システムソフトウェアと
オペレーティング・システム 71－13
マルチメディア通信と分散処理 73－13
（１９９５．１２．１）

Shaped Objectによる情報の分散共有

横田 実
NEC C&C研究所

情報の分散共有のための新しい方式として，オブジェクトのユーザインタフェース部分のみを共有するShaped Object Modelを提案する。本モデルでは同一オブジェクトのＧＵＩ部が複数の遠隔ユーザに送りつけられ，各々独立したプロセスとして実行されることにより，従来より柔軟でカスタマイズ可能な情報の分散共有を実現できる。
本論文ではShaped Objectのコンセプト，その実現に必要な要素技術，Machマイクロカーネルと Tcl/Tkを用いた試作について述べる。

Information Sharing based on Shaped Object Model

Minoru Yokota
C&C Research Laboratories
NEC Corporation

For the flexible information sharing, the new distributed object model, Shaped Object Model has been proposed. The key feature of this model is the separation of the GUI part from the object. Since the GUI part is sent to the remote terminals and executed as a process, each user can handle the object independently while they can share the same body part with other users.
The paper describes its concept, the required software architecture, and the experimental prototype using Mach microkernel and Tcl/Tk.

1 はじめに

インタネットに代表されるネットワーク利用の急速な拡大に伴い、ノートパソコンや電子手帳をネットワークにつないで利用する、いわゆるモバイルコンピュータが注目されている。しかしながらこのようなシステムのアーキテクチャは従来のスタンドアロンコンピュータのそれから全く進化していない。例えばハードディスクの内蔵は小型化や低消費電力化を難しくしている。

一方、このようなモバイルコンピュータを利用するためのソフトウエアに関してもパソコンと同じＯＳ／応用ソフトを利用するか、専用ソフトで構築された電子手帳タイプのシステムのいずれかしかなく、グループウエアのようなネットワークを利用した情報の分散共有のためのプラットフォームは確立されていない。

単体の分散応用システム構築のためにはクライアント／サーバシステム方式が定着し、具体的なソフト環境としてDCE[1]やCORBA[2]が提唱され利用が進んでいるが複数のクライアント間の連係処理はプログラマに委ねられている。

X Window Systemはクライアント／サーバ方式の採用で応用プログラムのGUI画面をネットワークワイドで共有可能にした。しかしながら情報共有がウインドウレベルであるため、ユーザ毎に異なるビューを提供する等の柔軟な共有が難しい。パソコンではXのようなクライアントサーバ型ウインドウシステムは存在しない。OLE等により異なる応用システムを一つのシステムに組込むメカニズムは実現されつつあるがネットワーク分散はまだこれからである。

本研究はこのような現状の問題点を解決し、誰にでも簡単に、しかも柔軟に情報共有ができるソフトウエア環境の実現を目指したものである。本論文ではそのための新しいオブジェクトモデルとしてShaped Object Modelを提案し、その試作について報告する。

2 Shaped Object Model

2.1 GUIの分離

基本的なアイデアはオブジェクトの外観（ビュー）とオブジェクトが本来実行する処理の分離である．更に言えばウイジェットの表示とイベント処理の分離となる。以降，本モデルでは各々をオブジェクトのシェイプとボディと呼ぶ。シェイプとボディはネットワークで常に結合されており，この通信チャネルを利用してメッセージを相互にやり取りして処理を進める。この双方向通信チャネルをネットワーク流にバーチャルサーキット（VC）と呼ぼう。

図1にシェイプ，ボディ，バーチャルサーキットの関係を図示する。

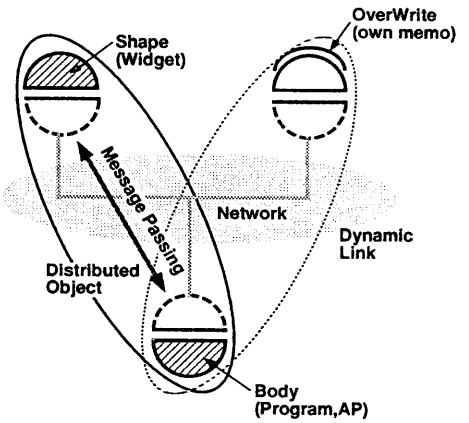

図 1: Shaped Object

オブジェクトは遠隔マシンからアクセスされるとシェイプを転送する。シェイプは糸のついた凧のようにバーチャルサーキットを伸ばしながら遠隔マシンへ辿りつく．シェイプは遠隔マシン上でウイジェットとなり，オブジェクトのGUIそのものを具現化する．ボタンが押されたり，文字が入力されると，対応するウイジェットがイベントを生成しバーチャルサーキットを介してボディに送り，ボディ側で必要な処理が行なわれる。

本モデルの利点の一つはＧＵＩの生成や制御がかなり利用者側のマシンにオフロードできる

ことである．例えば，ボタンが押された場合に「ボタンが引っ込む」という３Ｄ効果はサーバ側ではなく，利用者側のマシンでローカルに処理すれば良い。即ち，ユーザインタフェースに関わるきめ細かい応対を分散させることができる。しかもこのＧＵＩはサーバ側から転送されるのであらかじめユーザのマシンに組み込んでおく必要がない。

もう一つの利点はこのような形でボディを共有し，シェイプを分散させることにより，自由度の高い分散共有が実現できることである。即ち，共有しているオブジェクトのＧＵＩは各マシンで独立に動作しておりユーザはオブジェクトに対して勝手な操作をすることができる。一方，各ユーザからのイベントは一つのボディで管理されているので，操作結果は全員で共有可能である。

更に，ボタンの位置の変更や上書きのような個々のユーザによるカスタマイズもある範囲で許すことができる。また，必要なら，全員が同じ画面を見ることも可能であり、分散プレゼンテーションにも有効である。

このように本モデルにより、オブジェクトの操作に対してはユーザ毎の独立性を許しながら，一貫性を保った共有が実現可能となる。

2.2 スクリプト言語での実現

では，このようなシェイプの実現方法であるが，柔軟性、互換性、安全性の３つの理由から筆者はスクリプト言語による実現方法を採用した。

2.2.1 柔軟性

インタプリタ言語はテキストベースのソース記述を読み込んで即実行可能であり，非常に柔軟である。また，シェイプとボディのネットワーク結合の観点からはスクリプトをストリームデータとして入力し，リアルタイムにインタプリタで解釈することが可能である。コンパイラ言語では一度貯め込んでからコンパイル／実行というステップが要求される。デバッグ時も含めて，インタプリタの動的実行，柔軟性は大きな利点である。特にここで対象としているのはＧＵＩの実現や画面操作というユーザインタフェースであり、このような高いレベルの処理にはスクリプト言語が最適である。

2.2.2 互換性

情報を共有したい相手はワークステーション，PC，携帯端末など様々であり使用されているプロセッサもＯＳも異なる。このようなヘテロジニアスな環境での情報共有を可能にするにはバイナリコードのようなマシンに依存するデータ形式は使えない。各マシンが共通のインタプリタをインストールし，共通言語を解釈実行できて初めて実現可能となる。

2.2.3 安全性

シェイプの実行とは，もともと存在しなかったコードが他のマシンから転送されてきて実行されるということである。ある意味でワームと同じメカニズムを必要としている。バイナリコードはたとえ悪意が無くてもバグや互換性の違いでレジスタやメモリ領域を破壊する危険が常に存在する。このような何をするか不明なデータに実行を許すわけにいかない。インタプリタ言語なら入力はASCIIテキストであり，ユーザ側のインタプリタが危険なコマンド（例えばremoveなど）を解釈しないようにガードをかけることができる。

3 実現のためのソフトウエアアーキテクチャ

スクリプト記述による Shaped Object Model の実現には図2に示すようなソフトウエアアーキテクチャが必要である．

キーとなるコンポネントは以下の３つである。

- オブジェクトエンジン
- ディスプレイサーバ
- ネームサーバ

3.1 オブジェクトエンジン

オブジェクトエンジンとは、スクリプトで記述されているオブジェクト（シェイプ&ボディ）

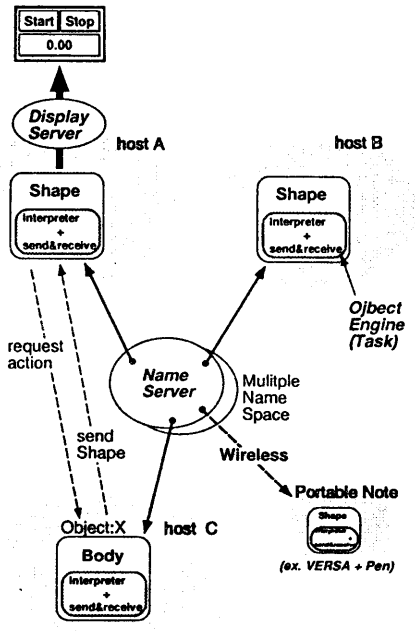

図 2: Software Architecture

を実体のあるプロセスとして動かすものであり、メッセージの送信／受信機能を持ったインタプリタで構成されている。

オブジェクトエンジンはあらかじめプロセスとして起動されており、ネットワークを介してスクリプトストリームを入力するとシェイプもしくはボディに変身する。

3.2 ディスプレイサーバ

実際の画面表示を行なうサーバであり、従来のウインンドウシステムに相当する。但し、本モデルでは「ウインドウ」という概念は存在する必要はなく、従来のウインドウシステムよりもっと簡素化された、ウイジェットを表示するシンプルな機能があればよい。そういう意味でここではあえてディスプレイサーバという名称を使用する。

3.3 グローバルネームサーバ

遠隔マシンから共有オブジェクトにアクセスするためには、目的オブジェクトの所在を知る手段が必要となる。このためにオブジェクト名とオブジェクトID（ポインタ）の対を管理するネームサーバが必要となる。

最もシンプルなネームサーバは単一名前空間を提供するものであるが、プロジェクトやグループ毎に自由に名前を定義したいので、ここではLAN/WANを通じて多重名前空間を提供するグローバルネームサーバを考える。名前を特定の名前空間に閉じることはプロテクションの一手段でもある。

3.4 オブジェクトの生成・共有

実際のオブジェクトの生成・共有の流れを述べる。尚、ボディ側をスクリプト言語で実現しなければならない絶対的な理由はないが、ここでは実現が容易なようにシェイプと同じスクリプトで記述することを考える。従って、ボディの生成もオブジェクトエンジンによって行なわれる。

1. まず、オブジェクトエンジンをプロセスとして生成し、これに目的オブジェクトのスクリプトを送信することによりオブジェクトが生成される。ユーザからの参照を可能にするためオブジェクト名とオブジェクトＩＤをネームサーバに登録する。

2. 一方、オブジェクトにアクセスしようとするユーザはネームサーバを検索して目的オブジェクトのオブジェクトＩＤを入手する。

3. 次にシェイプとなるオブジェクトエンジンを生成し、目的オブジェクトへ初期化要求を送信する。その際に、自分のオブジェクトＩＤをメッセージ中に埋め込んでおく。

4. 初期化要求メッセージを受けとったボディオブジェクトは、メッセージ中に埋め込まれているオブジェクトＩＤに向けてシェイプ用スクリプトを送信する。

5. シェイプ用スクリプトを受信したオブジェクトエンジンはスクリプトを解釈して必要なウイジェットを生成する。

以降の動作はユーザによるウイジェット操作とそれによって起動されるウイジェットに定義された動作（例えば、ボタン押下でボディへ特定メッセージを送る、等）によって遂行される。

4 プロトタイプ試作

4.1 ソフトウエア環境

本モデルを検証するのにTcl／Tkスクリプト言語[3]とMachマイクロカーネル[4][5]を利用した。

Tcl／TkはX用のウイジェットをインタプリティブに簡単に作成できることを特徴とするツールで、本モデルでの「シェイプをスクリプト言語で記述する」という目的に合致している。

一方、シェイプとボディは各々プロセスとして実行されねばならないこと、しかもボディはネットワークを介して複数のシェイプとメッセージのやりとりができなければならないこと，からMachの提供するメッセージによるプロセス間通信機能を利用することにした。Machではプロセス間通信が全てメッセージ転送によって行なわれるが、その受け取り口はポートという概念で実現されている。ポートはタスクにつながれた入力ホースのようなもので、複製して別のタスクに渡すことにより、複数タスクからのメッセージを受信することが可能になる。ポートがネットワーク透明であることもMachの優れた特徴であり、これにより遠隔ホスト上のプロセスにも、あたかもローカルなプロセスと同じインタフェースでメッセージを送ることが可能である。このようなMachのプロセス間通信機構は本モデルでのメッセージ通信に必要な機能をほぼ満たしている.

ネットワーク透明性を実現するのに必要となるのがネームサーバであり、Machには基本機能を提供する簡易ネームサーバNetMsgServer[6]が一緒にリリースされている。NetMsgServerは単一名前空間しか提供していない等の不便さはあるものの今回の試作にはこれを利用した。

まとめると、実際の試作は以下のようなソフトを利用して実現された。

- オブジェクトエンジン
  ＝＞ Tcl/Tk インタプリタ
  ＋ Mach プロセス間通信
  ＋ 送受信ルーチン
- ディスプレイサーバ
  ＝＞ X Window System
- ネームサーバ ＝＞ NetMsgServer

4.2 応用プログラム：会議開催通知

図3はShaped Objectにより会議開催通知を実現した例である。

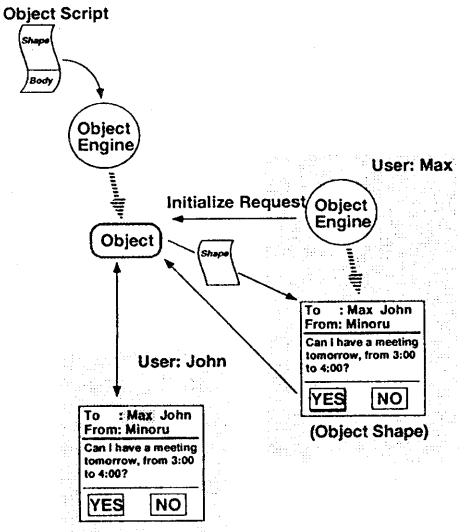

図 3: Meeting Notice Example

まず、オブジェクトエンジンを起動し、会議開催通知のスクリプトを入力するとそれが開催通知を一括管理するサーバオブジェクトとして振舞う。

一方、スクリプトのシェイプ部分は出席予定者に送付され、各自のマシン上で開催通知のイメージを表示する。ボディとシェイプは2本のポートで結合されており，ユーザが出席ボタン

を押下するとボディ側に「出席」の情報が伝達される。ボディの受信ポートは複数ユーザに共有されている。ボディは出欠等の情報を受け取ると，名前部分の背景色を変えるなどのウイジェットの属性を変更するために必要なスクリプトを全シェイプにブロードキャストし，これにより出席予定者全員が、誰が開催通知を見たか、誰が出席するのか、を知ることができる。また、ユーザが開催通知を開く度に最新のスクリプトが転送されるので、常に最新情報を表示することができる。

4.3 評価・議論

会議開催通知はウイジェットで実現し易い例であり，Tcl/Tkで記述されたスクリプトは１００行程度でネットワークの負荷もそれ程大きくない。ボタン押下等の対話処理における実際の転送データ量は百バイト程度であり，応答はスムーズである。今後，より本格的な情報サービスへの適応により，実用レベルでどの程度の転送量になるのかを検証する必要がある。

マルチキャスト機能については，試作ではTclのスクリプトで記述しており，ボディを共有しているユーザ数などを管理している。基本機能はプロセス間通信のレベルでサポートすべきと考えるが，ユーザ数の変動に対処するにはShaped Objectとしてどのレベルを分担すべきかの切り分けにはまだ議論が必要である。特に回線断などのエラー処理に関しては今後の課題である。

5 関連する研究

オブジェクトのビューを分離する方式はSmalltalkにおけるMVC(Model View Control)が有名である[7]。Shaped Object ModelをMVCの拡張と見ることもできるが、GUIの分離を目的としており、むしろX-Terminalのコンセプトに近い。スクリプトを転送するという考え方ではGeneral Magic社のTelescriptによるエージェントの実現があるが、そこではエージェントの一人歩きを目的としているが、Shaped Object Modelではオブジェクトの共有を目的としており、シェイプとボディの間が通信チャネルで結ばれていることこそ実用性があると考えている。

6 おわりに

本論文では情報共有の方法としてユーザインタフェースのみを独立させ、共有する方法としてShaped Object Modelを提案した。更に、Tcl/Tk、Mach Microkernelを用いた試作について述べた。提案した情報共有方式はネットワーク利用を前提とするこれからの携帯端末，情報サービス端末のアーキテクチャとして有効であろう。

謝辞

本研究を支えてくれたNEC北米C & C研究所システムアーキテクチャグループメンバに感謝する。特にDr. Maximilian Ott氏には討論を通じて貴重な意見を頂いた。また，本研究の機会を与えて頂いた渡辺孝次郎同研究所所長に深謝する。

参考文献

[1] John Shirley, *Guide to Writing DCE Applications*, O'Reilly & Associates, Inc., 1992

[2] OMG, *The Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.8.1 (Draft), August 1991

[3] J. Ousterhout, *Tcl: An Embedded Command Language*, Proc. of Winter USENIX Conf., January 1990

[4] R. Rashid, et.al., *Mach: A New Kernel Foundation for UNIX Development*, Proc. of Summer USENIX Conf., July 1986

[5] J. Boykin, D. Kirshen, A. Langerman, and S. LoVerso, *Programming under Mach*, Addison Wesley, 1993

[6] MACH Networking Group *Network Server Design*, Mach Technical Document, CMU, Aug. 31, 1989

[7] G.E. Krasner and S.T. Pope, *A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80*, Prac Place Systems, 1988