解　説

ユーザインタフェース管理システム (UIMS)

# 3. ユーザインタフェース管理システムの研究動向と将来†

橋　本　治††

## 1. はじめに

グラフィカルユーザインタフェース (GUI) は，マウスなどのポインティングデバイスを用いてグラフィックスと対話する視覚的な操作環境である．GUI では，グラフィックスを用いてデータを表現し，画面上でグラフィックスを変更するとデータも変わり，逆にアプリケーションの内部処理などでデータが変更されるグラフィックスも変化するので，ユーザは内部データを直接操作している（直接操作法）ような感覚をもつことができる[1]．近年，ユーザ層の拡大によって GUI の必要性が高まる一方，ワークステーションやパソコンの処理能力向上によって多くのシステム上で GUI の実現が可能になっている．

しかし，GUI を開発するのは大変に困難な作業である．たとえば，ワークステーション上ではXウィンドウを用いて実現するのが一般的であるが，その作業は熟練したプログラマでも多大な開発工数を必要とする困難なものである．また，現状では UI の品質向上は試行錯誤的な開発に頼らざるをえない点[1]や，ユーザの好みやアプリケーションの内容によって個別に UI を変更する必要がある点などからも，GUI を効率よく開発する技術は重要である．

本稿では，UIMS を GUI を効率よく構築するための実行ライブラリと支援ツールとしてとらえて，その研究動向と将来の方向性を紹介する．**2.** ではウィンドウシステム上の GUI 実行ライブラリの集合である GUI 構築環境について，**3.** ではプログラマ以外の人でも使えることを目指した支援ツールである GUI 構築ツールについて最新の研究を紹介する．最後にこれらの将来について UI の観点からいくつかの方向性を示す．

† The State-of-the-Art and Perspectives on Graphical UIMSs by Osamu HASHIMOTO (NEC Corporation, C&C Systems Research Laboratories).

†† NEC C&C システム研究所

## 2. GUI 構築環境

GUI 構築環境は，ウィンドウ/グラフィックスの基本的なライブラリ関数群（Xウィンドウの場合 Xlib[2]）の上に位置して，GUI 構築を容易にするような構造と機能をもった高水準な実行ライブラリの集合であり，Xウィンドウでは XToolkit[3] や InterViews[4),5] などが有名である．これらについてはすでに多数の解説や紹介があるので，本稿では CMU で開発された Garnet[6)~8)] の技術を紹介する．Garnet はオブジェクト指向システム (KR)，制約システム，グラフィックスシステム (Opal)，入力処理システム (Interactor) の4つの部分からなり，Unix とXウィンドウ (Xlib のみ利用) の上で Common Lisp によって実現されている．

### 2.1 オブジェクト指向システム

直接操作法では，ユーザは画面上の複数の独立したグラフィックスオブジェクトと対話するので，オブジェクト指向は直接操作法にフィットしたモデルである[9]．Garnet のオブジェクト指向システム KR[10] は，Smalltalk などで採用されたクラス-インスタンスモデルではなく，プロトタイプ-インスタンスモデルを採用している．このモデルではオブジェクトにはクラスとインスタンスのような区別がなくて，どのインスタンスでも新しいインスタンスのプロトタイプになれる．オブジェクトは任意のスロットをもつことができ，そこにデータやメソッドを格納する．このモデルはクラス-インスタンスモデルに比べて，①インスタンスにおいて上書きしなかったスロットはプロトタイプのスロットの値を継承する，②動的に追加可能なスロットをローカル変数として使うこと

ができる，③インスタンスもプロトタイプも画面に表示できる，という点で GUI 構築のベースに適している．

このモデルは，知識表現や獲得の研究に関連しており，人間が新しい知識を獲得する際にはこれまでに得た概念との類推（analogy）を働かせている，という考えをベースにしている[11),12)]．そのために具体的で視覚的な応用に適している．また，既得の知識（デフォルト）と異なる部分だけを記述すればよいので，“差分プログラミング”が可能である．さらに，動的で柔軟なので記述力が高いうえに，概念的に簡単なので理解しやすい．

### 2.2 制約システム

オブジェクト指向は直接操作法に適しているが，独立したオブジェクト群を統合して GUI を構築するには，オブジェクト相互の関係をうまく表現したり，グルーピングする仕組みが必要である．Garnet ではオブジェクト間の関係を制約という形式で宣言的に記述すると，以降はオブジェクトが変化しても制約システムがその関係を自動的に保つ（グルーピング機能は後述の Opal が提供する）．

制約は，オブジェクトのスロットに Lisp の式で記述するが，式の中で他のオブジェクトのスロットを参照できる．そして，参照先のスロットが変化すると，制約の式が自動的に再評価される．**図-1** に制約の利用例を示す．**図-1**（a）では，my-line の右端の座標（:$x2$, :$y2$）は制約（o-formula 以下）で表現されており，my-box 2 の位置情報を参照している．たとえば，:$x2$ は my-box 2 の :left スロットを参照（gv）している．左端の座標（:$x1$, :$y1$）は my-box 1 で決まる．ここで my-box 2 を :left＝200 の位置に移動すると，**図-1**（b）のように my-line の右端の座標も制約によって変更され（この場合 :$x2$ が 200 になる），my-box 2 とともに右に移動する．

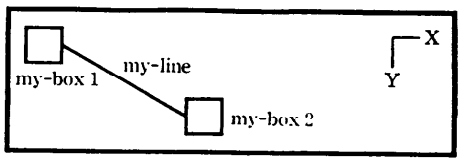
一般的な制約は，$x=y+z+10$ などのように双方向的な関係であるが，このような制約を効率よく解消して自然な結果を得るには課題が多く必ずしもうまくいっていない[13)]．そこで，Garnet では一方向の制約（$x \leftarrow y+z+10$ のように参照方向を指定）に限定している．また，グラフィックスオブジェクト同士の関係だけでなく，グラフィックスと内部データを制約を用いて結び付けることもできる．

制約は宣言的な表現で動的なグラフィックスを記述できるので，記述が容易であり読みやすい．また，グラフィックスとアプリケーションの関係も制約で表現できるので，画面上のグラフィックスと内部データが一体化してユーザと対話する，という直接操作法の構造を簡単に表現できる．

### 2.3 グラフィックスシステム

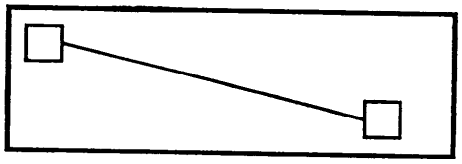
X ウィンドウ（Xlib）などでグラフィックスを描画したり，マウスなどの入力処理を実現するのは容易ではない．X ウィンドウの概念を理解するとともに，数多くの関数とそのパラメタの意味や使い方を修得する必要がある．特に，入力処理を実現するイベントモデルは原始的であり，実際の操作との間に意味的なギャップが大きい．

グラフィックスシステム Opal はグラフィックス表示の実現を支援する（入力処理の実現は後述の Interactor が支援する）．Opal はグラフィックスをオブジェクトとして管理し，各種グラフィックスのプロトタイプを提供する．これらプロトタ




```
(create-instance 'my-box 1 opal : rectangle
  (:left 10) (:top 10)
  (:width 20) (:height 20))
(create-instance 'my-box 2 opal : rectangle
  (:left 100) (:top 50)
  (:width 20) (:height 20))
(create-instance 'my-line opal : line
  (:x1 (o-formula (+ (gv my-box 1 :left)
        (gv my-box 1 :width))))
  (:y1 (o-formula (+ (gv my-box 1 :top)
        (/ (gv my-box 1 :height) 2))))
  (:x2 (o-formula (gv my-box 2 :left)))
  (:y2 (o-formula (+ (gv my-box 2 :top)
        (/ (gv my-box 2 :height) 2)))))
```

（a）



（b）

**図-1** 制　約

イプの属性には典型的な値が設定されている．したがって，グラフィックスプロトタイプからインスタンスを作成して，わずかな属性を指定するだけでグラフィックス表示を実現できる．たとえば，

```
(create-instance 'my-rect1 opal:rectangle
  (:left 10) …)
```

によって矩形プロトタイプ opal:rectangle からインスタンス my-rect1 を作成できる．グラフィックスの位置や大きさや色などを変えるには，対応する属性（スロット）を変えるだけでよい．たとえば，

```
(s-value my-rect1 :width 90)
```

によって my-rect1 の幅を変更できる．

このように，Opal ではグラフィックスの描画や消去や変更のために複雑なライブラリ関数を呼び出す必要がない．Xウィンドウの描画機能が全て隠され適切なデフォルトが設定されているので，Xウィンドウの知識がなくてもグラフィックス表示を実現できる．一方，細かに属性を指定すればXウィンドウのもつ豊富で強力な機能を容易に引き出せる．また，オブジェクトが移動／サイズ変更／消去／属性変更されたときには，Opal が自動的に他のオブジェクトを書き直す．さらに，オブジェクトをグルーピングする機能も提供している．

### 2.4 入力処理システム

GUI の操作法を分類してみると，まったく異なる振舞いはそれほど多くない．たとえば，スクロールバーにはさまざまな形態があるが，その操作法はほとんど同じである．Garnet では GUI の主な操作法を6種類に分類し[14]，これらに対応する汎用的な入力処理プロトタイプ（Interactor）を提供している：

- Menu-Interactor（メニュ操作）
- Button-Interactor（ボタン操作）
- Move-Grow-Interactor（移動／サイズ変更操作）
- Two-Points-Interactor（大きさ／範囲指示操作）
- Angle-Interactor（回転操作）
- Text-Interactor（文字入力操作）

これらプロトタイプの属性（振舞い）には，典型的な値が設定してあるので，インスタンスを作成してわずかな属性を指定し，操作の対象となるオブジェクトを指定するだけで入力処理が実現できる．たとえば図-2 において，killer（ア）は，button-interactor のインスタンスであり，my-text（"Goodbye World"）を左ボタンで押される（イ）と，:final-function スロットの関数を呼ぶ（ウ）．この関数の中で (opal:destroy my-win) は，ウィンドウ my-win を消去する．つまり，このプログラムは，まず "Goodbye World" という文字をもつウィンドウを表示し，この文字をマウスの左ボタンで押されるとウィンドウを消す，というものである．

このように Interactor は操作対象とするオブジェクトの名前だけを知っていればよいので，入力処理がグラフィックス表示から分離される．また，Xウィンドウのイベント処理による入力機能が完全に隠され適切なデフォルトが設定されているので，Xウィンドウの知識がなくても入力処理を実現できる．一方，細かにパラメータを指定すれば入力処理の振舞いを自由に制御することができる．

最後に，GUI の実現モデルとして有名な MVC モデル[15]で Garnet をみると，モデル (M) はアプ


```
(in-package "USER" :use '("KR" "LISP"))
(create-instance 'my-win
  inter:interactor-window
  (:width 150) (:height 100))
(create-instance 'my-agg opal:aggregate)
(s-value my-win :aggregate my-agg)
(create-instance 'my-text opal:text
  (:string "Goodbye World")
  (:left 20) (:top 40))
(opal:add-components my-agg my-text)
(opal:update my-win)
ア (create-instance 'killer
     inter:button-interactor
   (:window my-win)
  イ (:start-where (list :in my-text))
   (:continuous NIL)
  ウ (:final-function
      #' (lambda (inter final-obj-over)
           (opal:destroy my-win)
           (inter:exit-main-event-loop))))
(inter:main-event-loop)
```

図-2 Interactor を使った簡単なプログラム例

リケーション，ビュー(V)はOpal，コントローラ(C)はInteractorに相当する．VとCは分離されるべきだが，Garnetでは上述のように完全に分離されている．MとVの間は直接操作法実現のために密な情報交換が必要であるが，Garnetでは制約によって高速で密な参照が実現されている．

## 3. GUI構築ツール

GUI構築ツールは，プログラマ以外の人でも使えることを目指した支援ツールであり，GUI構築に便利な機能を提供し，プログラミングに頼る部分を減らしている．これによって，デザイナや認知心理学やアプリケーションの専門家が，GUI開発に直接参加できる．ここでは，まず現状を説明した後で，注目される研究動向を紹介する．

### 3.1 現　状

#### 3.1.1 エディタ方式

エディタ方式では，グラフィックスエディタで絵を描くように，画面上で対話的にGUI部品を配置したり，それらの形状や特性などを編集することによってGUIを作成する．作成したGUIはプログラムコードに変換され出力される．Next Step (NeXT社)，TeleUSE (TeleSoft社)，ゆず (NEC) などがすでに商用化されており，UIビルダと呼ばれている．ここで，GUI部品とはメニュやボタンなどのような対話操作の基本単位であり，ウイジェットと呼ばれることがある．

UIビルダはほぼ共通した構成であり，作業用ウィンドウ(**図-3** 上)と部品用ウィンドウ(**図-3** 中央)と部品属性の編集用ウィンドウ(**図-3** 下)をもっている．部品用ウィンドウ（パレット）にはGUI部品が置かれており，これらを使って作業用ウィンドウの上でGUIを作成する．部品属性の編集用ウィンドウには部品の属性リストが表示されるので，これらを編集して属性を設定する．作成したGUIはプログラムコードに変換される．

本方式は視覚的で具体的であり，常に出来映えを確認しながらGUIを作成できる．修得が容易なので，プログラマ以外の専門家が直接GUIの

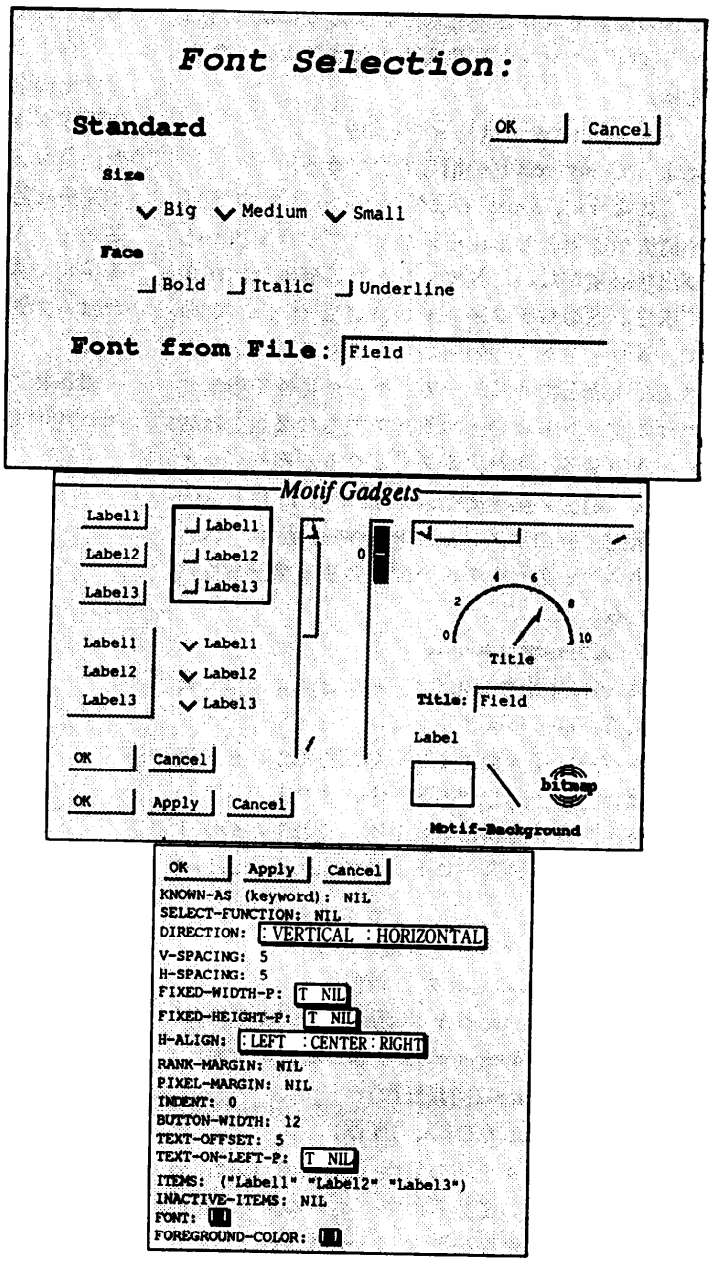



**図-3** UIビルダの構成[11]

作成に参加できる．しかし，手作りの面が強いのでアプリケーション全体を通して一貫したGUI設計を保証できない．また，大規模なアプリケーションの開発では生産性の面で問題がある．

### 3.1.2 ルール記述方式

ルール記述方式は古典的なUIMS方式[16]の流れを汲むものであり，事前にGUI部品の配置や属性の設定方法（ルール）を専用言語で記述しておく．アプリケーションの対話内容や流れを専用言語で記述すれば，ルールに従ってGUIの実現コードを自動生成する．ウィンドウの作成だけでなく操作手順も一部作成される．本方式は実用実績が乏しかったが，最近ITS[17]が実システムに利用されたことは注目に値する．

本方式は，LATEX[18]の文書作成方式に類似している．ルールがスタイルであり，対話内容がテキストに相当する．LATEXでは，標準的な文書を作成する場合はキーワードの下に文章を差し込むだけで見栄えのよい文書ができるが，標準（スタイル）から少しでも逸脱しようとすると，特殊なコマンドが必要になったり，スタイルを変更せざるをえない．しかし，コマンドやスタイルは複雑で容易には使いこなせない．ルール記述方式では，デザイナとルールを熟知した人が協力して記述した巧妙なルールがあれば，だれでも高い品質のGUIを効率よく容易に作成できる．また，作成したGUIの一貫性が保証される．しかし，一般にルールはかなりの部分をアプリケーションごとに変更したり修正する必要がある．だが，ルールの書き直しは，だれにでもできるわけではない．実用化されたITSのルールもかなり複雑なので，理解し使いこなすには相当の努力が必要と思われる．

## 3.2 研究動向

GUI構築ツールの研究は，エディタ方式をベースに進められている．これは，プログラマ以外のUI専門家が直接GUI作成に参加しやすいことと，常に出来映えを確認しながら作成したほうがGUI開発者にとっても作業しやすいことなどが理由であろう．以下では，GUIの表示部分（Look）の作成支援，つまりウィンドウ作成支援（**3.2.1**）と，ユーザの操作に対する振舞い（Feel）の作成支援，つまり応答の定義支援（**3.2.2**）に分けて注目される研究をそのキー技術別に紹介する．

### 3.2.1 ウィンドウ作成支援

（1） 制　約

OPUS[19]は部品の位置や形状などを一方向の制約によって記述できるUIビルダである．従来のUIビルダで作成されたGUIでは部品の位置は固定的だったが，OPUSで作成したGUIではウィンドウの大きさが変更されたり，他の部品が動かされたり，内部データが更新されたときには，関連する部品の位置や大きさが自動的に変化する．

OPUSでは，視覚的記法（フレーム，参照ライン，制約など）を用いて制約を視覚的に記述できる．フレームは部品をグルーピングするものであり，入れ子にできるので，画面設計を階層的に行える（フレーム自身は実行時には表示されない）．参照ラインはフレームに付着できる水平か垂直のラインであり，付着の仕方として最小，最大，平均，比率などがある．制約は矢付きの線分で表され，根元は参照相手を指し，先は制約相手を指す．線分をクリックすると制約式が表示されて，式を入力したり変更することができる．OPUSの制約は，Garnetの制約に類似しているが，UIビルダ上で視覚的記法によって設計できる点が注目される．

（2） レイアウトルール

従来のUIビルダでは，部品の端を揃えたり同じ幅にしたり等間隔に並べたりして，キチンと配置するのは案外面倒な作業であった．そこで設計者がだいたいの位置に部品を置くと，レイアウトルールを利用して自動的にキチンと配置するアプローチが研究されている[20),21)]．

Druid[20]では，設計者が新しい部品を作業用ウィンドウに置くと，レイアウトルールを使って位置的に関連する部品を見つけてそれに整列させる．適用するルールと相手部品を“推論（guess）”しているわけである．たとえば，あるラベルの下に新しいラベルを置くと，Druidは新しいラベルをそのラベルに左揃えに整列する．そして，ダイアログボックスを表示して推論の適否を確認する．設計者の答えがNoならば別のルールや相手を探す．

Druidでは，フィードバックがモーダルな点（いちいち設計者に確認する点）が問題である．そこでHudsonら[21]はレイアウトルールの適用結果を，即座に視覚的に設計者に示すスナップフィードバ

ックを実現した．設計者が部品をドラッグして，あるレイアウトルールに該当する範囲にくると，部品をそのルールに従う位置にジャンプさせる．ここで，適用したルールの意味を設計者に知らせるために，参照相手の部品から伸ばした補助線を表示するなどの視覚的な工夫（スナップフィードバック）を施す．設計者は，それが意図したものと異なるときは，ドラッグを続ければよい．

（3） スタイル登録

レイアウトルールでは，固定したルールによって部品配置を支援しているが，GUI作成では自分のスタイルでレイアウトしたい場合が多い．そこで，設計者が例を示す（デモをする）だけで自分のスタイルを登録することができ，その登録したスタイル（ルール）によってウィンドウ作成を支援する方法[22]が研究されている．

スタイルには部品の属性とレイアウト情報を登録できる．スタイルを登録するには，まず部品属性の編集用ウィンドウなどを使って，ある部品の属性を設定し，それを選択して“Define Style”コマンドを発行する．StyleEditingウィンドウ（図-4中央）が表示されるので，スタイルの名前を入力すると，その部品の標準と異なる属性だけがスタイルに記録される．レイアウト情報にはタブに関連して定義するもの（絶対ポジション）と，すでに配置された他の部品に関連して定義するもの（相対ポジション）がある．タブはウィンドウにおける水平または垂直の位置を表す線であり，これに部品を揃えることができる．タブの追加や位置決めはTabStopウィンドウ（図-4下）で行う．相対ポジションでは，タブの代わりに参照する部品を選択する．

図-4では，絶対ポジションのMain-Titleスタイルが，“Font Selection:”ラベルを使って，大型

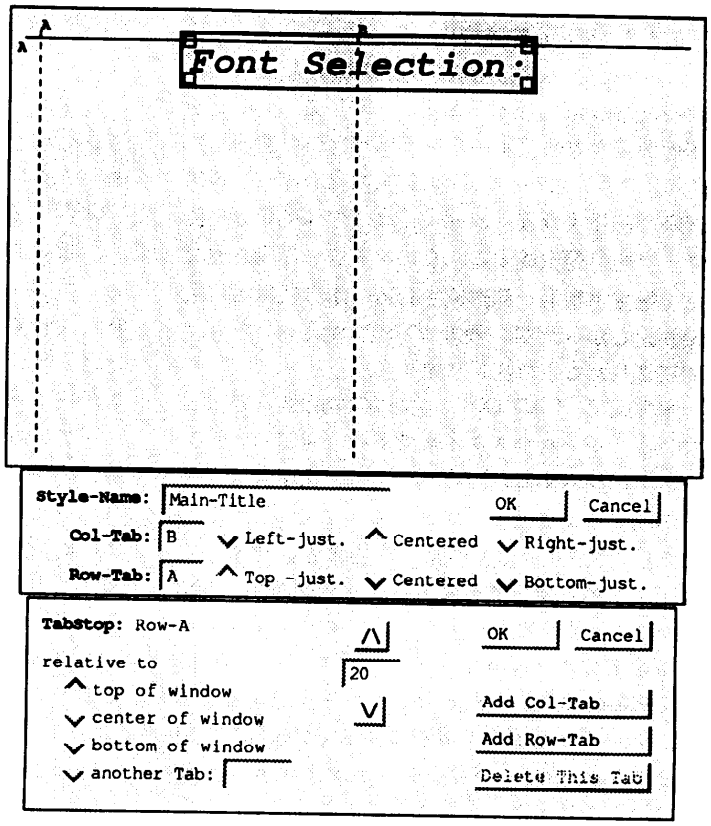



図-4　絶対ポジションのスタイル登録[22]

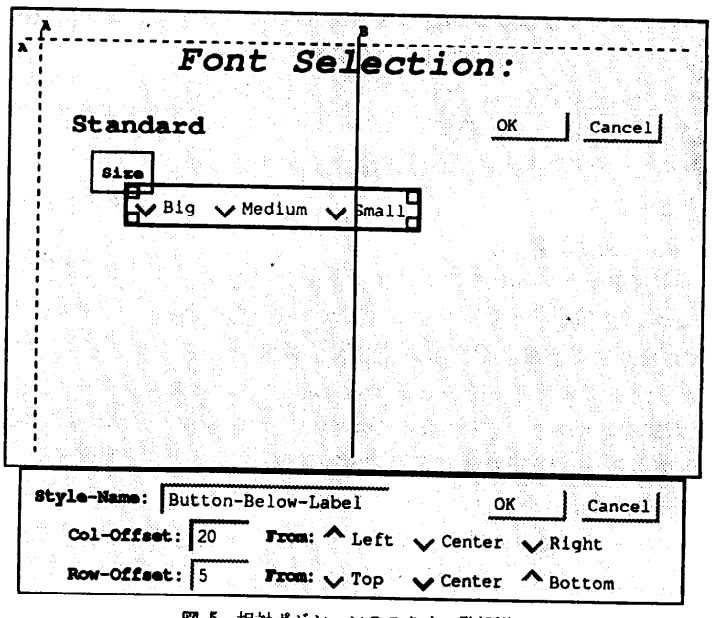



図-5　相対ポジションのスタイル登録[22]

のボールド／イタリック書体という属性情報と，水平タブ A (Row-Tab: A) に上揃え (Top-just.) で，垂直タブ B (Col-Tab : B) に中央揃え (Centered) というレイアウト情報をもつスタイルとして登録される．**図-5** では，相対ポジションの Button-Below-Label スタイルが，"Big-Medium-Small" ラジオボタンを使って，横並びで可変間隔という属性と，ラベル部品（この場合 "Size"）に対してその左から (From : Left) 20 ピクセル間隔 (Col-Offset : 20) で，その下から (From : Bottom) 5 ピクセル間隔 (Row-Offset : 5) というレイアウト情報をもつスタイルとして登録される．

このようにして登録されたスタイルを基にウィンドウ作成支援が行われる．設計者が新しい部品を作業用ウィンドウに置いたりドラッグすると，スタイルの推論が行われる．推論では，まずその部品と同じ種類の部品に関するスタイルを見つける．次にその部品の現在位置が，候補となっている各スタイルのレイアウト情報にどれほど近いかを調べる．相対ポジションのスタイルについては，その部品の周りにある部品で，参照部品と同じ種類のものとの間隔を調べる．そして，一番近いスタイルがその部品にただちに適用される．設計者はそのスタイルが好ましくない場合，"Try Again" ボタンを押せば，次に近いスタイルが適用される．

レイアウトルールやスタイル登録は，エディタ方式にルール記述方式のルールを取り込むことによって，自動化による生産性の向上を図るとともに，GUI の一貫性を保証するものであり今後の研究が注目される．

### 3.2.2 応答定義支援

（1） フィルタ

従来の UI ビルダでは，GUI の応答の定義支援が貧弱である．NextStep やゆずなどは部品間の定型的なデータ変換や部品からのアプリケーション呼出し（コールバック関数）の定義支援機能を提供している[23),24)]が，支援範囲が限定されている．ここでは，フィルタという統一概念で広い範囲の応答定義を支援する研究[25)]を紹介する．

フィルタには出力値フィルタと初期値フィルタがあり，各部品についてそれぞれ定義することができる．出力値フィルタは，その部品の通常の出力値（: value）を他の部品やアプリケーションが利用しやすい形式（フィルタ値，: filtered-value）に変換する．初期値フィルタは，部品の初期表示時の属性値を決めるものであり，他の部品のフィルタ値やアプリケーションの処理結果を参照したり条件判定を行って，属性の値を生成する．

フィルタの定義は，専用のウィンドウを呼び出して Lisp 式で記述する．他の部品を参照するときは，その部品をマウスで選択すれば，その部品のフィルタ値が変数として Lisp 式に挿入される．また，さまざまな型変換関数などがメニュ形式で用意されている．フィルタは Garnet 上に構築されているので，参照は全て制約によって定義される．したがって，参照先の部品やアプリケーションが更新されると，フィルタ値は自動的に再評価される．

これらの処理は，従来アプリケーション関数やコールバック関数においてプログラミングで実現されていたが，大部分がフィルタで実現できるのでプログラミング負担が大きく低減する．これにより，GUI 構築の生産性が向上するとともに，GUI 変更の際に書き直すコード量が大幅に減少し，GUI の洗練やカスタマイズが容易になる．

（2） デモンストレーション

従来の UI ビルダでは，部品に新たな振舞い (応答) の仕方を定義することはできない．ここでは，設計者に部品の動的な振舞いをデモンストレーションしてもらい，そこから応答定義の情報を導き出すという研究を紹介する．

Lapidary[26)] は Garnet 上に構築されているが，部品の振舞いのデモ結果を一般化して，その振舞いを応答にもった部品を生成することができる．具体的には，設計者はまず部品の現在の状態を指定し，次にその部品を編集して振舞いの結果の状態にする（これがデモである）．Lapidary は，部品を構成する各図形（オブジェクト）について，それらの位置や大きさや色などの属性（スロット）が前の状態と後の状態でどう変化したかを比較する．変化した各スロットについて，前の値と後の値の両方をもち，あるスロット（制御スロット）の値によってどちらかを選択する式を生成して埋め込む．ここで，変化スロットは制約によって制御スロットを参照する．制御スロットは自動的に追加される．たとえば，ある部品の色属性を白（前の状態）から赤（後の状態）に変更する，

というデモを行うと，Lapidary は元の部品に対して，

```
(:selected NIL); 制御スロット
(:color (o-formula (if (gvl:selected)
                       :red
                       :white)))
```

のようなスロットを追加した新たな部品を作成する．この部品は，:selected スロットに T をセットすると赤に変わり，NIL をセットすると白に戻る．

## 4. UIMS の将来

UIMS の将来について，UI の観点から GUI 構築ツールを中心にいくつかの方向性を述べる．

### 4.1 デモンストレーション UI

GUI／直線操作法では，繰り返しや抽象的な処理をうまく組み込むことができない[1]．この問題に対して，GUI 上で例をデモすると，それを一般化して繰り返しなどを含む処理手続きを生成するというデモンストレーション UI 方式が提案され[27]，研究が進められている[28]．エディタ方式の GUI 構築ツールは GUI／直接操作法そのものであり，すでにウィンドウ作成支援のスタイル登録(**3.2.1** の(3))や応答定義支援のデモンストレーション(**3.2.2** の(2))で一部実現されている．今後もこの方式の研究成果を GUI 構築ツールに積極的に導入すべきである．

### 4.2 UI のカスタマイズ

エンドユーザによる UI のカスタマイズは UIMS の一つの目的である．たとえば，マッキントッシュではエンドユーザが UI 部品の属性をエディタ（リソースエディタ）で変更できる[29]．しかし，現状では変更対象が限定されており，今後の本格的な研究が期待される．

### 4.3 適応型 UIMS

ユーザの操作履歴から，タスクの内容やユーザの好みに UI を自動的に適応（カスタマイズ）させる適応型 UIMS の研究は魅力的である．現状ではまだ実用的な成果は乏しいが，分散協調環境(マルチエージェントモデル)をベースとする新たな取組み[30]も出始めており，今後が注目される．

### 4.4 部品の高度化

現状では，ボタンやメニュなどの基本的な部品（ウイジェット）を単位とした GUI 構築ツールが中心だが，応用対象を限定することによって部品のレベルを高度化して，GUI 構築を容易にするという研究が行われている．代表的なものは，Inter-Views 上に作成された Unidraw[31] であり，グラフィックスエディタのためのフレームを提供する．部品からフレームへの進化と言えよう．今後，GUI の典型的な応用分野について，このような高度な部品（フレーム）が用意され，GUI 構築が一層効率化され容易になることが望まれる．

### 4.5 UI 評価機能

本稿では GUI 構築の効率化に関する技術についてみてきたが，GUI の評価技術は UIMS の重要なテーマである．作成した GUI にログ記録機能を埋め込んで，被験者に操作してもらい，その記録を分析する方式については，すでにいろいろな研究が行われている[1]．しかし，GUI 部分の設計結果だけを使って，実行せずに UI 評価を行う方式の研究はあまり進んでいない．モデル化や評価ルールなどの基本的な問題から研究する必要があるが，モデル化については，ユーザ（人間）のモデル化というアプローチではなく，UI のモデル化というアプローチ[32]が必要と思われる．

## 5. おわりに

本稿では UIMS の研究動向として，GUI を効率よく構築するための GUI 構築環境と，プログラマ以外の人が GUI 構築を行うための GUI 構築ツールの研究動向を紹介し，将来の方向性について述べた．最近，ペンやジェスチャや 3 次元などの新しい入力方式が注目されている．また，マルチメディアやネットワークなどの新しい UI 実行環境が普及し始めている．このように UIMS の研究対象は，今後 GUI に留まらず大きく広がっていくものと思われる．UIMS 研究のさらなる発展と成果の着実な普及が期待される．

## 参考文献

1) Shneiderman, B.: Designing the User Interface (Second Edition), Addison-Wesley (1992).
[初版の邦訳] 東，井関訳：ユーザインタフェースの設計，日経 BP 社 (1987).
2) たとえば，相沢博泰：Xlib とそのプログラミング，bit 別冊 X ウィンドウとその仲間たち，pp. 144-166 (1992).
3) たとえば，稲田 睦：Xツールキットの概要，bit 別冊 X ウィンドウとその仲間たち，pp. 167-180 (1992).

4) Linton, M. et al.: Composing User Interfaces with InterViews, IEEE Computer, Vol. 22, No. 2, pp. 8-22 (1989).
5) 斎藤 靖他：オブジェクト指向 UI ツールキット InterViews, bit, Vol. 23, No. 11, pp. 69-77 (1991).
6) Myers, B. et al.: Comprehensive Support for Graphical Highly-Interactive Interfaces: The Garnet User Interface Development Environment, IEEE Computer, Vol. 23, No. 11, pp. 71-85 (1990).
[邦訳] Garnet：対話性にすぐれ，柔軟な GUI の作成を支援するツール，日経エレクトロニクス，No. 522, pp. 187-205 (1991).
7) Myers, B. et al.: Environment for Rapidly Creating Interactive Design Tools, The Visual Computer, Vol. 8, No. 2, pp. 94-116 (1992).
8) 橋本 治：GUI 構築ツール Garnet—使い心地と課題—, Human Interface News & Report, Vol. 7, No. 2, pp. 159-166 (1992).
9) Lowgren, L.: History, State and Future of User Interface Management Systems, ACM CHI Bulletin, Vol. 20, No. 1, pp. 32-44 (1988).
10) Giuse, D.: Efficient Knowledge Representation Systems, The Knowledge Engineering Review, Vol. 5, No. 1, pp. 35-50 (1990).
11) Borning, A. H.: Classes Versus Prototypes in Object-Oriented Languages, Compcon '86 Fall Proceedings, pp. 36-40 (1988).
12) Lieberman, H.: Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems, OOPSLA '86 Proceedings, pp. 214-223 (1986).
13) Freeman-Benson, B. N. et al.: An Incremental Constraint Solver, Communications of the ACM, Vol. 33, No. 1, pp. 54-63 (1990).
14) Myers, B.: A New Model for Handling Input, ACM Transactions on Information Systems, Vol. 8, No. 3, pp. 289-320 (1990).
15) Krasner, G. E. et al.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80, Journal of Object-Oriented Programming, Vol. 1, No. 3, pp. 26-49 (1989).
16) たとえば，Foley, J. et al.: Defining Interfaces at a High Level of Abstraction, IEEE Software, Vol. 6, No. 1, pp. 25-32 (1989).
17) Wiecha, C. et al.: ITS: A Tool for Rapidly Developing Interactive Applications, ACM Transactions on Information Systems, Vol. 8, No. 3, pp. 204-236 (1990).
18) たとえば，Lamport, L. (Cooke, E., 倉沢良一他訳)：文書処理システム LATEX, アスキー (1990).
19) Hudson, S. E. et al.: Interactive Specification of Flexible User Interface Displays, ACM Transactions on Information Systems, Vol. 8, No. 3, pp. 269-288 (1990).
20) Singh, G. et al.: Druid: A System for Demonstrational Rapid User Interface Development, UIST '91 Proceedings, pp. 167-177 (1991).
21) Hudson, S. E. et al.: Smoothly Integrating Rule-Based Techniques into a Direct Manipulation Interface Builder, UIST '91 Proceedings, pp. 145-158 (1991).
22) Hashimoto, O. and Myers, B.: Graphical Styles for Building User Interfaces by Demonstration, UIST '92 Proceedings (to appear) (1992).
23) Webster, B.: The NeXT Book, Addison-Wesley (1989).
24) 杉山高弘他：鼎ユーザインタフェースビルダ「ゆず」の構築，情報処理学会第 45 回全国大会予稿集，pp. 5：99-5：100 (1992).
25) Myers, B.: Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs, UIST '91 Proceedings, pp. 211-220 (1991).
26) Myers, B. et al.: Creating Graphical Objects by Demonstration, UIST '89 Proceedings, pp. 95-104 (1989).
27) Myers, B.: Invisible Programming, 1990 IEEE Workshop on Visual Languages Proceedings, pp. 203-208 (1990).
28) たとえば，Cypher, A.: Eager: Programming Repetitive Tasks by Example, CHI '91 Proceedings, pp. 33-39 (1991).
29) たとえば，西林瑞夫：Macintosh [増補版]，共立出版 (1988).
30) 佐藤 豊他：適応型利用者環境支援システム VIA，情報処理学会第 43 回全国大会予稿集，pp. 5：451-5：452 (1992).
31) Vlissides, J. et al.: Unidraw: A Framework for Building Domain-Specific Graphical Editors, ACM Transactions on Information Systems, Vol. 8, No. 3, pp. 237-268 (1990).
32) 佐藤啓一：ユーザインタフェイス設計におけるシステム記述形式—ユーザインタフェイス設計方法論 1—，デザイン学研究，No. 76, pp. 33-40 (1989).

(平成 4 年 7 月 16 日受付)

**橋本 治** (正会員)

1958 年東京生. 1983 年早稲田大学大学院修士課程(電気工学)修了. 同年 NEC 入社. 以来，C&C システム研究所にて，ユーザインタフェースの研究に従事. 1990 年～1991 年米国カーネギーメロン大学に客員研究員として留学. 1992 年度東京工業大学非常勤講師. 現在，C&C システム研究所ターミナルシステム研究部主任.