メタユーザインタフェースを有するユーザインタフェース構築支援システム†

谷 正之†† 荒井俊史†† 谷越浩一郎††
横山孝典††† 谷藤真也††

ユーザインタフェース (UI) の開発が応用プログラム (AP) 開発のネックとなっており，UI の開発効率化が重要な課題となっている．そこで，1) 柔軟な UI 定義環境の提供，2) 試行錯誤的な UI 開発の支援，3) UI 定義環境自身の UI，すなわち UI を構築するための UI (メタユーザインタフェース，略してメタ UI と呼ぶ) を拡張可能にすること，により UI の開発を効率化する UI 構築支援システム MU を開発した．MU は以下の特徴をもつ．1) UI 記述言語 (UIDL) による定義方式と WYSIWYG (What You See Is What You Get) 方式とを即座に切り替えられる．WYSIWYG 方式は表示形状などの視覚的な仕様を定義するのに適し，UIDL は動作仕様や，規則的な画面配置などをアルゴリズムを用いて定義するのに適している．両者の方式を自由に組み合わせることで，UI の仕様をより柔軟に定義できる．2) UI の定義環境と，実行環境とを即座に切り替えられる．これにより，定義／実行／修正を効率的に繰り返せるため，実際に使いながら不具合を修正していくといった UI 開発に不可欠な試行錯誤が効率化できる．3) メタ UI 自身が UIDL で記述されている．これにより，AP の UI を開発するのと同じ要領で，メタ UI を修正できる．UI 開発者が自分の好みに合わせてメタ UI を手直ししたり，新たな UI 構築用のツールを組み込んだりすることが容易にできる．

1. はじめに

近年のハードウェアの進歩により計算機の処理能力に余力がでてきた．それに伴い，計算機をより使いやすくするユーザインタフェース (UI) への期待が高まっている．特に，グラフィックスの専用ハードウェアや，マウスなどのポインティングデバイスを備えたワークステーションでは，グラフィックスを利用した直感的で使いやすい UI が強く望まれている．

しかし，ユーザの要求を満足する UI を開発するには多くの時間とコストがかかる．UI の開発時間がプログラム全体の開発時間の半分以上を占めることも多い．これは，1) UI の設計，評価方法がいまだ確立しておらず，開発したシステムを実際に使いながら不具合を修正するといった試行錯誤的開発が必要なこと，2) UI に対する要求が個々のユーザによって異なるため，ユーザごとに UI を手直しする必要があること，などが主な理由である．

こうした UI の開発を支援することを目的に，ユーザインタフェース管理システム (UIMS : User Interface Management System) が研究，開発されている[1)~3)]．UIMS は下記アプローチにより UI の開発を支援することを目指している．

1) UI 部分を応用プログラム (AP) 本体から分離して管理し，UI 部分を AP 本体とは独立に開発，変更できるようにする．

2) 標準的な UI 用ソフトをあらかじめ部品として用意し，それらを組み合わせることにより，UI を効率的に開発できるようにする．

3) UI 記述言語 (UIDL : User Interface Description Language) や，WYSIWYG (What You See Is What You Get) 方式の UI 定義手段を提供し，UI をより簡単に定義できるようにする．

しかし，従来の UIMS では，1) 言語による UI の開発を主体にしており，WYSIWYG 方式の UI 定義環境や，WYSIWYG 方式と言語による開発とを統合する環境を十分に整備していない，2) 開発環境と実行環境とが分離されており，開発，実行，評価，修正を短時間で繰り返す必要のある試行錯誤的開発に適した環境を提供していない，3) UIMS 自身の拡張性が低く，UI 開発者が UIMS 自身の UI を変更したり，新しい UI 構築用の部品を組み込むことが難しい，という問題があった．

そこでわれわれは，UIMS の考え方を基本として，1) 柔軟な UI 定義環境の提供，2) 試行錯誤的な開発

---

† A User Interface Development System with Flexible and Extensible User Interface by MASAYUKI TANI, TOSHIFUMI ARAI, KOUICHIROU TANIKOSHI (The 10th Department, Hitachi Research Laboratory, Hitachi, Ltd.), TAKANORI YOKOYAMA (ICOT Research Center, Institute for New Generation Computer Technology) and SINYA TANIFUJI (The 10th Department, Hitachi Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所日立研究所第10部
††† (財)新世代コンピュータ技術開発機構

の支援，3) UI 定義環境自身の UI，すなわち UI を定義するための UI（メタユーザインタフェース，略してメタ UI と呼ぶ）を拡張可能にすること，を目的に，UI 構築支援システム MU (Meta User Interface の略）を開発した．

本論文では，MU の提供する UI 定義環境と，その実現方法について述べる．

2. MU の開発目標

MU 開発の最大の目的は，UI の開発効率を向上するメタ UI を提供することにある．メタ UI では，

1) UI の仕様を簡単に定義できること，

2) 試行錯誤による開発を効率的にできること，

3) メタ UI 自身を変更できること，

が必要である．これらの要件を満たすため MU では下記を開発目標とする．

（1） UIDL と WYSIWYG 方式を併用して UI の仕様を定義できること

UI の仕様を定義する方法として，適当な言語を用いて定義する方式[4)~7)]と，画面上で対話的に UI を定義する WYSIWYG 方式[8),9)]とがあるが，それぞれ一長一短がある．一般的には，表示形状の指定や，画面配置などを定義するには WYSIWYG 方式が優れている．一方，動作の定義や，規則的な繰り返しなどを定義するにはアルゴリズムを記述できる言語の方が簡便に定義できる[10)]．定義内容に合わせて両者の方法を自由に組み合わせることができれば，UI の仕様をより柔軟に定義できる．

従来の UIMS でも，UIDL と，WYSIWYG 方式による定義ツール（レイアウトエディタ，アイコンエディタなど）とを備えているものもあったが[11)]，WYSIWYG で定義した内容が UIDL ソースに反映されないため両者を併用して UI の仕様を定義するのが難しかった．

（2） AP の実行環境と UI の定義環境とを即時に切り替えられること

UI の開発では，実際に使いながら，使いにくいところをその場ですばやく修正するといった，試行錯誤的な作業が不可欠である．こうした試行錯誤を効率化するには，AP の実行と，UI の定義とを即時的に切り替える機能が有用である．この機能により，AP の実行を一時的に中断し，UI 定義環境を呼び出して定義内容を修正した後，AP の実行を再開して修正結果を評価するといった試行錯誤的手順を効率的に繰り返すことができる．

本機能は UI 開発時だけでなく，AP のエンドユーザが自分の好みに合わせて UI を手直しする場合にも便利な機能である．この機能を使ってエンドユーザは AP を使っている最中に，メニューの配置を変更したり，メニューに新しいコマンドを登録したりできる．

本機能は，Sassafras[6)] で提案されている Suspended Time Editing (STE) 機能と類似の機能であるが，STE では WYSIWYG 方式の定義機能を利用できず，UIDL の編集しかできない．MU では実行中断時に，UIDL の編集だけでなく WYSIWYG 方式も含めた UI 定義環境の全機能を利用できるようにする．

（3） UI 定義環境自身の UI を拡張，変更できること

AP の UI を個々のユーザの要求に合わせて手直しする必要があるのと同様に，UI 定義環境の UI もそのユーザである UI 開発者の要求に合わせて手直しする必要がある．従来の UIMS には，それ自身の UI を拡張，変更していくという考え方がなく，UI 開発者が UIMS 自身を手直しするのは難しかった．そこで，MU では，UI 開発者が，AP の UI を開発するのと同じ要領で，定義環境の UI を変更できるようにする．

UI 定義環境自身を変更できることは，新たな UI 構築用の部品を導入する上でも必要である．従来の UIMS の多くはメニューやアイコンなどの標準的な UI 用ソフトを部品として提供するというアプローチをとっているが，あらかじめ用意できる部品には限りがある．ある特定の応用分野に必要な特殊な部品まで用意するのは不可能である．このため，後から必要に応じて新しい部品を追加していくことになる．その場合には，その部品を利用するための定義環境も同時に整備する必要がある．それには，定義環境が簡単に拡張できなければならない．

3. MU の概要

3.1 オブジェクト

MU では，メニューやテキストエディタなどの，多くの AP で共通に使われる UI 構成要素を，オブジェクトとして提供する．UI 開発者は既存のオブジェクトを自分用に手直ししたり，組み合わせたりして UI を構築する．

MU のオブジェクトは，属性，メソッド，ルールからなる．属性，メソッドは通常のオブジェクト指向型

システム[12),13)]と同様のものである．ルールにはユーザ操作に対するオブジェクトの動作を規定するイベントルールと，他のオブジェクトやAPからメッセージを受け取ったときの動作を規定するメッセージルールとがある．

MU の UIDL（以下，MU の UIDL を単に UIDL と呼ぶ）ではオブジェクトを次のような形式で定義する．

```
object 名前（親の名前）
{
    属性，メソッド，ルールの定義
}
```

名前は定義するオブジェクトの名前である．オブジェクトは親の名前で指定されたオブジェクトの属性，メソッド，ルールを継承する．**図1**に UIDL によるオブジェクトの定義例を示す．この例では，meter というオブジェクトを継承して新たなオブジェクト meter 1を定義している．

以下，属性，メソッド，ルールの概略，および AP とのインタフェースについて説明する．

（1） 属　　性

属性はオブジェクトの静的な性質や内部状態を指定する．すべてのオブジェクトは，表示位置と，領域を指定する属性（図1（1）～（4））をもつ．また個々のオブジェクトはそのオブジェクトに固有の属性をもつ．例えば，数値データを表示するためのメータは，スケールの最大，最小値などを属性としてもつ．

オブジェクトの組み合わせは，自分の台紙となるオブジェクトを指定する属性（図1（5））によって定義する．この属性によって，オブジェクトを階層的に組み合わせることができる．オブジェクトの位置は台紙となるオブジェクトに対する相対座標で指定する．台紙となったオブジェクトは，自分の上に乗っているすべてのオブジェクトに個々の名前を陽に指定せずにメッセージを送ることができる．これによって，組み合わせた複数のオブジェクトを一括して操作（移動，複製など）する機能が実現できる．

（2） メ ソ ッ ド

メソッドは，オブジェクトの基本的な動作を実現する．例えば，多くのオブジェクトは自分を表示したり，表示を消したりするためのメソッドをもつ．また，メータオブジェクトは数値データに従って針の位置を設定するメソッドをもつ．メソッドはそれが定義されているオブジェクト内でしか呼び出すことができない．他のオブジェクトから利用したい場合はメッセージルールを新たに定義する．

```
/* オブジェクトの定義 */
object meter 1 (meter)
{
/* 属性の設定 */
pos_x=10                                    /*-(1)*/
pos_y=10                                    /*-(2)*/
width=100                                   /*-(3)*/
hight=100                                   /*-(4)*/
min_value=0
max_value=100
title="FUEL"
current_value=0
default_value=0
/* 台紙の設定 */
base='sheet 1                               /*-(5)*/
/* 仮想オブジェクトの設定 */
virtual_obj=['Any, 'Fuel]                   /*-(6)*/
/* メソッドの定義 */
method output(x:int){                       /*-(7)*/
        if (x < min_value){x=min_value}
        if (max_value < x){x=max_value}
        set_value(x)
        get_value(current_value)
  }
/* イベントルールの定義 */
event hold(1){                              /*-(8)*/
        move_bar(1)
        get_value(value)
      /* AP 関数の呼び出し */
        call set_fuel_value(value)          /*-(9)*/
  }
/* メッセージルールの定義 */
message output(x:int)    {output(x)}        /*-(10)*/
message get_value(x:int){x=current_value}
message appear()         {appear() activate()}
message disappear()      {disappear() deactivate()}
}
```

図1 オブジェクトの定義例
Fig. 1 Example of object definition.

UIDL では次のような形式でメソッドを定義する．

```
method 名前（引数並び）{動作}
```

引数並びには引数名と型を指定する．動作は，メソッドの実行，他のオブジェクトへのメッセージ送信，AP 内のルーチン呼び出し（（5）参照）を組み合わせて定義する．図1（7）では output というメソッドを定義し，それを外部から実行できるように図1 (10)でメッセージルールとしても定義している．

（3） イベントルール

MU では，オブジェクト単位に定義されたイベントルールに基づいてユーザ操作を解釈する．イベントルールは，ユーザの操作とそれに対するオブジェクト

の動作の対を定義する．UIDL ではイベントルールを次のような形式で定義する．

event イベント名（デバイス）{動作}

イベント名はユーザ操作の種類を指定する．具体的には，マウスボタンの押し下げ，解放，キーボードからの入力などの，入力の種類を指定する．デバイスはイベントを発生するデバイス（どのボタンか，どのキーか）を指定する．動作の項は，入力イベント発生時に実行すべき処理の内容を定義する．図1(8)ではマウスボタン1が押し下げられたときに実行すべき処理を定義している．

入力イベントはそれが生成されたときにカーソル位置にあるオブジェクトに渡される．オブジェクトはその入力イベントに適合するルールがあれば，そのルールの動作部を実行する．カーソル位置に複数のオブジェクトが存在する場合には，最も最近活性化したオブジェクトに入力イベントを渡す．活性化とはオブジェクトが入力イベントを受け取れる状態にすることをいう．すべてのオブジェクトは自分を活性化するメソッドと不活性化するメソッドとをもつ．入力イベントは適合するルールがみつかるまで，候補のオブジェクトに新しく活性化した順に渡していく．カーソル位置にあるすべてのオブジェクトに適合するルールがなければその入力イベントは無視する．

(4) メッセージルール

オブジェクトは他のオブジェクトにメッセージを送って処理を依頼できる．メッセージルールはオブジェクトがメッセージを受け取ったときにどう応答するかを定義する．UIDL ではメッセージルールを次の形式で定義する．

message 名前（引数並び）{動作}

引数並び，動作の定義方法はメソッドの場合と同じである．

(5) AP とのインタフェース

MU では，現在，C言語で記述された AP に対するインタフェースを提供している．MU から AP へは，関数呼び出しによってアクセスする．すなわち，イベントやメッセージを受け取ったときの動作として，AP の機能を関数単位で呼び出せる．図1(9)では set_fuel_value という AP 内の関数を呼び出している．

一方，AP から MU へは，メッセージ送信によってアクセスする．ただし，メッセージの送り先は，直接オブジェクト名を指定するのではなく，仮想的なオブジェクト（仮想オブジェクトと呼ぶ）を介して指定する．各オブジェクトは自分がどの仮想オブジェクトに対応するかを指定する属性 virtual_obj をもつ．この属性値として複数の仮想オブジェクト名を指定できる．図1(6)では仮想オブジェクトとして Any と Fuel の2つを指定している．AP からのメッセージは送り先の仮想オブジェクトに対応するすべてのオブジェクトに送信される．すなわち，仮想オブジェクトと，実際のオブジェクトとは多対多で対応付けできる．図2に例を示す．同図において，オブジェクト bar1 は仮想オブジェクト Any に対応し，オブジェクト meter 1 は2つの仮想オブジェクト Any と Fuel に対応している．AP が appear というメッセージを仮想オブジェクト Any に送ると，このメッセージは bar 1 と meter 1 の両方のオブジェクトに送られ画面に2つのオブジェクトが表示される．次に AP が output というメッセージを仮想オブジェクト Fuel に送ると，このメッセージは Fuel に対応するオブジェクト meter 1 にだけ送られ，meter 1 の針が AP のデータに合わせて設定される．

仮想オブジェクトを導入することにより，AP を再コンパイルせずに，メッセージの実際の送り先を変更することができる．例えば図2において，メータで表示していたデータをバーグラフ表示に変更したい場合，オブジェクト bar 1 の属性 virtual-obj に仮想オブジェクト Fuel を追加し，オブジェクト meter 1 の属性 virtual-obj から仮想オブジェクト Fuel を削除すればよい．

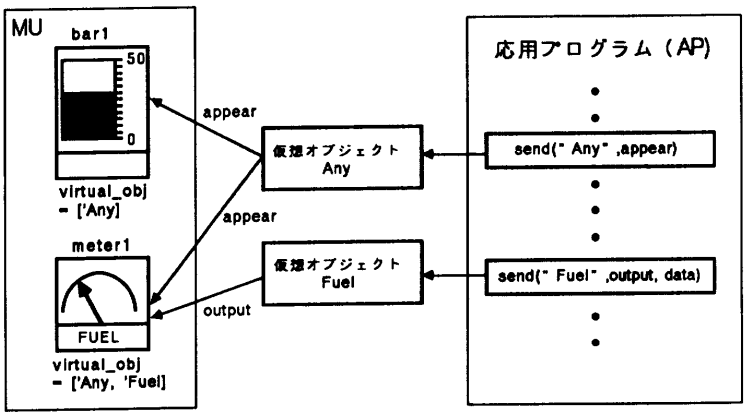


図2 仮想オブジェクト
Fig. 2 Virtual object.

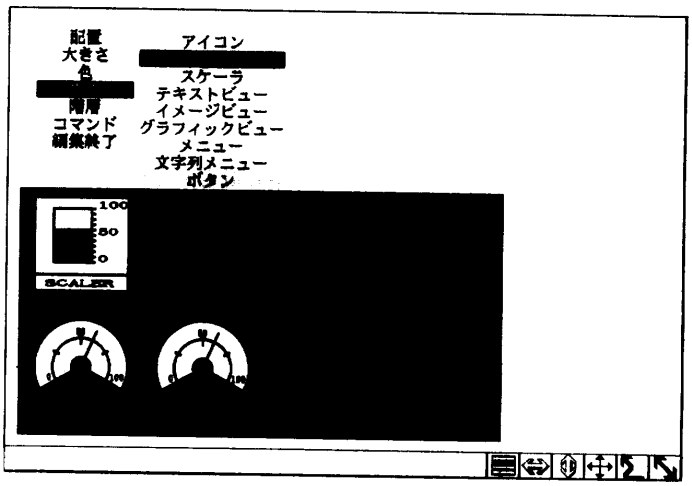


図 3 オブジェクトの生成
Fig. 3 Generating object.

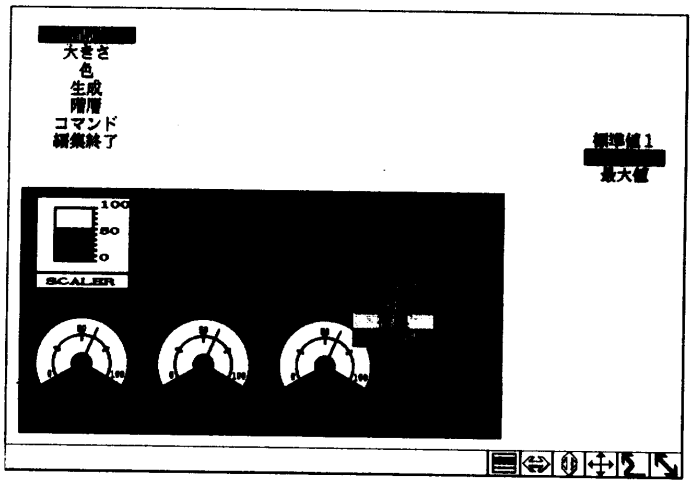


図 4 オブジェクトの複製
Fig. 4 Duplicating object.

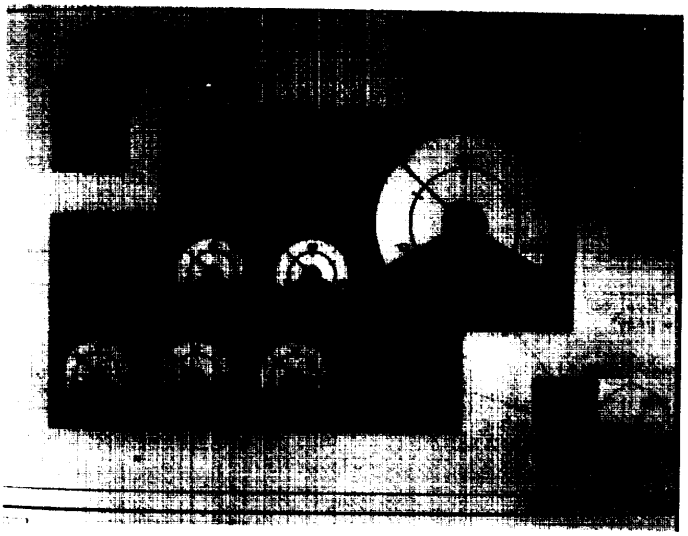


図 5 色の設定
Fig. 5 Setting color.

3.2 UI 定義環境

MU における UI の開発は，UIDL と WYSIWYG 方式を併用して進められる．UIDL だけを用いて開発することも可能であるが，表示形式や画面配置に関する属性の設定は WYSIWYG 方式でも設定できるようにしてある．本節では，WYSIWYG 方式の内容と UIDL との併用方法，定義環境と実行環境の切り替え操作などを具体的に説明するため，MU における典型的な UI 開発手順を示す．

（1） オブジェクトの生成

まず，既存のオブジェクトの中から自分の用途に最も合ったものを選択し，その属性，機能を継承するオブジェクトを生成する．オブジェクトは，**図 3** に示すようにメニューを選択して生成する．このメニューによって MU が標準的に提供するオブジェクトを生成できる．また，**図 4** に示すように画面上のオブジェクトを直接指定してその複製を生成することもできる．オブジェクト上でマウスのボタンを押し下げると，ポップアップメニュー（図 4 中央のメニュー）が表示される．そのメニューの“複製”という項目を選択すると，マウスで指定したオブジェクトの複製が生成される．複製されたオブジェクトは，元のオブジェクトの属性，メソッド，ルールを継承する．この方法は似たようなオブジェクトを幾つも使うときに便利である．

（2） 画面デザイン

生成したオブジェクトの大きさ，配置，組み合わせ，色などは UIDL でも設定できるが WYSIWYG 方式で設定した方が簡便な場合が多い．WYSIWYG 方式では，オブジェクトの配置，大きさはマウスのドラッグ操作（ボタンを押し下げた状態でマウスを動かす操作）で指定する．オブジェクトの組み合わせは，台紙としたいオブジェクトをマウスとポップアップメニューを使って選択し，次にその上に載せたいオブジェクトを同様な方法で選択するという手順で定義する．オブジェクトの色は，色を変更する部分を選択するポップアップメニュー（**図 5** 中央のメニュー）とカラーパレットメニュー（図 5 右上のメニ

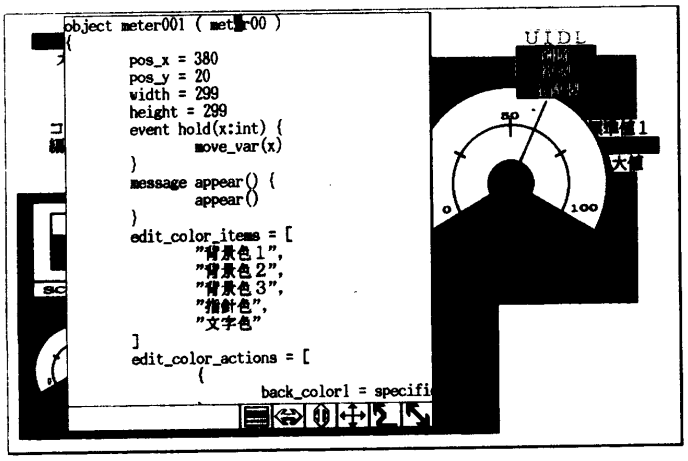
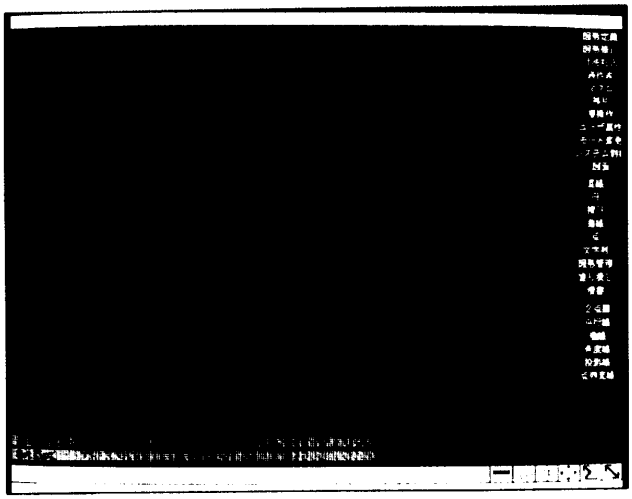
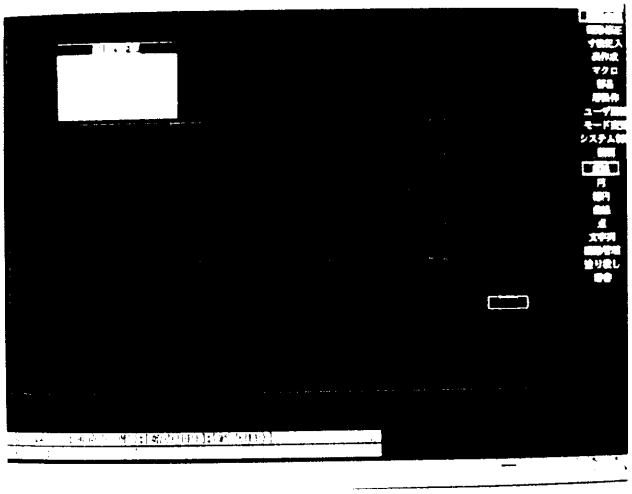


図 6 UIDL の編集
Fig. 6 Editing UIDL



(1) 図面作成時



(2) UI 変更時

図 7 エンドユーザによる手直し
Fig. 7 Customized by end user.

ュー) とを使って設定する.

(3) 動作の定義

生成したオブジェクトのメッセージやイベントを受け取ったときの動作を新たに定義する場合は, そのオブジェクトの UIDL ソース内のルールを編集する. UIDL ソースを編集したいオブジェクトをマウスでピックし, そのとき表示されるポップアップメニューの "UIDL" という項目を選択すると, **図 6** のようにテキストエディタが開き, ピックしたオブジェクトの UIDL ソースが表示される. この UIDL ソースを編集してルールを定義する. もちろんその場でルールだけでなく属性値も設定できる. エディタを閉じると, UIDL の編集結果が自動的にコンパイルされ, WYSIWYG 方式の定義環境に戻る. UIDL 編集時に表示に関連する属性を変更した場合には, 変更内容がすぐ画面上に反映される.

(4) 実行, デバッグ

上記の手順により UI の定義がある程度完了したら, それを実際に実行してデバッグする. メニューで編集終了を指定すると, 即座に実行環境に移行する. 実行環境では, 例えばマウスで画面上のオブジェクトをピックすれば, 手順(3)で定義したルールに従って所定の動作が実行される. これにより, イベントルールのデバッグができる. 実際に動作させてみて不具合がみつかれば, 実行を一時的に中断して定義環境に戻り, 定義内容を修正する. 現在の MU では実行時に特定のキーを押すと定義環境に移行するように設定してある. この設定はイベントルールとして定義してある. このキーを AP に対する操作用に使いたいときには, ルールの入力イベントの項を AP では使わない別なキーに設定する.

また, メッセージルールの単体デバッグも可能である. 定義環境のメニューで "コマンド" という項目を選択するとキーボードから入力した文字列を UIDL のソースコードとして実行できる. メッセージルールをデバッグしたいときには, この機能を使ってデバッグしたいオブジェクトに直接メッセージを送ってみればよい.

(5) エンドユーザによるカスタマイズ

上記の手順によりAPが完成した後も，APのエンドユーザが自分の好みに合わせてUIを変更したいということも多い．そうした場合，エンドユーザはAPの実行を一時中断して，MUの定義環境を呼び出しUIを変更する．**図7**は，MUで開発したCADのシステムで，図面作成中に定義環境を呼び出し，メニューの位置をドラッグ操作で変更しているところを示している．図7(1)は図面作成中のCADシステムの画面例である．この状態で特定のキーを押すとMUの定義環境が呼び出され，図7(2)のように画面左上にUI定義用のメニューが表示される．このときにメニュー上にカーソルを移動しマウスをドラッグするとメニューの位置が変更される．図7(2)の右下の矩形はマウスのドラッグ操作によって移動中のメニューを示している．

4. MUの実現

4.1 システム構成

MUのシステム構成を**図8**に示す．コンパイラはUIDLソースコードを定義データに変換し，それをインタプリタが解釈実行する．デコンパイラは指定されたオブジェクトの定義データをUIDLソースコードに変換する．組み込みメソッドは，UIDLレベルでは記述できない低レベルの入出力に関連するメソッドなどで，C言語で実現されている．APは，MUと同一のロードモジュールとしてリンクされる．APは関数の集まりとして構成することもできるし，APからMUのインタプリタを関数として呼び出すこともできる．

本システムは68020ベースのUNIXワークステーション上で開発した．メタUIすなわちMU自身のUIはUIDLで記述し，その他の構成要素はC言語で記述した．それぞれの概略のステップ数を図8中に付記する．コンパイラの開発にはUNIXのプログラム開発ツールであるLEXとYACCを使用した．組み込みメソッドと，インタプリタ内のイベント入力ルーチンはターゲットマシンに固有のグラフィック入出力ルーチンを使用しているため，他のマシンへの移植の際には変更が必要であるが，他の部分に関しては容易に移植可能である．

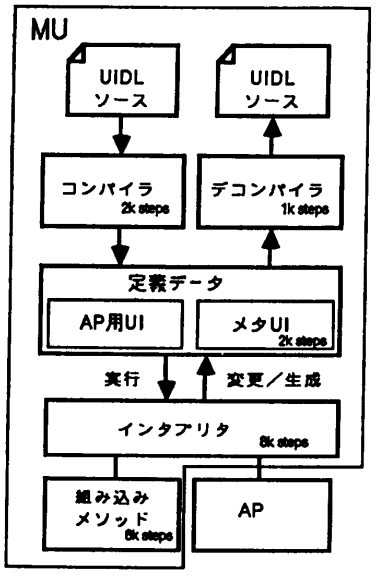


図8 システム構成
Fig. 8 Architecture of MU.

4.2 定義データの構造とその実行

図9にコンパイラによってメモリ上に展開された定義データの構造を示す．オブジェクトディレクトリはすべてのオブジェクトデータへのポインタを格納する．オブジェクトデータは親のオブジェクト，台紙オブジェクト，自分より1つ前に活性化されたオブジェクトへのポインタをもつ．これにより，オブジェクトデータは継承関係，組み合わせ(3.1節参照)に従ってそれぞれ木構造上につながれ，活性化順序(3.1節参照)に従ってリスト状につながれる．オブジェクトの属性，メソッド，イベントルール，メッセージルールに関する情報はそれぞれテーブルに格納され，オブジェクトデータはそれらのテーブルへのポインタをもつ．属性テーブルは属性名，データの型，属性値からなる．メソッドテーブル，ルールテーブルはメソッド，

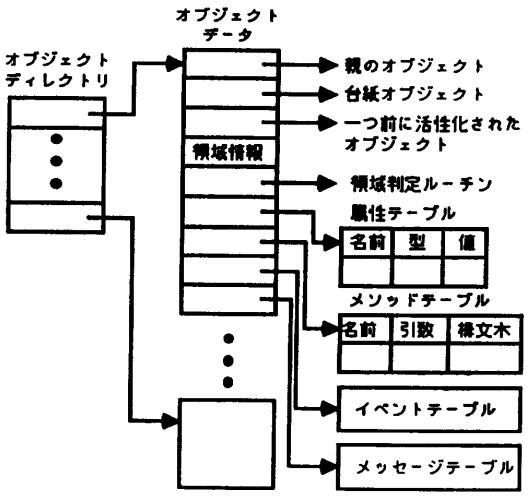


図9 定義データの構造
Fig. 9 Structure of definition data.

イベント，メッセージなどの名前，引数情報（個数，型），動作を規定する中間コードからなる．中間コードはコンパイラで解析した結果の構文木である．さらに，オブジェクトデータは領域情報（画面上でそのオブジェクトが占める領域）と領域判定ルーチン（イベントが領域内で発生したか否かを判定する）へのポインタをもつ．

インタプリタは，ユーザからのイベントを入力すると，活性化順序のリストに従って最も最近活性化されたものから順に各オブジェクトの領域判定ルーチンを起動する．イベントが領域内にあるオブジェクトがみつかると，そのオブジェクトのイベントテーブルから適合するイベントルールを検索する．そのオブジェクトに適合するイベントルールがない場合は親のオブジェクトのイベントテーブルを探索する．適合するイベントルールがみつかるとその中間コードを実行する．

4.3 UIDL と WYSIWYG 方式との切り替え

デコンパイラの役割は，WYSIWYG 方式による定義と UIDL による定義との一貫性を保つことにある．UIDL ソースはコンパイラで定義データに変換された後，メモリ上に展開される．WYSIWYG 方式ではメモリ上の定義データを直接変更する．メモリ上の定義データは変更された時点で元の UIDL ソースコードとは一貫性がなくなる．WYSIWYG 方式による定義と UIDL による定義とを併用するには，メモリ上の定義データの内容を正しく反映する UIDL ソースが必要になる．デコンパイラはメモリ上の定義データを UIDL ソースコードに逆変換することで，メモリ上の定義データと一致する UIDL ソースを生成する．

ユーザが WYSIWYG 方式から UIDL 編集への切り替えを要求すると（3.2 節(3)参照），MU はデコンパイラを呼び出して指定されたオブジェクトの定義データを UIDL ソースコードに変換し，ファイルに書き出す．続いてそのファイルを編集するためのエディタ（現在，UNIX の vi を使用）を呼び出す．ユーザが編集を終了するとコンパイラを呼び出してファイル内の UIDL ソースを定義データに変換し，元のオブジェクトデータと置き換える．

4.4 メタ UI の実現

メタUIは画面上でのWYSIWYG方式によってオブジェクトの属性を設定する機能（例えば，オブジェクトの表示位置や大きさをマウスによるドラッグ操作によって設定する機能など）を主に提供する．メタ UI を実現する上で考慮すべき点は，2章で述べたように，

1) UI 開発者がメタ UI を簡単に手直しできること，

2) 新たなオブジェクトの追加が容易なこと，

である．MU では上記要件に対してそれぞれ，

a）メタ UI 自身を UIDL で記述する，

b）共通の定義機能は継承の階層において上位のオブジェクトで実現し，個々のオブジェクトに固有な機能は下位のオブジェクトで実現する，

ことにより対処している．a）により，UI 開発者は AP の UI を定義するのと同じ要領でメタ UI 自身を手直しできる．また，b）により，新たなオブジェクトを追加しても，その上位のオブジェクトで定義されている定義機能はそのまま使うことができる．新しく追加したオブジェクトに固有な属性は，とりあえずは UIDL を用いて定義できる．それらを WYSIWYG 方式によって定義したいときだけ，そのための定義機能を新たにそのオブジェクト内に実現すればよい．

上記b）を具体的に説明するため，まず MU におけるオブジェクトの継承に基づく階層構成を説明する（**図 10** 参照，ただし本図は実際の構成より若干単純化してある）．最上位のテンプレートにはすべてのオブジェクトに共通な機能が定義されている．その下には，UI の基本的な構成要素となる，メニュー，アイコン，メータといったオブジェクトがある．これらのオブジェクトの下位に，UI 開発者が AP の UI 用に定義したオブジェクトが位置する．

ドラッグによるオブジェクトの位置，大きさの指定，複製の生成，UIDL ソースの編集，などの定義機能は，すべてのオブジェクトに共通に使われる機能でありテンプレートで実現する．下位のオブジェクト

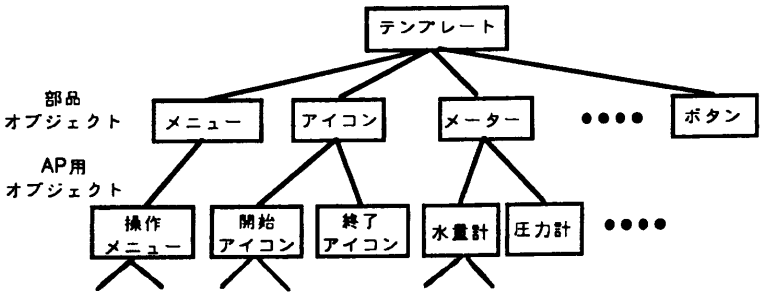


図 10 オブジェクトの継承に基づく階層構成
Fig. 10 Inheritance hierarchy of objects.

が，位置，大きさの属性と，表示および表示を消去するメソッドさえもっていれば，テンプレートで実現する定義機能はそのまま使用できる．一方，メニューの項目を設定する機能や，メータの目盛りを設定する機能などは，それぞれのオブジェクトに固有な機能であり，それぞれのオブジェクトで実現する．新たに UI 構成要素となるオブジェクトを追加する場合には，そのオブジェクトに固有な定義機能だけを追加すればよい．

4.5 定義環境と実行環境との切り替え

メタ UI を上記のように UIDL で実現するには，1つの入力イベントに対して AP 実行時と UI 定義時とで実行すべき処理を切り替える必要がある．例えば，メニューの上でマウスのボタンが押し下げられた場合，AP 実行時にはメニュー項目の選択処理を実行し，UI 定義時にはメニュー位置の移動処理を実行しなければならない．この切り替えは，実際の入力イベントと UIDL 上でのイベント名との対応表を切り替える方式で実現している．

例えば，メニューオブジェクトに次の2つのイベントルールを定義しておく．

```
event hold (1) {select_item (1)}
event ev_hold (1) {move (1)}
```

そして，ボタンの押し下げというイベントを AP 実行時にはイベント名 hold に対応させ，UI 定義時にはイベント名 ev_hold に対応させる．これにより，AP 実行時にボタン1を押し下げると hold という名前のイベントが発生しメソッド select_item が実行される．一方，UI 定義時にボタン1を押し下げると ev_hold という名前のイベントが発生しメソッド move が実行される．なお，上記イベント名と実際のイベントとの対応付けは次のメソッドを呼び出して行う．

```
event_map ([name 1, …, name 5])
```

これにより name 1～name 5 がそれぞれボタンの押し下げ，解放，マウスのドラッグ，マウスの移動，キーの押し下げに対応するイベント名として設定される．

この方式では，AP 実行環境と UI 定義環境との切り替えが，イベントの対応表を切り替えるだけでできき，高速に実現できる．また，メタ UI に関するイベントルールの記述と AP 用のイベントルールの記述とが完全に独立しており，AP 開発時に定義環境用のイベントルールに注意を払う必要がない．

5. 評価と検討

（1） UIDL と，WYSIWYG 方式による定義との切り替え

UIDL と，WYSIWYG 方式による定義とは，短時間で切り替えられる．実際，平均的なオブジェクト（UIDL ソース 50 行程度）の場合，オブジェクトを指定してから，UIDL ソースの編集が可能になるまでの所要時間は，1.2 秒程度である．この間にはデコンパイルと，UIDL を編集するためエディタの呼び出しを行っている．逆に，UIDL ソースの編集終了後，WYSIWYG 方式による定義が可能になるまでの所要時間も同程度である．この間には，コンパイルと，画面に表示中のオブジェクトの再表示を行う．したがって，表示オブジェクトの数が多くなると再表示に時間がかかるが，表示オブジェクトが十数個程度のときには，2秒以下で切り替えられる．このため，UI 開発者は気軽に UIDL と WYSIWYG 方式による定義とを併用できる．

（2） 定義環境と実行環境との切り替え

実行環境から定義環境へは，特定のキーの押し下げによって，また定義環境から実行環境へはメニューの選択によって簡単に切り替えられる．どちらも切り替えを指定してから 0.5 秒以下で切り替えが完了するので，ユーザが待ちを感じることはない．このように定義環境と実行環境とを即座に切り替えられることは試行錯誤的な開発には有効である．

（3） 定義環境の変更，拡張

MU の定義環境は UIDL を用いて，1人の開発者が約3週間で実現した．これは，UIDL の開発効率の高さを示すとともに，定義環境自身の手直しの容易さをも示している．

（4） 実 用 性

従来の UIMS の多くはラピッドプロトタイピングを目的としており，実際，それらを使って開発された AP のほとんどは小規模なプロトタイプであった．MU では単なるプロトタイプの開発支援だけでなく，すでに実用規模の CAD システムの開発に適用している．その CAD システムの UI は階層型メニューシステム（図7参照）で，100 以上のオブジェクトから構成されている．最上位のメニューを選択するとデフォルト値に従って，3から4階層下のメニューまで展開されるが，それに要する時間は 0.5 秒程度である．また，その UI は UIDL の行数にして約1万行の大き

さであるが，実行時の定義データの大きさは約 600 k バイト程度である．応答速度，メモリ効率の点からも十分な実行性能が得られていることがわかる．

6. おわりに

MU は実際に日立の CAD システム (HICAD/W)，文書処理システムの開発に適用し，その実用性を確認している．しかし，部品として提供しているオブジェクトがメニューやテキストエディタなどの基本的なものに限られているため，より幅広い分野の AP に適用していくには部品の種類を増やしていく必要がある．そこで，われわれは，画面上に自由に描いた絵をそのまま UI として利用する機能を MU 上に実現中である．これにより，場当たり的に部品を追加することなく，広範囲の UI を実現できるようになる．

謝辞 本研究に当たり御助言，御討論頂いた日立製作所ソフトウェア工場図形部の方々に深謝いたします．

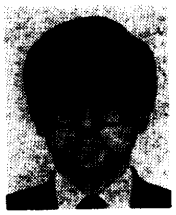
参考文献

1) Thomas, J.J. et al.: Graphical Input Interaction Technique Workshop Summary, *ACM Comp. Graph.*, Vol. 17, No. 1, pp. 5-30 (1982).
2) Pfaff, G.E.: *User Interface Management Systems*, Springer-Verlag (1985).
3) Olsen, D.R., Jr. (ed.): ACM SIGGRAPH Workshop on Software Tools for User Interface Management, *ACM Comp. Graph.*, Vol. 21, No. 2, pp. 71-147 (1987).
4) Schulert, A.J. et al.: ADM—A Dialog Manager, *ACM CHI '85 Proc.*, pp. 177-183 (1985).
5) Green, M.: The University of Alberta User Interface Management System, *ACM SIGGRAPH '85 Proc.*, Vol. 19, No. 3, pp. 205-213 (1985).
6) Hill, R.D.: Supporting Concurrency, Communication, and Synchronization in Human-computer Interaction—The Sassafras UIMS, *ACM Trans. Graph.*, Vol. 5, No. 3, pp. 179-210 (1986).
7) 竹村ほか：ユーザインタフェース作成支援システムの設計と試作について，情報処理学会論文誌，Vol. 28, No. 5, pp. 506-515 (1987).
8) Myers, B.A. and Buxton, W.: Creating Highly-interactive and Graphical User Interfaces by Demonstration, *ACM SIGGRAPH '86*, Vol. 20, No. 4, pp. 249-258 (1986).
9) Olsen, D.R., Jr.: MIKE: The Menu Interaction Kontrol, *ACM Trans. Graph.*, Vol. 5, No. 4, pp. 318-344 (1986).
10) Bentley, J.: Programming Pears: Little Languages, *ACM Communications*, Vol. 29, No. 8, pp. 711-721 (1986).
11) Olsen, D.R., Jr. et al.: Input/Output Linkage in a User Interface Management System, *ACM SIGGRAPH '85*, Vol. 19, No. 3, pp. 191-197 (1985).
12) Barth, P.S.: An Object-oriented Approach to Graphical Interfaces, *ACM Trans. Graph.*, Vol. 5, No. 2, pp. 142-172 (1986).
13) Sibert, J.L. et al.: An Object-oriented User Interface Management System, *ACM SIGGRAPH '86 Proc.*, Vol. 20, No. 4, pp. 259-268 (1986).

(昭和 63 年 9 月 30 日受付)
(平成 元 年 6 月 13 日採録)

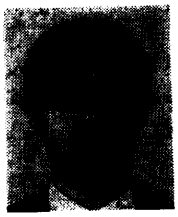
谷 正之 (正会員)

1956 年生．1980 年東京工業大学工学部電気・電子工学科卒業．1982 年同大学院物理情報工学専攻課程修了．同年(株)日立製作所日立研究所に入社．文書処理，連想記憶，オブジェクト指向プログラミング環境の研究・開発を経て，ユーザインタフェース関連ソフトウェアの研究・開発に従事．現在 MIT メディアラボにて客員研究員としてユーザインタフェースの研究中．



荒井 俊史 (正会員)

1964 年生．1986 年東京工業大学理学部情報科学科卒業．現在，(株)日立製作所日立研究所にてユーザインタフェース関連ソフトウェアの研究・開発に従事．プログラミング言語，プログラミング環境，OS 一般に興味をもつ．



谷越浩一郎 (正会員)

1962 年生．1985 年東京工業大学理学部情報科学科卒業．1987 年同大学院情報科学専攻課程修了．同年(株)日立製作所日立研究所に入社．ユーザインタフェース関連ソフトウェアの研究・開発に従事．マイマシンインタフェース，ヒューマンファクタ，プログラム言語／環境に興味をもつ．

横山　孝典（正会員）

1959年生．1981年東北大学工学部通信工学科卒業．1983年同大学院工学研究科電気及通信工学専攻修士課程修了．同年(株)日立製作所入社．日立研究所にて図形認識，ユーザインタフェースの研究開発に従事．1987年より(財)新世代コンピュータ技術開発機構に出向．知識表現の研究に従事．電子情報通信学会会員．

谷藤　真也（正会員）

昭和22年生．昭和48年早稲田大学理工学研究科修了．同年(株)日立製作所日立研究所入社．鉄鋼プロセスの計算機制御，知識工学応用システムの研究に従事．現在はエンジニアリングワークステーションのソフトウェアおよびユーザインタフェースの研究を担当．計測自動制御学会，電気学会などの会員．