データベースシステム 91－9
（1992. 11. 5）

分散オブジェクト指向シェルHyperShellでの共有オブジェクト管理方式

岩崎 未知, 岡本 道子, 新　　淳, 川越 恭二

NEC C&Cシステム研究所

HyperShellは、分散オブジェクト指向マルチメディア・システムであるHyperStation上のソフトウェア構築環境である。HyperShell環境ではユーザが利用する各種情報は共有オブジェクト空間中のオブジェクトとして管理され、オブジェクトに対するネットワーク透過なアクセスや複数ユーザによるオブジェクトの共有を容易に行なうことが可能となっている。本稿では、HyperShellの分散機能検討用プロトタイプでの共有オブジェクト実現方式と、共有オブジェクトを用いた電子メールシステムについて説明する。

A Synchronization Mechanism of the HyperShell Distributed Object-Oriented Environment

Michi IWASAKI, Michiko OKAMOTO, Atsushi ATARASHI, Kyoji KAWAGOE

C&C Systems Research Laboratories,
NEC Corporation

HyperShell is an object-oriented programming system for distributed environment. It offers "shared object space", so that users can share objects and access them without knowing their actual location. It also enables users to manipulate shared objects simultaneously. This paper presents the outline of HyperShell and an implementation of the shared object space in a HyperShell prototype. An electronic mail system based on shared objects is also described.

はじめに

ワークステーションやネットワークの利用の普及により分散環境における複数のユーザ間での情報の共有や交換は徐々に容易なものとなってきている。しかし一方で、ネットワーク上での当該情報の参照、分散環境で動作するアプリケーションの作成、テキスト以外の情報の作成や操作等は依然として困難である。これに対して、我々は個々人の能力を最大限に引き出す様な計算機環境の構築を目標に、計算機利用者の通信 (Communication)、共同作業 (Collaboration)、創造活動 (Creation) を支援するための次世代計算機環境として、分散オブジェクト指向マルチメディア・システム HyperStation の研究開発を進めている [5]。本稿では、このシステムの上のオブジェクト指向環境 (シェル) である HyperShell[2][3] の概要と、分散オブジェクト機能実験用プロトタイプでの分散共有オブジェクト実装方式、及び、共有オブジェクトを用いた電子メールシステムとその試作システムに関して説明する。

1 HyperShell の設計目標

マルチユーザ・アプリケーション向けのインフラストラクチャには、シングルユーザの場合と異なるいくつかの機能が要求される [10][4]。HyperShell では、エンドユーザやプログラマが分散環境で各種オブジェクトを容易に作成、利用できるよう、以下の機能を設計目標としている。

- オブジェクトを画面上で視覚的に操作可能であること
- ネットワーク透過なオブジェクトへのアクセスが可能であること — ユーザがネットワークを意識し、その上の特定の位置を陽に指定する必要のないこと。
- 分散環境でのマルチユーザ・アプリケーションの構築が容易であること
- マルチメディアの利用が容易であること
- オブジェクトが永続性を持つこと — ユーザがオブジェクトの保存を陽に指定すること無しに、アプリケーションの状態が保存されること。

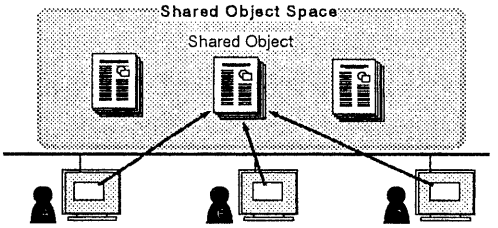

図 1: 共有オブジェクト空間

- 自律的に動作するオブジェクトの作成、利用ができること

また、全体としてユーザにとって快適な使用感を提供することを目標としており、例えば、スループットよりはユーザの操作に対するレスポンスを重視する。

2 HyperShell の概要

2.1 HyperShell 環境の特徴

HyperShell は、上述の様な目標を満たす環境を実現すべく、以下の機能 / 特徴を提案している。

- すべてのオブジェクトは原則として外観を持つ
- 共有オブジェクト空間の提供 — 後述 (「2.2 共有オブジェクト空間」参照)。

また、環境として以下の機能を提供する。

- アプリケーション構築用オブジェクト・ライブラリ、オブジェクト・エディタの提供
- マルチメディア・オブジェクトの提供
- 永続オブジェクト・サービスの提供
- 自律オブジェクト・サービスの提供

HyperShell の枠組自体に関しては、本稿では、以下その概要に関してのみ説明する。

2.2 共有オブジェクト空間

HyperShell では、ユーザにネットワーク透過な情報空間を提供するために共有オブジェクト空間を実現する。ここでの共有オブジェクト空間のイメージは、[図 1] の様なものである。 ユーザは、実際には分散して存在しているオブジェクトをあたかも

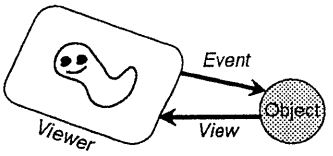

図 2: オブジェクトと Viewer

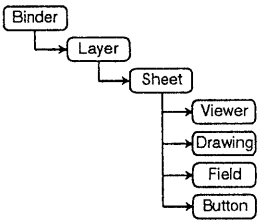

図 3: Binder オブジェクトを構成するオブジェクト

単一の空間に存在しているものであるかのように操作することができる。

オブジェクトは外観を持ち、各ユーザはこれを Viewer というオブジェクトを通して覗き込み、共有した状態で操作する ([図 2])。 Viewer を通して覗くオブジェクトとしては、我々が長年慣れ親しんで来た紙と本のメタファ [1] に基づく Binder オブジェクトを提案している。

2.3 Binder オブジェクトの構成

Binder オブジェクトは本のメタファを実現するオブジェクトであり、その構成要素として紙のメタファを実現する Sheet と Layer、及び紙の上に載せる Viewer、Drawing、Field、Button 等がある。これらオブジェクトの階層関係を [図 3] に示す。

それぞれのオブジェクトの役割は、以下の通りである。

- **Sheet**　後述の Viewer、Drawing 等のオブジェクトを貼り付けるための「台紙」の役割を果たす。
- **Layer**　何枚かの Sheet を重ねたもの。本の 1 ページに相当する。異なる Layer に共通の Sheet を挟み込むことにより、外観の共有やイベント処理のためのスクリプトの共有等が可能となる。
- **Binder**　複数の Layer を製本したオブジェクト。

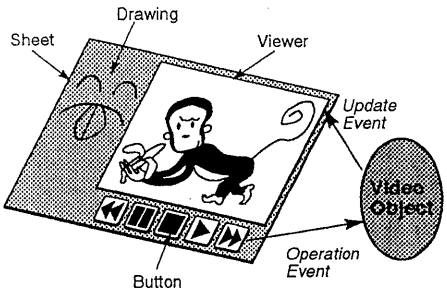

図 4: Sheet の例

Sheet 上に載せることのできるオブジェクトとしては以下の様なものを用意する。

- **Viewer**　他のオブジェクトを覗き込むためのオブジェクト。
- **Drawing**　グラフィカルな情報を表現するためのオブジェクト。
- **Field**　テキスト情報を表示するためのオブジェクト。
- **Button**　ユーザからのマウスを中心としたイベントに対して、実行すべきスクリプトを貼り付けるオブジェクト。

これらオブジェクトを組み合わせた利用例を [図 4] に示す。 ここでは Sheet の上に、Drawing、ビデオ・オブジェクトを覗き込んでいるビデオ再生用 Viewer、ビデオ・オブジェクトを操作するためのスクリプトが貼り付けられた Button 群が配置されており、これらによって、VTR のメタファが実現されている。

2.4 Binder の計算メカニズム

Binder オブジェクトは、ユーザからのマウス / キーボード等からの入力イベントを非同期に処理する。Binder オブジェクトがこれらイベントを検出すると、イベント発生位置と現在その近傍に存在するオブジェクト群との位置関係からメッセージの伝達順序を計算し、最初のオブジェクトにメッセージを伝える。処理がそのオブジェクトのみで完了しない場合には、オブジェクトの上下関係から定められ

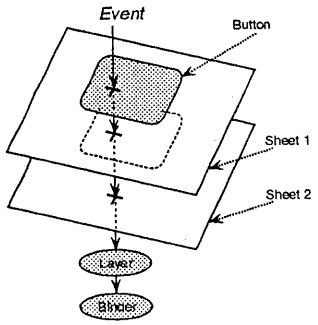

図 5: Binder オブジェクトの計算メカニズム

る伝達順序に従い、順次、次のオブジェクトへと伝えられる。[図 5] は、この伝達順序を示したものである。

3 プロトタイプの構成

3.1 制約条件

今回の HyperShell プロトタイプ [8] は、分散環境において複数のユーザが Binder を共有し、同時に編集作業を行なう実験を可能とすることを主な目標として設計されている。Binder を構成する個々のオブジェクトのマイグレーションは想定していない。Binder 単位での移動は可能であるが、当該 Binder 利用中には移動できないものとした。分散環境としては LAN 上での利用を想定している。システムの構成としては、既存の計算機環境との連続性を持ち UNIX 等の既存の計算機環境からの移行が容易である様、UNIX 上の通常のアプリケーションとして実現するものとした。また、新たな計算機環境の枠組の実験が容易であることを重視し、エンドユーザに見せる環境のモデル、あるいは実装方式 / 制御方式を容易に変更可能とすることも目標とした。

3.2 プロセス構成

構成を [図 6] に示す[1]。各モジュールは以下の様な機能を備える。

[1] 本構成は現在のプロトタイプで用いているものであり、HyperShell 本来のモデルとは異なる部分が存在する。

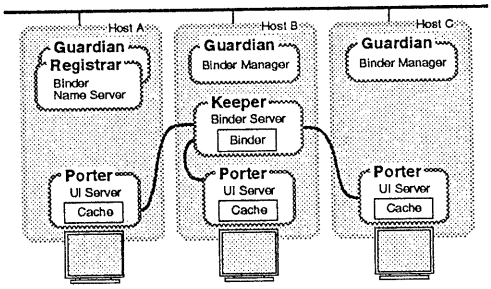

図 6: プロトタイプの構成

- **UI サーバ**　UI 実現部。次に述べる Binder サーバ内部のオブジェクトをキャッシュし表示を行なう。また、ユーザからの操作を受け付ける。以下 Porter と呼ぶ。
- **Binder サーバ**　Binder に対応して存在する。オブジェクトを管理する。以下 Keeper と呼ぶ。
- **Binder マネージャ**　ホスト毎に存在する。そのホスト上に存在する Binder を管理し、必要に応じて Keeper を起動する。以下 Guardian と呼ぶ。
- **Binder ネームサーバ**　LAN 上に存在する Binder を管理する。Binder 利用要求に対して、対応する Guardian の所在位置を返す。以下 Registrar と呼ぶ。

この構成は、Porter-Keeper 間に抽象度の高いプロトコルを利用しユーザの操作に対するフィードバックを極力 Porter で実現することによる Keeper の負荷軽減、ユーザの操作に対する高速なレスポンスの実現、特定のウィンドウシステムへの依存部の分離等を狙ったものである。

各種オブジェクトの表示等は Porter が行なっており、Keeper は、Porter 内でのインスタンス生成に必要な情報のみを Porter に渡している。ユーザの操作に対するフィードバックも、それが Keeper 内オブジェクトの状態を変更しないものであれば Porter でローカルに処理できる。また、両者の間での通信に HyperShell 独自のプロトコルを用いることにより、Binder のプログラミングは特定のウィンドウシステムに依存したプリミティブを使用すること無しに可能である。すなわち、Porter はウィンドウシステムの隠蔽をも行なっている。

4 分散共有オブジェクトの管理方式

4.1 Binderの存在位置の隠蔽

実際には特定のホスト上に存在しているBinderを共有オブジェクト空間中に見せるため、Binderが存在する位置をユーザから隠す必要がある。プロトタイプでは通信にTCP/IPを用いているので、Binderが存在するホストのIDとBinderに対応して同じホスト上に存在するKeeperが持つ通信用のポート番号がこの存在位置に相当する。この隠蔽はRegistrarとGuardianの連係により実現される。Guardianはこの他、Keeperが走行中か否かの管理、及び走行していない場合の起動等を行なっている。Porterによる Binder検索に対しては、最終的には常にKeeperが接続を待つポート番号が通知される。これによりユーザはBinderの位置を知る必要がなく、位置とは独立した名前体系を用いてBinderを管理することも可能となる。また、各ユーザは、共有されたBinderの利用の開始や終了をいつでも自由に行なうことができる。

4.2 オブジェクトID

オブジェクトに対するID付与の方式は、多くの場合直ちにシステムの機能に対する制約となる。例えば、ホスト名を含む形式のオブジェクトIDを採用するとオブジェクトがホスト間を移動する際にIDを付け替える等の何等かの処理が必要となる。

今回は、前述(3.1)の制約条件から、オブジェクトIDは単一のBinder内での一意性のみを保証するものを採用した。具体的にはオブジェクト生成順にシリアルナンバを付与し、これが再利用されないことのみを保証している。

4.3 分散共有オブジェクトの実現

ここでの分散共有オブジェクトの実装は文献[6]のシステムのものを拡張しており、必要に応じてオブジェクトのレプリカを作成し、それらを連動させるものである。すなわち、いくつかのオブジェクトがホスト間でキャッシングされる形となる。

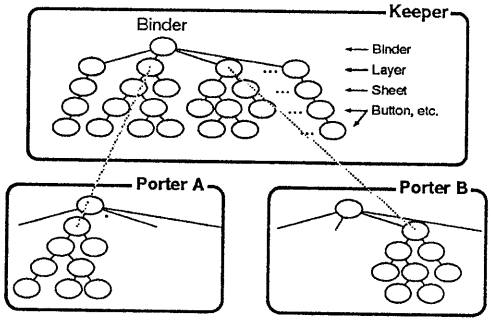

図7: オブジェクトのキャッシング

Binderは[図7]の上部の様な構造で実現されている。 オブジェクト管理は文献[7]の枠組を利用した木構造としている。オブジェクトのキャッシングはLayer単位で行なわれる。ある時点でPorterが表示する画面は一つのLayerに対応し、Layerをrootとする部分木に含まれるオブジェクト群の総和に相当する。Keeperは各Porterがどの Layerをキャッシュしているかを管理しており、オブジェクトの操作が行なわれた場合、その情報を必要とするPorterに通知する。

4.4 一貫性維持方式

4.4.1 基本方式

制御はKeeperでの集中実行方式である。オブジェクト操作の輻輳に対する一貫性の維持には基本的には以下の様なアルゴリズムを用いた制御方式を用いている。これは鼎談システム[14]での制御方式の内、明示的なロックを用いないものと同等なものであり、ユーザの操作に対するレスポンスを重視したものである。

1. Porterで変更操作が行なわれると、Porterは、変更要求としてKeeperに変更内容とPorter内でのタイムスタンプを送り返事を待つ。

2. Keeperは自分のタイムスタンプと比較する。同一であれば要求を受理する。タイムスタンプを更新し、操作対象となったオブジェクトをキャッシュしているPorterのそれぞれに対

して新しいタイムスタンプと共に変更内容を通知する。一方、Porterからのタイムスタンプが Keeper のものと異なる場合には、その要求を拒否する。この場合オブジェクトの操作は行なわれない。

3. Porterは、Keeperからの変更通知は無条件に実行し、タイムスタンプを更新する。この処理は、Keeperからの返事を待っている間にも行なわれる。

4.4.2 メソッド実行順序

HyperShellでは、編集対象のもつメソッドが実行されることがあること、さらにそのメソッドの中から別のオブジェクトのメソッドが起動され得ること等から、ユーザによるオブジェクトの操作とそれに起因する一連のオブジェクトの状態変更が短時間で完了しない場合が存在する。操作に対する快適なレスポンスを実現するためにはユーザの操作に対する応答処理を優先的に行なうことが望ましいが、実行中のメソッドを無条件に中断するとシステムの一貫性が失われる恐れがある。Keeperの実行形態としては、例えば、「一連のメソッド実行が終了するまで実行を中断しない」、あるいは、「Keeperをマルチスレッドとし Porterからのリクエスト毎にスレッドを与え、必要に応じて排他制御を行なう」、といったものが考えられる。またマルチスレッドの場合、さらに、「各スレッドはプライオリティを持ちタイムスライスも行なう」、「各スレッドは、少し実行しては自発的に制御を手放す」、といった選択も存在する。今回は、実現が容易であることからBinder内オブジェクトのメソッドは基本的にKeeper内でシングルスレッドで実行することでシステムの一貫性を保ち、これを Porterからの要求の処理に優先して処理する方式とした。よって、Keeperの状態によっては Porterからの要求は待たされることになる。ただし、Keeper内でのメソッド実行によるオブジェクトの状態変化は、その都度 Porterに通知している。

4.4.3 オブジェクト毎のタイムスタンプ保持

今回のプロトタイプでは、オブジェクト毎にタイムスタンプを備えている。タイムスタンプをオブジェクト毎に保持することにより、同一画面上で同時に操作が行なわれても操作対象が異なれば変更を受けなかったオブジェクトのタイムスタンプは更新されないため、タイムスタンプが古いといった理由での要求の拒否が発生する度合を少なくすることができる。

構造を持ったオブジェクトの操作に際してはオブジェクト管理のための階層構造の中で子供のオブジェクトに変更が及ぶ場合があり、この場合はタイムスタンプも順次更新する。オブジェクトの生成、破壊を行なう際は、その親となるオブジェクトのタイムスタンプを使用する。

4.5 永続オブジェクト機能の実現

4.5.1 Binderの内容保持

Keeperが、その起動時と終了時に Binderの状態のファイルへの保存や復旧を行なうことで疑似的に永続オブジェクトを実現し、ユーザが明示的に保存/復旧を指示しなくとも済むようにしている。

4.5.2 揮発性資源の管理

UNIX上で X-Window等のウィンドウシステムを用いて永続的なオブジェクトを実現しようとした場合の問題点の一つに、プロセス終了時にポインタや Windowの IDの様な揮発性(volatile)の資源が失われ、次回プロセス起動時に復旧できないという問題がある。

HyperShellプロトタイプでは、X-Windowの資源としての Windowは不可視となっており画面上へのオブジェクトへのアクセスはすべて HyperShellの世界のオブジェクトに対するアクセスという形で行なわれる。つまり Window資源は HyperShellの世界のオブジェクトとしては存在せず、Buttonや Fieldといったオブジェクトの中に隠蔽されている。このため次回プロセス生成時に Windowを作成し直し、それが前回と異なる IDとなっ

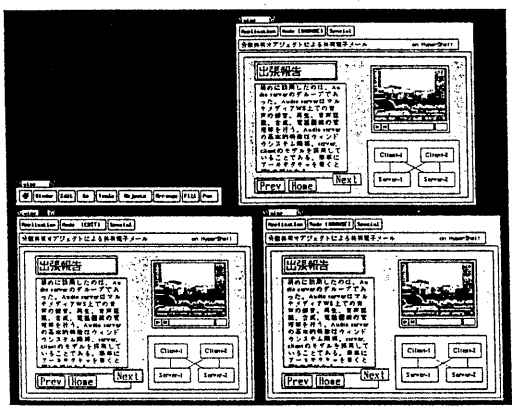

図 8: HyperShell の利用画面例

ても問題は発生しない。

Binder 内の各種オブジェクトへのアクセスに関しても、ポインタを用いたアクセスを一部に限定する一方、HyperShell で提供する ID やユーザ定義可能な名前をオブジェクトに付与し、これを用いてアクセスを行なうことで問題を回避している。

4.6 マルチメディア・オブジェクトの実現

図形等の利用は、鼎 [13] の各種エディタ部品を HyperShell 上のオブジェクトとして導入することで行なっている。動画の表示に関しては、Media-Viewer-Context モデル [15] [17] を用いており、マルチメディア・エディタ mbuild [15][17] で作成した動画像データを再生可能となっている。

5 動作例

HyperShell プロトタイプの動作中の画面を [図 8] に示す。1 つの Binder の同一の Layer を 3 人のユーザで共有している。

プロトタイプは UNIX 上に X11R4、X-Toolkit、鼎 [13] を用いて作成されており、単一の Binder の複数ユーザによる共有、複数ユーザによる同一の、または異なった Layer の操作、Button の様な動作を持ったオブジェクトや Field 等のオブジェクトの画面上での作成、操作等が可能となっている。ウィンドウ右上のオブジェクトには動画が表示されており、画面上で再生 / 停止の制御が可能な他、オブジェクト下部のスライダで特定の表示フレームを指定することもできる。これらの操作も複数ユーザ間で共有可能である。

6 共有オブジェクトを用いた電子メール

HyperShell 上のアプリケーションとして、共有オブジェクトを用いた電子メールシステムを検討、試作中である [11][12]。ここでは、本電子メールシステムの枠組と HyperShell プロトタイプ上での試作システムに関して説明する。

6.1 設計目標

既存の電子メールシステムの多く、例えば UNIX 上の電子メールには以下のような問題点が存在する。

- 送信者は、送信後のメールが如何に処理されるかに関与できない
- メールの整理機能、検索機能が貧弱
- メール上での議論の支援が不十分
- テキスト以外のメディアの利用が困難

これに対して、本メールシステムは以下の様な手法によりこれらの問題の解決を図っている。

- 共有オブジェクトをメールとして用いる —複数のユーザでメールを共有可能。また、送信者は送信後のメールの操作が可能。
- 共有オブジェクト空間を覗いて、メールを整理された形に見せるオブジェクトを提供 —特定の視点に基づいて整理されたビューを複数提供可能。
- 複数のオブジェクトでメールを構成 —共有されたメールに対する各ユーザの編集過程を Hypertext 的構造として管理可能。
- 様々なメディアを実現するオブジェクトを利用可能

共有オブジェクトとしてのメールは、送信者、受信者の間で共有されており、メールへのアクセス権の授受が通常のメールシステムでの送信や受信に相当する ([図 9])。

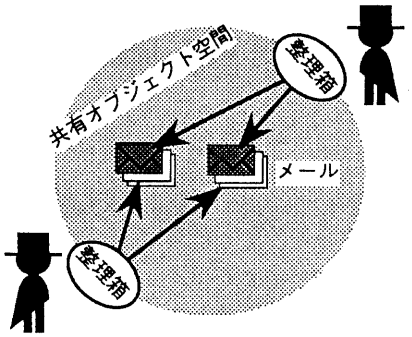

図 9: 共有オブジェクトを用いたメール

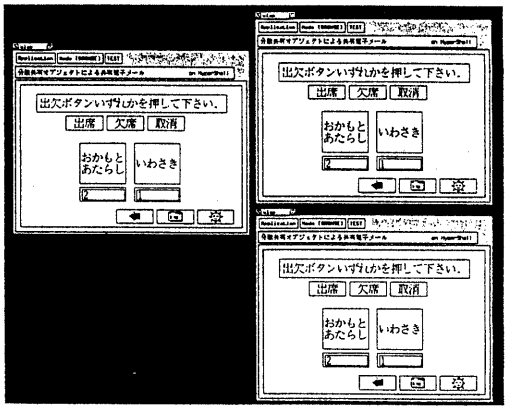

図 10: 出欠の返事とその集計

6.2 メールシステムの構成

6.2.1 メールを構成する主なオブジェクト

メールを構成するオブジェクトには主に以下のものがある。

- **便箋** 様々なオブジェクトをこの上に添付するためのオブジェクト。
- **封筒** 宛先、差出人、題、日付、メールの有効期限、開封確認指定の有無、緊急度などを指定するためのオブジェクト。
- **メール** 複数の便箋を Hypertext 的に構成したものと一つの封筒とを合わせた(まとめる)オブジェクト。

6.2.2 メールを管理するためのオブジェクト

メールの管理に用いられる主なオブジェクトには以下のものがある。

- **引き出し** 特定のメールの検索条件を保持するオブジェクト。その検索条件に従って自動的に整理された形でメール群を見せるビューをユーザに提供する。
- **整理箱** 複数の引き出しを管理する(まとめる)オブジェクト。個々のユーザに対して存在する。

ユーザは、整理箱の中の引き出しを必要に応じて開き、引き出しに指定された条件に従った形でメール群が整理されたビューを得ることができる。

6.3 共有電子メールの利用例

本モデルの電子メールは、共有オブジェクトとしてメールを作成することにより、「複数の受信者からの返事(例えば会議への出欠)をまとめて見ることができる」、「議論の際にも各自の発言を一つのメール上で Hypertext 的に構造化することで議論の流れを分かり易く表現 / 管理することが可能である」、「テキスト以外のメディアも扱うことができ、Button を初めとする動作を持ったオブジェクトもメールに含めることが可能である」等の特徴を持つ。

6.4 試作システムの構成

上記の出欠集計機能を持ったメールを HyperShell プロトタイプ上に作成した。試作中の電子メールシステムでは、HyperShell の Binder をメールとして用いている。メールが到着すると、受信者の整理箱にメール本体にアクセスするためのボタンが現れる。受信者はこれを操作してメール本体にアクセスする。[図 10] は、3 人のユーザがメール本体の同じページ(Layer)を見ているところである。出欠の状況を共有でき、集計は自動的に行なわれる。返事の取消も可能となっている。

おわりに

HyperShellは、分散オブジェクト指向マルチメディア・システムであるHyperStation上のソフトウェア構築環境である。HyperShellでは各種情報は共有オブジェクト空間中のオブジェクトとして管理され、ネットワーク透過なアクセス、複数ユーザによる共有を容易に行なうことが可能となっており、共同作業向けアプリケーションの容易な構築が可能である。本稿では、HyperShellの概要、及びプロトタイプでのオブジェクト共有方式、共有オブジェクトを用いた電子メールシステムに関して説明した。

今回のHyperShellプロトタイプは分散機能を検討するためのものであり、実現された分散共有オブジェクトの管理、制御手法を現在評価中である。HyperShellの次期プロトタイプではその機能として、共有するオブジェクトと共有しないオブジェクトの使いわけ、ユーザ毎のコンテキストの管理形態、コンテキストを共有するか否かを選択するためのUI、オブジェクトのマイグレーション、マルチスレッドによるコンカレントなメソッド実行とスレッド間の同期/排他/タイムスライス制御、等に関し、それをHyperShellの機能として用意するか否かも含めて検討し、これをInterViews[9]上に構築する予定である。また、この様な用途を考慮に入れた並列オブジェクト指向言語[16]を使用しての言語レベルでの排他制御やプリエンプティブなマルチスレッド制御機能の導入、自律オブジェクト実現形態等も検討していく予定である。

最後に、本研究の機会を与えて下さると共に有益な御指導を下さいましたNEC C&Cシステム研究所の山本昌弘所長、久保秀士主管研究員、小池誠彦部長、西谷隆夫部長、NEC U.S.A. Inc. C&C Research Laboratoriesの横田実Senior Reseach Staff Member、HyperShellプロトタイプの分散機能及び電子メールシステムの機能に関して貴重な御助言、御助力を下さいました、C&Cシステム研究所の中島震主任、マイコンソフトウェア開発研究所の佐治信之主任、ソフトウェア生産技術開発研究所の暦本純一主任を初めとする皆様に深く感謝致します。

参考文献

[1] Apple Computer, Inc., "HyperCard User's Guide"

[2] 新 淳, 他, 「ユーザインタフェースから見た分散オブジェクト」, ソフトウェア科学会第9回大会予稿集, 1992.

[3] 新 淳, 他, 「HyperStation: 分散オブジェクト指向シェルHyperShell」, 情報処理学会第45回全国大会, 6Q-06, P.5-123, Oct. 1992.

[4] P. Dewan, et al., "Primitives for Programming Multi-User Interfaces", ACM UIST'91, Nov. 1991.

[5] 濱川 礼, 他, 「分散オブジェクト指向マルチメディアシステムHyperStation — その構想と試作 —」, 情報処理学会第45回全国大会, 1B-01, P.3-289, Oct. 1992.

[6] 岩崎 未知, 「異機種分散環境におけるUI実現方式」, 情報処理学会研究会報告 91-OS-52, vol. 91, No. 73, Sep. 1991.

[7] 岩崎 未知, 「階層構造を利用したUIオブジェクト管理手法について」, 情報処理学会第44回全国大会, 5J-2, P.5-233, Mar. 1992.

[8] 岩崎 未知, 他, 「HyperStation: 分散オブジェクト指向シェルHyperShellのオブジェクト共有方式」, 情報処理学会第45回全国大会, 6Q-05, P.5-121, Oct. 1992.

[9] M. Linton, et al., "Composing user interfaces with InterViews", Computer, Feb. 1989.

[10] J. Patterson, et al., "Rendezvous: An Architecture for Synchronous Multi-User Applications", ACM CSCW'90, Oct. 1990.

[11] 岡本 道子, 他, 「分散共有オブジェクトに基づく電子メールシステム」, 情報処理学会第44回全国大会, 6M-2, P.1-301, Mar. 1992.

[12] 岡本 道子, 他, 「HyperStation: 分散オブジェクト指向シェル HyperShell の電子メールへの応用」, 情報処理学会第45回全国大会, 6Q-04, P.5-119, Oct. 1992.

[13] 暦本 純一, 他, 「エディタを部品としたユーザインタフェース構築基盤: 鼎」, 情報処理, vol. 31, No. 5, pp. 602-611, May. 1990.

[14] 暦本 純一, 「共有作業空間における柔軟な同時実行制御方式」, ソフトウェア科学会第8回大会, E3-3, 1991.

[15] 暦本 純一, 他, 「マルチメディアオブジェクトモデルとその実現」, ソフトウェア科学会第9回大会, 1992.

[16] 佐治 信之, 他, 「HyperStation: 分散オブジェクト指向言語の構想」, 情報処理学会第45回全国大会, 2Q-1, P.5-27, Oct. 1992.

[17] 坂上 秀和, 他, 「HyperStation: オブジェクト指向 GUI ツール InterViews の AV 拡張」, 情報処理学会第45回全国大会, 5B-01, P.3-309, Oct. 1992.