データベースシステム 89－3
（1992. 7. 22）

# オブジェクトサーバとその応用

小坂 一也，梶谷 浩一，何 千山，森本 康彦，福田 剛志

日本アイ・ビー・エム 株式会社 東京基礎研究所

**あらまし** ネットワークで結合された異機種の構成を考えた時、マルチメディアを扱うシステムには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。我々はそのような環境において、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 を構築している。本稿では、オブジェクト・サーバを中心とするシステム全体の概要について述べた後、最初のプロトタイプについて報告する。

# An Object Server and Its Applications

Kazuya Kosaka, Kouichi Kajitani, Qianshan He, Yasuhiko Morimoto, Takeshi Fukuda

Tokyo Research Laboratory, IBM Japan Ltd.

**Abstract** Extensibility and flexibility are very much important for multimedia systems so that they can handle emerging media when we consider heterogeneous computers in a networking environment. We are working on COSMOS project, which is an object-oriented environment with an object server for development and execution of multimedia applications in the circumstance. In this paper, we describe an overview of COSMOS and its first prototype system.

# 1 はじめに

従来、情報の交換はデータの交換を意味していたので、データ変換や共通データ・フォーマットは重要な技術的課題であった。しかし、現在に至るまで単一のフォーマットは存在せず、それどころか、年々新たなフォーマットが増加している。そのため、受けとったデータの処理はそのフォーマットごとに各アプリケーションが個別に対応する必要があった。ところがマルチメディアのような複雑なデータを扱い始めると、単なるデータ交換ではそれを処理するアプリケーション作成の負担が大きくなり過ぎて対応し切れなくなって来た。そこで、最近注目されてきたのが、「オブジェクト交換」と呼ばれる、データとそれを処理するための手続きをカプセル化したオブジェクトを情報の単位として交換する方式である。

オブジェクト交換の利点は、データを受け取る側のアプリケーションが、そのデータ構造を知る必要も無く、それを直接処理する必要も無いため、個別対応の負担から解放される点である。そのため、個々のアプリケーションはオブジェクト化された各メディアを組み合わせることに専念できるので開発コストの低減や期間の短縮が期待できるのである。

オブジェクト交換の仕様としては、ローカルな環境でのみ機能する OLE[11] などが現在実用化されている。また、ネットワーク環境で機能する仕様としては CORBA[7] などが提案されており実用化は時間の問題となっている。

以上のようなネットワークで結合された計算機間で自由にオブジェクトを交換できるような世界を考えた時、それら交換可能なオブジェクトを組織的に管理するためのデータベースを考えることは重要であり、また、そのような環境にオブジェクト指向データベース管理システム (OODBMS) が応用できると考えるのは自然であろう。それでは、現在商用化されている OODBMS[8],[10] を使用すればなにも問題がないのであろうか？ 確かに商用 OODBMS はオブジェクト指向インタフェースを提供しているが、共有管理の対象となるのはメタ・データ (一部のクラス情報) とデータ (インスタンス変数) のみであり、手続き (メソッド) に関しては各アプリケーションに静的にリンクされており共有の対象にはなっていない。従って、オブジェクト交換を行うためにはメソッドの共有および実行まで含めた意味でのオブジェクトの管理を行なう OODBMS が必要になると思われる。本稿のタイトルに OODBMS という言葉を使用せず、"オブジェクト・サーバ" としたのは以上のような理由による。

図 1 は上記のようなネットワーク環境におけるマルチメディア・オフィスシステムの例を示している。現在、筆者らはこのような環境におけるマルチメディア・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。本稿では、2 節で COSMOS 全体の構成について述べ、3、4、5 節でオブジェクト・サーバの各構成要素について述べる。また、6 節でライブラリを中心としたマルチメディア対応について述べた後、7 節で最初のプロトタイプについて概説する。

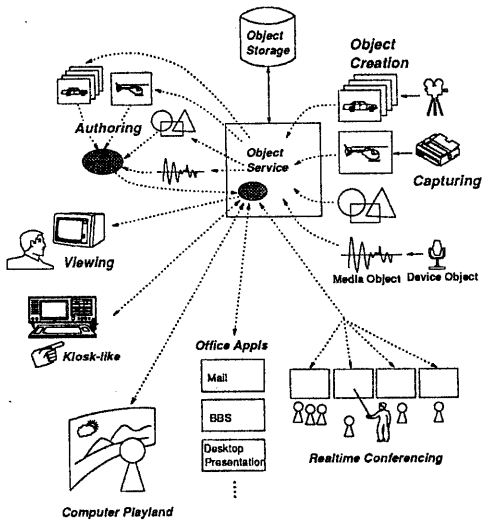



図 1: マルチメディア・オフィス・システムの例

# 2 全体の構成

COSMOS 全体の構成を図 2 に示す。COSMOS 環境は大きく分けて 5 つの構成要素からなっており、それぞれ (1) アプリケーションから直接利用可能なマルチメディア・ライブラリ、(2) それらを実現するクラスを記述するための C++ インタフェース、(3) 各クライアントとサーバの間でオブジェクトを転送するためのオブジェクト・マネジャ、(4) オブジェクトをリレーションに変換するためのデータベース・インタフェース、そして、(5) 基本的なデータ管理を行う関係データベース管理システム (RDBMS) である。

現在の構成ではサーバとして IBM RS/6000 上で稼働する AIX を、そしてクライアントとして IBM PS/55 上で稼働する OS/2 を採用している。また、RDBMS には現在米国 IBM で開発中の拡張可能 RDBMS である Starburst[5] を使用している。

# 3 C++ インターフェイス

## 3.1 オブジェクトモデル

C++ から見た COSMOS のオブジェクトモデルは、次のような特徴を持つ。

**永続性** — 全てのオブジェクトは、永続 (persistent) かまたは一時 (transient) である。全ての永続オブジェクト

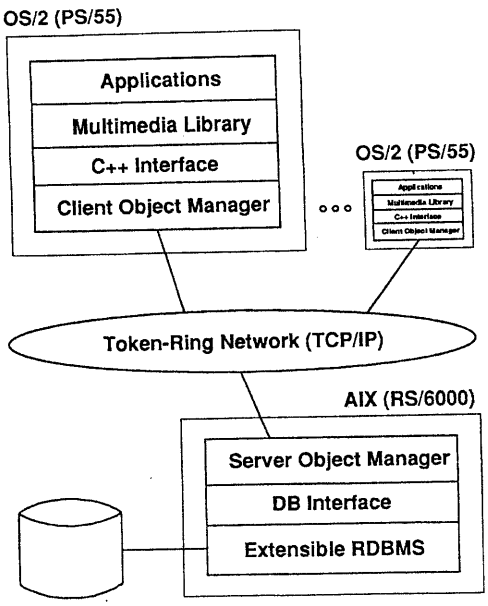



図 2: COSMOS 全体の構成

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、後に利用される。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。

永続クラスのインスタンスは一時オブジェクトとして作ることもできる。

PObj から導出されるクラスに属さないインスタンスは全て一時オブジェクトである。

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のオブジェクト ID を持つ。このオブジェクト ID は、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。

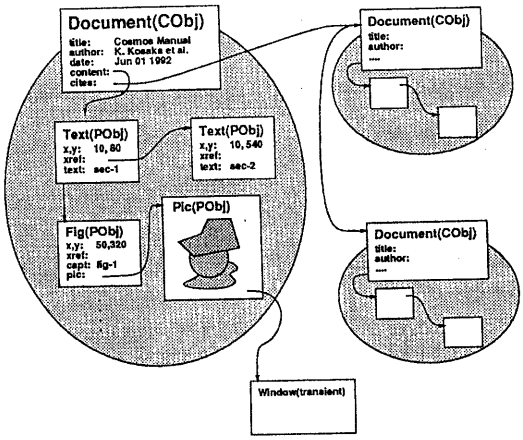



図 3: complex オブジェクトの例

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 [3]。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に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 (PObj 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・クラスの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) — オブジェクトサーバへの指示 (トランザクションのコントロール、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。

図 3 に、COSMOS オブジェクトモデルに基づくコンプレックス・オブジェクトの例を示す。

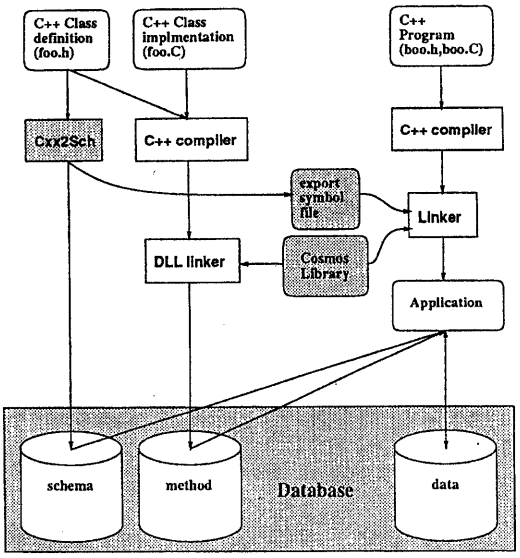



図 4: コンパイル時の処理のながれ

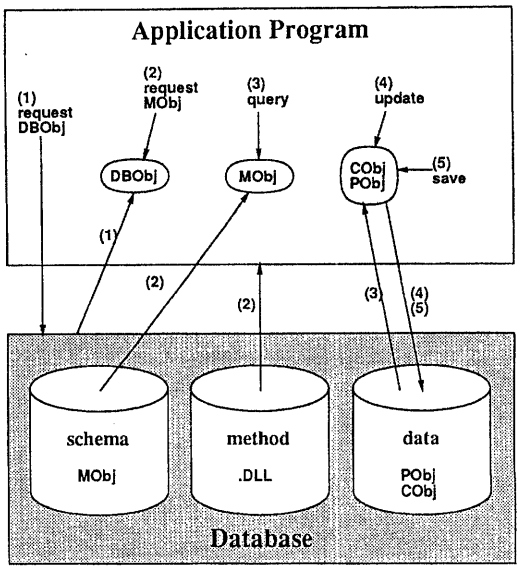



図 5: 実行時の COSMOS システム

### 3.2 コンパイル時

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 [8], [2]。COSMOS では3.1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は特殊な言語プロセッサを用いないため、実行環境を変更することが容易である。図 4 に COSMOS のアプリケーションの生成方法を示めす。

COSMOS が用意する言語プロセッサ c++2sch (C++ to schema - 現在開発中) は、C++のクラス定義を読み込んで、永続クラスのスキーマ情報をデータベースに登録し、データベースとのロード、セーブ等のためのプログラムを生成する。同時に、ダイナミック・リンク用の外部シンボル・ファイルを生成する。

### 3.3 実行時

図 5 にアプリケーション・プログラムの実行時の動作を示し、概略を説明する。

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をリンクする。
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。
4. 永続オブジェクトに変更を加える。
5. 永続オブジェクトを格納する。

## 4 クライアントーサーバ・アーキテクチャ

### 4.1 オブジェクトサーバとページサーバ

クライアントーサーバ・アーキテクチャを取っている OODBMS はオブジェクトサーバとページサーバに大別できる [4]。オブジェクトサーバでは、ネットワーク上での転送単位はオブジェクトで、サーバが問い合わせ機能を持って必要なサブセットを送る。ページサーバでは、ネットワーク上での転送単位はページで、サーバが問い合わせ機能がなくクライアントが問い合わせ機能を持っている。ページサーバでは、よいクラスタリング及びインデックスを付けている場合、ページフォールトが少ないので、ネットワーク上のページ転送量は少なくなり、また、クライアント側

が速いCPUと大きいキャッシュを持っている場合、高速アクセスができる。一方、オブジェクトサーバでは、上述の条件があまりよくない場合ではページサーバと同じぐらいのパフォーマンスが得られる[4]。異機種のサーバとクライアント環境では、ページの構造が異なるので、ページサーバのアプローチは取れない。COSMOSのプロトタイプでは、クライアントはOS／2の上で、サーバはAIXの上で構築されるので、オブジェクトサーバのアプローチを取る。図6はCOSMOSのクライアントーサーバ・アーキテクチャの構成図である。

## 4.2 オブジェクト転送

COSMOSでは、TCP/IPを基づいたRPCを使ってオブジェクトをクライアントとサーバの間に転送する。RPCでオブジェクトを転送する場合、一つのパケットサイズ以内（0から1．5 KBぐらいまで）のオブジェクト転送時間はほとんど同じであるので、オブジェクトが小さくなるほど、転送のオーバヘッドが大きくなる。そこでなるべくいくつかのオブジェクトをまとめ、パケットサイズ以上の単位で転送することが望ましい。商用のページサーバを基本とするOODBMSの転送単位はページ（4 KBぐらい）であるので、これはパフォーマンスを上げる一つ重要なポイントである。COSMOSでは、RPCの転送単位はコンテナ・オブジェクトであり、コンテナ間にリンクを張っている場合、リンクを辿ってリンクされるコンテナを送る。クライアントからサーバにコンテナを送る場合、まずコンテナをバイド列にパックする。アプリケーション・プログラムをコンパイルする時に、永続クラスの中のリンクが検出され、リンクを辿るパック用メソッドが作成される。実行時にパック用メソッドを呼び出し、ヒープ上にあるコンテナをRPCバッファにパックしてバイド列に変換してサーバに送る。

## 4.3 トランザクション管理

各クライアントからの同時アクセス、障害回復を制御するために、トランザクション管理が必要となる。各クライアントにはローカル・データベースの存在を仮定していないので、トランザクション管理はすべてサーバで集中管理する。アプリケーションには二相ロック（two-phase locking）と楽観的ロック（optimistic locking）の二種類の並行制御プロトコルを提供する。二相ロックは主に資源を確保してアクセスしたいトランザクションに使われ、楽観的ロックは主にチェック・インーチェック・アウト使い方の長いトランザクションに使われる。二相ロックではコンフリクトが起こる時、時間軸の順にトランザクションがブロックされる。楽観的ロックではコンフリクトが起こる時、トランザクションはブロックされないが、トランザクションのコミットフェーズにアボートされる。また、二つのアプリケーションが異なったプロトコル（一つは二相ロック、もう一つは楽観的ロック）を使用する場合、楽観的ロックのトランザクションが先にコミットしようとしても、もし二相ロックのトランザクションとコンフリクトすれば、楽観的ロックのトランザクションがアボートする（二つの楽観的ロックのトランザクションの場合、先にコミットしようとするトランザクションは必ずコミットする）ので、二相ロックのトランザクションは資源確保を保証する。

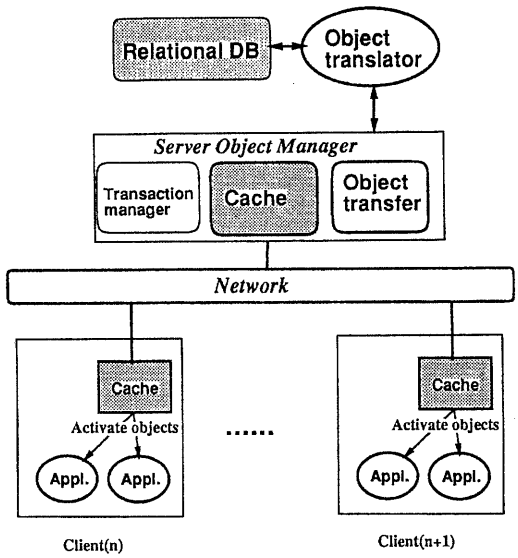



図6: クライアントーサーバの構成

リンクでつながっているツリー構造あるいはグラフ構造の複合（complex）オブジェクトに対してロックをかける場合、一度親にロックをかければすべての子孫（グラフ構造では辿られる範囲）もロックされるのは効率的であるが、極端な場合は全データベースをロックする可能性があるのであまり現実ではない。しかし、すべてのオブジェクトをロックの対象にすると、オーバヘッドが大きくなる。我々はその中間を取り、いくつかの独立していないオブジェクトからなるコンテナ・オブジェクトをロックの単位とし、コンテナのオブジェクトIDに対してロックをかける。そうすると、アプリケーションでサーバにアクセスできる単位はコンテナで、コンテナの中から他のコンテナをリンクを持っている場合、必要に応じて（on demand）でロックをかけてアクセスする。コンテナ・オブジェクトはマッピング・ルールによっていくつかのテーブル、タプルに跨る可能性があるので、RDBMSのトランザクション・マネジャを利用する場合、トランスレータはどのタプルをロックするかを決める。ロックの最小単位がタプルではなくテーブルあるいはページである場合、この方法は適切ではなく、RDBMSと独立にサーバ・オブジェクト・マネジャはトランザクション管理機能を持たなければならない。今回我々は使用しているStarburstはテーブルに対してロックして

いるので、Starburst 本体のロックを外してサーバ・オブジェクト・マネジャでトランザクションとロック管理を行なっている。サーバ・オブジェクト・マネジャにおけるトランザクション・マネジャとロック・マネジャは Starburst のトランザクション・マネジャ、ロック・マネジャを改良して利用し、オブジェクト ID にロックがかけられるようにする。

### 4.4 オブジェクト・キャッシング

データベースのアクセスとネットワーク上のオブジェクト転送を減らしてパフォーマンスを上げるために、キャッシングは不可欠である。サーバは 10MB ぐらいの大容量キャッシュを持ち、各クライアントからのアクセスがある場合、まずキャッシュにあるかどうかをチェックする。クラス情報は頻繁にアクセスされるので、サーバの初期化時に RDBMS にあるすべてのスキーマ情報がキャッシュに置かれる。また、クライアントからサーバのオブジェクトを取り出す時に、まず RDBMS のタプルをオブジェクトにトランスレートしてキャッシュに置いてからクライアントに送る。クライアントにもキャッシュがあり、同一クライアント内の複数のアプリケーションから共用される。キャッシュはオブジェクト ID の hash テーブル、キャッシュ・ディレクトリ、キャッシュ本体からなり、コンテナ・オブジェクトの単位でキャッシングを行ない、サーチの対象はオブジェクト ID である。キャッシュが満ぱいになる時、キャッシュを更新する必要がある。キャッシュ更新アルゴリズムとして least-recently-used(LRU) アルゴリズムは効率でよく使われているが、オブジェクトサーバでキャッシュしたオブジェクトが可変長で、キャッシュの空きスペースのリアロケーションは非効率であるので、first-in-first-out の更新アルゴリズムを採用する。サーバとクライアントの両方はキャッシュを持っているので、キャッシュ一貫性管理が必要となる。あるクライアントのあるオブジェクトが更新される場合、このクライアントのキャッシュとサーバのキャッシュは更新されるが、このオブジェクトを持っているすべてのクライアントのキャッシュを更新する必要がなく、ダーティ・マークを付ければよい。ダーティ・マークを付いたオブジェクトをアクセスしたい時、必要に応じてサーバからコピーするので、無駄なオブジェクト転送を避けることができる。

## 5 オブジェクトとリレーション

より高機能のアプリケーションが求められるに従って、データベースに登録するデータに対してもより複雑で高機能なものが求められている。しかし、同時にデータベースに登録されるデータは多くのアプリケーションあるいはユーザに広く利用され得なければならない。高度に構造化されたデータ (以下、コンプレックスデータ、コンプレックスオブジェクトと呼ぶ) はそれ自身、ある特定のアプリケーションに依存しているため、本質的に多くのアプリケーションで利用することが難しい。一方、従来の関係モデルにおけるデータは多くのアプリケーションから広く利用することができるが、コンプレックスデータをモデリングするには表現力に乏しい。

そこで我々はこれらの 2 つの相反する要求を満足させる第一歩として、関係モデル上のリレーションをオブジェクト指向の概念でとらえ、操作する手法を提供し、また逆にオブジェクトをリレーションに変換して保持する手法を提供する。これにより本システムでは、コンプレックスオブジェクトを含めすべての永続オブジェクトはリレーションの形で保持され、必要な時にオブジェクト化して利用する。これらの関係モデルのデータはシステムが提供する豊富なデータベース操作機能を使って検索することも可能である。

本節ではオブジェクト指向モデルと関係モデルのマッピング手法とシステムが提供するデータベース・クラスついて述べる。

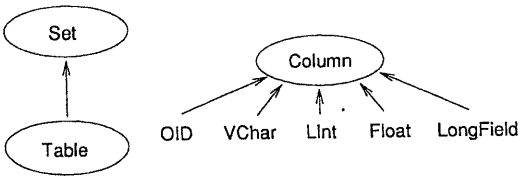
### 5.1 データベース・クラス

COSMOS オブジェクトサーバーを利用するアプリケーションは通常、C++上で利用できる Starburst 用の基本的なアプリケーション・プログラム・インターフェース (API) を利用して作られた Table クラス、および C++のオブジェクトとして認識されたタプルの各種操作を提供する Set クラスを通じて関係データベースにアクセスする。図 7 にデータベース・クラスのクラス階層および、それぞれのアグリゲーション階層を示す。

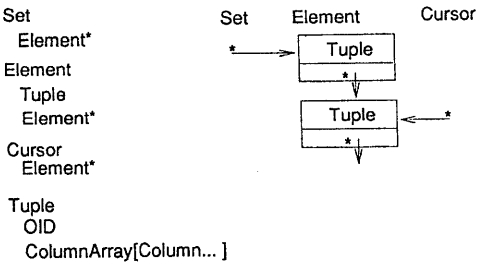
ユーザ及びアプリケーションプログラマにデータベース機能を提供する Set クラスは Element オブジェクトへのポインタを持つ。Element クラスは Tuple オブジェクトと Element オブジェクトへのポインタを持つ。Set オブジェクトはこの Element オブジェクトリストの先頭を指すポインタである。Set クラスはまた、その Set が指す Element のリスト中にある任意の Element オブジェクトへのポインタである Cursor オブジェクトを操作して各種データベース操作を提供する。

Set クラスが扱っているデータが C++のオブジェクトであるのに対して、Set クラスの導出クラスである Table クラスは直接データベース上のテーブルあるいはタプルを API を利用して操作する。したがって Set クラス上で提供される操作は実際の API 用にオーバーライドされる。さらに、cursor の更新操作など、Set クラスには定義されていないものを含んだより充実したデータベース操作が可能である。

Element オブジェクトの内容である Tuple オブジェクトは OID と Column オブジェクトの配列である ColumnArray からなる。関係データベース上のタプルは Set クラスおよび Table クラスを通じて見た場合の永続オブジェクトの単位として認識されるため、COSMOS 環境上でユニークな ID を与えられる。抽象クラスとして定義された Column クラスの導出クラスとしては、OID 、LongField と Starburst




(a) Class Hierarchy




(b) Aggrigations

図 7: データベースクラス

| ClassID | variables | vtable | constructor | super |
|---|---|---|---|---|
| #people | name, age,.. | | | |

(a) Catalog Table

People

| OID | name | age | ... |
|---|---|---|---|
| | | | |

(b) Table

図 8: リレーショナルテーブル

上でのプリミティブである VChar 、Lint 、Float などのクラスが定義されている。OID が ColumnArray の要素である場合、他のオブジェクトへの参照 (reference) を意味する。LongField は Starburst によって提供される長大データフィールドである [9] 。これは主として画像データなどの大容量のデータやコンテナ・オブジェクトに含まれるオブジェクトのバイナリーコードの保持に使われる。

## 5.2 マッピングの概要

C++インターフェース上で定義された永続クラス、及び永続オブジェクトはデータベース・インターフェースを介して、Table クラス、Tuple クラスなどのデータベースクラスにマッピングされうる形で、データベースに登録される。

永続クラスはデータベースには CatalogTable の 1 タプルとして登録される。CatalogTable は図 8(a) のように ClassID 、変数名の配列である variables 、メソッドなどに関する情報をもつダイナミック・リンク・ライブラリ (DLL) を指すポインタ vtable 、各種構築子へのポインタの配列 constructor 、基底クラス ID の配列である super からなる。さらに、CatalogTable の情報を用いて、それぞれの永続クラスのためのテーブルが図 8(b) のように作成され、永続オブジェクトはそのクラスのテーブルにおける 1 タプルとして保持される。それぞれの永続クラスのテーブルは OID と、そのクラスおよび導出されるクラスの変数名すべてに対してのカラムをもつ。

このようにテーブルの 1 タプルとしてデータベースに保持されている永続オブジェクトは、必要な時、Table クラスおよび Set クラスの提供するメソッドを使って検索し、Tuple オブジェクトとして取り出すことができる。それぞれの永続クラスはそれに対応する Tuple オブジェクトを引数とする構築子 (activator) をもつ。その構築子には CatalogTable などの情報をもとに、その永続クラスのオブジェクトを構築する手続きが記述され、プログラマはそれを利用することでデータベース上に保持されたデータを、もとの C++オブジェクトに戻すことができる。

## 5.3 データベース操作

C++上からデータベース機能を利用する場合、ハンドルクラスをインスタンシエイトして特定のデータベースを起動する。その起動されたデータベースはハンドルにコネクトされ、データベースへの API をうけつけるようになる。このハンドルは string 型で準備された (基本的な) SQL の構文をを受けつける。

SELECT 文の場合、そのあとに続くカラム名で示される Column オブジェクトを要素とする ColumnArray オブジェクトをもつ Tuple オブジェクトが作られ、それを Element にもつ Set 型のオブジェクトがかえされる。FROM 節に続く名前は Table 名 ( Table クラスのインスタンス名として扱われるがデータベース上の実際のテーブル名と同じ。) であっても Set 名であってもよい。SELECT 文の起動とともに Cursor がオープンされ、WHERE 節の条件にあうタプルがリターン値である Set の要素に Element オブジェクトとして加えられる。

# 6 マルチメディア対応

いくつかの基本的で重要なメディア (データ型) は COSMOS 対応の C++ で書かれたクラスライブラリ (マルチメディア・ライブラリと呼ぶ) によってサポートされる。

マルチメディア・ライブラリが提供するものは、(1) メディアを扱うためのメソッドと (2) メディアを二次記憶装

置上に格納したり、そこから取り出したりする機能である。マルチメディア・ライブラリのクラスは、永続クラスとして作成されるので、(2) の機能は、ほとんどの場合自動的に行われるが、一部のメディア (例えば、動画像) は、独自のメカニズムを備える場合がある。

ライブラリの設計で考慮したのは、メディアの "理解の容易なモデル化" と将来の "拡張性" である。メディアのモデル化に関しては、様々な研究 (例えば、[6]) が行われているが、マルチメディア・ライブラリにおけるモデルの特徴は、メディアとそれを実現するためのハードウエア (デバイス) の明確な分離である。実際のマルチメディア・システム上では、デバイスの受け持つ機能は多く、重要であり、コードの再利用の観点からもデバイスとメディアを別のライブラリで表現する。また、メディアの変換を行う機能をフィルタと呼び、これも独立したライブラリとした。

### 6.1 マルチメディア・ライブラリ

COSMOS のマルチメディア・ライブラリは、

1. メディア・ライブラリ (ビデオ、オーディオ等の抽象的なデータ型を定義)
2. デバイス・ライブラリ (レーザディスク・プレイヤ、スピーカ等メディアを扱うためのデバイスを制御)
3. フィルタ・ライブラリ (メディアを変換する)

の三種類からなる。

一般には、あるメディア・オブジェクト (メディア・ライブラリのインスタンス) は複数のデバイス・オブジェクト (デバイス・ライブラリのインスタンス) と複数のフィルタ・オブジェクト (フィルタ・ライブラリのインスタンス) を用いて計算機上で実現される。あるメディアの実現方法は複数存在する。例えば、アナログビデオは、レーザディスクを用いても実現できるし、ビデオテープレコーダやビデオカメラを用いる場合もある。しかし、アプリケーションによっては、その実現方法を意識せずに "アナログビデオ" とだけ、あるいは、"動画" とだけ認識したいことがある。このような場合、COSMOS の環境ではアプリケーションは、抽象度の高い (メディア・ライブラリの)"アナログビデオ"(あるいは、"動画") というクラスに属するオブジェクトのみを用いれば、その実現方法を関知しなくてよい。この場合、この "アナログビデオ"(あるいは、"動画") オブジェクトは、例えば、レーザディスク・プレイヤを制御する "レーザディスク・プレイヤ" とアナログビデオの映像を表示する "アナログビデオ・ウインドウ" という二つのデバイス・オブジェクトを内部で制御しているのである。アナログビデオの供給元がレーザディスクではなくてビデオテープレコーダの場合は、"レーザディスク・プレイヤ" オブジェクトを "ビデオテープレコーダ" オブジェクトに置き換えるだけでよい。このように、メディア・オブジェクトとデバイス・オブジェクトの分離は有効である。COSMOS では、アプリケーションは抽象度の高いメディアライブラリを用いることを想定しているが、アプリケーションによっては、デバイス・ライブラリを直接利用する場合もある。マルチメディア・ライブラリは、各クラスごとにダイナミック・リンク可能なモジュール (DLL) として実現され、アプリケーションの実行時に必要なものがロードされる。

### 6.2 組み込みユーザインタフェース

各オブジェクトは、組み込みのユーザインタフェース (Built-in User Interface、以下 BUI) を持つ。あるオブジェクトの BUI は、ユーザのオペレーションを解釈してオブジェクトへメッセージを送る働きをする。例えば、レーザディスク・プレーヤを制御するためのデバイス・オブジェクトは、レーザディスク・プレイヤの操作パネルのような BUI を提供している。このような、BUI がマルチメディア・ライブラリと共に提供されることによって、ユーザは、自分でプログラムを書く事なしに、オブジェクトサーバからロードしてきたオブジェクトを即時に見たり (聞いたり)、操作したりすることができる。その BUI が適当でなければ、ユーザが自分で BUI を書いて、システムに登録することもできる。BUI は、そのオブジェクトが受け取ることのできるメッセージすべてに対応したユーザインタフェースを持っていることが望ましい。

メディア・ライブラリの各オブジェクトは、"openUI" という (標準の) メソッドを持っており、これを呼ぶと画面上にウインドウとして BUI が現れる。"closeUI" というメソッドで、BUI は画面上から消えるがオブジェクト本体はまだメモリ上にある。当然ながら、オブジェクトは BUI を開かなくても直接アプリケーションから呼び出し可能である。

## 7 最初のプロトタイプ

COSMOS 環境に関する研究を開始するに当たり、我々のアプローチの有効性を検証するために迅速なプロトタイプを行ったので、その概要を紹介する。

### 7.1 アプリケーション

マルチメディアに対するオブジェクト指向アプローチの有効性を示すために、アプリケーションとしてマルチメディア電子メールを採用したデモ・システムを作成した。通常の電子メールは文字のみを対象にしているのでメールの受信者はだれでも容易にその内容を表示することができる。ところがマルチメディア電子メールを考えた時、受信側のシステムが考えられる限りのメディアに対応することが現実的ではないことから、送信者が処理可能なメディアと受信者が処理可能なメディアが必ずしも一致するとは限らない。つまり、この種の問題にうまく対処するためにはオブジェクト交換の機能が有効なのである。

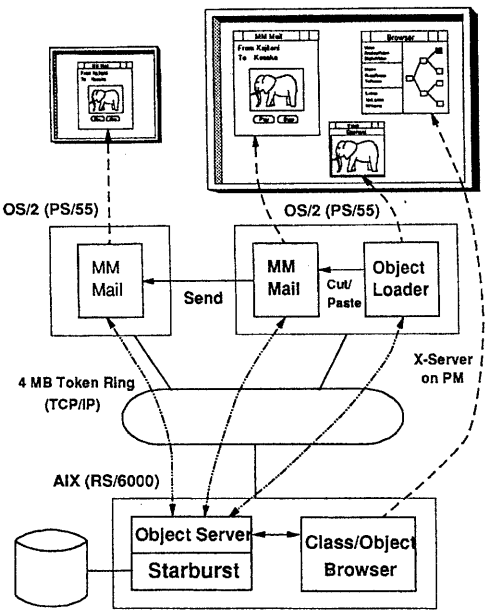



図 9: プロトタイプ・システムの構成

### 7.2 システムの構成

システム全体の構成を図 9 に示し、主な構成要素を以下に説明する。

オブジェクト・サーバ Starburst 上に TCP/IP の RPC を利用した簡単なオブジェクト・マネジャを実現した。ただし C++ インタフェースがないので、後に述べるマルチメディア・ライブラリの記述の中で将来提供するインタフェースの機能を予想して同様の機能を組み込んだ。

クラス・ブラウザ サーバ上で作成し、ユーザ・インタフェースの部分は OS/2 の X-server を利用した。オブジェクト・サーバと密接に関連しており、クラス階層や指定したクラスに属するオブジェクトのリストが表示できる。また、簡単な条件検索も可能になっている。

オブジェクト・ローダ クラス・ブラウザからの指令によって、実際にオブジェクト・サーバからオブジェクトをクライアントにロードし実行 (表示) する。また、カット操作により実行中のオブジェクトの ID を OS/2 ウインドウ・システムのペースト・バッファに転送する機能を持つ。

マルチメディア・ライブラリ メディアとして、アナログビデオと 3 次元グラフィックスを作成した。C++ で記述した OS/2 のダイナミック・リンク・モジュールとして実現しており、BUI も実装している。

マルチメディア電子メール OS/2 上で動作する Smalltalk を利用して作成した。システムのペースト・バッファからオブジェクト ID を受取り、オブジェクト・サーバからオブジェクトをロードし実行することができる。また、ボタンを配置することができ、さらにそれぞれのボタンにスクリプト言語 (Smalltalk) を記述できる。この機能によりオブジェクト・サーバに対して直接オブジェクトを検索したりすることができる。実現上特に工夫したのは、オブジェクト・サーバからロードしてきたオブジェクトを、あたかも Smalltalk のオブジェクトのようにスクリプト言語で記述できるようにしたところである。

実際に送信されるメールには、宛先や文章などの通常のデータに加えてペースト操作により張り込まれたオブジェクトの ID とそれらが配置される領域の情報のみが含まれており、オブジェクトの本体は受信したシステムが改めてオブジェクト・サーバからロードするような仕組みになっている。従って、受信者は受け取ったメールに含まれるオブジェクトに BUI 等を通じて自由にメッセージを送ることができる。例えば、ビデオを再生したり、3 次元グラフィックスのオブジェクトを回転したりすることができる。

## 8 むすび

本稿では、ネットワーク環境におけるマルチメディア・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 COSMOS について紹介した。まず、マルチメディアを対象とするネットワーク環境においてはオブジェクト指向によるアプローチが有用である点について述べた後、本システムの中核であるオブジェクト・サーバについて紹介し、そのマルチメディアへの応用を示した。更に、本稿で示した設計指針に基づき作成した最初のプロトタイプについて概説した。

### 8.1 マルチメディア対応について

本研究を進める上でマルチメディア対応について多くの議論を行ってきが、特に OODBMS を用いた動画像等の連続メディア (continuous media) に対する限界について我々なりの結論が出たので報告する。

ここでは動画像をオブジェクトとし、それを画面上で再生するアプリケーションを例として考える。処理すべきデータ量に関しては、例えば MPEG[1] 方式の圧縮を行ったデータは、画像部分だけで 1.15 ～ 10 メガビット/秒 であるから、高々 1 分間の動画像データでも、69 ～ 600 メガビットすなわち 8.6 ～ 75 メガバイトのデータ量となる。考慮すべきことは、

- クライアントのメモリに動画像データを全て持つことは容量的に困難であること

- 従って、再生中にデータをサーバから、必要な部分が再生される前に、再生用メモリ領域に転送しなければならないこと
- ただし、使用できるメモリ領域が空くまで、次のデータのロードはできないことである。

つまり、従来の CAD 等のアプリケーションでは、データは“なるべく速く”ロード出来ればよかったのであるが、動画像の場合には“再生よりも前に”という条件がつく。

商用 OODBMS の多くは応用分野としてマルチメディアを挙げているが、基本的にはクライアント・マシンのメモリとサーバ・マシンのハードディスク装置間のオブジェクトの転送を必要に応じて行うものだとみなせる。サーバに多くのクライアントが接続している状況を考えると、データのロードをサーバに要求した時点でサーバがクライアントの要求をすぐに処理できることを保証することは、ハードディスク装置の制御方法やクライアントとサーバ間の通信プロトコル等を考えると現在の OODBMS のアーキテクチャでは困難であると思われる。動画像を対象としたサーバに対する技術的考察に関しては現在研究中であり、まとまり次第別途報告したい。

## 8.2 本研究の現状と今後の展望

最初のプロトタイプを作成した段階での主な目的は、実用性を重視した詳細な設計および実現ではなく、必要最小限の構成で迅速な実現を行ない、我々のアプローチの有効性を確認するために多数のアプリケーション開発者および使用者に提示してその反応を調べることにあった。その結果、マルチメディアに対するオブジェクト指向アプローチの有効性およびオブジェクト・サーバによるメソッドまで含めたオブジェクトの共有管理、そしてマルチメディア・オブジェクトのスクリプト言語による簡単な操作に対して好評を得た。ただし、細部までの設計および実現がなされていないため、システム全体のパフォーマンスやマルチメディア対応に対する評価は出来なかった。

本研究は現在第二段階にあり、最初のプロトタイプからの経験を生かしつつ、特に COSMOS 環境において中心的な役割を担うオブジェクト・サーバとマルチメディア・ライブラリに焦点を絞って詳細設計および実現を行っている。今回の報告では間に合わなかったが、パフォーマンスに関する評価ができるようになったら別の機会に改めて報告したいと思っている。

さらに今後の展望 (第三段階？) として、最初のプロトタイプで好評を得たツール群やスクリプト言語の充実も目指したい。また、クライアントーサーバ形式のオブジェクト交換だけではなく、複数のオブジェクトが相互に対等の立場で通信し合うような分散オブジェクト管理なども目標にして行きたいと思う。マルチメディア対応に関しては、ライブラリの充実も重要であるが、特に動画像をサービスするための特殊なサーバについての研究も重要であると考えている。

# 参考文献


[1] I. J. 1/SC29. Coding of moving pictures and associated audio - for digital storage media at up to 1.5 mbit/s, 1991.

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

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

[4] D. J. DeWitt and D. Maier. A study of three alternative workstation-server architectures for object oriented database systems. In *Proceedings of the 16th VLDB Conference*, pp. 107–121, 1990.

[5] G. L. et al. Extensions to starburst: Objects, types, functions, and rules. *Communications of the ACM*, 34(10):94–109, Oct. 1991.

[6] S. Gibbs. Composite multimedia and active object. In *Proceedings of OOPSLA*, pp. 97–112, 1991.

[7] O. M. Group. The common object request broker: Architecture and specification, Aug. 1991. OMG Document Number 91.8.1 Draft 26.

[8] 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.

[9] T. J. Lehman and B. G. Lindsay. The starburst long field manager. In *Proceedings of the 15th VLDB Conference*, pp. 375–382, 1989.

[10] V. Soloviev. An overview of three commercial object-oriented database management systems: Ontos, objectstore, and o2. *SIGMOD Record*, 21(1):93–103, Mar. 1992.

[11] 日経バイト. Windows アプリケーション間連係機構 OLE. 日経バイト, pp. 239–252, Feb. 1991.