の DBPL は Jasmine/C である．この言語はCプログラミング言語に Jasmine DB 中の永続オブジェクトを定義・操作する機能を追加した拡張Cである．Jasmine/C が扱うオブジェクトの属性には静的属性と手続き（動的）属性とがあり，後者がいわゆるメソッドに当たる．このメソッドを書くのに Jasmine/C を使う．また，応用プログラムもこの言語で書く．

Jasmine/C の特徴は単一オブジェクト変数のほかに集合オブジェクト変数を許し，それに条件検索の結果を代入できることである．

```
SUPPLIER fujis multiple;
fujis=SUPPLIER where SUPPLIER. Sname
                        =="Fuji";
```

で複数のオブジェクト（の OID）が fujis に代入される．この変数を使ってさらに検索を進めることができる．

```
baseparts=fujis. Supply where fujis.
                    Supply. Cost>10;
```

個々の BASEPART インスタンスの属性を表示する動的属性 'display ( )' が定義されているとすると 'baseparts. display ( );' は 'For all b in baseparts b. display ( )' の意味である．動的属性は静的属性とまったく同じに使える．

```
fujis. Supply. display ( ) where fujis.
                    Supply. Cost>10;
```

あるいは条件節の中で

```
COMPOSITEPART compositeparts multiple;
compositeparts=COMPOSITEPART where
              COMPOSITEPART. Cost ( )
                        >50;
```

Jasmine のアーキテクチャを**図-12** に示す．XDE は拡張関係 DB エンジンである．NF^2 テーブルをサポートしていること，結合演算にハッシュ結合方式も採用していること（したがって高速結合が可能），さらにバッファサイズが 4K から 64K まで可変であることに特徴がある．

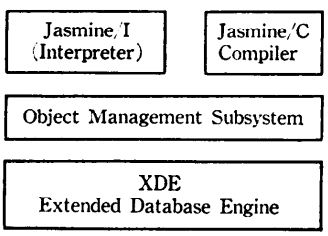

図-12 Jasmine のシステム構成

下層に関係 DB エンジンがあることは IRIS と同じである．IRIS と同様に，オブジェクト検索式は拡張関係演算式に変換される．ただし，この変換はコンパイル時に行われるので実行時オーバヘッドは小さい．クラスは XDE テーブルに，インスタンスはそのタップルに対応させる．多値は NF^2 で実現されるから高速アクセスが保証される．

OID はテーブルの一つのフィールド値として実現される．各テーブルの OID フィールドには索引が張られアクセスの高速化が図られる．ディスク上にあるオブジェクトのアクセスは OID または TID（タップル識別子）によるアクセスが可能である．TID によるほうが OID によるより早い．しかし，いつでも TID が利用可能というわけにはいかない．したがって，Jasmine では「可能なときには利用する」最適化を行う．一方，主記憶中にロードされたオブジェクトはその番地で直接アクセスと高速巡航が可能になる．当初，Jasmine は ORION，O_2 などと同様に二重バッファ方式をとっていた．その場合，主記憶中のオブジェクトに対する拡張関係演算が適用できなくなる．そこでわれわれは上記欠点を改良すべく（1）主記憶 DB（拡張関係演算適用可能），（2）ページバッファ中のタップル（オブジェクト）への番地による直接アクセスを考案した．

7. おわりに

本稿で紹介したシステムが OODBMS を代表するということで紹介したわけではない．たまたま筆者がそれらの論文を目にして，内容を知りえたに過ぎない．また，本稿で用いた OODBMS の分類法が世に流通しているわけでもない．しかし，本稿を通読された読者諸兄姉は独立に設計されたシステムのアーキテクチャに共通点が結構あることにお気付きかと思う．筆者もこの稿を書きながらこの点に興味をいだいた．

参 考 文 献

1) Gardarin, G.: Towards the Fifth Generation of Data Management System, in New Application of Data Bases, Academic Press, London (1984).
2) Copeland, G. and Mayer, D.: Making Smalltalk a Database System, Proc. ACM SIGMOD Int. Conf. on the Management of Data (1984).
3) Krasne, G.: The Smalltalk-80 Virtual Machine, BTYTE, pp. 117-124 (Aug. 1981).
4) Mayer, D., Stein, J., Otis, A. and Purdy, A.: Development of an Object-Oriented DBMS, OOPSLA '86, pp. 472-482 (Sep. 1986).
5) Cattel, R. G. G. and Skeen, J.: Engineering Database Benchmark, Technical Report, Sun Microsystems (Apr. 1990).
6) Kim, W., Garza, J. F., Ballow, N. and Woelk, D.: Architecture of the ORION Next-Generation Database System, IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 1, pp. 109-124 (Mar. 1990).
7) Stonebraker, M., Rowe, L. A. and Hirohama, M.: The Implementation of POSTGRES, IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 1, pp. 125-142 (Mar. 1990).
8) Wilkinson, K., Lyngoboek and Hasan, W.: The Iris Architecture and Implementation, IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 1, No. 2, pp. 63-75 (Mar. 1990).
9) Atkinson, M. et al.: The Object-Oriented Database System Manifesto, Proc. of the First International Conf. on DOOD, pp. 40-57 (Dec. 1989).
10) The Committee for Advanced DBMS Function, Third-Generation Data Base System Manifesto, Memorandum No. UCB/ERL M 90/28 (Apr. 1990).
11) Blasgen, M. W. and Eswaran, K. P.: Storage and Access in Relation Databases, IBM Syst. J., Vol. 16, No. 4, pp. 363-377 (1977).
12) Premeriani, W. T., Blaha, M. R., Rumbaugh, J. E. and Varwig, T. A.: An Object-Oriented Relational Database, C. ACM, Vol. 33, No. 11, pp. 99-109 (Nov. 1990).
13) Deux, O. et al.: The Story Of O_2, IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 1, pp. 91-108 (Mar. 1990).
14) Aoshima, M, Izumida, Y., Makinouchi, A., Suzuki, F. and Yamane, Y.: The C-based Database Programming Language Jasmine/C, Proc. 16th Intl. Conf. on VLDB, pp. 539-511 (Aug. 1990).
15) Agrawal, R. and Gehani, N. H.: ODE (Object Database and Environment) : The Language and the Data Model, Proc. of ACM SIGMOD Intl. Conf. on the Management of Data, pp. 36-45 (1989).
16) Nguyen, T. A., Wagner, M. and Hoffman, B.: PC++: An Object-Oriented Database System for C++ Applications, Proc. of the 2nd Intl. Symp. on DASFAA pp. 109-115 (Apr. 1991).
17) Stonebraker, M.: The Design of the POSTGRES Storage System, Proc. of 1987 VLDB Conf., pp. 289-300 (Sep. 1987).
18) Rowe, L. and Stonebraker, M.: The Postgres Data Model, Proc. of 1987 VLDB Conf. pp. 83-96 (Sep. 1987).
19) Fishman, D. H. et al.: Iris : An Object-Oriented Database Management System, ACM Trans. Office Information Syst., Vol. 5, No. 1, pp. 48-69 (Jan. 1987).
20) Shipman, D.: The Functional Data Model and the Data Language DAPLEX, ACM Trans. Database Syst., Vol. 6, No. 1, pp. 140-173 (Mar. 1981).
21) Ullman, J. D.: Principle of Database and Knowledge-Base Systems, Computer Science Press (1988).
22) Chou, H.-T. et al.: Design and Implementation of the Wisconsin Storage System, Software-Practice and Experience, Vol. 15, No. 10 (Oct. 1985).
23) Minsky, M.: A Framework for Representing Knowledge, In The Psychology of Computer Vision, McGraw-Hill, Editted by Winstone, P. H. (1975).
24) 1990年 ACM SIMID Conf. におけるパネル.

(平成3年1月22日受付)

牧之内顕文 (正会員)

1967年京都大学工学部電子工学科卒業. 1970年グルノーブル大理学学部応用数学科 Docteur-Ingènieur 取得, 同年富士通(株)入社. 以後, コンパイラ-コンパイラ, データベース, 知識ベース, 自然言語インタフェースの研究開発に従事, 京都大学工学博士. (株)富士通研究所を経て現在九州大学工学部教授 (情報工学科). 電子情報通信学会, ACM, IEEE Computer Society, 人工知能学会各会員.