1R−10

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

梶谷 浩一 T.H.Fin

日本アイ・ビー・エム株式会社 東京基礎研究所

1 はじめに

オブジェクトサーバを中心としたアプリケーションの開発/実行環境 COSMOS(COmmon Service for Multomedia ObjectS の略) [1] を研究中である。本稿では、COSMOS 環境でのマルチメディア・オブジェクトの実現方法について述べる。

マルチメディアという言葉は、現在様々な意味で使われているが、文字通りとれば、"複数の (多くの) 種類のメディア" という意味である。(本稿ではメディアという単語を、データ型と同じ意味に使う) しかし我々は、マルチメディアのシステムの設計をする上で考慮しなければならないのは "メディアの種類が多い" ということだけではなくて、"メディアの種類が増える" ことであると考えた。例えば、あるアプリケーションでは、N 種類のメディアを扱うことができるが、N+1 種類目のメディアが新たに登場した時に容易に対応できるか、ということである。我々は、COSMOS の環境ではできる限りメディアとアプリケーションの独立性を高めることを目標とした。すなわち、任意のアプリケーションが任意のメディアを扱うことができるような環境を目指した。このような環境では、アプリケーションとメディアを独立して開発することができるのでアプリケーション開発のコストや期間を縮小することができる。また、メディアの開発者はメディアの開発だけに集中できる。

2 マルチメディア・オブジェクトの実現例

アナログ動画のオブジェクトを実際に COSMOS の環境で C++ を用いて試作したが、その時に気づいたことを以下に示す。

[構成と分析] 動画の供給源としてレーザーディスク・プレイヤを用い、RS-232C を通じて制御する。レーザーディスクから再生された動画はビデオオーバーレイ・ボードによってクライアントマシン (PS/55* OS/2* V1.3) の画面上 (あるウインドウ内) に出力する。ハードウェアが決まっているので、ある程度機能も決まってくる。(このようにマルチメディア・データを扱う場合、使用するハードウェアによって機能の一部またはほとんどが決まってしまうことが多い) ここで、動画が本来持っている性質や機能をオブジェクトとして表現したいのであるが、注意しなければならないのは何をオブジェクトとするかということと、各機能はどのオブジェクトが受け持つかということである。

表 1: 各オブジェクトの機能

動画のクラス	最初から再生 一時停止 再生の再開 早い速度で再生 遅い速度で再生 ...
レーザーディスクのクラス	先頭フレームを設定 終了フレームを設定 再生 停止 一時停止 再生速度の変更 現在のフレームを尋ねる ...
動画ウインドウのクラス	明るさ/コントラスト/色合い等を設定 画面の凍結 画面の大きさの設定 ...

アプリケーションの立場で見ると、動画がレーザーディスクから再生されているか、あるいは VTR から再生されてるか、というようなことはあまり重要でないかもしれない。要は、その時必要な動画データが表示できればよい。このことから、動画オブジェクトの中に、ハードウェアを制御するためのコードを含めるよりはハードウェアを制御する部分をオブジェクトとして独立させた方が柔軟性が高いことがわかる。また、アプリケーションによっては、ハードウェアを直接制御して、より細かい制御を行いたいかもしれない。この場合も、ハードウェアを制御するオブジェクトが明確に独立している方が便利である。結果的には、動画のクラス、ある機種のレーザーディスク・プレイヤを制御するためのクラス、動画を画面 (ウインドウ) に出力するためのクラスに分けて試作した。(実際は、各クラスはそれぞれあるクラス階層に含まれている。実際、ハードウェアは機種によって制御の方法が異なる可能性があるので、そのハードウェア全般を表すクラスのサブクラスとして各機種用のクラスを作っ

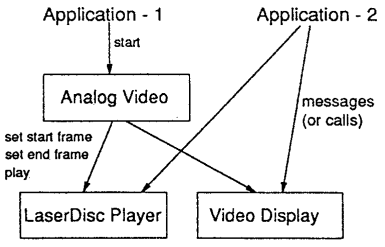

図 1: オブジェクト間の関係

Multimedia Objects in the Object Server
Kouichi KAJITANI T.H.Fin
Tokyo Research Laboratory, IBM Japan

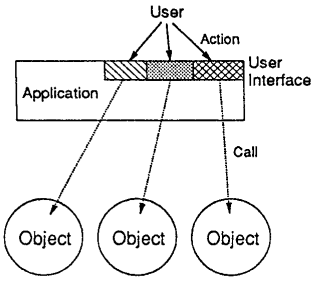

図 2: BUI のない場合

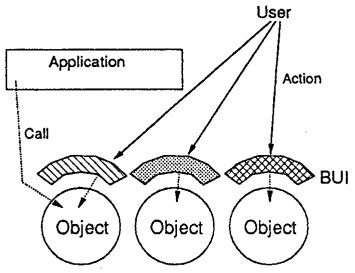

図 3: BUI のある場合

ているが、本稿では、その説明は省略する)

表 1 に、各オブジェクトが受け持つ機能を示す。“動画のクラス” は、他の二つに比べて、より抽象度の高いオブジェクトである。図 1に、各クラスのインスタンスのメッセージの流れを記す。例えば、動画オブジェクト (Analog Video) がアプリケーション (Application-1) から “play”(最初から再生) というメッセージを受け取った場合は、それに対応するメッセージをレーザーディスク・プレイヤ制御用のオブジェクトに転送する。この場合は、“set start frame”(そのビデオデータの先頭フレームを設定)、“set end frame”(そのビデオデータの終了フレームを設定)、“play”(先頭フレームから終了フレームまでを再生) の 3 つのメッセージを送る。

Application-2 は、直接ハードウェア制御用のオブジェクトを使用している例である。

レーザーディスク・プレイヤの代わりに他のハードウェアが接続されていてもよいように、動画オブジェクトの中でレーザーディスクを指している変数の型はレーザーディスク (のクラス) ではなくて、その (クラス階層の) 先祖である “動画データの供給元一般を表すクラス” である。

[共通メッセージ] COSMOS の環境では、アプリケーションとメディアの独立性が高い。逆にいうと、アプリケーションは未知のクラスのオブジェクトを扱う必要がある。これを実現する最も簡潔な方法は、全ての (マルチメディア) オブジェクトに共通なメッセージ (C++の場合は、メンバ関数) を定義することである。試作では、“load”(指定したオブジェクトをアプリケーション領域にをロードする)、“openBUI” (組み込みユーザインタフェース-後述-を開く)、“closeBUI” (組み込みユーザインタフェースを閉じる) 等の共通なメッセージを定めた。“load” は、クラス関数である。アプリケーションはこれらのメッセージを知っていれば、全てのマルチメディア・オブジェクトをその領域にロードし使用することができるが、メディア固有のメッセージまで知る必要はない。

[組み込みユーザインタフェース] 例えば、画面のコントラストや明るさをを変えるような機能はアプリケーションが持つべきではなくて、動画を画面に出力するためのクラスが持つべきである。当然そのためのユーザインタフェースも、そのクラスが持つべきである。このように、クラス固有の機能を実行するためのユーザインタフェースを**組み込みユーザインタフェース** (Built-in User Interface、以下 **BUI**) と呼ぶ。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。例えば、レーザーディスク・プレーヤを制御するためのオブジェクトは、レーザーディスク・プレイヤの操作パネルのような BUI を提供している。

図 2は、BUI が無い場合のユーザの操作 (Action) とメッセージの送付 (Call) の関係を示している。アプリケーションが、各オブジェクトのユーザインタフェースを持たなければならないのでアプリケーション開発・保守の手間がかかり、かつ、新たなメディアに容易には対応できない。(現在のアプリケーション開発コストのかなりの部分はユーザインタフェースの作成に使われる)

図 3に、BUI がある場合を示す。アプリケーションは、BUI がない場合に比べて簡潔に構成でき、かつ、どのようなメディアが現れても共通メッセージ “openBUI” によって、そのメディアの BUI を開くことができる。

BUI によって、ユーザは、自分でプログラムを書かなくてもに、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。また、オブジェクトだけでなく BUI も共有できるのでコードの再利用の面からも有用である。

[組み込みユーザインタフェースの実現] BUI の本体は、OS/2 プレゼンテーション・マネジャ*のウインドウ・プロシージャである。あるウインドウのウインドウ・プロシージャとは、そのウインドウに到着したイベントに対応する処理を行う特別なサブルーチンである。あるオブジェクトの “openBUI” が呼ばれた時に、その BUI のウインドウ・プロシージャをウインドウ・システムに登録する。通常は、そのオブジェクトがロードされたアプリケーションのコンテキスト (context) で実行される。

* IBM 社の登録商標です。

参考文献

[1] 小坂他: オブジェクトサーバとその応用, 情報処理学会 データベースシステム 研究報告 No.89, 1992