データベースシステム 93−11
システムソフトウェアと
オペレーティング・システム 59−11
（1993. 5. 28）

関係データベースを利用したオブジェクトサーバのデータ管理

森本 康彦, 福田 剛志, 何 千山, 小坂 一也

日本 アイ・ビー・エム 株式会社 東京基礎研究所

あらまし マルチメディア・アプリケーションの開発および実行を支援するためのオブジェクト指向環境 COSMOS と、そのデータ管理機構について述べる。マルチメディアを扱うシステムには、従来データベースに必要とされる各種データ管理機構に加え、新たに出現するメディアに容易に対応できる拡張性や柔軟性が求められる。そこで、本システムではオブジェクト指向データベースでなく関係データベースをデータベースエンジンとして採用し、そのデータ管理機能の有効性を生かしたオブジェクトサーバの実現を行なっている。

Data Management in an Object Server based on a Relational Database

Yasuhiko Morimoto, Takeshi Fukuda, Qianshan He, Kazuya Kosaka

Tokyo Research Laboratory, IBM Japan Ltd.

Abstract Data management facilities of COSMOS is presented in this paper. COSMOS, which is an object-oriented environment with C++ object server for development and execution of multimedia applications, uses relational databases to store persistent objects. Therefore, it can utilize various database management facilities of relational databases in an object-oriented environment without sacrificing extensibility and flexibility. COSMOS translates every persistent objects and their manipulations into simple tuples and SQL statements.

1 はじめに

我々はマルチメディア・アプリケーションの開発および実行を支援するための環境 COSMOS (COmmon Service for Multimedia ObjectS) を構築している。この論文では COSMOS のオブジェクトサーバ機能、データベース機能を中心にオブジェクトのデータ変換、データ管理、質問検索等について述べる。

より高機能なアプリケーションが求められるに従い、データベースに保持すべきデータもより複雑な構造を持つ必要性がでてきた。近年、オブジェクト指向モデルに従って、その操作法とともに構造化されたデータ（オブジェクト）は、複雑で機能的な実体を簡単に管理することが可能なためひろく使われている。また、ネットワークで結合された異機種間でマルチメディアデータを共有することを考えた時、新たに出現するメディアに容易に対応できる柔軟性と強力な質問機能を備えたサーバが必要となる。このため我々は関係データベース (RDBMS) の (1) 成熟した技術に基づく頑健性、(2) 普及度、(3) 質問機能、(4) 柔軟性、(5) 既存のデータベースの活用 などに着目し、RDBMS の上にマルチメディアオブジェクト指向環境COSMOS[7] のオブジェクトサーバを構築している。

本稿では、2章で COSMOS 全体の構成、3章で COSMOS システムおよびアプリケーションが使用する各種のデータモデルの解説、4章ではクラス定義、オブジェクトの検索等のランタイムオペレーションの振舞い、5章ではマルチサーバの管理機能について述べる。

2 COSMOS システム構成

COSMOS 全体の構成を図1に示す。図のように COSMOS を構成するコンポーネントはクライアント、サーバーそれぞれに存在する。

COSMOS のオブジェクトサーバを構成するコンポーネントとしては、クライアント側に、(1) C++ Interface 、(2) Client Object Manager 、(3) Client Communication Manager の3つのコンポーネントがある。サーバへのリクエストを制御する Client Object Manager は Communication Manager を通じて必要なサーバへアクセスする。マルチサーバの制御も、このコンポーネントと各サーバが連携して行なっている。アプリケーションプログラムは C++ Interface を通じて Client Object Manager のプリミティブ関数をコールし、COSMOS の様々な機能を利用する。また、各種メディアを有効に利用するための (0) マルチメディアライブラリ [4, 6] も用意されており、アプリケーションプログラムはこの C++ Interface を通じてライブラリを使うことができる。クライアント、サーバの各 Communication Manager はそれぞれのサイトにあるオブジェクトキャッシュを利用して、適切なオブジェクト転送を制御する。ＯＳの違いによって影響するコードの変換もここで行なわれる。

一方、サーバ側には (4) Server Communication Manager 、(5) Server Object Manager 、(6) DB Interface の3つのコンポーネントがあり、クライアント側からのリクエストに応じて、必要なオペレーションを実行したり、必要なオブジェクトを Communication Manager を通じて授受したりする。Server Object Manager は各 Client Object Manager からのリクエストを受けつけ、その内容に応じて DB Interface の提供する API をコールし、必要なデータ操作を行ない、さらに Communication Manager にオブジェクトの転送をさせたり、キャッシュをアクセスしたりする。さらに Client Object Manager と連携してオブジェクトの Migration 、Replication 機能を始めとした種々のマルチサーバ機能および分散データベース機能の制御を行なう。DB Interface はコールされた API とその内容に応じて、リクエストを具体的なデータベース操作に変換しそれを実行する。

C++ Interface アプリケーションが COSMOS オブジェクトサーバを利用する時のインターフェースで、オブジェクトサーバ内のオブジェクトとアプリケーションでの C++ オブジェクトとの変換を行なったり、アプリケーション内のオブジェクトの操作を Client Object Manager への命令に変換したりする。

Client Object Manager 分散トランザクション、マルチサーバのコントロールを行なう。分散トランザクションでは2相コミットのプロトコルが使われ、関連するトランザクションすべてコミット可ならコミットする。どのサーバに接続するか等のマルチサーバコントロールもこのレイヤーで処理される。

Client-Server Communication Manager オブジェクトの転送とキャッシュのコントロールを行なう。COSMOS では TCP/IP をベースに RPC による（オブジェクト単位の）オブジェクト転送をしている。また、ディスクアクセスと転送の回数を減らすため、Client 、Server 双方にオブジェクトがキャッシュがされこのレイヤーでその管理をしている。

Server Object Manager オブジェクトのレプリケーション、マイグレーション等分散オブジェクトの管理と自身の物理サーバの管理を行なう。レプリカを持っている場合の代理応答、レプリカ間の一貫性管理をこのレイヤーで処理しマルチサーバの効率をあげている。

DB Interface Object Manager からの命令を実際の関係データベースの命令に翻訳し実行する。ここで用意される API は各データベース間の違いを吸収する抽象的な関数となっているので、Object Manager はデータベースの違いを意識することなくプログラムできたり命令を実行できたりする。

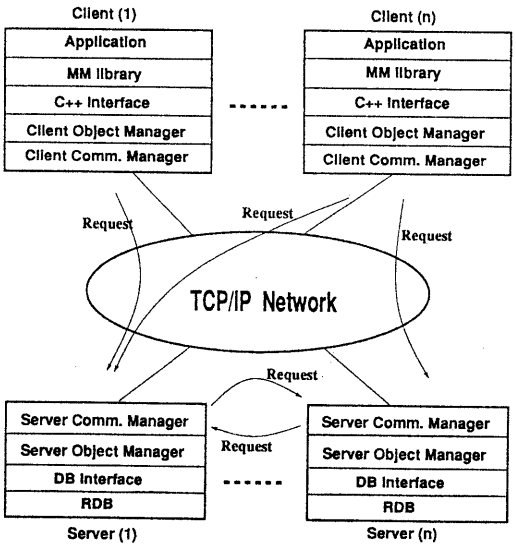

図 1: COSMOS のシステム構成

3 COSMOS データモデル

COSMOS では C++ 上で作られたオブジェクトを関係データベースに保存することができる。さらには関係データベース上に元からあるデータも C++ のオブジェクトとして利用することができる。

しかし、オブジェクトを単純な関係データベースのテーブルに対応づけたのでは、パフォーマンスが上がらず [2]、OODBMS に対する根源的要求を満足できない。このため、パフォーマンスを向上させ、しかも関係データベースの長所を損なわないオブジェクトモデルを設計する必要がある。

また、従来データベース言語とホストプログラミング言語との間に、いわゆるインピーダンスミスマッチ [3] の問題が存在している。OODBMS では、これを避ける言語インターフェイスが重要である。

さらに、既存の OODBMS は、オブジェクトの内部状態はデータベースに格納し共有管理するが、メソッドはアプリケーションと直接リンクし管理しない。このため、マルチメディアのような次々に新しいオブジェクトが出現する環境に対応することができない。

3.1 オブジェクトモデル

COSMOS では上記の問題を考慮してオブジェクトのモデルをデザインし、それに基づいたデータの変換（オブジェクト－関係データ）を行なっている。主な特徴としては以下の点があげられる。

永続性 永続オブジェクトは、データベース内に格納され、オブジェクトを作成したプログラムの終了後も存在し利用される。永続オブジェクト以外のオブジェクト (一時オブジェクト) はプログラムの終了と共に消滅する。

COSMOS が定義した、永続基底クラスである `PObj` から導出されるクラスのインスタンスのみが、永続性を持つことができる。この制約によって、永続性を持たないクラスに対する、不必要な制限をなくし、無駄なスキーマをデータベースに格納することを避けることができる。

プログラマは `PObj` を継承することによって、新しい永続クラスを作ることができる。

オブジェクト ID 全ての永続オブジェクトは、その作成時に COSMOS システムが割り振るオブジェクト ID を持つ。このオブジェクト ID は、単一の物理的 COSMOS オブジェクトサーバ内で、唯一であることを保証する。

コンテナオブジェクト (container object) 全ての永続オブジェクトは必ず一つのコンテナオブジェクト (COSMOS システムが定義する、コンテナ基底クラス `CObj` (`PObj` のサブクラス) から導出されるクラスのインスタンス) に所有される。コンテナオブジェクトはデータベースとのロード、セーブ及びロックの単位となる。

C++ のオブジェクトは小さいことが多く、細粒度のオブジェクトを独立にデータベースに格納すると大幅な性能低下を招く [2]。しかし一般に、小さなオブジェクトは共有の単位にならず、大きなオブジェクトに含まれる形で使用されることが多い。そこで、アプリケーション間で共有される大きなオブジェクトをコンテナオブジェクトとし、コンテナオブジェクト及びそれが主として参照する多数の細粒度のオブジェクトをまとめて一つのクラスタ (cluster) として扱うことにより、性能の向上が期待できる。

プログラマは `CObj` を継承することによって、新しいコンテナクラスを作ることができる。

メタオブジェクト (meta-object) COSMOS システムが定義するクラス `MObj` (`CObj` のサブクラス) のインスタンスは、永続クラスのスキーマやデータベースへのオブジェクトの格納、問い合わせ方法などを保持するオブジェクトである。

コンテナクラスから導出されるクラスオブジェクトに対して質問を発することにより、クラスに属している条件を満たすインスタンスをデータベースより取り出すことができる。

データベースオブジェクト (database object) 物理的オブジェクトサーバへの指示 (トランザクショ

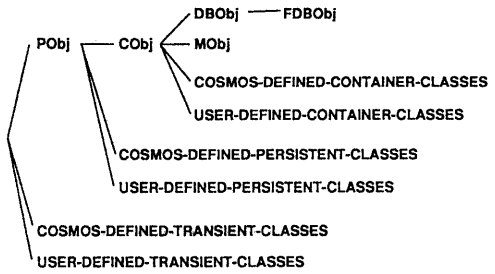

図 2: COSMOS ライブラリのクラス階層

ンのコントロール、メタオブジェクトの取得など) は、データベースオブジェクトへのメッセージを通して行なわれる。

連合データベースオブジェクト (federated database object)
複数の物理的オブジェクトサーバを 1 つの論理的なオブジェクトサーバとして定義できる。この連合データベースへのメッセージを通じて COSMOS のマルチサーバ機能を活用することができる。

図 2に COSMOS ランタイムライブラリのクラス階層を示す。アプリケーションはこれらのライブラリを利用することで COSMOS の機能を利用することができる。

3.2 COSMOS テーブルモデル

COSMOS オブジェクトのモデルにしたがって関係データベースには図 3に示される 3 つのテーブルからなる COSMOS Catalog があり、データマッピングのメタ情報として利用される。CLASS テーブルにはサーバ内でユニークなクラス ID (CID) 、クラス名、C++ 上のヘッダファイルと等価な情報をもつ型情報、メソッドのダイナミックリンクライブラリが保持されるインプリメンテーション情報等が管理されている。SUPER テーブルではクラス間の継承関係が保持され、ATTR テーブルではコンテナクラスの中に定義されている検索対象の変数の名前、そのクラス中での ID 、変数の型などの情報が管理されている。

クラス定義時にはこれら 3 つのテーブルに必要なデータがインサートされ、コミットに成功すると 1 つのコンテナクラスにつき 1 つ図 4のようなオブジェクトテーブルが作成される。各オブジェクトテーブルは CID に基づいてユニークな名前が付けられ、OID 、検索対象となる変数、その他の変数用のカラムをもつ。検索対象となる変数は、そのコンテナクラスで定義されたポインタや PObj 以外の値を持つ変数で、そのクラスで定義されたものに加え、上位クラスで定義されたものも含まれる。したがってあるコンテナクラスのオブジェクトは外部リ

CLASS

CID	CNAME	TypeInfo	Imple.Info
7350	YourContClass	.....	
0001	PObj		
0002	CObj		
0003	MObj		
.....			

SUPER

CID	SUCID
7350	0002
7350	xxxx
.....	

ATTR

CID	Attr_Name	Attr_Id	Attr_Type	....
7350	YourAttrib_0	0	.....	
7350	YourAttrib_1	1	.....	
.....				

図 3: COSMOS Catalog

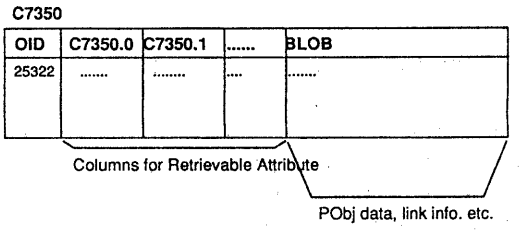

図 4: オブジェクトテーブル

ンクの先のオブジェクトを除き、1 つのテーブルにしまわれる。カラムの型はある決められたルールにしたがって C++ の型からデータベースの型にマッピングされる。検索対象以外の変数値およびそれらのリンク情報はある決められたフォーマットで 1 つのカラムにまとめてしまわれる。データベースやそのコンテナクラスによってそれらは BLOB 、TEXT 、BYTE 等の型を持つカラムとなる。BLOB の利用できないデータベースシステムではファイルシステムを BLOB として利用することもできる。

このようなマッピングを行なうことで懸念されていたモデル変換のオーバーヘッドが少なくなる。さらに各 ID 、クラス名等をインデックス化したり、カタログ検索などの定型的で頻繁に使われる操作をデータベース上でチューニングしてパフォーマンスを向上させている。

3.3 Packed オブジェクト

C++ のオブジェクトをそのまま各 COSMOS コンポーネント間で受渡しするのは効率か悪いうえ扱いにく

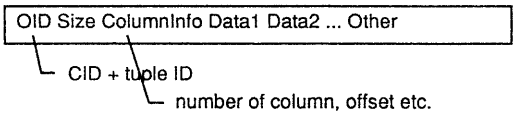

図 5: Packed オブジェクト

い。そこで COSMOS ではオブジェクトは（クラスオブジェクト、DDL オブジェクトを含め）すべて図 5にあるような Packed オブジェクトと呼ばれるバイトストリームとして扱っている。アプリケーションプログラム中の C++ オブジェクトは C++ Interface を経て Client Object Manager へ引き渡される場合、Packed オブジェクトに変換される。逆に Object Manager から得られた Packed オブジェクトはこれを引数とするコンストラクタによりアプリケーションプログラムで利用する時 C++ のオブジェクトに戻される。

Packed オブジェクトは OID 、ストリームのサイズ、そのオブジェクトのクラスの検索対象変数の数とそれらの値のオフセットなどの情報、実際の値、そして検索対象外の値およびリンク情報のデータからなる。

4 ランタイムオペレーション

商用のオブジェクト指向データベースのほとんどは、アプリケーションインターフェイスに C++ を採用しているが、独自の言語拡張を行なっているものが多い [5, 1]。COSMOS では上述のオブジェクトモデルを、C++ 3.0 を用いて言語拡張なしに実現する。これによって、プログラマは言語の拡張部分を新たに学習する必要がないばかりでなく、アプリケーション開発に特殊なコンパイラを必要としないため、実行環境を変更することが容易である。

図 6 にアプリケーションプログラムとオブジェクトサーバの実行時の関係を示す。

1. データベースオブジェクトを要求する。オブジェクトサーバと接続し、データベースオブジェクトができる。

2. データベースオブジェクトに、メタオブジェクトを要求する。クラスの名前または属性を元に、メタオブジェクトを検索し、アプリケーションに返す。この時、そのクラスのメソッドの入ったダイナミックリンクライブラリをデータベースから取りだしアプリケーションとリンクする。

3. メタオブジェクトに、質問を送る。メタオブジェクトが表すクラスに属すオブジェクトの中から、条件を満たすオブジェクトを取りだし、アプリケーションに返す。

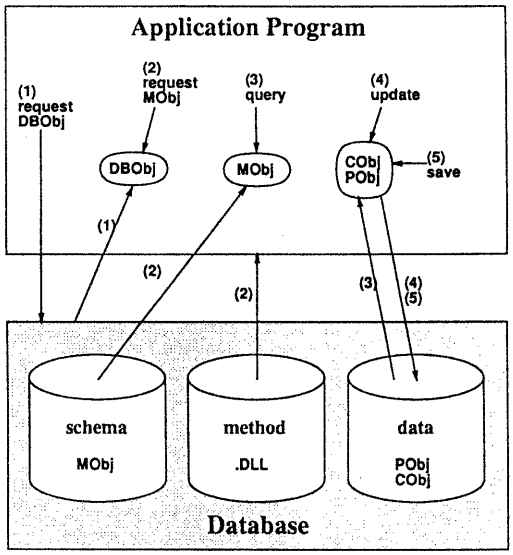

図 6: 実行時の COSMOS システム

4. 永続オブジェクトを変更する。通常の C++ のメッセージパッシングの機構を用いる。

5. 永続オブジェクトをデータベースに格納する。

この章では COSMOS のデータベース機能を中心とした個々の振舞いを実行例とともに説明する。

4.1 クラスの定義

以下に COSMOS でのクラスの定義例を示す。まず、コンテナクラスは次のように定義される。

```
class YourContClass : public CObj {
public:
  void YourMethod();
  /* ... */
protected:
  int YourAttrib;
  /* ... */
};
```

CObj クラスから導出されるすべてのクラスはコンテナクラスであり、そのオブジェクトはデータベースに保存することができ、ロック、転送などの単位となる。それぞれのコンテナオブジェクトには、接続されている物理サーバ内でユニークな ID が与えられる。したがって「サーバの ID + OID 」で COSMOS の世界中でオブジェクトをユニークにアイデンティファイできる。

```
class MyClass : public PObj {
public:
  void YourMethod();
  // ...
protected:
  int YourAttrib;
  // ...
};
```

PObj クラスから導出されるクラスは持続クラスであり、そのインスタンスはコンテナ中に保存することができる。PObj クラスは CObj クラス同様、サーバにユニークな ID が与えられる。そのインスタンスには、その PObj の属すコンテナ中でユニークな ID が与えられる。PObj クラスのインスタンスは COSMOS システム内では独立したオブジェクトとしては扱われず、コンテナに附属するデータとして扱われる。

4.2 クラスオブジェクトの生成

図 7 に COSMOS のアプリケーションの生成方法を示めす。

まず、プログラマはコンパイルされ動作の確認されたクラスをデータベースに登録する。このとき、そのクラスに対してサーバでユニークな CID を与え、それにしたがってスキーマ情報、DLL などがデータベースに送られる。送られたデータは図3で示した COSMOS Catalog に保存され、コミットに成功すると図4で示した CID に基づいた名前をもつオブジェクトテーブルがデータベース内に作成される。

COSMOS では、従来アプリケーションと静的にリンクされていたオブジェクトのメソッドをダイナミックリンクライブラリとしてオブジェクトサーバが管理し、実行時に必要なモジュールをアプリケーションプログラムとリンクする。これにより、アプリケーションプログラムを作成した後に、それが使用するメソッドの実現を変更したり、新しく作られたサブクラスのオブジェクトを使用することができ、アプリケーションプログラムの安定性と新しいオブジェクトに対する柔軟性が向上する。

4.3 データベースへの接続とトランザクション

```
DBObj* db = new DBObj("PS-1");
       //DB obj.
 ...
FDBObj* db = new FDBObj("LS-1");
       //FDB obj.(by defined FDB name)
 ...
FDBObj* db = new FDBObj("PS-1","PS-2",...);
       //FDB obj.
```

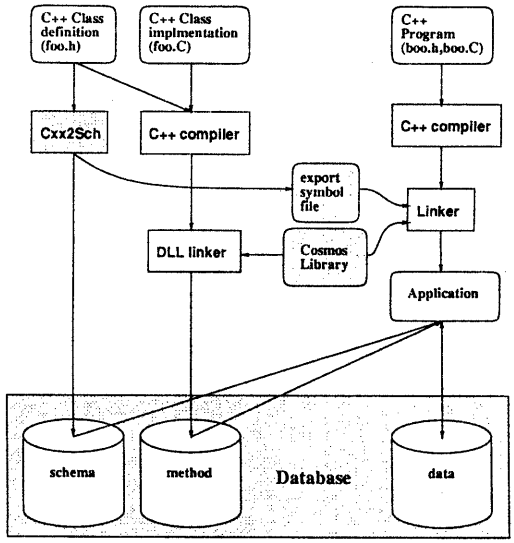

図 7: コンパイル時の処理のながれ

アプリケーションプログラムはデータベースオブジェクト DBObj または連合データベースオブジェクト FDBObj を作ることによりサーバにアクセスすることができる。

連合データベースは複数の物理サーバの集合を 1 つの論理サーバとして扱う。Client Object Server は FDBObj への要求があった場合、引数で示されるサーバの集合を 1 つの論理サーバとみなしてサービスをする。引数には物理サーバ名あるいはすでに登録してある論理サーバ名を指定する。

```
db->transactionStart();
  operation1();
  operation2();
  ...
db->transactionCommit();
```

一連のデータベースオペレーションを 1 つのトランザクションとして実行するには、上記のメッセージを DB オブジェクトまたは FDB オブジェクトに送れはよい。何も指定しない場合は 1 つの物理サーバへの接続の開始から終了までがトランザクションとみなされる。ただし、2 相コミットをサポートしない関係データベースでできた物理サーバを含む FDB の場合、1 つの物理サーバへのトランザクションに留まるアトミックなトランザクションのみを許す。

4.4 オブジェクトの生成と操作

```
FDBObj* fdb;
```

```
...
YourContClass* cobj
        = new(fdb) YourContClass;

YourPersistClass* pobj
        = new(cobj) YourPersistClass;
```

コンテナオブジェクトは DBObj または FDBObj を引数とする new オペレータにより作られる。また、持続オブジェクトはコンテナオブジェクトを引数として作られる。コンテナオブジェクトはそれをデータベースにセーブしようとするか、他のオブジェクトからリンクされる以前に明示的に OID を与えなければならない。COSMOS はデータベースオブジェクトに接続した際に現在の OID の情報をサーバから受けとり、そのクラスの持つ CID を基にユニークな OID を与える。

```
YourClass* obj;
obj->YourMethod(); // YourMethod updates
obj->save();       // save to DB
obj->remove();     // remove from DB
```

新しく作られたコンテナオブジェクトや、データベースから読み込んだコンテナオブジェクトは C++ のオブジェクト同様にメッセージを送ることによりメソッドを実行でき、そのなかでデータベースに反映される update 等を含む操作についてはそれぞれ Object Manager を通じて実際の関係データベースの操作に変換され実行される。

4.5 オブジェクトの検索

```
MObj* mobj = fdb->getMObj("YourContClass");

Iterator* itr
    = mobj->select("a predicate you like");

YourContClass* obj = itr->next();
while (obj != NULL) {
    obj->YourMethod();
    obj = itr->next();
}
```

オブジェクトを質問により検索するには、まず必要とするオブジェクトのクラス名を引数とするメタオブジェクト MObj を得なければならない。MObj は引数で与えられたクラスのスキーマ情報やメソッドの情報をデータベースから受けとりアプリケーションとリンクする。次にこの MObj に検索条件をメッセージとして与えるとそれを実際関係データベース上のテーブル名カラム名に変換した SQL 文が実行される。これを COSMOS が提供している（関係データベースの cursor に相当する）Iterator オブジェクトに代入してフェッチすることにより検索条件に適合するオブジェクトを得られる。

```
YourContClass* obj1
   = itr->next(); // get an obj by query

AnotherClass* obj2
   = obj1->link;  // get another via Link
obj2->AnotherMethod();
```

さらに、オブジェクト間のリンクをたどることによってデータベース内のオブジェクトを読み込むことができる。リンクは COSMOS のコンテナオブジェクト間の参照関係を表し、オブジェクトの論理的な ID を持つ。この論理的な ID は自動的に COSMOS オブジェクトサーバの物理的 ID (サーバ名 + OID) にマップされ処理されるのでプログラマはオブジェクトの場所を意識する必要はない。また、この場合 MObj を必要とせず自動的にオブジェクトを読み込むことができる。

4.6 ビューの定義と既存テーブルのオブジェクト化

関係データベース上にある既存データは COSMOS ビューを定義することにより C++ 上のオブジェクトとして利用可能となる。COSMOS ビューは既存の COSMOS クラスからも定義することができるので必要に応じて各クラスのビューを作ることができる。

ビュー定義の概要を図 8に示す。

1. 元の関係データベースに定義されているテーブル、ビュー、または COSMOS クラスを参照し、これから定義するビューの元となるものとその中のキーとなるカラムを調べる。つぎに新たに作るビューの名前を決め、その新クラス名、元のテーブルまたはクラス名、キーとなるカラムまたは属性名を引数とするインタラクティブなスクリプトを起動する。新たなビューが述語を必要とするものである場合、あらかじめ SQL によりその述語によるビューを定義しておき、それを元のテーブルとすれば良い。

2. ビューを管理するカタログに必要なデータを書き込む。これらのカタログテーブルは主に元のテーブル名とビュー名（ビュークラス名）、元のタプルとビューとしてのそのタプルの OID のマッピングに利用される。

3. 元テーブルのスキーマから、そのビュー（クラス）のヘッダファイルが持つべき最低限の属性をもつヘッダファイルのテンプレートを生成し、クライアントに返す。

4. テンプレートをもとにビューのためのメンバー関数（メソッド）を定義する。このときビュー（クラス）のスキーマの変更を必要とするような変更はできない。

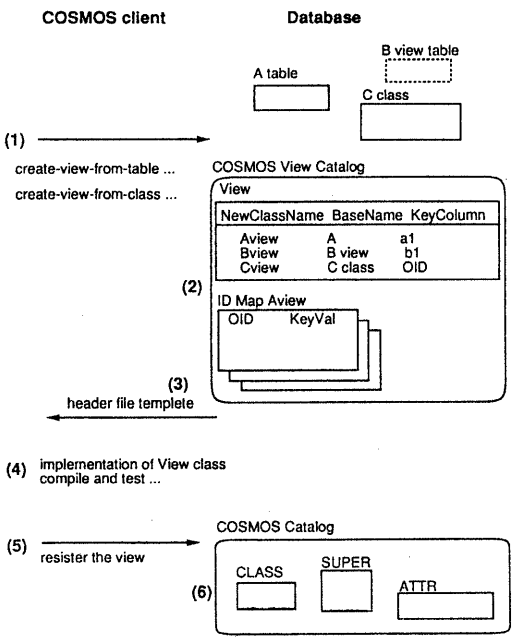

図 8: ビュー定義の概要

5. コンパイルし、振舞いを確認した後、そのクラスをビューとして登録する。このときオブジェクトテーブルが新たに作られないこと以外は、ほぼクラス定義と同様の振舞いをする。

6. COSMOS Catalog に必要なデータを書き込む。

4.7 セキュリティ管理

COSMOS は WAN を通して不特定多数のユーザが利用することを想定してデザインされている。また、ベースにある関係データベースの種類ならびに OS によってユーザ、アカウントの管理法がかなり異なっている。そこでアクセスコントロールは個人ユーザのアカウントではなくユーザのクラスを単位として行なっている。ユーザは自分の属すクラスの権限で Object Server にアクセスし、各関係データベースはそのクラスの権限をもとに各種操作の制限を受ける。システム管理者、サーバの所有者はクラスごとに各オブジェクトテーブル、カタログ、ビューなどに権限を与える。

5 マルチサーバ管理

COSMOS は WAN 上にいくつかのサーバを有するマルチサーバに対応できるよう拡張されている。したがってネットワーク上の適切なサーバを指定して、そのサーバにあるデータを利用できる。さらにはいくつかのサーバを含む論理的なサーバを定義することにより、オブジェクトマイグレーション、レプリカ、フォールトトレラント等の各種の分散マルチサーバ機能を利用できる。

5.1 論理サーバの制御

データベースオブジェクトによりネットワーク上の利用可能な任意の物理サーバを利用することができる。それに加え COSMOS では連合データベースオブジェクトを利用することにより、マルチサーバの特徴をいかしたサービスを提供する。

各 Client Object Server には必要に応じて連合データベースを定義することができる。例えば図 9のように PS-1 PS-2 PS-3 の 3 つの物理サーバからなる論理サーバ LS-1 を定義した場合、この LS-1 を連合データベースとしたアプリケーションでは次のような振舞いをする。

LS-1 に対する連合データベースオブジェクトへの接続リクエストが来と、その Client Object Manager 内の論理サーバ定義をさがす。なければその名前の物理サーバに接続要求を出す。この場合、LS-1 の定義によりデフォルトサーバとして PS-1 に接続される。(先に定義されているものほど優先順位の高いサーバとなる)

Client Object Server は接続されたディフォルトの物理サーバにアプリケーションからの連合データベースに対するリクエストを送る。クライアントからのリクエストがそのサーバで処理できるものであれば、それを処理し、できない場合、サーバはエラーコードを返す。

ディフォルトサーバで処理できなかった場合、Client Object Manager は論理サーバ定義中の次のサーバ、あるいはそのリクエストを処理できる論理サーバ内の他の物理サーバにリクエストを出し、それを処理する。各物理サーバはオブジェクトのマイグレーション情報、レプリカオブジェクトの情報などを参照できるため、リクエストによっては他のサーバへのリクエストでも自身で処理できたり、そのリクエストの処理が可能なサーバを知っている場合 (エラーコードにより適切なサーバを指示することができる場合) もあるので、ただ論理サーバを順に接続していくよりも効率が良い。

このような Client Object Manager の連合データベースオブジェクトの管理を利用することにより、プログラマはマルチサーバを意識することなく COSMOS のマルチサーバ機能を利用することができる。(ただし、2 相コミット等の分散機能のないデータベースを利用する、連合データベースによるトランザクションには仕様上の制約が生じるケースがあり注意が必要である)

効果的な利用法としては次のような場合が考えられる。安定したサービスを必要とする場合は、そのサービスで必要なオブジェクトのレプリカを 2 つ以上の含むような論理サーバを構成することによって物理サーバやネットワークの故障に対処することができる。さらに、これら

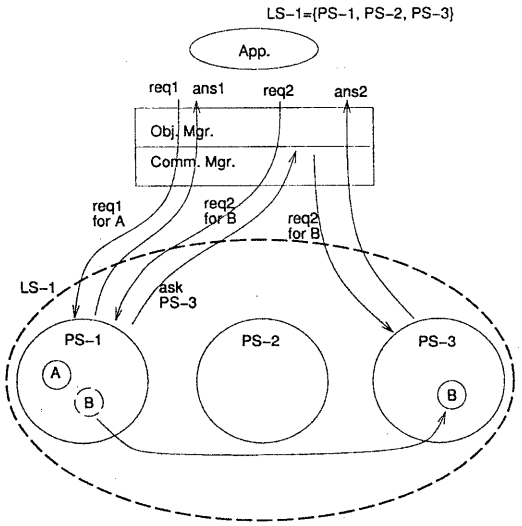

図 9: 論理サーバ

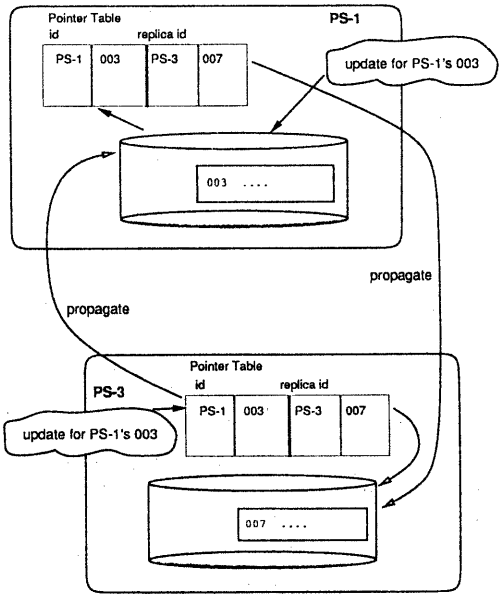

図 10: ポインタテーブルと分散オブジェクト管理

を速いネットワーク上のサーバを優先サーバとして定義した論理サーバとすれば、いくつかのレプリカオブジェクトのうちもっとも速く届くものにアクセスできる。

5.2 分散オブジェクト管理

Client Object Manager でのマルチサーバ管理を実現するためには、各物理サーバにマルチサーバ上の分散オブジェクトの管理機能が必要になる。サーバ上で管理しなければならないのは主として、等価なオブジェクト群の一貫性の管理と各サーバ間でのそれらの OID のマッピングである。

まず、等価なオブジェクトとそれらの OID を効率良く管理するため、各サーバにはポインタテーブルと呼ばれる OID の管理テーブルがある。このポインタテーブルには、自サーバが持つオブジェクトのレプリカ情報、マイグレートされたオブジェクトの移転先の情報、他のサーバにあるオブジェクトのレプリカで自サーバで持っているものの情報が管理されている。したがってこのポインタテーブルを参照することにより、かつてあったオブジェクトのマイグレート先とそこでの OID 、今あるオブジェクトのレプリカのありかとそこでの OID 、自身がどこのどのオブジェクトのレプリカを所有しているか、の 3 つの情報が常に把握できる。

マルチサーバ間でのマイグレーション、レプリケーションなどの操作はポインタテーブル等のメタ情報を必要とするため、そのためのインタラクティブなスクリプトを通しておこなう。スクリプトを起動した際、必要なメタ情報は自動的にポインタテーブルに格納される。図 10 にリプリケーションされたオブジェクトの概要を示す。例では物理サーバ PS-1 の OID 003 のオブジェクトのレプリカを物理サーバ PS-3 に作った状態を示している。

レプリカを作るプロセスでは、まず、元のオブジェクトのクラスに相当するものが、レプリカを作る先のサーバにあるかチェックし、なければ元のサーバからクラス情報を新たなサーバにコピーする。(新たなサーバで同じクラスを新たに定義する) 次にオブジェクトをコピーする。その際、新しく OID が与えられるのでその情報をもとに両サーバのポインタテーブルの情報を書き込む。

レプリカが作られた後はサーバによって一貫性の管理が行なわれる。例では PS-1 の 003 に対する更新は PS-3 をはじめとする各レプリカにプロパゲートする。逆の場合も同様な処理が起こる。

レプリカを持つ側に本物のオブジェクトに対するリクエストが来た場合、(例では PS-3 に PS-1 の 003 に関する要求があった場合、) PS-1 に対するリクエストなので通常の処理はおこなわれない。しかし、ポインタテーブルを参照した際、PS-1 の 003 に関してはレプリカを持っていることがわかり、そのリクエストはレプリカに対する (PS-3 の 007 に対する) リクエストに翻訳されて処理される。

オブジェクトのマイグレーションをする場合は、レプリカを作る場合同様にクラス情報、ポインタテーブル情報の書き込みを行なった後、元のオブジェクトを削除する。

6 まとめ

本稿では、WAN で接続されたネットワーク環境におけるオブジェクトサーバ COSMOS について、その関係データベースを利用したデータ管理機能を中心に紹介した。本システムはマルチメディア・アプリケーションを始めとした C++ アプリケーションの開発および実行のための支援システムで、そのデータベース機能は言語拡張なしの C++ を通して容易に利用することができる。

また、COSMOS オブジェクトサーバは関係データベース (RDBMS) を利用しており、RDBMS が本来有する基本機能をいかした設計をしている。C++ アプリケーションでのデータはオブジェクトとして存在するが、COSMOS の DB Interface により RDBMS 中では関係データとして存在する。COSMOS ではこれらのモデル変換を自動的に行なっている。当初、オブジェクトをそのままモデル変換なしに格納することのできるオブジェクト指向データベース (OODBMS) にくらべモデル変換のオーバーヘッドからパフォーマンスの低下が心配されたが、OID を与えられ単独でデータベースに格納できるコンテナオブジェクトとその他の持続オブジェクトと区別したオブジェクトモデルを利用することで、細粒度のオブジェクトをクラスタリングし、まとめて扱うことができるため、シュミレーション実験ではオブジェクトの構造が複雑であるものにおいては OODBMS よりもよい結果が得られた。我々は主としてマルチメディア等、ある程度大きなオブジェクトを対象としているのと、故意に細粒度のオブジェクトの多い設計をしないという仮定のもとでパフォーマンス上問題はないと考えた。

RDBMS を利用するさらなる利点はその普及度と標準機能の充実度にある。COSMOS の DB Interface はモデルの変換を行なうだけでなく、関係データベース機能の抽象化を行なっており、提供される抽象化された RDBMS の API を利用することで各種の RDBMS の違いを吸収することができる。さらにオブジェクトビューを定義することにより、既存の関係データベース上のデータをオブジェクトとしても利用することができる。

現在、COSMOS はシングルサーバで単一の関係データベースのもとで作られたプロトタイプをもとに、マルチサーバと各種データベースおよびＯＳに対応できるよう拡張しつつある。現在、１部の機能しか稼働していないが、全体が稼働可能になったら、そのパフォーマンスを含めた仕様感等についても報告したい。今回は関係データベースを用いたオブジェクトの管理を中心に述べたが、今後はさらにマルチメディアに特化したデータ管理手法、マルチメディアライブラリとの効率的な連携についても考えてゆきたい。

謝辞

日頃、ご討論いただいた Fin 氏、梶谷氏、平賀氏ならびに研究所の諸氏に感謝致します。

参考文献

[1] S. Ahmed, A. Wong, D. Sriram, and R. Logcher. A comparison of object-oriented database management systems for engineering applications. Technical report, Massachusetts Institute of Technology, May 1990.

[2] R. G. G. Cattell and J. Skeen. Enginicering database benchmark. Technical report, Sun Microsystems, Apr. 1990.

[3] G. Copeland and D. Maier. Making smalltalk a database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 316–325, 1984.

[4] T. H. Fin and K. Kajitani. A framework for supporting multimedia in distributed object server. 情報処理学会第４５回全国大会講演論文集（４）, pp. 115–116, Oct. 1992.

[5] J. V. Joseph, S. M. Thatte, C. W. Thompson, and D. L. Wells. Object-oriented databases: Design and implementation. In *Proceedings of the IEEE*, volume 70, pp. 42-64, Jan. 1991.

[6] K. Kajitani and T. H. Fin. オブジェクトサーバを中心としたネットワーク環境におけるマルチメディア・オブジェクトの実現. 情報処理学会第４５回全国大会講演論文集（４）, pp. 117–118, Oct. 1992.

[7] K. Kosaka, K. Kajitani, Q. He, Y. Morimoto, and T. Fukuda. オブジェクトサーバとその応用. 情報処理学会研究会報告, volume 92-DBS-89, pp. 19–28, July 1992.