ヒューマンインタフェース 37－3
（1991．7．4）

ユーザインタフェース開発ツールの構成と実現

伊知地 宏*　　倉部 淳**

富士ゼロックス株式会社
*システム技術開発センター
**ワークステーション事業部　ソフトウェア開発部

ユーザインタフェース開発ツールの構成と、それを実現し評価した結果について述べる。ユーザインタフェース開発ツールの多くは、ユーザインタフェース部品に依存して実現されているので、各種情報機器のユーザインタフェース開発に同じツールを使用することは難しい。汎用的なユーザインタフェース開発ツールを実現するために、ユーザインタフェースを設計するグラフィカルエディタ、設計情報からプログラムを生成するコードジェネレータと、設計情報や部品などのデータでツールを構成し、データをS式で表現することで部品と開発過程を分離するメカニズムを獲得した。しかし、アプリケーション本体との結合には問題を残している。

The Structure of User Interface Development Tool and Its Implementation

Hiroshi Ichiji† *and Jun Kurabe*‡

Fuji Xerox Co., Ltd.
† System Technology Development Center
‡ Software Development Dept., WorkStation PBU

KSP R&D Business Park Building ,100-1, Sakado,
Takatsu-ku, Kawasaki-shi, Kanagawa, 213, Japan

e-mail: hiro@ksp.fujixerox.co.jp
jun@ssd8b.ksp.fujixerox.co.jp

This paper describes the structure of user interface development tool and its implementation. It is too hard to develop any user interfaces on any information medias with one user interface development tool. For realizing the common user interface development tool, one consists of graphical editor for designing some user interfaces, code generator for translation from user interface descriptions to actual programs, and data which are user interface descriptions and elements. To represent data as S-expressions achives the mechanism of separating user interface elements from processes of user interface developments. But there is the problem of binding the application body to the user interface.

1 はじめに

情報機器が高機能になると、誰にでも理解がしやすく操作性の良いユーザインタフェースが求められる。グラフィカルユーザインタフェースは、この点で優れているため、コンピュータはもとより、複写機、ファクシミリでも利用されるようになった。

これまで、ユーザインタフェースの設計には、絵で画面構成を示し、状態遷移図や言語で動作を記述する方法がとられていた[7]。この方法では、設計の誤りが実現や検証の段階で見つかることが多く、再度設計からやり直すために、開発の遅れを招く問題があった。

この問題を解決するために、プロトタイプを作成して設計の正しさを検証することが考えられた。プロトタイピングにはSmallTalk-80[3]が便利であるが、プログラミング作業がともなう。またHyperCardを用いても、画面の構成はグラフィカルエディタで容易に作成できるが、動作は言語HyperTalkでプログラミングをしなければならない[11]。このため、ユーザインタフェースの設計者にはプログラミングの知識が要求された。

しかし近年、グラフィカルエディタ上でユーザインタフェースを設計し、その正しさを検証し、さらにプログラムを生成するユーザインタフェース開発ツールが開発されている[4]。シーハイムモデル[7]のプレゼンテーション部が設計できるツールとしてdevGuide[13]がある。さらにダイアローグ部まで設計できるツールもある[6]。Interface Builder[8]やPrototyper[12]ではグラフィカルエディタで動作を設計でき、Interface Design Tool[2]では動作を記述するための言語が用意されている。また別のモデルを採用したツールとして、MVCを採用した鼎[10]、並列オブジェクト指向モデルを採用したGUIDE[9]がある。

ユーザインタフェース開発ツールは、プログラミングに関する知識が少なくても、それぞれの環境におけるユーザインタフェースの開発が容易に行えるという点で、大きな効果を上げている。しかし、各種情報機器のユーザインタフェースを扱うための枠組みを与えているとは言えない。我々は、汎用的なユーザインタフェース開発ツールの作成を試み、実験を通じて、その可能性について考察を行った。

本論文では、2章でユーザインタフェース開発ツールのモデルと構成、実現について説明する。3章でユーザインタフェースの開発実験について記述し、4章で考察を行なう。5章で結論、6章で今後の課題を述べる。

2 ユーザインタフェース開発ツール

2.1 制約オブジェクト指向モデル

ここでは、ユーザインタフェースの要素をオブジェクトとしてとらえる。オブジェクトxに対してある関係に基づいて近傍を定義し、近傍を開集合としてオブジェクト空間に位相を入れる。オブジェクトxとオブジェクトxのある近傍に属するオブジェクトの間に成り立つ関係を制約条件と呼び、オブジェクトxの近傍と制約条件の対をオブジェクトxの制約という。制約オブジェクト指向モデルは、オブジェクト指向に制約を取り入れたもので、メッセージと制約によりオブジェクトの動作が決定されるモデルである。

制約オブジェクト指向モデルでは、制約は二種類に分類できる。メソッドの実行で制約条件が成り立たなくなるときにメソッドの実行を禁止する強い制約と、メソッドの実行を禁止しない弱い制約である。弱い制約では、制約条件が満たされなくなった近傍でオブジェクトが自らの状態を変化させ、制約条件を成立させる。

ユーザインタフェースにおけるBox-AとBox-Bの位置関係は制約である。Box-AがBox-Bより右側になければならないとすると、Box-Bは Box-Aのある近傍 $U_{Box\text{-}A}$ に属し、Box-A, Box-BのX座標の値をBox-A.x, Box-B.xで表せばBox-A.x > Box-B.xと表現できる。つまり、Box-A.x > Box-B.xが成り立つ全てのBox-Bを近傍の要素と考えることができ、条件とそれを満たすオブジェクトを示すことで、制約を表現できる。

2.2 ユーザインタフェース部品の記述

ユーザインタフェースの要素は、位置、大きさなどの属性と、要素が持つ手続きとで構成されており、S式で表現される。図2.1にAthena Widgetの CommandWidgetを記述した例を示す。6行目は属性名x(X座標)の値が100であることを表している。カテゴリーはカテゴリー名と属性またはカテゴリーのリストで記述される。3行目から10行目はカテゴリーCommon-Resourceに属する属性を表している。16行目は手続き名ButtonDownとその手続きを表している。

```
1:  (Object
2:    (data
3:      (CommonResource
4:        (class: commandWidget)
5:        (name: quit)
6:        (x: 100)
7:        (y: 200)
8:        (width: 10)
9:        (height: 20)
10:     )
11:     (OtherResource
12:       (label: Quit)
13:       (callback: quit)
14:     ))
15:   (method
16:     (ButtonDown: select)
17:   )
18:   (Struct
19:     (ON formWidget)
20:   ))
```

図2.1 ユーザインタフェースの記述例

ユーザインタフェースの構造もS式で表現され、属性ONで上にあるWidgetを示している。19行目はCommandWidgetがformWidgetという名前のWidgetの上にあることを示している。

このようにS式を用いて記述すると、ツールキットごとの異なる属性の構造、種類、内容に、容易に対応できる。

2.3 ツールの構成

制約オブジェクト指向モデル、ならびに部品の記述に基づいて、ユーザインタフェース開発ツールPrefaceを実現した。Prefaceは、図2.2に示すようにグラフィカルエディタとコードジェネレータ、ユーザインタフェース要素の定義ファイル、テンプレートで構成される。

2.3.1 グラフィカルエディタ

グラフィカルエディタは、ユーザインタフェース部品の定義ファイルを読み込み、定義ファイルに記述されている要素を用いたユーザインタフェースの設計エディタとなる。

エディタの画面は、図2.3に示すように編集エリアとブラウザ、コマンドエリアで構成される。編集コマンドには、ユーザインタフェースの要素を編集するコマンド、ファイルを操作するコマンド、エディタを操作するコマンドがあり、各コマンドはメニューで選べるようになっている。ユーザインタフェースの設計では、編集エリアに要素を直接操作で配置していく。要素はブラウザで管理され、カテゴリー単位でブ

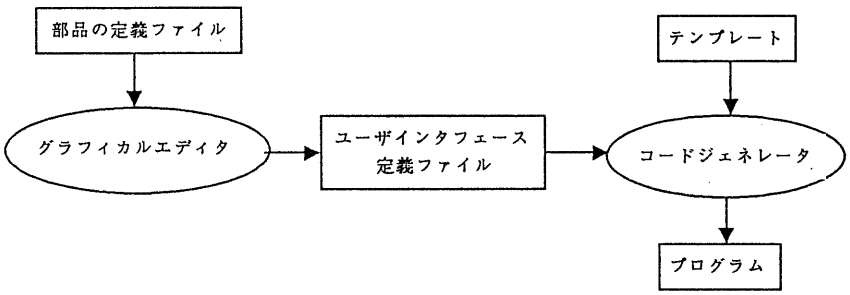

図2.2 ツールの構成

ラウザのビュウに表示される。設計結果はユーザインタフェース定義ファイルに出力される。

2.3.2 コードジェネレータ

コードジェネレータは、構文解析部とコード生成部、ファイル出力部で構成され、ユーザインタフェース定義ファイルとテンプレートファイルからプログラムを生成する。テンプレートファイルには、プログラムの出力書式が記述されている。

構文解析部は、ユーザインタフェースの定義ファイルの構文解析を行い、要素のクラス、要素の名前、要素の階層構造、要素の属性に分解し、構文解析テーブルに出力する。コード生成部は、構文解析部が生成したテーブルからプログラムを生成する。プログラムの生成方法は、要素のデータを表す変数、要素を生成するプログラム、要素の属性を設定するプログラム、要素がイベントを受け取ったときに呼び出す関数のスケルトンを別々に生成し、最後に合成する方法である。この方法により各種のツールキットに容易に対応できる。ファイル出力部ではプログラムを機能単位ごとにファイルに分け出力する。プログラムの変数名は、要素の名前と要素の識別子を連結し一意に決定される。

3 ユーザインタフェースの開発実験

3.1 ユーザインタフェースの開発手順

Prefaceを用いたユーザインタフェース開発では、まず部品の配置を行ない、次に部品がイベントを受け取ったときの動作を定義し、最後にコードの生成を行なう。

部品の配置はエディタで行い、その手順は以下のようなものである。まず、ブラウザから部品を取り出して編集エリアに置き、好みの大きさに変更する。次に、部品の色や枠の太さ、ラベルなどの値をプロパティシートを用いて変更する。編集エリアにある部品をコピーして使うこともできる。

ラベルにDoItと書かれたCommandWidgetを置くには、ブラウザでCommandWidgetがあるカテゴリを選び、表示されたCommandWidgetを取り出して置く位置を指定し、好みの大きさに変更する。プロパティシートを開き、ラベルの属性値をDoItに変更すると、WidgetのラベルがDoItになる。このような作業の繰り返しで、ユーザインタフェースを定義できる。

部品の配置の次に、部品の動作の定義を行う。プロパティシートを開き、コールバック関数の属性値にイベントを受け取ったときに呼び

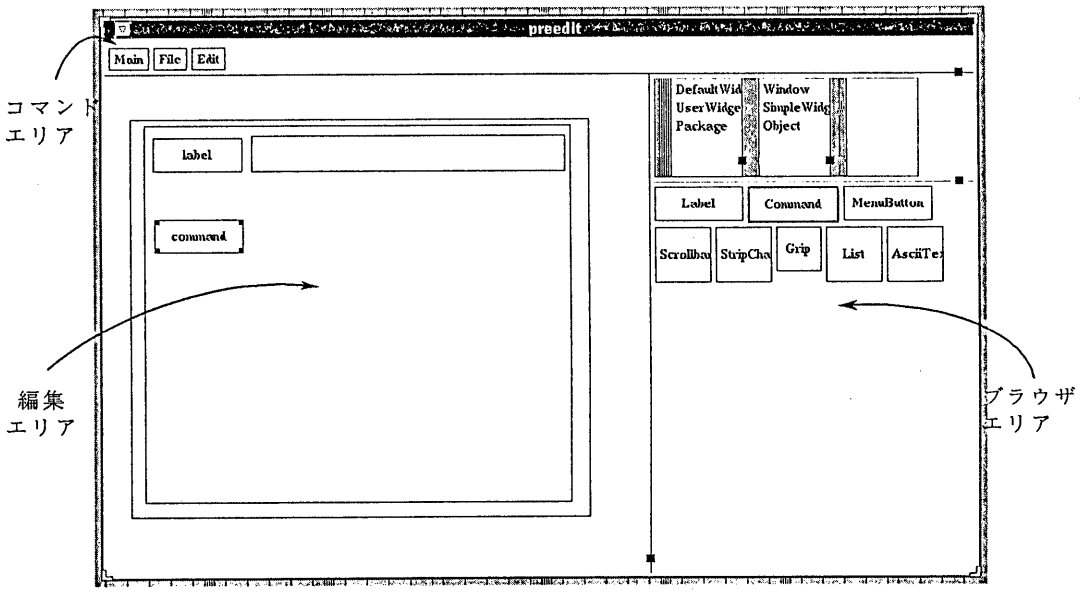

図2.3 グラフィカルエディタの画面

出す関数名を登録する。たとえば、コマンドボタンDoItが押されたときに、doItという関数を呼び出すのなら、コールバック関数の属性値をdoItにすればよい。

コードジェネレータが、定義したユーザインタフェースを自動的にソースプログラムに変換する。

3.2 実験

3.2.1 ユーザインタフェースの作成

Prefaceを利用することにより、どれだけユーザインタフェースの作成が容易になったかを知るために、Athena Widgetを用いた電卓ツールを作成した。電卓ツールのユーザインタフェースは、図3.1に示すように0から9までの数字キーと四則演算キー、等号キー、表示用のエリアで構成されている。数字キーを押すと表示用のエリアに値が表示される。四則演算キー、等号キーが押されると計算結果を表示用のエリアに表示する。

電卓ツールの各キーにはCommandWidgetを用い、結果表示用エリアにはTextWidgetを用いた。Prefaceでは、自動的に各Widgetを上下左右等間隔に並べられないので、各WidgetのX、Y座標の属性を編集して、等間隔になるようにしなければならなかった。以上の作業に3分35秒かかった。

動作の定義では、各CommandWidgetが押されたときに呼び出す関数名を、各Widgetのコールバック関数の属性値に設定した。動作の定義に13分56秒必要であった。

プログラムの生成にかかった時間は11秒であった。

3.2.2 既存アプリケーションとの結合

次に既存のツールのユーザインタフェースを作成し、アプリケーション部との結合実験を行なった。プリントツールは、ファイルとネットワーク上にあるプリンタを指定すると、指定されたプリンタにファイルを出力するツールで、そのユーザインタフェースは図3.2のようなものである。

プリントツールのユーザインタフェースは、PanedWidget, TextWidget, CommandWidgetなどを用いて作成した。部品の配置を行なうのに8分30秒費やした。

動作の定義では、アプリケーションのプログラムを接続するために、アプリケーションを起動する関数を、各CommandWidgetのコールバック関数に設定した。この作業に10分23秒要した。

プログラムの生成に7秒かかった。

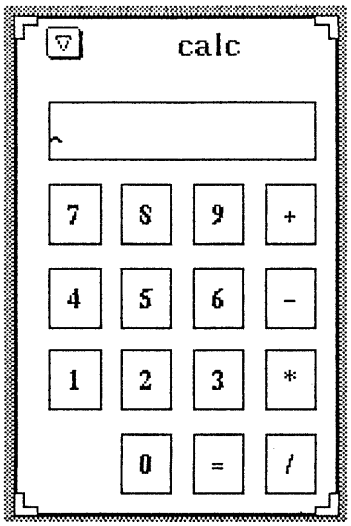

図3.1 電卓ツールのユーザインタフェース

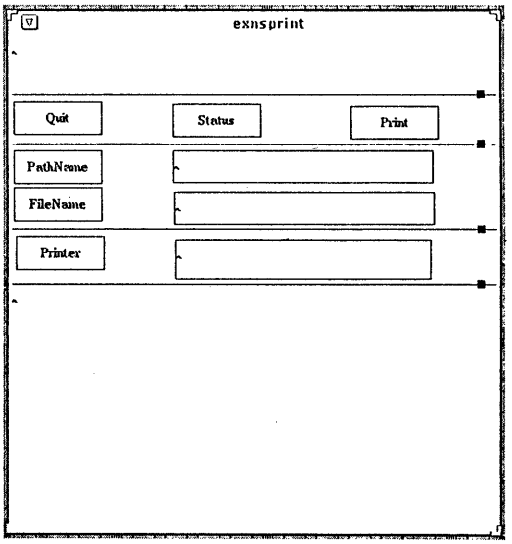

図3.2 プリントツールのユーザインタフェース

コード生成後、ユーザインタフェースとアプリケーションの接続のために、アプリケーションプログラムからユーザインタフェースに値を渡している部分の修正を行なう必要がある。これはアプリケーションが使うAthena Widgetの変数名とPrefaceが生成した変数名が一致しないためで、変更しないとアプリケーションの実行結果がユーザインタフェースに反映されないことになる。変数名の不一致を一つ一つ確認し修正していくため、かなり大変な作業となる。この作業に約4時間費やした。

4 考察

4.1 部品と開発過程の分離

上記の実験からわかるように、Prefaceグラフィカルエディタは Athena Widgetの定義ファイルを読み込んで、Athena Widget用グラフィカルエディタとして機能した。またPrefaceコードジェネレータは、ユーザインタフェース定義ファイルとAthena Widget用のテンプレートファイルから、Athena Widgetを用いたC言語によるユーザインタフェースのプログラムを生成した。つまり、ユーザインタフェース開発ツールPrefaceはAthena Widgetの情報を受けて、Athena Widgetのユーザインタフェース開発ツールに変化したわけで、(Athena Widgetが提供する)部品と(Prefaceが提供する)開発過程の分離に成功したと言える。

Prefaceでは、ユーザインタフェースの部品にも開発過程にも特別なメカニズムを導入したわけではないので、データを表現し、ツールの性格を決定づけたS式の果たす役割は非常に大きいと考えられる。S式の表現力の豊かさにより、各種の部品の属性を余すとこなく記述できたし、S式の柔軟な構造でユーザインタフェースの構造を忠実に記述することができた。またLISPと同様に、PrefaceのS式ではデータとプログラムの違いがないため、エディタの性格を決定づけるプログラムとしても有効に働いた。制約の実現はその一例とも言える。

しかし、細部においては各種の問題が発生した。特に制約の実現において顕著に現れた。部品の上下関係や部品Aの右隣りに部品Bを置いてはいけないというような部品が元々持っている配置の規則に関しては、部品を定義するS式でリスト構造や不等号を用いて表すことができ、ユーザインタフェース設計の際にも有効に働いた。しかし、ひとたびエディタを使って配置した特定の部品に制約を付けようとすると問題が生じた。エディタで部品の状態を変化させるには、プロパティシートを用いて属性値や呼び出す関数を設定する方法しか提供されていない。つまり、エディタで制約を定義する機能が提供されていないので、例えば3.2.1で述べたように部品を上下左右等間隔に並べるといったことを指定できないのである。

特定の部品に制約を定義する方法としては、プログラムによる方法とグラフィカルな指定方法が考えられる。プログラムによる方法であれば、プロパティシートを使って設定を行うことも可能である。しかし、直接S式を編集させることにはいろいろな問題があるので、制約の記述にむいたプログラミング言語を提供することがよいと思う。グラフィカルな指定方法としては、Borning[1]の様に部品の関係づけを直接操作で行えるようにするのがよいが、これだけで全ての制約を表現できるかどうかは明らかでない。またS式で全ての制約を表現できるかも、まだ明らかになっていない。

4.2 ユーザインタフェースの開発

ユーザインタフェースのともなうソフトウェア開発をPrefaceを使った場合と、従来の方法によるものとで比較する。

表4.1に、電卓ツールのユーザインタフェースの開発にかかった時間を、Prefaceを使った場合と従来の方法による場合について、それぞれ作業項目ごとに比較した結果を示す。なお、従来の方法で開発を行った被験者は、初心者ではないが、C言語によるX Windowのプログラミングに精通はしているとも言えないレベルである。まず合計時間からみると、従来の方法では2時間40分強かかるものが、Prefaceを用いると18分弱で開発でき、開発時間はおよそ9分の1になる。配置の設計では、Prefaceでは視

覚的にそして直接操作が行えるため、紙の上で設計するよりも3分の2程度の時間で作業が終了している。動作の設計が終わると、Prefaceではプログラムの抽出が行えるので、従来のような詳細設計やプログラミング作業がいらないことも明らかになっている。Prefaceを用いてのユーザインタフェース開発の効果の程がよくわかる。

また、従来の方法でユーザインタフェースを開発する場合、ユーザインタフェースの開発者には、プログラミングの知識とユーザインタフェースの部品の意味、部品の配置の規則、そしてわかりやすく使いやすいユーザインタフェースを構築するための知識が必要であった。これに対し、Prefaceを用いるとグラフィカルエディタの支援により、はるかに少ない知識でユーザインタフェースを作成することができることが上記の実験、ならびにユーザインタフェースのプログラミングをしたことがない人にPrefaceを使ってもらった結果から明らかになった。

しかし、現在のPrefaceでは、ユーザインタフェースの動的な面の設計が正しいことの検証は行えない。すなわち動作に関しての設計に間違いが生ずることがある。動作に関する検証を行うために、ユーザインタフェースの動作シミュレータが必要だと考えられる。

作業	従来の方法	Prefaceを使用
配置	5分20秒	3分35秒
動作の設計	14分00秒	13分56秒
モジュール設計	15分10秒	--
詳細設計	31分30秒	--
コーディング	66分29秒	0分11秒
テストとプログラムの修正	28分50秒	--
合計	161分19秒	17分42秒

表4.1 電卓ツールのユーザインタフェース開発にかかった時間

4.3 アプリケーションとの結合

新しいアプリケーションプログラムを作成するときに、Prefaceがたいへん効果的であることはこれまでに見てきた。しかし、実験からも明らかなように、既存アプリケーションのユーザインタフェースをPrefaceを使って作り直す場合に、大きな問題が生じている。ユーザインタフェースの設計が簡単にでき、ユーザインタフェースのプログラムを自動生成できても、アプリケーションとの結合でユーザインタフェースのプログラミング相当の時間を費やしては、ユーザインタフェース開発ツールの価値があるとはとても言えない。アプリケーションとの結合に関しては、全面的に方法を考え直す必要がある。

アプリケーションとのインタフェースの問題に関して、ユーザインタフェース側からのアプローチとアプリケーション側からのアプローチ、そしてその両面からのアプローチが考えられる。

ユーザインタフェース側からのアプローチとしては、コールバック関数の本体までユーザインタフェース開発ツールで生成して対処することが考えられる。しかし、これではプログラミングの知識が少なくても使えるという利点を失うことになる。

アプリケーション側からのアプローチとしては、ユーザインタフェース開発ツールが生成するプログラムを想定して、アプリケーションプログラムを書き直すということが考えられる。しかし、ユーザインタフェース開発ツールが生成するプログラムを、アプリケーション開発者が予想するということは、ユーザインタフェースの設計の幅を狭め問題である。

ユーザインタフェース、アプリケーション両面からのアプローチとしては、GUIDE[5]のようにスタブを作成して、サーバー・クライエントモデルに基づいたプログラムを生成し、リモートプロシジャーコールで結合を行うことが考えられる。GMWでは、この方法が成功して

いるようであるが、他のシステムでの実現については今後検討してみる価値がある。

ユーザインタフェース開発ツールを利用するようになると、アプリケーションプログラムの作り方にも変化が現れるようになる。ソフトウェア開発の方法自体が、近い将来大きく変わる可能性を示唆しているとも言える。

5 結論

Prefaceの S式を中間媒体にしたメカニズムは、ユーザインタフェースの部品と開発過程の分離に効果があることが明らかになった。汎用的なユーザインタフェース開発ツールの基盤として使える可能性が高い。また、ユーザインタフェースの開発に、時間面、品質面で大きな効果があることもわかった。

一方、アプリケーションとの結合において、多くの問題が残されており、この解決なしにはユーザインタフェース開発ツールの価値が半減することも明らかになった。また制約の定義方法の提供が重要であることもはっきりした。

6. 今後の課題

Athena Widget 以外のユーザインタフェース部品に Preface を適用し、Preface のメカニズムが汎用的なユーザインタフェース開発ツールの基盤として有効であることを確認するのが、第一の課題である。

ユーザインタフェースとアプリケーションを容易に結合するための方法を、理論的、実験的考察を行って見つけ出すことが第二の課題である。

制約の表現と操作性の良い定義方法を開発することが第三の課題である。

謝辞

Prefaceのアイデアの提供と実装をしてくれました岩田正武氏、さらに有益なコメントをいただいた渡辺美樹氏、高橋政樹氏、北見俊一氏に感謝いたします。

参考文献

[1] Borning, A: Defining Constraints Graphically, *CHI'86 Proceedings*, pp.137-143 (1986).
[2] 福岡久雄, 宮崎一哉, 辻順一郎, 坂下善彦: ユーザインタフェース設計ツール, 情報処理学会・文書処理とヒューマンインタフェース, Vol.17, No.3, pp.1-8 (1988).
[3] Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley (1983).
[4] 萩谷昌己他: ウインター・チュートリアル「ヒューマンインタフェースの最先端」, 日本ソフトウェア科学会 (1990).
[5] Hagiya, M. and Ohtani, K: Parallel object-oriented UIMS with macro and micro stubs, *Proceedings of the Winter 1990 USENIX Conference*, pp.259-273 (1990).
[6] 萩谷昌己: 視覚的プログラミングと自動プログラミング, コンピュータソフトウェア, Vol.8, No.2, pp.27-39 (1991).
[7] Hartson, R. H. and Hix, R.: Human-Computer Interface Development: Concept and Systems for Its Management, *ACM Comput. Survey*, Vol.21, No.1, pp.6-92 (1989).
[8] Next User's Guide V1.0, Next (1989).
[9] 大谷浩司, 角野宏司, 児島彰, 萩谷昌己, 服部隆志, 劉樹苓: GMW ウインドウ・システム上のアプリケーション構築について, コンピュータソフトウェア, Vol.7, No.1, PP.45-60 (1989).
[10] 暦本純一, 垂水浩幸, 菅井勝, 山崎剛, 猪狩錦光, 森岳志, 杉山高弘, 内山厚子, 秋口忠三: 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, 情報処理学会誌, Vol.31, No.5, pp.602-611 (1990).
[11] 佐原伸: HyperTalk, コンピュータソフトウェア, Vol.7, No.1, pp.155-162 (1990).
[12] *Prototyper manual*, Smethers Banes (1989).
[13] Developer's Guide ユーザマニュアル, Sun Micro Systems (1990).