データベースシステム 99－1
（１９９４．７．２０）

高速性と言語透過性を重視したオブジェクト指向データベース

川村 敏和　脇園 竜次　土屋 武彦　田中 立二
株式会社東芝　重電技術研究所

プラント監視制御システムのようなリアルタイム分野への応用を考慮した、高速で使い易いオブジェクト指向データベース（ＯＯＤＢ）を開発した。このような分野では性能のバランスとともに、既存ソフトウェア資産の再利用性が重視される。そのためＯＯＤＢのナビゲーション検索の高速性と応用プログラムの生産性の良さという特徴を最大に生かすよう、機能を絞り込みコンパクトで高性能なものを目標とした。また既存ＯＯＤＢでは別々に扱われていた外延と集合の操作性の統合を行うことにより、シンプルで強力なオブジェクト操作機能を実現した。本稿ではシステムの基本的な仕様とアーキテクチャについて述べる。

Object-Oriented Database with high performance and transparency

Toshikazu Kawamura, Ryuji Wakizono, Takehiko Tsuchiya, Tatsuzi Tanaka
Heavy Apparatus Engineering Lab., Toshiba Corp.

1.TOSHIBA-CHO, FUCHU-SHI, TOKYO, 183 JAPAN

We have developed an object-oriented database management system(OODBMS) for real-time system like plant control systems. For real-time system, high reusability of application programs is important as well as high performance of data management. Our target is developing a compact and high performance OODBMS with high transparency to programming language and high productivity of program. We achieved the integration of 'extent' and 'collection' concept, so that the system provides powerful object manipulation facilities and operations with simple notation.

This paper describes basic specifications and architecture of the OODBMS.

1 はじめに

プラント監視制御システムや系統監視システムのグラフィカル・ユーザ・インタフェース(GUI)などのソフトウェア開発では、データ管理にファイルシステムが多く使用されている。その理由は、標準的で汎用的な格納手段であることと、処理速度の速さや使い方の簡便さなどである。しかしソフトウェアの高度化によって扱うデータ構造が複雑化するにつれて、データ入出力処理の生産性の悪さ、およびデータ構造の変更に伴う保守コストの高さなどの欠点が無視できなくなってきている。関係データベースの利用も行われているが、下記のような問題があるため広く使われるには至っていない。

- 処理速度の遅さ。速度を優先して結合演算を減らすとデータの汎用性が犠牲になる
- プログラミング言語とのインピーダンス・ミスマッチ
- 複雑なデータ構造をそのまま扱えない
- 導入コストや実行コスト（メモリやディスクの消費、CPU負荷）の高さ

これに対してオブジェクト指向データベース（OODB)は、複雑なデータ構造を直接扱えるモデリング能力の高さ、ナビゲーションによるデータ操作の高速性、オブジェクト指向プログラミング言語（OOPL）との言語透過性の高さなど、上記のような分野における標準的なデータベースとなりうる優れた機能を備えている。そこでリアルタイム分野への応用を目的に、高速で言語透過性に優れたオブジェクト指向データベースの試作を行った。

以下2章で開発の背景、3章で機能仕様、4章で実現方式について述べる。

2 開発の背景

2.1 現状の問題点

我々は当初OODB応用の立場から研究を開始した。しかし一般に指摘されているOODBの技術的な問題点、例えばDBMSとしての信頼性の低さ、C++言語に依存したデータベース・モデルが持つスキーマ進化に対する脆弱性や継承機能の弱さ[8]などの他に次のような問題が出てきた。

(1) C++の構文やクラス定義の型に制約があるなど言語透過性が十分でない。そのため既存アプリケーションにOODBを適用する際の修正作業が煩雑になる。例えばスキーマに記述できる型に制限がある場合には、アプリケーションのデータ構造を変更しなければならない。あるいは基本型のデータをそのまま永続化できない場合には、永続性を持つ小さなクラスをたくさん追加しなければならない、などの問題がある。既存プログラムには構造化が不十分なものが少なくないため、後者は作業効率に大きく影響する。

(2) 外延と集合が統合されていない[1]。例えば、外延の機能がなくユーザが集合機能を使って管理しなければならない、外延の機能はあるものの集合に関する操作に比較して外延に関する操作が貧弱(外延の検索結果が集合にならないなど)、集合の機能がDBMSと別体系になっていて使いにくい、などの問題がある。

(3) 性能に関する客観的な評価基準がない。カタログ上の機能が実際のアプリケーションの速度にどう影響するか見えない。またOODBのカスタマイズや性能のチューニングが容易でない。

2.2 要件

以上の点を考慮して、開発するOODBの要件を次のようにまとめた。

(1) OODB必須機能の提供

オブジェクト指向データベースに必要とされる必須機能[6]を提供する。多機能よりもコンパクトさを重視し高性能化を図る。

(2)分散データベース

クライアント／サーバ型の分散データベースの構成とする。

(3)高速性

オンライン・リアルタイム・システムへの適用を可能にするため、主記憶上での永続オブジェクト操作と一時オブジェクト操作において同等の速度が得られるようにする。またサーバ／クライアント間のデータ転送量および転送回数を最小限に押さえることにより高速化を行う。なお一定時間以内の応答を保証する必要があるハード・リアルタイム分野への対応は今後の検討課題とする。

(4)言語透過性

データベース言語はC++と完全に透過な仕様とする。C++の構文拡張を行わず、データベースとのインタフェースはクラス・ライブラリと多重演算子として提供する。スキーマ定義における型の制限を最小限にし、ポインタ型も直接記述できるようにする。

(5)永続性

永続性はC++の記憶域クラスの拡張とし、それに対応したオブジェクトの定義形式を新たに導入する。永続性が指定されたデータは永続領域（仮想記憶空間）に割り付け、コミット時に2次記憶領域に保存する。プログラムからの参照により、データが永続領域になければ2次記憶領域から仮想記憶空間に読み込む。C++基本型に関してはシステムでスキーマを定義することで、変数に対しても永続化を可能とする。これにより永続／一時オブジェクトの操作をプログラム上で同じ形式で記述できるようにする。

(6)ポインタによる関連付け

永続オブジェクト間の関係付け（参照）をポインタで行う。これにより永続／一時オブジェクトの参照効率を同等にする。仮想記憶空間と2次記憶領域のオブジェクト移動に伴うアドレス変換は、データベース管理システムが自動的に行う。

(7)外延

すべての永続オブジェクトをクラスごとのインスタンス集合としてシステムで管理する。永続オブジェクト生成時には、永続領域に割り付けたオブジェクトを要素としてクラスのインスタンス集合に追加する。

(8)集合

ユーザ定義の集合のために集合クラスを提供する。集合クラスとして要素の重複を許すBag と、要素の重複を許さないSet を提供する。これらの集合は要素として他の集合を持てる。また集合オブジェクトごとに永続／一時の指定ができる。

(9)外延と集合の操作の統一

外延と集合の操作性を統合する。外延をシステム定義の集合(Set)と位置付け、外延に対しても集合と同じ演算・操作体系を提供する。ユーザから見た外延と集合の違いは、生成／追加と削除インタフェースの違いのみとする。

ＯＯＤＢの開発には2つのアプローチがある。1つは言語主導であり、ベースとなるオブジェクト指向プログラミング言語に、永続性機構、トランザクション，障害回復などの機能を加えていく考え方である。言語依存性が欠点になるが、言語透過性の高さによるプログラミングの容易さは大きな長所である。また機能を絞りこみやすく、性能を得やすい。もうひとつはデータベース主導であり、従来のＤＢＭＳの機能にオブジェクト指向の性質を加えていく考え方である。言語独立のデータモデルを設計できるため、伝統的なＤＢＭＳに求められる機能を実現しやすい。

我々は上述の要件を満たす上で適している前者のアプローチをとった。

3 機能仕様

3.1 言語透過性を実現するための仕様

２章の要件に従ってデータベース言語はＣ++と完全に透過な仕様とする。すべてのデータベース機能は、クラス・ライブラリおよびそのメンバ関数と多重定義された演算子として提供する。プリプロセッサによる言語変換を行わないため、既存の言語開発環境をそのまま利用できる。

- オブジェクト定義：Ｃ++のクラス宣言をスキーマ定義として利用する。
- オブジェクト操作：データベース管理システムの提供するクラスライブラリを利用して、オブジェクトを操作する。永続オブジェクトの生成・削除は多重定義されたnew(), delete()演算子で行い、参照・更新はＣ++言語の構文に従う。

データベースとアプリケーションのインタフェースとして次のクラスを提供する。データベース管理に関してはDatabaseクラス、スキーマ操作に関してはTypeクラスを提供する。さらにクラスタリング（セグメント）管理にSegmentクラス、トランザクション管理にTransactionクラス、集合に関してはCollection, Set および Bagクラスを提供する。これらのクラスはそれぞれの機能を利用するためのメンバ関数を提供する。図１にそれらの概要を示す。

永続オブジェクトを生成するためには、まずデータベースをオープンしてアクセス権を得る。次に指定クラスへのアクセス権を得る。取得にはDatabaseクラスの type() メンバ関数を使用する。永続オブジェクトの生成と削除は、Ｃ++言語のオブジェクト生成／削除演算子を拡張した構文で提供する。

例)

```
Database* db = Database::open("Politician.DB");              // データベースをオープン
Trsansaction::start();                                       // トランザクション開始
Type* politicianType = db->type("Politician");               // Politicianクラスを取得
politicianType* hosokawa = new(politicianType) Politician("Hosokawa",50,MAN);
                    :                                        // 永続オブジェクトを生成
```

集合を利用するには、まず集合オブジェクトを生成する。生成時に永続／一時を指定する。集合へのオブジェクト挿入はinsert()メンバ関数を用いる。引数には挿入するオブジェクトのポインタを渡す。

例)

```
Type* setType = db->type("Set");                             // Setクラスを取得
Set* party = new(setType) Set();                             // 永続集合を生成
patry->insert(hosokawa);                                     // 集合にオブジェクトを追加
        :
Trsansaction::commit();                                      // コミット
db->close();                                                 // データベースをクローズ
```

- 集合操作機能

要素の重複に関してBagとSetの２つのモデルを提供する。要素の順序に関しては、順序概念を持たない集合の比較と包含演算、データ並びの順序概念を持つカーソルによるデータ操作、データ大小関係による順序概念を持つ索引操作の機能を提供する。主な機能とその記述形式を図２に示す。

- 集合のデータ構造

集合はカーソルと索引集合とコンテナから構成する。コンテナは集合要素の管理を行うもので、要素として集合オブジェクトも許す。カーソルは集合要素の逐次データ操作機能を提供する。索引は検索を高速化するための機能であり、要素となるオブジェクトの属性ごとに作成できる。デフォルトでは集合オブジェクトは索引を持たないが、索引を作成することで検索を効率よく行える。

データベース定義・操作機能	記述形式	記述例

(1)オブジェクト定義	C＋＋言語のクラス定義
	データ構造以外に以下の定義が可能
	・データ初期化（構成子）
	・データ操作関数
		class Person {
		char* name; //名前
		int age; //年齢
		Person* spouse; //配偶者
		Person(char* aname, int anage); //初期化（構成子）
		};

(2)データベース操作		
データベース作成	Databaseクラスの静的関数	Database::create("ﾃﾞｰﾀﾍﾞｰｽ名");
データベース削除	同上	Database::purge("ﾃﾞｰﾀﾍﾞｰｽ名");
データベース処理開始	Databaseクラスの静的関数	Database* db = Database::open("ﾃﾞｰﾀﾍﾞｰｽ名");
データベース処理終了	Databaseクラスの関数	db->close();
トランザクション処理開始	Transactionクラスの静的関数	Transaction::start();
トランザクション処理終了	同上	Transaction::commit();
トランザクション処理廃棄	同上	Transaction::abort();
(3)スキーマ操作		
スキーマ作成	スキーマ作成・変更ツール	データ構造定義をデータベースに登録
スキーマ参照	Databaseクラスの関数	Type* person = db->type("Person");
(4)オブジェクト操作		
オブジェクト作成	C＋＋言語のnew演算子の多重定義	Person* taro = new(person) Person("TARO",25);
オブジェクト削除	C＋＋言語のdelete演算子の多重定義	delete jiro;
オブジェクト検索	C＋＋言語の[]演算子の多重定義	Person* saburo = (Person*)(*person)["name == ","SABURO"];

オブジェクト参照
オブジェクト更新
	C＋＋言語のデータ参照
	C＋＋言語のデータ代入
		if(taro->spouse == NULL) {
		taro->spouse = hanako; taro->spouse->age = 23;
		}

図１　データベース言語の概要

コレクション宣言・操作機能	記述例　コレクションクラスの関数・演算子
宣言	Collection teenAgers("Person"), c_object2, c_object3; Collection *c_object;

コレクション
オブジェクト作成
	Type* collection = db->type("Collection");
	c_object = new(collection) Collection("Person");
	c_object = new Collection("Person");

要素追加	c_object->insert(person1);
要素削除	c_object->remove();
代入演算	c_object2 = *c_object;
集合演算	c_object3 = c_object2 + *cobject; 集合和(+)、集合積(*)、集合差(-)、部分集合(>,<,>=,<=)、等値比較(==,!=)
索引作成	c_object->createIndex("name"); c_object->setPrimaryKey("name");
索引削除	c_object->deleteIndex("name");
索引更新	c_object->update();
検索	c_object2 = c_object->find("name ==", "SABURO"); Person* saburo = (Person*)c_object->find1("name ==", "SABURO"); Person* saburo = (Person*)c_objectj["SABURO"];
カーソル	Person* p = (Person*)c_object();

要素の逐次取り出し	for (c_object->first(); c_object->more(); (*c_object)++){
	p = (Person*)(*c_object)();
	if (p->age<12 && p->age<20) teenAgers.insert(p);
	}

複数カーソル	c_object3 %= *c_object1;

図２　集合操作機能の概要

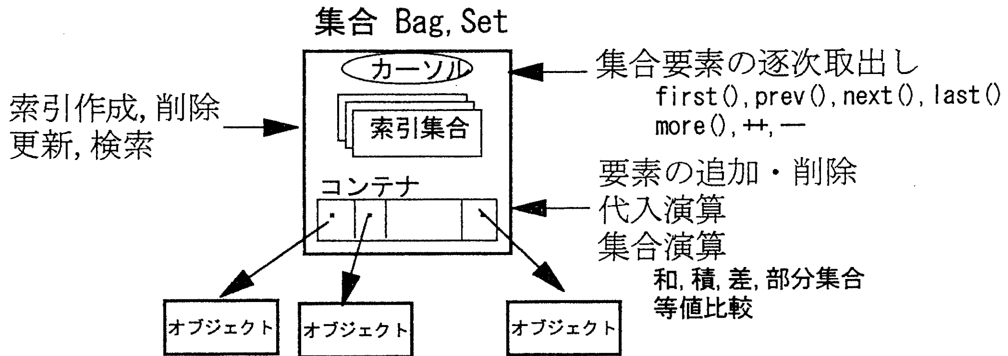

図3 集合のデータ構造の概念

- 集合の永続性

集合オブジェクトごとに永続／一時の指定ができる。ただし永続集合の要素に一時オブジェクトを持つことはできない。

3.2 クラスタリング

デフォルトでは、ユーザ・オブジェクトは生成順にデータベースに格納する。ＣＡＤ図面データやＣＡＳＥ文書などのような多クラス・小オブジェクトの場合にはページの無駄を少なくでき、関連データが局所的に配置されるため参照効率がよい。クラスタリングを指定すると、オブジェクトはクラスごとのセグメントに分けて格納される。クラス数が少なく、特定クラスのオブジェクトにアクセスが集中する場合は効率的である。

4 実現方式

4.1 システム構成

システムはクライアント／サーバに基づいた分散処理システムの構成である（図４）。データベースサーバは、複数ユーザによる永続オブジェクトへのアクセス、障害回復、２次記憶の管理を集中して担当する。ただし前述したように、サーバの管理単位は複数オブジェクトを含むページであり、オブジェクト単位の管理は行わない。個々のオブジェクトの管理はクライアント・プロセスが担当する。オブジェクトの参照や更新など負荷が大きい部分をクライアント側で処理することで、負荷のサーバへの集中を防ぐ。

4.2 高速性の実現方式

サーバ／クライアント間のデータ転送を高速化するために次の方式を採用する。

- ページ転送制御

ネットワークの通信回数を少なくするために、サーバとクライアント間のデータ転送は論理ページ単位で行う。論理ページは物理ページサイズの任意の整数倍であり、論理ページのサイズ指定はクラスタ(セグメント)単位で指定できる。用途に応じて論理ページサイズを指定することにより、効率的な転送が可能になる。

- ローカル・キャッシュ

ネットワークの通信量の削減のために、一度参照したオブジェクトは、アプリケーション(クライアント)プロセスごとに確保したローカル・キャッシュに保持する。ローカル・キャッシュは障害回復、トランザクション制御にも利用する。

- データベースサイズの削減

データベース化に伴うデータ全体のボリューム増加を最少にした。データベース化されたデータのボリュームはファイル管理の場合と比較して２～４倍にもなることがある。元のデータ量を１０ＭＢとすると、ＯＯＤＢの転送する最大データ量は２０～４０ＭＢにもなり、サイズの増加を押さえることは、データ転送量の削減だけでなく主記憶のスワップ回数の削減にもなる。

また永続オブジェクトの管理を効率化するため次の方式を採用する。

- ポインタ書換え

永続オブジェクトを２次記憶領域から仮想記憶空間への読み込むタイミングの判定、およびオブジェクトの再配置に伴うポインタ書換えは Swizzling at Page Fault Time方式[3]を用いた。

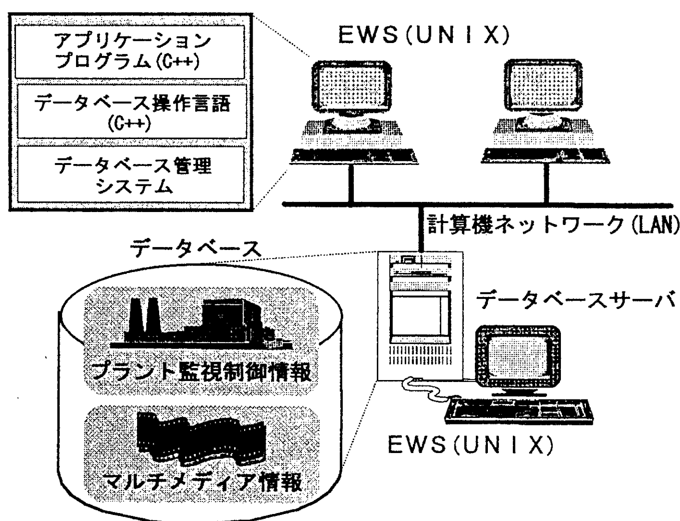

図４　システム構成

5　あとがき

機能を絞りこみコンパクトで高速なオブジェクト指向データベース管理システムの試作を行った。今回の試作はＯＯＤＢの課題の解決というよりも、現状の技術で得られる性能や操作性、拡張性(カスタマイズ性)など、応用上問題となる点の技術的ポイントの把握に主眼を置いたものである。現在は基本機能の開発を終え、アプリケーションへの性能評価と適用実験を進めている。

参考文献

[1] 脇園竜次, 土屋武彦, 川村敏和, 田中立二：オブジェクト指向データベースの開発 −機能仕様−, 情報処理学会第48回全国大会, 1994/3.

[2] 土屋武彦, 脇園竜次, 川村敏和, 田中立二：オブジェクト指向データベースの開発 −システム構成と実現方式−, 情報処理学会第48回全国大会、1994/3.

[3] 鈴木慎司、喜連川優、高木幹夫：永続的プログラミング言語Ｐ３Ｌの実装方式,情報処理学会データベース研究会資料 89-9,1992.

[4] W.Kim: Introduction to Object-Oriented Databases,MIT Press, 1990.

[5] M.A.Ellis, B.Stroustrup: The Annotated C++ Reference Manual, Addison-Wesly, 1992.

[6] M.Atkinson, etal.: The Object-Oriented Database system Manifesto,Proc. of DOOD89, pp.40-57,1989.

[7]Object Database Management Group: The Object Database Standard ODMG-93, Version 0.9:revision of 7/1/93,1993.

[8]第3回Obaseシンポジウム資料,1994.