プログラミング－言語・基礎・実践－ 15－9
（1994．1．21）

分散オブジェクト指向UIMSの実行時アーキテクチャの設計と実現

佐藤豊　大蒔和仁
ysato@etl.go.jp, ohmaki@etl.go.jp

電子技術総合研究所

本論文ではユーザインタフェース管理システム(UIMS)の実行時システムとして開発中のVIA-UIMSについて、その設計と実現の概要を述べる。UIMS実行時のエンジンである実行時システムのアーキテクチャは、開発時における対話部品と応用部品の分離を実現し、部品の再利用を促進する基礎である。VIA-UIMSでは、ネットワーク上に分散した部品間の接続、オブジェクト指向部品の記述と実行、そして並列計算モデルによる部品間の実行制御を通じた対話制御という、3レベルの機能を提供する。この構成は、既存のユーザインタフェースの拡張や既存のソフトウェア部品の取り込みなど、従来のUIMSの適用しにくかった領域の支援を容易にしている。

Design and Implementation of a Run-time Architecture for a Distributed and Object-Oriented UIMS

Yutaka Sato,　Kazuhito Ohmaki
ysato@etl.go.jp, ohmaki@etl.go.jp

Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan

This paper presents design and implementation of a run-time architecture of user interface management system(UIMS) named VIA-UIMS which is under development. A run-time architecture, which works as an engine of UIMS at run-time, gives a base to realize separation and reuse of modules at developing time. The VIA-UIMS support three types of module integration; connecting modules distributed on networks, description and interpretation of object-oriented modules, and dialogue control via global sequencing control among modules based on process calculus. This architecture supports restructuring or extenstion of existing user interface, and incorporating existing software modules into UIMS, which have been left out of scope of UIMSs in the past.

1 はじめに

昨今では、新たに開発されるソフトウェアのほとんどが、グラフィカルユーザインタフェース (GUI) をベースに作られている。その背景には、グラフィカルな入出力装置の普及や、利用者層の拡大などの要因がある。一方で、キーボードとキャラクタ・ディスプレイをベースとする従来型のソフトウェアが、依然として大きな割合を占めている世界もある。インターネット上で流通している公開ソフトウェアなどである。このようなソフトウェアでは、ひとつのプログラムが広範囲の実行環境で使用できることが重要であるため、多様に想定される環境の最大公約数がキャラクタ入出力インタフェースとなっているという側面がある。また、本質的にキーボード入力を好むユーザや、それにより最も効率的に使いこなせる種類のツールがあることも要因である。

キーボードベースのインタフェースとGUIとでは、それぞれの記述に適したプログラムの制御構造が異なる。このため、既存の応用プログラムを「ウィンドウ対応」にするために、GUIの制御構造に合わせた応用プログラムの大幅な作り直しが行われて来た。しかし同一の機能のプログラムを、異なるユーザインタフェースごとに作り、それらの間の一貫性を保って維持するのは極めて非効率である。この問題を解決することが、ユーザインタフェース管理システム (UIMS) の本来の目的の一つであった。すなわちユーザインタフェースと応用プログラムとを分離して、一つの応用に複数の (異種の) ユーザインタフェースを簡単につなぎ込めるように支援することである。ところが従来のGUI構築システムやほとんどのUIMSでは、GUI以外を支援しておらず、応用プログラムに固定的な制御構造を課しているために、既存のプログラムの構造やユーザインタフェースを大幅に変えることなくGUIを付加したいと言う要求には応えていない。

本稿で述べるVIA-UIMSの開発は、このような要求に答えるUIMSを筆者自身が必要としたことを契機として始められた。ここで問題となるのは、応用プログラムの内部に固定的な制御構造を強制できない状況で、UIMSの行うべき制御を外部から実行しなければならないことである。我々はこれを実現するために、ソフトウェア・モジュール (部品) の間の緩やかな接続と、部品の外部での柔軟な実行制御の機能を提供するメカニズムが必要であると考えた。そのために、

- ネットワーク指向の緩やかな部品接続方式
- 並列計算モデルに基づく柔軟な実行制御方式
- オブジェクト指向な部品構造を支援するカーネル

から成るVIA-UIMSの実行時アーキテクチャを設計し実現した。これはUIMSの上位機能としてどのような機能を提供するかとは独立に、UIMSを実行するシステムに共通な基盤となるアーキテクチャを目指すものである。開発したシステムは現在、多数のユーザに利用されている既存のツールに適用されている。

以下では、実行時アーキテクチャの構成要素に関して検討し、VIA-UIMSの実現の概要を述べ、いくつかの適用例を示す。

2 実行時アーキテクチャの設計

2.1 UIMSの実行時アーキテクチャの役割

ユーザインタフェースは、具体的な実行メカニズムは別として、図1に示すSeeheimモデル [1] のように、3種類の構成要素に分離して考えるのが共通のモデルとなっている [2][3]。すなわち、フロントエンドにあってユーザとの物理的な交信を実現する部分 (表示・入力管理)、バックエンドにある応用機能とインタフェースする部分 (応用インタフェース)、そして、両者の間の交信を仲介しつつ対話を制御する部分 (対話制御) である。

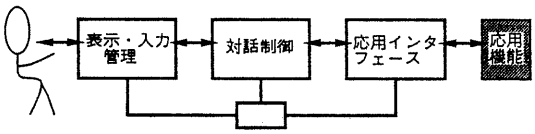

図 1: Seeheim モデル

ユーザインタフェースと応用機能の独立な開発・改良、柔軟な組合せと再利用を支援するために、両者をいかに「分離」するかという問題は、UIMSの中心的課題である。分離の方針は結合する手段に大きく依存するから、実行メカニズムの面から見れば、分離された要素間をいかに「結合」するかが問題である。

ここで部品間の結合には、部品間を物理的に結合する (通信を可能にする) ことと、論理的に整合した通信を成立させるという2つのレベルがある。

物理的な結合のレベルでは、実行性能面や部品の独立性の実現のために、部品間の結合度に応じて結合方式を柔軟に選べることが重要である [4][5]。論理的な結合のレベルでは、多様なユーザインタフェースの実現のために、対話制御を柔軟に記述できることが重要である。また、制御構造の規定されていない既存のソフトウェア部品をUIMSで支援するためには、能動的な部品の制御構造と制御構造との間を結合できる方式が必要である。UIMSの実行時アーキテクチャには、このような部品間の結合を柔軟に実現する手段を提供することが求められる。

2.2 柔軟な部品間接続と実行制御のために

並列計算モデルに基づく対話制御の記述と実行 – PIE

対話制御の記述はUIMSの中心的な課題として多くの研究がなされてきた[1][11]。近年では、非同期な対話と同期的な対話の両者を実現する対話制御のモデルとして、並列に動作可能な部品の実行を制御して部分的に逐次化しながら制御するモデルが主流である。

対話制御のための専用の記述言語として、逐次的な対話に対しては、状態遷移グラフをベースとする対話記述が使われて来た。これは対話の開発者にとって記述しやすいが並列対話の記述ができない。一方、並列的な対話のための記述言語としては、Sassafras/ERL(Event Response Language)[6]や、PPS(Propositional Production System)[9]などがある。これらの言語では対話はルールの集まりとして記述され、ルールの起動はシステムに任されている。これによって高い記述力が実現されている一方、記述する側に状態の遷移が陽に見えず全体の流れが掴みにくいという問題点がある。このことから我々は、

- 状態遷移グラフと同様な明示的な順序記述
- 並列性と逐次性の制御

を両立する記述方式が必要であると考えた。そこで、並列計算のモデルであるπカリキュラス[8]を対話記述のベースとして利用することにし、πカリキュラスのインタプリタ"PIE"を、VIA-UIMSの実行制御メカニズムとして開発した[16]。

ここで、制御構造をUIMSから規定できない既存の応用プログラムを管理するには、応用内部で取り得る状態の遷移と現在の状態を、必要十分に把握している必要である。例えば従来型インタフェースを持つプログラムは「コマンドモード」のような内部状態を持っており、現在の状態に即して対話制御を切り替える必要がある。PIEではそのような応用の状態の遷移の記述が自然に行える。

ネットワーク指向の部品結合メカニズム – VIABUS

部品の物理的な結合方式はUIMSの実行効率と実現の柔軟性を左右するが、応用対象によって適した方式が異なる。そのため、部品間の静的な結合、動的な結合、1:1の結合、1:Nの結合などが、部品内部の実現に変更を加えることなく、適宜選択できることが必要である[4][5]。

また、近年ではユーザインタフェースに接続される部品群が、ネットワーク上に分散しているのが一般的である。さらに、グループウェアなどへの応用を考えると、分散環境への対応は必須の課題である。そこで我々はVIA-UIMSに接続する部品プロセスの間で、1:Nの通信を含むプロセス間通信を、分散を意識せずに行えるメカニズムとして"VIABUS"を開発した。

コンパクトなオブジェクト指向カーネル – COOK

GUIの表示・入力管理部分では、見た目や制御が同一または類似の部品を多数作成する必要があるため、その記述にはオブジェクト指向の枠組が適している。ただし、可搬性や柔軟性の観点からは、大がかりなオブジェクト指向言語や、ウィンドウシステムに密着した処理系、コンパイラベースの処理系を使用することは避けたい。既存の言語で書かれたユーザインタフェース部品や応用プログラムに組み込めて、オブジェクト指向機能を追加できるような実行時ライブラリが望まれる。そこで、オブジェクト指向の機能を提供するコンパクトなカーネル"COOK"を開発した。

3 VIA-UIMSの概要

VIA-UIMSの実行時アーキテクチャを、図2に示す。これは、前節に述べたような以下の3つの構成要素で構成されてる。

- **VIABUS**: 分散した部品(プロセス)間での、物理的な分散を意識しないメッセージ通信による接続
- **PIE**: 並列計算モデルに基づいた、部品間の接続管理と部品間にまたがる広域的実行制御
- **COOK**: オブジェクト指向な部品の組み立てと、低レベルのイベント応答処理(部品内部の局所制御)

Seeheimモデルに対応させると、COOKが表示・入力管理部、PIEが対話制御と応用インタフェースの記述と実行に用いられる。従来のUIMSの実行時アーキテクチャのほとんどは、UIMSの構成要素間の接続関係が固定化されているが、VIA-UIMSではこの図のようにUIMSの構成要素も応用プログラムも、VIABUSという共通の通信媒体に対等な関係で接続する。

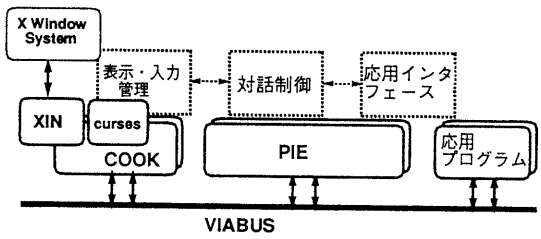

図2: VIA-UIMSの実行時アーキテクチャ

分散したCOOK処理系やPIE処理系の間は、VIABUS上のメッセージ交換によって接続されている。これら3つの構成要素はいずれも、それぞれに役割の異なる部品接続のメカニズムであり、それぞれに機能が特化されたメッセージの転送メカニズムである。いずれにおいても共通に、転送されるメッセージは単純な構造のバイト列である。

3.1 VIABUS

VIABUSは、分散環境上で文字列ベースのメッセージ交換を提供するプロセス間通信メカニズムである[15]。

VIABUSのプロトコル

VIABUSのメッセージは、空白文字で区切られた

header body

という単純な形式である。*body*の構造は規定されない。メッセージの転送先は、*header*に含まれる文字列によって決まるが、これは具体的な転送先のアドレスを示していない。メッセージの配送は以下のように行われる。

*selector body*というパターンのメッセージの受信を希望するプロセスは、VIABUSインタフェースに対して、SENDME *selector*という形式のメッセージを送る。以降、*selector*を*header*に含むメッセージが、VIABUSインターフェースから受信できるようになる。ここで、VIABUSインタフェースは文字列ストリーム型の入出力である。

送信側のプロセスは、受信プロセスの所在や数を意識しない。複数のプロセスが同一の*selector*の受信を要求していれば、マルチキャストが起こる。受信者がいないメッセージはそのまま消滅する。

階層化

VIABUSの構造は、ローカルエリアネットワーク(LAN)のネットワーク構造を模したものである。LANにおいて、個々のネットワーク単位(セグメント)の間は、パケットのフィルタリングやルーティングを行う「ルータ」で接続される。これと同様な機能を持つ"VIAルータ"を用いてVIABUSの構成単位の間を接続し、例えば図3のように階層的に構成することができる。このような構成により通信の局所化が計れる。

実装

VIABUSのバス構造は、実際には「VIABUSサーバプロセス」により集中的に集配を行う星型の接続を用い、stream型のソケットを用いて実現している。この実現では、VIABUS上のメッセージの転送性能は以下のようになった(メッセージサイズ=64bytes, SparcStation10/41上で測定)

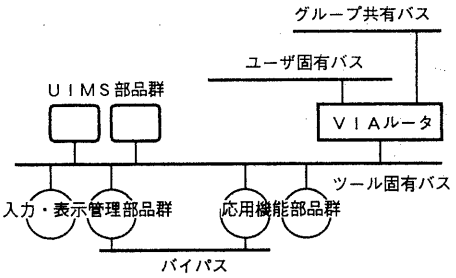

図3: VIABUSの階層的な構成の例

一方向, 受信者数=0： 350メッセージ/秒
一方向, 受信者数=1： 280メッセージ/秒
双方向, 受信者数=1： 170メッセージ/秒

Lindaとの関連

通信相手を明示的に指定せず、メッセージの内容に対するマッチングでメッセージの転送先が決まる通信方式としては、Linda[7]が知られている。[13]では、このような通信方式が、利用者インタフェースやグループウェアを柔軟に構築するのに適していることが示されている。

一方VIABUSは、Lindaが提供しているような通信に伴う同期の機能や、データの蓄積、構造的なメッセージのマッチングなどの機能は持たない。これらの機能は、VIA-UIMSの中においてはVIABUSではなく、VIABUS上に接続されるUIMS部品によって提供されるべきものである。

3.2 PIE

PIE(PI-calculus Engine)は、並列計算モデルの一つであるπカリキュラス[8]をベースとする、スクリプト言語とそのインタプリタである[16]。

並列計算モデルのUIMSへの応用

並列計算モデルは、何らかの通信により結ばれた複数の処理(プロセスまたはエージェントと呼ばれる)の間での、逐次実行、並列実行、非決定的選択や、通信路の確立、送信や受信とそれに伴う同期などを抽象的に記述するものである。一般に仕様記述と検証などの分野に応用されているが、VIA-UIMSでは、これを並列実行する部品の接続・実行制御の表現言語として、また実行の機構として使用している。

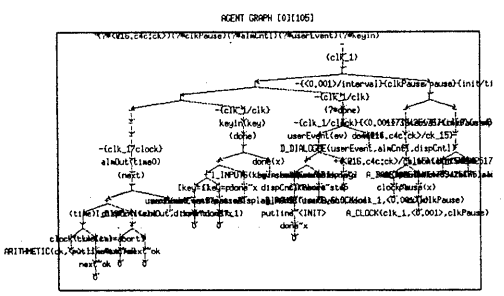

図 4: 実行中の PIE エージェントの木構造表示

並列計算モデルとしてのπカリキュラスの特徴は、プロセス間の接続関係が動的に変化できる点にある。これは、プロセス間を接続する通信路 (ポート) の名前自体を、ポート上で転送できることで実現されている。これは、UIMS 実行時に起こる動的な部品の生成と接続を表現するのに都合が良い。

PIE の実現

PIE の言語仕様は、[8] に示されたモデルや記法にほぼ沿ったものであるが、実現上の若干の記法の変更と、VIA-UIMS での使用のために後述のような若干の拡張を加えている。πカリキュラスにおける概念的な計算の単位 (エージェント) の間の並列性や逐次性は木構造として、ポートへの入出力はノードの属性として表現できる。実際に PIE 処理系は、エージェントの式を木構造として内部表現し、その変形を行いながら評価を進める。評価の各ステップで、現在の状態を図 4のような木構造で表示することにより、視覚的に計算の進行状況を知ることができる。

外部とのメッセージの送受信

PIE 処理系の外部 (VIABUS など) から受信したメッセージは (もし存在すれば) 空白文字を境に *name body* のように分割され、*name* の部分がポートの (名前) として使われる。PIE 処理系の中で、*body* の部分は名前に付随する属性として受け渡され、PIE 処理系の外部に送出される時に、*name body* の形に戻される。

PIE エージェントの転送

別の PIE 処理系に (VIABUS を経由して) PIE のエージェントを転送する場合には、内部の木構造表現をテキスト表現 (PIE スクリプト) に変換して転送する。この際、局所的な (ポートの) 名前に対する参照は、広域的に通用する名前で置き換えられる (一種の *name extrusion* を起こす)。この機能を用いて、処理系をまたがる PIE エージェント間での双方向の通信などを実現している [16]。

3.3 COOK

COOK(Compact Object-Oriented Kernel) は、既存の言語にオブジェクト指向機能を加えるための実行時ライブラリである。これはまた、COOK のプリミティブ関数に 1 対 1 に対応する単純な構文を持つ “COOK スクリプト” のインタプリタとしても使用できる。

COOK の主な機能

COOK は以下の機能を提供する。

- オブジェクトの作成と消去
- 動的な継承の追加 (多重継承)
- 動的なメソッドの付与
- 記憶管理 (インスタンス変数の割り当てと開放)

基本的なプリミティブ関数は以下のようなものである。

```
createObject(char *objectName)
addStorage(int objectId,
     char *storageName, int size);
findStorage(int objectId,
     char *storageName);
addMethod(int objectId,
     char *methodName, (*methodFunc)())
callMethod(int objectId,
     char *methodName, char *method_args)
```

これらを用いて簡単な画面を作成して表示した例を図 5に示す。具体的なメソッドの実現、例えば図 5の例における “geometry” や “map” の実現は、C 言語で作成して 関数 addMethod() によってオブジェクトのメソッドとして付加する。

COOK にはクラス / オブジェクトという区別はなく、既存の任意のオブジェクトからメソッドを継承する方式を取っている。メソッドの継承はオブジェクト間でのメッセージの委譲 (delegation) という形で実現されている。このような実現をとったのは、動的な継承の追加を含む処理系の実現を簡略化するとともに、既存の全てのオブジェクトの再利用を可能にするためである。実現値を持つインスタンスを雛型 (テンプレート) としてオブジェクトが作成できることで、例えばグラフィカルな部品の差分的プログラミング、段階的なカスタマイズなどが容易である [12]。

全てのオブジェクトは “OBJECT” を雛型として生成される。OBJECT の持つメソッド inherit では、メッ

C プログラムによる記述

```
int w;
w = createObject("win1");
callMethod(w,"inherit","WINDOW");
callMethod(w,"inherit","GC");
callMethod(w,"inherit","TEXT");
callMethod(w,"inherit","TEXTVIEW");
callMethod(w,"title","simple win");
callMethod(w,"geometry","${GEO:-80x40+1+1}");
callMethod(w,"text","シンプルな窓の例 ");
callMethod(w,"map/WINDOW"); }
callMethod(w,"map/TEXTVIEW");
```

COOK スクリプトによる記述

```
OBJECT win1 new      # define new object
  inherit WINDOW     # simple window
  inherit GC         # graphic context
  inherit TEXT       # internal text
  inherit TEXTVIEW   # text viewer
  METHOD initialize
    title    simple win
    geometry ${GEO:-80x40+1+1}
    text     シンプルな窓の例
    map/WINDOW       # display the window
    map/TEXTVIEW     # display the text
  END
```

図 5: 簡単なグラフィックオブジェクトの記述例

セージを委譲する経路を指定するとともに、その経路の先に存在する全てのオブジェクトの "new" というメソッドを起動する。インスタンス変数のための記憶領域をオブジェクトに割り当てるのは、通常 C プログラムで実現されたメソッド new であり、ここで関数 addStorage() を用いて領域が割り当てられ、雛型となるオブジェクトの実現値が複写される。以下、OBJECT に組み込みの主なメソッドとそれらの意味について説明する。

オブジェクトの包含関係

inherit と同様に、メッセージのルーティングのパスを指定するメソッドとして partof がある。これは、メッセージをクラス階層ではなくインスタンスの包含関係に従って委譲するルートを指定する。ウィンドウシステムにおけるイベントの上位部品への伝搬や、ブロック構造言語における名前のスコープに類する働きを持つ。

メソッド探索経路

メソッドは、まず inherit の経路で探索され、そこで見つからなかった場合には partof の階層で探索される。言い替えれば、まずメソッドを呼び出したオブジェクト自身の属すクラスの中を探索し、見つからなかった場合には部品階層の中の上位の部品にメッセージを委譲する。複数の inherit 経路、partof 経路がある場合には、後から追加されたものが優先する。

明示的な経路指定

多重に継承したクラス内に同一のメソッド名がある場合に、以下のようにクラス名を後置してクラスを明示的に指定できる。

methodname/objectname

また、メッセージの送り先のオブジェクトを明示指定するには、そのオブジェクト名をメソッド名に前置する。

objectname : *methodname*

正規表現を用いたブロードキャスト

methodname/* のように書くと、*methodname* を名前として持つ全ての上位クラスのメソッドが起動される。また、*objectname* 中に文字 '*' が含まれると、それは任意の文字列と見なされ、そのパターンにマッチするオブジェクト全てにメッセージが送られる。

イベントとリアクション

オブジェクトは通常のメソッド呼び出しに応答する他、COOK 処理系などで発生するイベントに対して応答する。例えば、以下のように記述する。

```
select   ButtonPress
reaction Button1Press reverseTheButton
```

メソッド select で反応すべきイベントを選択し、メソッド reaction でそのイベントが発生した場合に起動するメソッドを指定する。

オブジェクトの転送

異なる COOK 処理系間でのオブジェクトの転送は、オブジェクトのテキスト表現 (COOK スクリプト形式) を VIABUS 上で送受信することで実現している。

COOK 上のユーザインタフェース部品

COOK を用いて、図5中の WINDOW や GC と言った X ウィンドウシステムのオブジェクトに対応するオブジェクトの作成と管理を行う "XIN/COOK"、curses ライブラリによる文字端末上のウィンドウ機能を操作する "curses/COOK" が作成されている。

4 適用例

筆者が開発を行ってきたニュースリーダ vin[14] を対象として、VIA-UIMS の適用実験を行っている。3 つの実験例を示す。

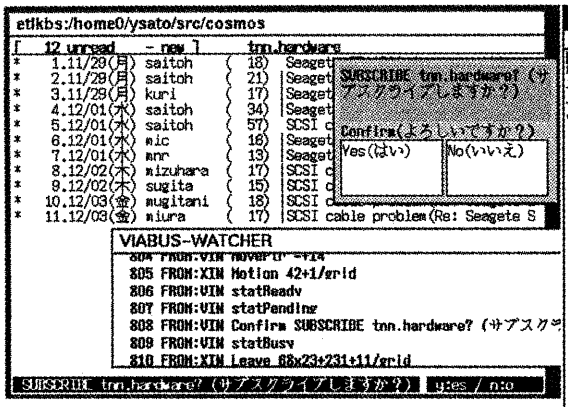

図 6: VIA-UIMS で実現した vin の拡張インタフェース

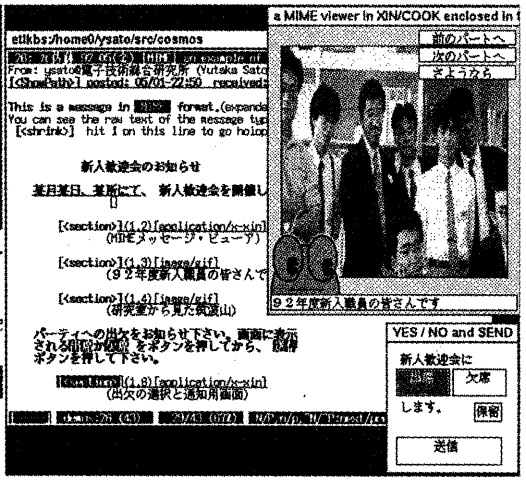

図 7: VIA-UIMS を用いたインタラクティブメイル

4.1 既存のユーザインタフェースの拡張

vin の X ウィンドウシステム対応の操作インタフェースとして、マウスによるポインティングとメニューの操作機能を追加した。これは、ウィンドウ上でのマウスポインタの移動に同期させた vin の画面上の文字カーソル移動や、マウスボタンの操作に応じた vin のコマンドの起動 (ユーザインタフェース側からの応用の起動)、処理実行前の確認要求 (応用側からのユーザインタフェースの起動) を含んでいる。図 6にその実行の様子を示す。その実現は、以下のように行われた。

(1) ウィンドウ操作部品の作成

ウィンドウ上での入力と表示を管理する部品を、XIN/COOK を用いて作成した。この部品は、VIABUS から受け取った指令 (マウスポインタ移動や、メニュー作成) メッセージを受けて、COOK オブジェクトの生成やメソッドの起動を行う。一方、ウィンドウ上で発生したイベントは、vin に必要な情報だけを含む抽象化された形で、VIABUS 上にメッセージとして送出する。この部品への指令は VIABUS からの単純な形式のメッセージなので、メッセージを手で入力して与えることにより、vin とは切り離して開発や修正を行うことができた。

(2) vin の応用インタフェースの記述

まず、vin の中で対話のきっかけとなる入出力を行っている箇所やコマンドモードが遷移する箇所で、それを PIE 処理系に知らせるメッセージを送るようにした。次に、それらのイベントの発生順序などの制御構造を表現するエージェントを、PIE スクリプトで記述した。そして、vin のコマンドを起動する外部からのメッセージと、その実行を要求するために vin に送るメッセージの記述を行った。これは、vin の内部の制御構造の大枠と外部との通信を記述したものになっている。

(3) 対話制御の記述

(1) で作成した COOK オブジェクトの行うメッセージ入出力と、(2) で記述した vin の枠組を表す PIE のエージェントのポート間での、関連する入出力を仲介するエージェントを PIE スクリプトで記述した。これを (2) のエージェントと結合して動作させることで、ウィンドウインタフェース用の対話制御が実現された。この実現では、PIE 処理系は vin のプロセスの中に組み込まれている。

4.2 インタラクティブメイルへの応用

インターネット上の多目的マルチメディアメイル形式として標準化が進められている MIME(Multipurpose Internet Mail Extensions) 形式 [10] を用いて、vin の多目的メイルへの対応を行っている。

図 7に示すような、画像や音声などを含むマルチメディアメイル、あるいは読み手と対話してアンケート調査を行うようなインタラクティブメイルが実現されている。インタラクティブメイルは、メイルの中に、読み手と対話するためのユーザインタフェースが同封されたメイルである。

この例ではメイルの中に、PIE スクリプトで書かれた対話制御、XIN/COOK スクリプトで書かれたグラフィカルなユーザインタフェース、および画像データなどをそれぞれ “part” として、MIME の “multipart” 形式の中に同封されている (図中で [<section>] と表示されている部分)。読み手が VIA-UIMS 付きの vin でこのメイルを読むと、それぞれの part が対応する処理系に送られて実行されることで、図 7のような画面が作られ、読み手との対話が行われる。

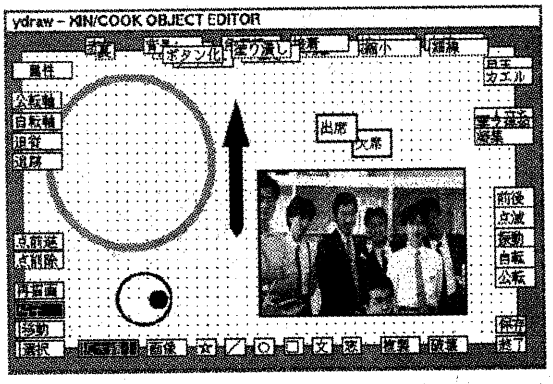

図 8: グラフィカルな COOK オブジェクトエディタ

4.3 オブジェクトエディタ

XIN/COOK 上の視覚的なオブジェクトをグラフィカルに編集するエディタ (ydraw) を試作している (図 8)。このエディタにより、オブジェクトの作成、継承の追加、属性やリアクションの付加などをマウスの操作を基本として行うことができる。作成した結果は、COOK スクリプトの形で出力して保存できる。

このエディタ自身も COOK でスクリプトで記述されているため、自身を編集することが可能である。非視覚的なオブジェクトを視覚化する機構を導入すれば、COOK オブジェクト全般に適用可能なエディタが実現できる。

5 おわりに

適応範囲の広い UIMS を実現する基礎として、並列計算モデルによる実行制御、分散環境上での柔軟なプロセス間通信、オブジェクト指向の部品操作という機能を提供する VIA-UIMS の実行時アーキテクチャについて述べた。それぞれの機能は組み合わせても単体としても利用できるため、応用プログラムは多様なレベルで、必要な部分に対して、VIA-UIMS を利用することができる。実用のツールへの応用実験を通じて、VIA-UIMS が既存のユーザインタフェースの拡張をも有効に支援できることが示された。VIA-UIMS を用いて、グラフィカルなオブジェクト指向部品を作成するエディタを試作したが、このツールの延長として、対話制御の非プログラム的な開発を支援する構造エディタの開発を検討している。

謝辞

本研究の機会を与えていただいている、太田公廣情報アーキテクチャ部長に感謝します。

参考文献

[1] Green,M.: Report on Dialogue Specification Tools, User Interface Management Systems, Pfaff,G.A. (ed), Springer Verlag, pp.9-20 (1985).

[2] Edmonds,E.(ed): The Separable User Interface, Acacemic Press (1992).

[3] Hartson,H.R. and Hix,D.: Human-Computer Interface Development: Concepts and Systems for Its Management, ACM Comp. Surv. Vol.21, No.1 (1989).

[4] Coutaz,J.: Architecture Models for Interactive Software: Failures and Trends, Proc. of the IFIP TC2/WG2.7 Working Conf. on Engineering for Human-Computer Interaction (1989).

[5] Manheimer,J.M., Burnett,R.C. and Wallers,J.A.: A Case Study of User Interface Management System Development and Application, CHI'89 Proceedings (1989), pp.127-132.

[6] Hill,R.D.: Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS, ACM Trans. Graphics, Vol.5, No.3, pp.179-210 (1986).

[7] Carriero,N. and Gelernter,D.: How to Write Parallel Programs: A Guide to the Perplexed, ACM Computing Surveys, Vol.21, No.3, pp.324-357 (1989).

[8] Milner,R. Parrow,J. and Walker,D.: A Calculus of Mobile Processes, LFCS Report Series ECS-LFCS-89-85 (1989).

[9] Olsen,E.R.: Propositional Production System for Dialogue Description, CHI'90, pp.57-63 (1990).

[10] Borenstein,N. and Freed,N.: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, RFC1521 (1933).

[11] 宮崎一哉: ユーザインタフェース管理システムと対話制御, 情報処理, Vol.33, No.11, pp.1295-1303 (1992).

[12] 橋本治: ユーザインタフェース管理システムの研究動向と将来, 情報処理, Vol.33, No.11, pp.1331-1339 (1992).

[13] 増井俊之, 花田恵太郎, 音川英之: 共有空間を用いたグループウェアの構築, 情報処理学会研究報告, PRG10-7, pp.49-56 (1993).

[14] 佐藤豊, 田沼均, 小島功, 植村俊亮: 内容を解釈実行するメイル / ニュースリーダとその応用, 電子情報通信学会, DE88-23, pp.25-32 (1988)

[15] 佐藤豊, 真野芳久: UIMS の試作とそのニュースリーダへの応用, 情報処理学会研究報告, SE77-14, pp.81-88 (1991).

[16] 佐藤豊, 大蒔和仁: 対話型ソフトウェアの動的な部品間接続の方式について, 情報処理学会研究報告, SE89-7, pp.49-56 (1992).