マイクロコンピュータと 58－2
ワークステーション
（１９８９．１２．１）

並行プログラム用デバッガにおける ビジュアルインターフェース

羽鳥和重　平田勝裕　山田　剛　小原啓義
早稲田大学理工学部

我々は，並行プログラムのためのテストデバッグ環境 mimsyのユーザーインターフェース部をマルチウィンドウシステム環境を用いて実現するために，Xウィンドウシステムバージョン11のwidgetを導入した．ユーザーインターフェースを構築するにあたって，生じた問題点とその解決策を述べ，現状のインプリメント結果についての評価を行う．さらに，評価の結果から，並行プログラムのプロセス・プロセス間通信の状態を動画として表示するために，適切なユーザーインターフェースモデルの改良案を提案する．また，新たなモデル上では，ユーザーインターフェース部の持つ機能はどの様に実現されるか，そのインプリメントの方法について報告する．

User Interface of "mimsy", A Testing and Debugging Environment for Concurrent Programs

Kazushige Hatori　Katsuhiro Hirata　Tsuyoshi Yamada　Hiroyoshi Ohara

School of Science and Engineering, Waseda University

3-4-1 Okubo, Shinjuku-Ku, Tokyo, Japan

We have implemented a user interface for "mimsy", a concurrent programming environment, to interact with the user in a multi-window environment. This was conducted through the use of the "widget" mechanism of the X Window System V11R3.

The first topic of discussion concerns the problems which have arisen in building the user interface. The solutions to these problems are taken up next. Moreover, the implementation will be evaluated so as to provide suggestions for a more adequate model in producing animated displays of program behavior.

1．はじめに

本稿では，並行プログラムデバッグ環境mimsyのユーザーインターフェースについて報告を行なう．

並行プログラムでは，各プロセスが並行に実行され，プロセス間通信などの相互作用は非同期的に発生する．そのため，プログラマは起こり得るあらゆるプロセス間の相互作用を考慮する必要がある．このように，並行プログラムの開発は，逐次プログラムのそれと比べ著しく困難である．そこで高機能な開発，支援環境を提供することは，並行プログラムの生産性や信頼性の向上につながる．この開発，支援環境として，並行プログラムのデバッガビリティの向上を目的としたものが，並行プログラム開発支援環境mimsy である． mimsy の主要な特徴としては，高機能なユーザーインターフェースが挙げられる．mimsy のユーザーインターフェースは，マルチウィンドウシステム上へ構築され，動画によるデバッグ情報の表示が可能であり，また，マウスを用いたデバッグ機能の実行も特徴となっている．

今回は，mimsy のユーザーインターフェース部をＸウィンドウシステムバージョン11（以下Ｘ11と略す）のwidgetを使用して作成したことと，そのインプリメント結果の評価を行っている．さらに，評価の結果とDIB との連動を踏まえて，新たに改良を行うユーザーインターフェースモデルを報告をする．

また，mimsy のユーザーインターフェースの機能の詳細については文献[2][3]を，mimsy の全体像およびルールベース機能によるデバッグについては，文献[1] を参照されたい．

2．mimsy の構成とユーザーインターフェース

mimsy の構成は，Fig.1 に示すように，披検プログラムとのインターフェースをもつProve/Stub・Driver モジュール，ルールベースとエンジンからなる DIB（Debug Information Base）モジュール，プログラマとのインターフェースを形成するExplanation モジュールおよびVisual Shellに分けられている．

mimsy のユーザーインターフェース部は，Explanation モジュールと，Visual Shellの２つから構成されており，前者はテキストを用いたユーザーインターフェースであり，後者は動画を用いたユーザーインターフェースの機能を実現している．

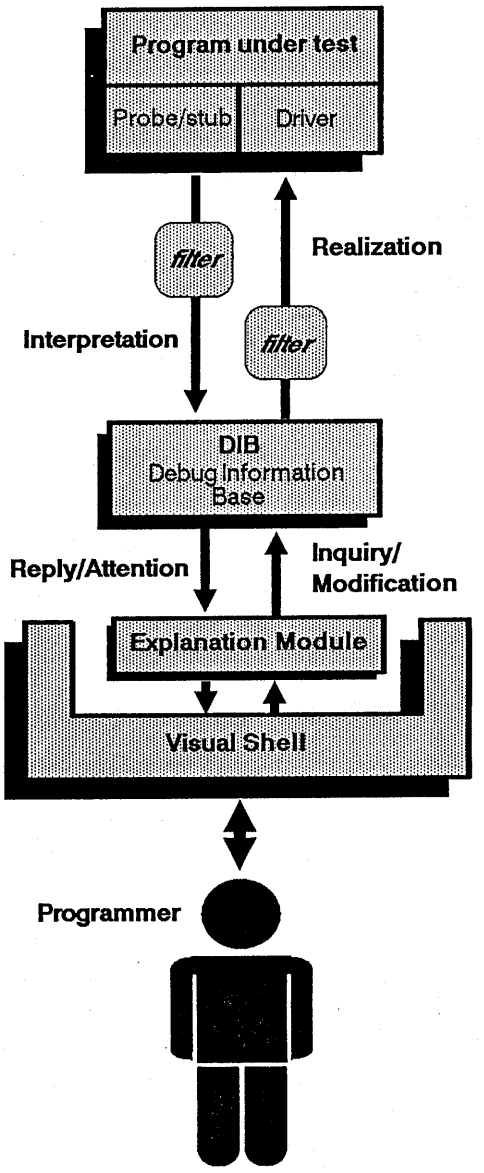

Fig.1 Internal Structure of "mimsy"

DIB からプログラマへのデバッグ情報は，主に動画を用いて示される．DIB で観測されたプロセス・IPC の状態をグラフィックスで表示することによって，プログラマは視覚的にプロセス・IPC の状態を理解することが可能である．さらに，グラフィックスを動画として表示することによって，プロセスや通信の生起・終了の過程を非常に分かり易くプログラマに示す．また，プログラマは複数のウィンドウを開き，各々のウィンドウでグループ化などの機能を行うことによって，多面的な観測も行うことができる．

また，プログラマによるDIB への操作は，キーボードよりもマウスによる操作を用いて操作性の向上を図る．プログラマは，マウスを用いて図形プリミティブの移動やマーキングを行ったり，メニューを開いてグループ化など機能の実行を行う操作によって，ルールの生成およびDIB への登録を可能にする．

mimsy のユーザーインターフェース部の開発は，これらの機能をサポートするVisual Shellのインプリメントが中心である．さらにDIB との連動では，ウィンドウアプリケーションとデータベースとの間にどの様なモデルを構築するかを考察しなければならない．

3．ユーザーインターフェース部のインプリメント

ユーザーインターフェース部の試作当初は，X ウィンドウシステムのバージョン10（X10）を用いて，開発が行なわれていた．X10には，Xlibと呼ばれる描画やウィンドウ作成のためのローレベル関数ライブラリのみしか備わっておらず，Xlib を用いて，ウィンドウアプリケーションを実現するには多くの手続きを必要としたため，アプリケーションの生産性が非常に悪かった．バージョン11（X11）がリリースされるにあたって，新たにwidgetと呼ばれるオブジェクト指向のアプリケーション構成法が導入され，mimsy もwidget（Athena widget ）によるユーザーインターフェース部（visualwidget）の作成を行った．

3.1 widget化の問題点と解決策

widgetを用いると簡単なウィンドウアプリケーションは容易に作成できるが，次の機能はwidgetには備わっておらず，widgetでmimsy のユーザーインターフェースの機能を実現するために．

・グラフィックスの使用

widgetは，ウィンドウへのグラフィック表示をサポートしていないために，描画を行なうためには新たにグラフィックスの使用をwidget内で実現しなければならない．

・マウスによるグラフィックスへの操作

widgetはテキストレベルのインターフェースが中心になっている．しかしmimsy では，マウスでのグラフィックスの操作機能を実現するために，ウィンドウに表示されたグラフィックスにもオブジェクト指向の構成をもたせ，マウスのイベントを受け取るような機能にする必要がある．

visualwidgetを作成する際には，次のように解決を図った．

・Xlibの関数を用いてグラフィックルーチン作成し，widget内に組み込むことによってウィンドウへのグラフィック表示を可能にした．また，グラフィックスの操作は，visualwidget専用の描画関数を作成し，オブジェクト指向の構成を持たせた．

・widgetには，イベントによって呼ばれる関数をプログラマが自由に定義できるコールバックと呼ばれる機能がある．まず，マウスのイベント入力から描画上のマウスのポインタ位置を取得する機能をサポートし，コールバック関数には様々なユーザーインターフェースの機能を設定した．こうすることによって，描画されている図形（オブジェクト）の移動などの機能を可能にした．

以上のような解決策をもとに，visualwidgetのインプリメントを行なった．

3.2 ユーザーインターフェース部の構成とその問題点

旧ユーザーインターフェース部の設計モデルにwidgetを用いてインプリメント行った結果，ユーザーインターフェースはfig.2 のような構成となっている．

Visual Shellの機能は，ジョブウィンドウから送られてきたVisual Shellの操作コマンドに基づいて動画を表示することと，visualwidgetに表示されているもののステータス情報を格納することである．さらに，visualwidgetとの関わりでは，Visual Shell操作コマンドをvisualwidget用の描画関数にパージングを行う．プログラマがマウス操作でグループ化などの機能を実行した場合には，Visual Shell内の描画データの変更を行って，続いてvisualwidgetでの書き直しを実行する仕様となっている．

ジョブウィンドウは，各Visual Shellのステータス情報をもとに，ルールの生成を行ってDIB に登録を行う機能を持つ．

しかし，DIB との連動を行う上で次のような問題点が明らかになった．

(1) ルールの生成には，プログラマの操作による変更が行われていない描画データとステータス情報との比較が必要だが，Visual Shellは限定された情報しか持たないため，ルールを生成するのが困難である．

(2) Visual Shellの管理は常にジョブウィンドウが行っているために，デバッグに必要なステータス情報などをDIB が把握していることは難しい．

(3) Visual Shellどうしの関係がないために，一つのVisual Shellから他のVisual Shellに対して，グループ化などの制御を実現しにくい．

以上の問題点により，次のように改良されたユーザーインターフェースモデルを導入し解決を図る．

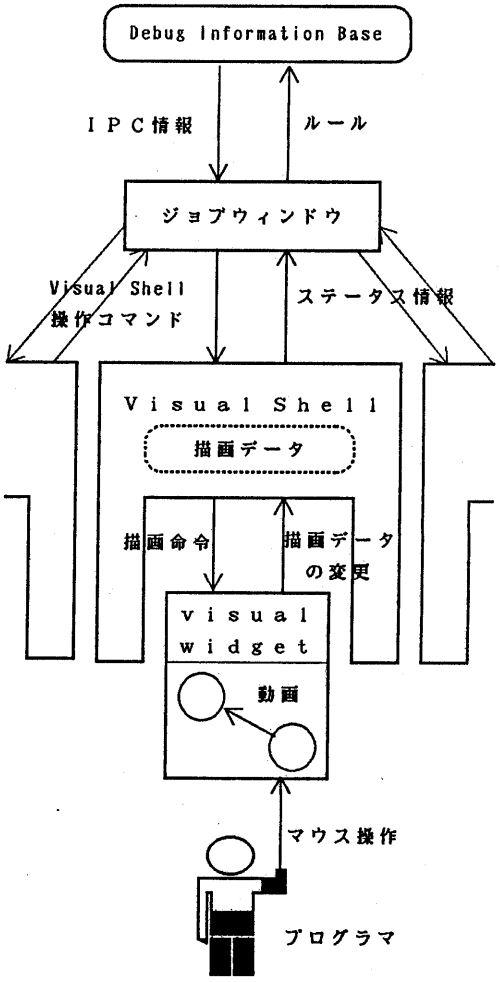

Fig.2　現ユーザーインターフェース部の構成

3.3 新たなモデルによる解決策

新たに導入するユーザーインターフェースモデルを，fig.3 に示す．

このモデルでは，ジョブウィンドウとvisualwidgetで構成されたアプリケーションをVisual Shellとして実現する．描画データは，ジョブウィンドウのレベルで扱うように変更を行い，また各々

のvisualwidgetに対して描画データの変換を行うフィルタを設けるようにする．DIB は，ジョブウィンドウの描画データ書き換えを直接行うようにする．また各々のvisualwidgetの描画は，それぞれ設けられたフィルタによって描画データを変更しこれを表示する．

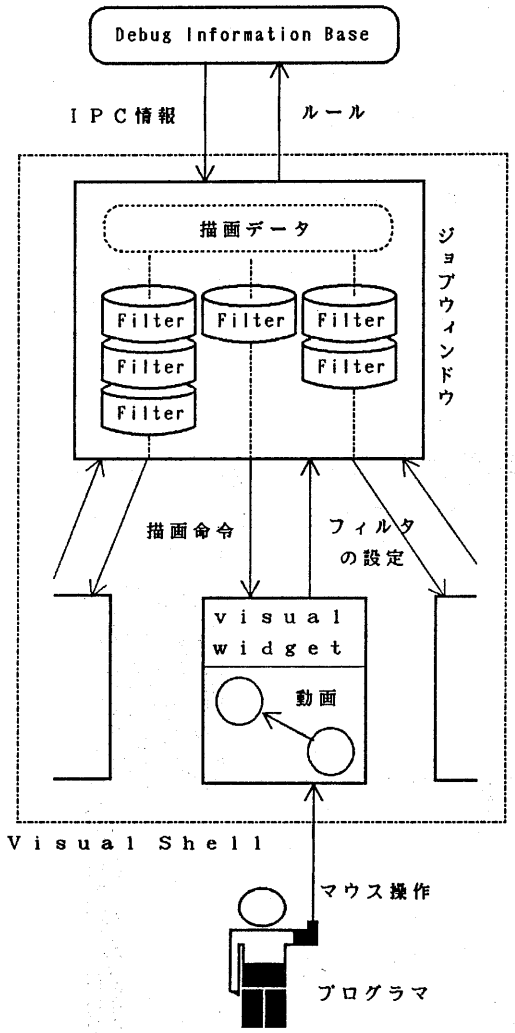

Fig.3　新たなユーザーインターフェースモデル

3.2 の問題点は，次のように解決される．

(1) プログラマが実行した機能は，描画データの変換を行うフィルタとして設定され，グラフィックスはこのフィルタを通して変換された描画データを基に表示される．DIB へのルールの生成は，このフィルタの機能をそれぞれルールにパージングすればよい．
(2) visualwidgetの扱う描画データは，DIB によって直接書き直しが行なわれる．また，描画データのフィルタは，ルールにパージングを行ってDIB へ登録される．これによりDIB は，複数のvisualwidgetで表示に使用されている描画データを常に把握していることができる．
(3) 一つのvisualwidgetによって生成したフィルタを，同様に他のvisualwidgetへのフィルタとして登録するだけで，同等な機能の実行を他のvisualwidgetに対して行ったことに等しくなる．

さらに，DIB から送られてきた描画データを順に格納しておく機能もインプリメントする．これにより，Visual Shellは自らの環境内だけで動画表示の再現が可能となる．また，描画データを用いたデバッグ支援機能の試行が可能であり，試行した後にDIB へルールとして登録を行うことができるなど，Visual ShellはDIB のためのルールエディタ機能を実現することができる．

4．Visual ShellとDIB の連動

前章で示したモデルを導入することによって，描画を行う機能はすべてVisual Shellが受け持つことになり，DIB は描画機能とは切り離されている．そのため，DIB とVisual Shellの間でやり取りされるデータの形式についてさらに詳細を検討する必要がある．特に，プロセスとIPC の状態を表すのにどれだけのパラメータが必要であるか，またDIB の処理の段階では扱うことのないプロセス名や動画のプリミティブの大きさなどの描画上

のシンボリックな情報をどこに格納するかなどが問題となっている．

ルールの生成においても大きく分けて２つの問題点がある．まず，ジョブウィンドウにある描画データのフィルタから，どの様なルールを生成するかという問題である．次に，生成したルールをどの様にしてDIB に登録するかという問題がある．現在の段階では，DIB のインプリメントがこの段階まで進んでおらず，ユーザーインターフェース側からDIB へ行う操作の詳細については具体化されていないために，順次検討を重ねインプリメントを行っていく方針である．

5．おわりに

以上，mimsy のユーザーインターフェース部をＸウィンドウシステム上に構築したことと，その機能について，報告を行った．さらに，提案を行ったモデルを基に，現在は順次インプリメントを進めている．

また，DIB との連動において，Visual Shellは，DIB が持つプロセスとIPC の状態データを複数の描画データに置き換えて表示を行う．DIB とVisual Shell間でのデータの制御や対応を系統立てて扱えるように，整理されたMVC （Model-View-Control）の導入を検討するのも今後の課題である．

参考文献

[1] 山田他，”並行プログラムのためのテスト・デバッグ環境｀ｍｉｍｓｙ’”，電子通信学会技術研究報告，Aug.1989

[2] 山田他，”並行プログラムデバッガｍｉｍｓｙのユーザインタフェース”，電子通信学会技術報告，CPSY88-40,pp.13-18 Aug.1988

[3] 小松他，”並行プログラム用デバッガｍｉｍｓｙのユーザーインターフェース”，第29回プログラミングシンポジウム，pp.1-11 Jan.1988

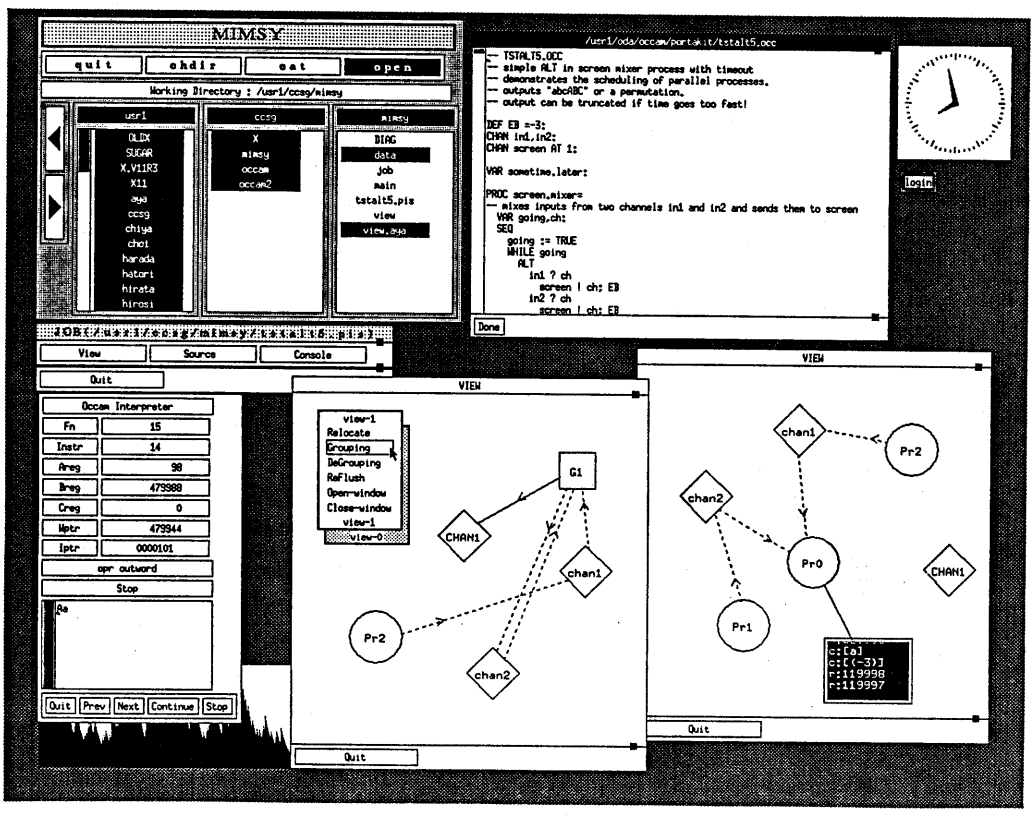

Fig.4　ｍｉｍｓｙのユーザーインターフェース