Vol. 35 No. 9 情報処理学会論文誌 Sep. 1994

アプリケーション実行時 GUI レイアウト変更機能

増田英孝† 笠原 宏†

筆者らは，GUI の作成と実行の境界を無くし，ユーザが GUI に対して徹底的に手が加えられるような環境がなければならないと考えている．そこで，筆者らは，GUI の見かけの定義，対話の定義をアプリケーション実行時にユーザごとに対話的・視覚的にカスタマイズすることが可能な GUI 変更・修正環境 Metamer を提案する．Metamer は個人の計算機環境をアプリケーションにとらわれずに動的に変形 (metamorphose) させることを目的とする動的 GUI モディファイアである．Metamer は，GUI の構成要素の変更，レイアウト変更，対話の変更を可能とすることを目的とする．本研究では，まず GUI の設計要素の見かけの定義のうち，効果的なビジュアル・コミュニケーションを行うために重要な，GUI 構成要素のレイアウトに注目し，GUI レイアウト変更機能の実装を行ったのでそれを報告する．レイアウト変更機能として，GUI 部品のレイアウトやビューのツリー階層を動的に変更できる DEVO を提案し，Metamer では，この DEVO を GUI レイアウト変更機能のベースとして利用している．本論文では Smalltalk 上に実装中の Metamer の概要，アプリケーション実行時の GUI 部品レイアウト変更機能およびその評価について述べる．

Dynamic GUI Layout Modification during Application Runtime

HIDETAKA MASUDA† and HIROSHI KASAHARA†

There exists a barrier between GUI *development* and GUI *running* which impedes GUI modification during runtime. We consider that the elimination of this barrier to allow end-users to thoroughly modify GUI is of crucial importance. We propose *Metamer*, a platform that allows users to interactively and visually modify GUI's appearance definitions and dialog definitions during application runtime. Metamer is a GUI modifier that allows dynamical GUI transformation (metamorphosis) independently of the applications running on particular user environments. Metamer objective is to enable GUI component modification, layout modification, and dialog modification. In the current stage of our research, we focused our attention on the implementation of the *GUI layout modification function*, which must support effective visual communication functionality. This function is part of the GUI design component called appearance definition. We have formerly proposed DEVO which allows dynamic modification of GUI objects layout and Tree structures. The *layout modification function* of Metamer is based on DEVO. In this paper, we summarize the outline of Matamer which is being implemented currently on top of Smalltalk. We also explain the application runtime GUI object layout modification function and its evaluation.

1. はじめに

ユーザインタフェースが備えるべき性質の一つとして，ユーザごとの特性に対する柔軟性がある[1]．ユーザにとって，積極的にユーザインタフェースのメインテナンスができ，自分だけのユーザインタフェースを自分自身で育てることができるということは，人に優しいインタフェースとして重要である[2]．特に，エンドユーザに対しては視覚的な GUI のカスタマイズ環境が用意されていることが望ましい[3]．そして，このような要求は古くからあるが，GUI の実装の複雑さから，設計レベルの段階で切り捨てられてしまっているのが現状である．

優れた構成の使いやすいユーザインタフェースを作成するために，ユーザインタフェースガイドラインがある[4]．ガイドラインでは，現存するさまざまな動作モデルから共通的な要素を抽出し[4]，異なったアプリケーションやプラットホーム間のユーザインタフェースの一貫性を保つことに重きを置いている．このガイドラインに沿って作成すれば，大多数のユーザに共通して利用できるユーザインタフェースを作成できるが，ユーザが自己の特性にあったユーザインタフェースとして使用する[1]ためには，自由にユーザインタフェースをカスタマイズできることが必要である．

† 東京電機大学工学部
Faculty of Engineering, Tokyo Denki University

筆者らは，GUI の作成と実行の境界を無くし，ユーザが GUI に対して徹底的に手が加えられるような環境がなければならないと考えている[5]．ユーザが望むであろういかなる変更をもすべて動的に実行でき，積極的にユーザインタフェースのメンテナンスを行う気を起こさせるようにすべきである[2]．

X Window System では，リソースファイルを記述することで GUI のカスタマイズが行える[6),7]．しかし，アプリケーションを起動する前にテキストの形で記述する必要があり，望むとおりの変更を実現するためには試行錯誤が必要である．X Window System では，リソースはアプリケーション起動時に読み込まれ，ウィジェット生成時にしか設定できないリソース等もある[7]ため，アプリケーション実行時に GUI を動的に変更することは難しい．Objectworks\Smalltalk[8),9] (以後 Smalltalk と表す) では，アプリケーション実行時に動的にオブジェクトを生成しているため，動的に GUI を変更できる可能性がある[10]が，実際には動的な GUI 変更は実現されていない．また，X Window System のリソースのような機構は用意されていない．

そこで筆者らは，アプリケーションに手を加えることなく，GUI の見かけの定義，対話の定義をアプリケーション実行時にユーザごとに対話的・視覚的にカスタマイズすることが可能な GUI 変更・修正環境 Metamer を提案する[5]．Metamer は個人の計算機環境をアプリケーションにとらわれずに動的に変形 (metamorphose) させることを目的とする動的 GUI モディファイアである．Metamer は，GUI の構成要素の変更，レイアウト変更，対話の変更を可能とすることを目的とする．

本研究では，まず GUI の設計要素の見かけの定義のうち，効果的なビジュアル・コミュニケーションを行うために重要な[11]，GUI 構成要素のレイアウトに注目し，GUI レイアウト変更機能の実装を行ったのでそれを報告する．レイアウト変更機能として，GUI 部品のレイアウトやビューのツリー階層を動的に変更できる DEVO[12)~14]を提案し，Metamer では，この DEVO を GUI レイアウト変更機能のベースとして利用している．

本論文では，まず Metamer の目的について述べ，GUI のレイアウト変更に関する要求調査とその結果について述べ，GUI プレゼンテーションフレームワークである DEVO および，アプリケーション実行時の GUI 部品レイアウト変更機能とその評価について述べ，最後にまとめを行う．

2. Metamer の目的

Metamer は，すべてのアプリケーションに渡って，マウスベースで手を加えられ，アプリケーションにとらわれずに自分の計算機環境を変形 (metamorphose) 可能な動的 GUI モディファイアを目指している[5]．

2.1 Metamer の構想

従来の GUI では，GUI の構成要素はウィンドウの固定した構成要素として見なされていた[15]．GUI 構築ツールの登場により，マウスベースの視覚的，対話的なツールが利用可能となり，GUI 作成時には構成要素の生成，削除，レイアウト変更等の動的な取り扱いが可能になっている．しかし，作成された GUI は，アプリケーションが完成してしまうと，一部の属性を除き変更することができない．**図 1** にアプリケーション作成と実行のフェーズを Smalltalk の例を挙げて示す．

図 1 (a) に示すように，アプリケーションはプログラマによって作成される．その後，図 1 (b) に示すように，ユーザがアプリケーションを利用することになる．アプリケーションに何らかの変更を加える場合には，図 1 (a) のプログラミングと図 1 (b) の実行のフェーズを行き来する必要がある．

そこで，Metamer ではアプリケーション実行時にもユーザがマウスベースで視覚的，対話的に GUI の各構成要素を動的に変更可能とすることを目的としている[5]．Metamer は，GUI の作成と実行の境界を無くし，ユーザが GUI に対して徹底的に手が加えられるような環境を目指している．ユーザインタフェース

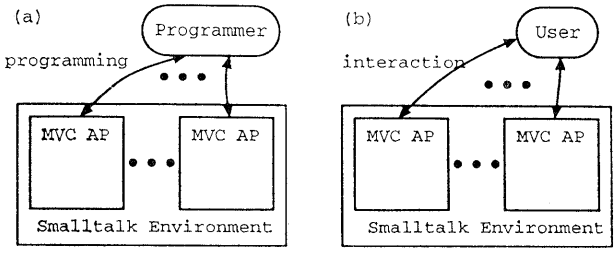


図 1 アプリケーションの作成と実行フェーズ (a) アプリケーション作成時 (b) アプリケーション実行時
Fig. 1 The difference between: (a) application development phase, and (b) running phase.

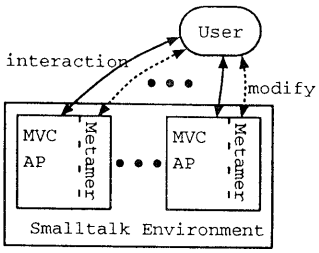


図 2 Metamer の構想
Fig. 2 The concept of Metamer.

のカスタマイズは，ユーザが望むであろういかなる変更をもすべて動的に実行でき，積極的にメンテナンスを行う気を起こさせるようにするべきである[2]．

通常，ユーザが GUI の変更やカスタマイズを行う場合には，プログラムやリソースの記述を変更しなくてはならない．しかし，そのためにはプログラムと実行のフェーズを渡り歩かなければならない．特に，プログラミングの知識がないユーザには，プログラムのフェーズに移行することができないという問題がある．そこで，Metamer では，ユーザがマウスベースで視覚的に GUI の変更を可能とすることを目的とする．**図 2** に Metamer の構成を示す．

図 2 に示すように，ユーザはプログラマのフェーズに移行することなく，Metamer から GUI のそれぞれの構成要素を変更する機能を呼び出すことができる．また，プログラムやリソースファイルの形式で GUI のカスタマイズを行うのではなく，マウスやメニュー操作によってカスタマイズを可能にする．

現在，筆者らはこの構想を元に Metamer を Smalltalk 上に実装中である．ユーザインタフェースのセマンティクスとの関連が少ない部分をユーザに開放し，ユーザの好みに応じて，同等機能を持つ GUI 部品同士の入れ替え，ウィンドウ上の GUI 部品のレイアウト変更，簡単な対話の流れの変更を支援する．ほかにも，ユーザにとって不要なものを画面上から消すことができる機能や，他の欲しい機能を持つアプリケーションがあればそのビューを加えることができるようなアプリケーション同士の併合などがある．Metamer では，これらの変更をアプリケーションに手を加えることなく，しかもアプリケーション実行時に変更を可能とする．本論文では，Metamer のアプリケーション実行時 GUI レイアウト変更機能について述べる．

2.2 GUI カスタマイズの現状

X Window System では，リソースを記述することによって，GUI のカスタマイズを行うことができる[6]．しかし，リソースはアプリケーションを起動する前にテキストの形式で記述しなければならず，また，目的とする変更を行うためには試行誤差が必要である．

Metamer では，アプリケーション起動後に視覚的に GUI の変更を行うことを特徴とする．アプリケーションを起動してから画面上で直接対話的に GUI の変更を可能とする．

X Window System 上でアプリケーションの暫定的な GUI を自動生成し，アプリケーション実行時に GUI 修正を行う試みが GhostHouse[16] で行われている．GhostHouse はアプリケーション実行時に GUI 修正可能な C++のクラスライブラリである．GhostHouse では，まずアプリケーションのデータ定義からデフォルト GUI 部品を用いた対話処理ソフトウェアを自動生成する．次にマウスを用いた対話的な編集操作により，自動生成した GUI を，GUI 部品の置換，構造化，レイアウトの変更，メニューの設定などを利用してユーザが望む GUI にカスタマイズすることができる．

Metamer では，このような特別なライブラリを用いて作られたアプリケーションではなく，システムに用意された既存のライブラリを利用して作られたアプリケーションであれば，そのユーザインタフェースを変更できる点が異なる．

Smalltalk 上の明クラスライブラリ[10]では，Look Preference ツールおよびテキストスタイルツールを用意しており，アプリケーション起動後にインスタンスレベルでフォント，文字サイズ，色，ボーダなどの変更が可能となっている．しかし，部品のレイアウト表示に関する変更はできない．

このように，専用のクラスライブラリを用意し，そのクラスライブラリを利用した GUI に対してアプリケーション実行時に GUI の変更を許すものもある．

本研究では，ユーザに対して既存のアプリケーションに手を加えることなく，視覚的に GUI のカスタマイズを可能にすることを目的としている．既存の MVC のクラスライブラリ[9]を元に作成されているアプリケーションであれば，システムが用意しているもの，ユーザが独自に作成したものに関わらずアプリケーション起動後に GUI のレイアウトを変更することができる．

2.3 Smalltalk 上の GUI 構築ツールおよび UIMS

Smalltalk 上の GUI 構築ツール[15]および UIMS[17),18)]

が既にいくつかある．これらはいずれも，MVC (Model-View-Controller)[9]を実装する際に，MVCの各構成要素の分離が明確に行えないために再利用性が低下するという密結合問題を解決するために，MVCの構成を変更している．例えばGlazier[15]では，MVC構成に Manipulator を導入し，GlazierView を通してユーザがGUIを作成することができる．MoDE[17]では，MVCの代わりに，Appearance, Interaction, Semantics から構成される Mode Framework に基づく Mode Composer を用意している．Talkie[18]では，見た目の情報，モデルと情報を授受するための情報などをGモデルに分離する MVCG を提案している．これらのツールや UIMS に共通する点は，GUI作成を重視した GUI 構築環境とするため，従来のMVC を改善し新たなフレームワークとしていることである．

また，Smalltalk のウィンドウの機能強化も様々に行われている．例えば CWS (Constraint Window System)[19] では制約を導入し，それを記述することによって，fixed-scale window や fixed-size window を実現し[19]，ウィンドウの中のサブウィンドウのサイズ変更も可能としている．

しかしながら，これらは新たにアプリケーションのユーザインタフェースを作成し，そのユーザインタフェースを保守するためには非常に有効となったが，反面，既存の MVC から構成されるアプリケーションのユーザインタフェースには適用できないという問題点がある．

3. GUI のレイアウト変更に関する要求調査

本章では，現状の Smalltalk の GUI レイアウトに関して，実際のユーザが持つ変更要求を調査した結果を示す[5]．実際のユーザがどのような変更要求を持ち，それにどう対処したのかを調査した．調査方法として，GUI レイアウト変更に関する紙面上のアンケート調査およびインタビューを行った．調査対象者は，Smalltalk を利用している工学部電気工学科の学部4年生，大学院生，および仕事で Smalltalk を利用しているユーザ23名である．

3.1 レイアウト変更要求調査

以下にアンケートおよびインタビューを行った結果の一部を示す．GUI のレイアウトを変更したいと思ったことがあるかどうか，また変更したいと思ったことがある場合どのように変更したいと思ったかを尋ねた．

- ビュー（GUI 部品）の境界線をマウスで動的に変更できて，それが保存でき，好きな時に呼び出せる機能が欲しい．
- ビュー（GUI 部品）の表示領域をマウスで動的に変更できて，必要な時にデフォルトの状態に戻すことができる機能が欲しい．
- 小さい画面でブラウザ（Browser）を使う時に，レイアウトを変更したいと思ったのでソースに手を加えて変更した．
- ブラウザで，メソッドの検索時と，メソッドの作成時ではレイアウトの割合を変更したかったので，ソースを試行錯誤で試しながら変更した．
- 長いクラス名があるシステムで，ブラウザのクラスの窓だけ幅広くなって欲しかった．
- レイアウトをどうすれば変更できるかわかっているが，面倒なのでそこまでして変更したくはない．
- スクロールする手間を省きたいのでレイアウトを変更したい．
- レイアウトについてはあきらめているので，あまりこだわらない．

このように，ユーザによって様々な要求があり，対処の方法も異なる．変更を希望し，実際に変更をすることができたユーザは，ソースコードに手を加えることによって GUI の初期設定の変更を行っている．しかし，アプリケーション起動後に必要な変更を行うことができたユーザはいなかった．必要であろうと思われる状況に合わせた初期設定を試行錯誤で行っている．つまり，現状の Smalltalk では，GUI の変更やカスタマイズを行うためにはプログラムを変更しなければならないという問題があり，さらに変更は初期設定に限られている．また，初心者ユーザでは，変更要求があっても，実際にどこをどのように変更すればよいかわからないため，プログラムによる変更もできない．そのため，アプリケーション起動後に視覚的なレイアウト変更機能が有効になる．

Metamer を利用することによって，これらの変更要求を持つユーザに対して，既存のアプリケーションに手を加えずに視覚的に GUI の変更を支援することができる．

3.2 既存のアプリケーションのレイアウト上の問題

次に，既存のアプリケーションのレイアウトで問題となると思われる実例を示し，その対処方法を調査

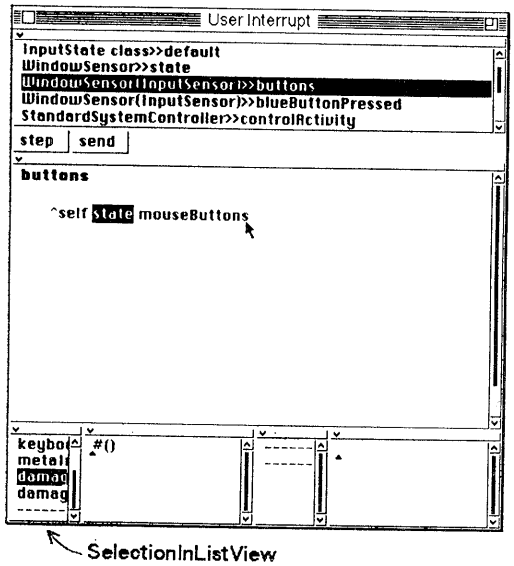


図 3 Debugger のレイアウト上の問題
Fig. 3 A problem of GUI layout in Debugger.

した.

ウィンドウ中の GUI 部品に割り当てられた表示領域が，実際に表示のために必要とする領域よりも小さい場合にレイアウト上の問題が発生することがある．例えば，テキストビューでは割り当てられたサイズに合わせて自動的に文字が折り返されるが，リストビューの場合にはリスト項目のはみ出した文字を見ることができない．

例として**図 3** に示すような，Smalltalk の標準的なツールであるデバッガ[8]を取り上げ，ユーザが実際にどのようにこの問題を回避しようとしているのかを調査した．

一般的に，Smalltalk では変数名が長いことが多く，リストビューに変数名のすべてを表示することができない．ここでは，インスタンス変数を表示するリストビューに注目し，そのインスタンス変数名を識別するにはどうするかを聞いた．

- ウィンドウ全体をリサイズする（16 名）
- インスペクタ（別のツール）を起動する（2 名）
- わからない（5 名）

ウィンドウ全体のリサイズによって，インスタンス変数のリストビューのサイズが変更されることを期待している．この中には，「それでも見えない場合にはあきらめる.」と答えたユーザもいる．このように，ほとんどのユーザはウィンドウ全体をリサイズすることによってこの問題を回避しようとしているということがわかった．アプリケーションに適した GUI に変更できることが必要であるにもかかわらず，実際には，必要な変更はプログラミングによってしかできない．

このような場合にも，Metamer のアプリケーション実行時 GUI レイアウト変更機能を利用することによって，ユーザの好みの GUI レイアウトに変更することが可能になる．

4. GUI プレゼンテーション・フレームワーク

ここでは，アプリケーション実行時 GUI レイアウト変更機能のベースである DEVO (Dynamicaly Extended View Objects)[12)~14)]について述べる．まず，通常の Smalltalk のウィンドウ構成について述べ，本論文で提案する DEVO について述べる．

4.1 Smalltalk のウィンドウ構成とビューのレイアウト

Smalltalk 上のウィンドウを構成するコンポーネントのクラスは，Dependent と Autonomous に分類できる[8]．Dependent は MVC フレームワーク[9]においてモデルを表現する依存物であるビューであり，Autonomous はマルチプルビューのレイアウトや座標を管理し，モデルとは直接関わりを持たない自律的なビジュアルコンポーネントである[8]．

Dependent に属するクラスとして，

- View とそのサブクラス

Autonomous に属するクラスとして，

- 他のコンポーネント集合を保持する CompositePart とそのサブクラス
- コンポーネントにボーダのような一般的な機能を加える Wrapper とそのサブクラス

がある[9]．

View は Dependent に属し，リスト，テキスト，ボタンなどのユーザとの対話を行う GUI 部品である．Composite は Autonomous に属し，コンポーネ

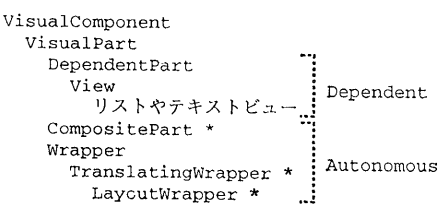


図 4 ビジュアルコンポーネントクラス階層の一部
Fig. 4 A part of VisualComponent class hierarchy.

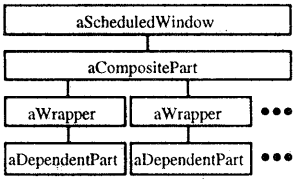


図 5 Smalltalk のビューインスタンスツリー
Fig. 5 A View instance tree.

ントのコンテナの役割を果たす．Wrapper もまた Autonomous に属し，部品の装飾やレイアウトの機能を持つ．**図 4** に Smalltalk のビジュアルコンポーネントを構成するクラスの一部を示す．

図 5 に通常の Smalltalk のウィンドウインスタンスツリーを示す[9]．このように，ウィンドウが複数の View から構成される場合には，View を Wrapper で包み，CompositePart がその集合を保持する．明示的に Wrapper を指定しない場合には，CompositePart に View を付け加える時にデフォルトの Wrapper が自動的に付加される．

Wrapper と Composite が Autonomous なビジュアルコンポーネントに相当し，Composite と Composite 上の Wrapper が GUI 部品 (Dependent) の配置管理を行う．

Smalltalk ではレイアウト情報やボーダなどの部品の装飾は Dependent が持たず，Wrapper として分離している．このため，Wrapper を複数重ね合わせたり，Wrapper を取り換えることによって様々な機能を実現できる．例えば**図 6** の右側では，Image を CompositePart 中の右半分に貼り付けるために BoundedWrapper で Image を包み，左側のテキストビューに 3 種類の Wrapper を重ねることによって，書き込み禁止 (PassivityWrapper) でスクロール可能な (ScrollWrapper) ボーダ付きで左半分の領域を占める (BorderedWrapper) テキストビューを作成することができる[8]．

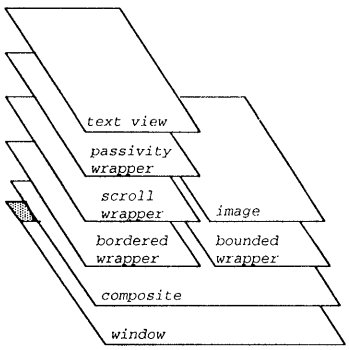


図 6 Smalltalk のビュー階層の例
Fig. 6 An example of View hierarchy.

Smalltalk では，GUI 部品のレイアウトはウィンドウ生成時に与えられ，与えられたレイアウトが初期レイアウトとなり，ウィンドウがリサイズされた場合に表示領域を再計算するために使われる[8]．

GUI 部品のレイアウトは Wrapper (具体的には図 4 中の LayoutWrapper) のインスタンス変数 layout によって決定される[8]．通常，Smalltalk の layout には矩形領域として Rectangle が保持され，ウィンドウ上にそのビューが占める領域の割合を保持する．更に，固定サイズビューやエッジからの距離固定ビューを実現するために LayoutOrigin や LayoutFrame が用意されている[8]．これによって，GUI 部品間に簡単なコンストレイントが実現できる．

レイアウトを設定する layout : aLayoutObject メソッドのセンダはインスタンスの生成時に呼び出される add : aVisualComponent in : aLayoutObject 等のメソッドに限られている．部品の初期レイアウトを変更するには，該当するメソッドのレイアウト記述を変更すればよいが，数値情報 (Rectangle や LayoutFrame) として記述する必要があり，変更はクラス単位に及んでしまい，インスタンスレベルの変更はサポートされていない．

4.2 DEVO (Dynamically Extended View Objects)

本研究で提案する DEVO[12)-14)]は，アプリケーション実行時に，GUI 部品のレイアウトやビューのツリー階層を動的に変更することができる．Metamer では，DEVO の機能を呼び出して利用している．

DEVO は，Smalltalk の Autonomous なビジュアルコンポーネントに着目し，既存の MVC 構成のまま GUI のレイアウト，ビュー階層の変更を可能とする．このため，既存のアプリケーションであっても手を加えることなく DEVO の機能を利用できる．

通常，プログラマが作成するビューはユーザとの対話を行う Dependent なビジュアルコンポーネントである[8]．**図 7** のように，DEVO ではプログラマが通常手を加えることがない Autonomous なコンポーネントにレイアウト変更のための機能を付け加えている．実際の実装では，図 4 中の「*」を付した Autonomous なクラスに従来との互換性を保ちながら機能を拡張している．また，プログラマが Autonomous な

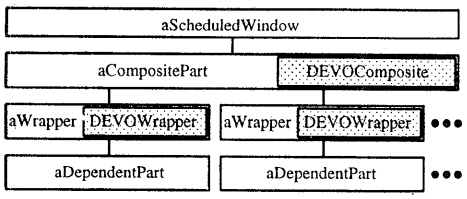


図 7 DEVO Framework
Fig. 7 DEVO Framework.

クラスを作成した場合にも，DEVO の機能拡張を行った CompositePart や Wrapper クラスのサブクラスとなるため DEVO の機能を利用できる．そのため，既存のアプリケーションであっても，プログラマが作成したビューを利用するアプリケーションであっても，ウィンドウのインスタンス作成時には機能拡張された Autonomous なコンポーネントが利用されるため，DEVO の機能が利用できることになる．このように，DEVO は，通常の MVC プログラミングによって作成されたアプリケーションに自動的に含まれる．既に動作しているアプリケーションに対しても Smalltalk の実行時動的メソッド決定によって DEVO の拡張機能が利用できる．

DEVO の基本機能として，

- レイアウトの変更
  移動，リサイズ
- ビュー階層の変更
  グループ化，アングループ化
- レイアウトの整形
  グループ内のビューの整形
- スクロールバー
  スクロールバーの付加，削除

がある．これらの基本機能を組み合わせて，次章に示すような更に複雑な機能を実現できる．ウィンドウ内の GUI 部品（ビュー）に対して，移動，リサイズ，グループ化等を行うことができるように対応するメソッドを各 Autonomous なクラスに実装している．例えば，TranslatingWrapper には move: aPoint メソッド，LayoutWrapper には resize: aRectangle メソッド，CompositePart には group: wrappers メソッドの記述がある．DEVO ではこれらのメッセージを送信することによって，初期設定されているレイアウトからの変更を可能としている．ただし，ユーザ操作の進行につれて GUI のレイアウトが動的に変わるようなアプリケーションの場合には，ユーザによる変更を行うこともできるが，レイアウト上に不具合が発生する可能性がある．

Wrapper や CompositePart に対してメッセージを送信することによって DEVO の機能を利用できる．グループ化によって，複数の部分をまとめて取り扱うことができ，枠をつけることができる．また，グループとなる CompositePart に並びの変更を指示することによって，グループ内部の GUI 部品の配置を変更することができる．同一階層内の複数の Wrapper を一つの CompositePart で取りまとめてグループを実現している．また，グループ化したものをアングループ化することも可能である．

GUI 部品は既存の部品をそのまま利用すればよく，DEVO を組み込んでも，既存のアプリケーションは以前のようにそのまま利用可能である．すべての Autonomous なクラスのインスタンスに DEVO の機能が貼り付くが，Smalltalk の効率的なオブジェクト，メソッド管理機構によって消費する資源は少なくて済む．日常的に Metamer および DEVO を組み込んでいる２人のユーザからは，他のアプリケーションに影響を与えるようなパフォーマンスの劣化は特にないと報告されている．

DEVO はまた Smalltalk の動的なオブジェクト管理の機構を利用している．例えばグループ化の場合には，アプリケーション実行中にグループ化が必要な部分でグループとなる CompositePart のインスタンスを生成し，ウィンドウのインスタンスツリー中に挿入している．スクロールバーが必要な場合には Border-Decorator[8]のインスタンスをツリー中に挿入または既にある Wrapper との取り換えを行っている．Smalltalk を利用すれば，アプリケーション実行時にこのようなウィンドウインスタンスツリーに関する動的な変更を行うことができ，オブジェクト生成，削除の負荷も小さい．

アプリケーション実行時にウィジェットを動的に取り扱うことができるウィンドウシステムであれば，Smalltalk 以外のウィンドウシステムでも DEVO の考え方を適用できる可能性がある．

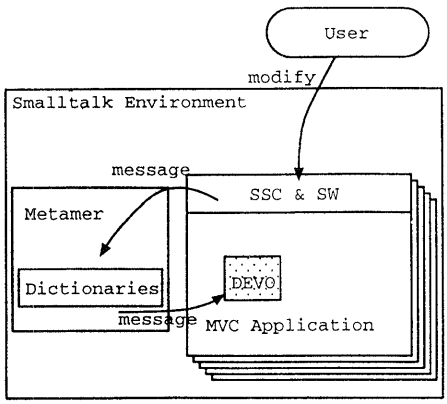
5. アプリケーション実行時 GUI レイアウト変更機能

本章では，Metamer のアプリケーション実行時 GUI レイアウト変更機能について述べる[5]．これらの機能は DEVO の基本機能を組み合わせて実現している．Metamer では，ScheduledWindow[8]のコントローラである StandardSystemController[8] からレイ

アウト変更機能を呼び出すことができる．ScheduledWindow および StandardSystemController は，ウィンドウを利用したアプリケーションの実行中に貼り付いて，ウィンドウの管理を行うウィンドウツリーのトップに位置する[9]．Control キーとマウスボタンを利用して，単一選択，複数選択により任意のビュー階層を選択でき，メニューを利用して GUI のレイアウト変更に関する指示を出すことができる．

ユーザから見れば，Metamer の機能を利用しなければ通常の Smalltalk システムと等価であるが，Smalltalk システムから見るといくらか Metamer の機能のために資源を消費する．また，既存のアプリケーションのユーザインタフェースを変更することができるが，DEVO を利用しているため，従来の MVC の構成に変更はない．Metamer のアプリケーション実行時 GUI 変更機能のうち，レイアウト変更およびビュー階層の変更を行うのが DEVO の役割であり，変更のためのユーザインタフェースおよび変更に関する管理は Metamer が行う．DEVO の基本機能は，移動，リサイズといった GUI 部品のレイアウト変更およびグループ化，アングループ化によるビュー階層の変更である[13]．

図 8 に Metamer の位置付けを示す．Metamer は選択のための辞書，変更管理のための辞書を持つ．さらに，GUI 部品の選択，領域変更のための表示やマウス入力，メニュー表示等を行い，ユーザが決定した操作を DEVO にメッセージの形で送信する．DEVO はその指示に従って GUI のレイアウトを変更する．この操作の流れによって，ユーザは Metamer を利用して，アプリケーションの GUI の任意のビュー階層を選択し，マウスやメニューを利用して希望する指示を出すことができる．



SSC: StandardSystemController
SW: ScheduledWindow

図 8 Metamer の構成
Fig. 8 The configuration of Metamer.

5.1 ビューの領域を動的に変更する機能

本機能は，GUI 部品の表示領域をアプリケーション起動後に動的に変更できる機能である．

図 9 に Smalltalk の標準的ツールであるブラウザを示す[8]．**図 10** にブラウザのリストビューの一つのレイアウトを変更した例を示す．表示された枠をマウスでドラッグすることによりビューの領域を変更することができる．他の関連するビューのレイアウトも自動的に更新される．枠を表示し，ドラッグにより枠のサイズを変更し，決定する部分は Metamer が行い，その枠のサイズによって実際にビューの領域の変更を行い，関連するビューの領域を調整するのは DEVO である．

5.2 指定したビューをウィンドウいっぱいに広げる機能，別ウィンドウとして開く機能

本機能は，指定した GUI 部品の表示領域をアプリケーション起動後にウィンドウのサイズいっぱいに広

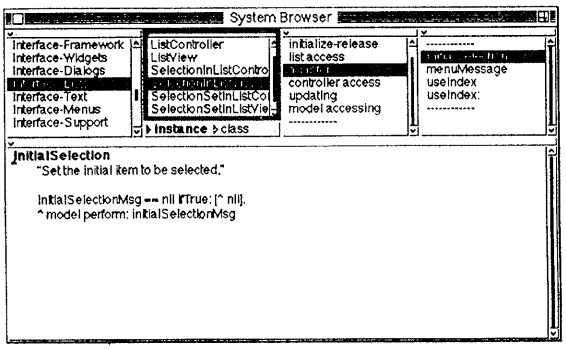


図 9 標準のブラウザ
Fig. 9 A standard Browser in Smalltalk.

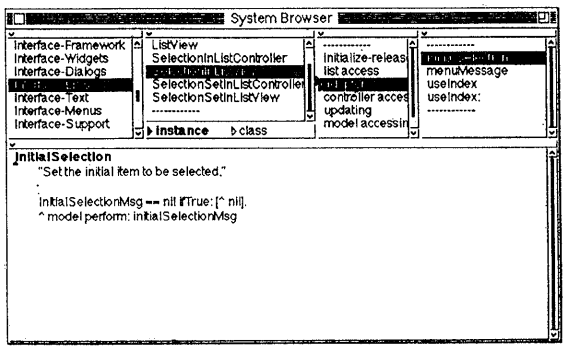


図 10 ビューのレイアウトの動的な変更
Fig. 10 Dynamic modification of GUI layout.

げることができる機能である．前述の調査によって，ユーザから欲しいという要求があった機能である．ビューを指定し，メニューから expand を選択することによりビューの領域を広げ，shrink を選択するこ

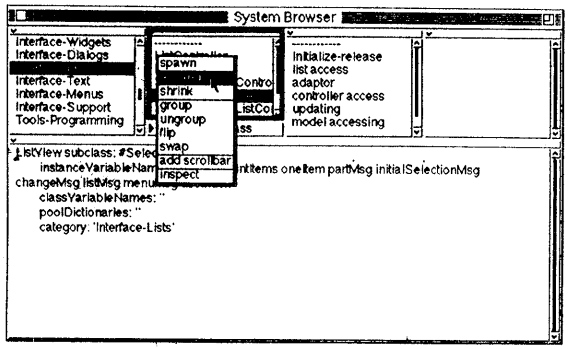


図 11 ビュー選択とメニュー表示例
Fig. 11 A selected view and attached pop-up menu.

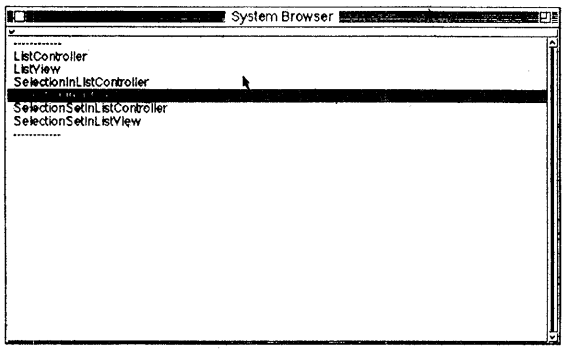


図 12 ビューをウィンドウいっぱいに広げる機能
Fig. 12 The function where one view can be expanded to the bounding of a window.

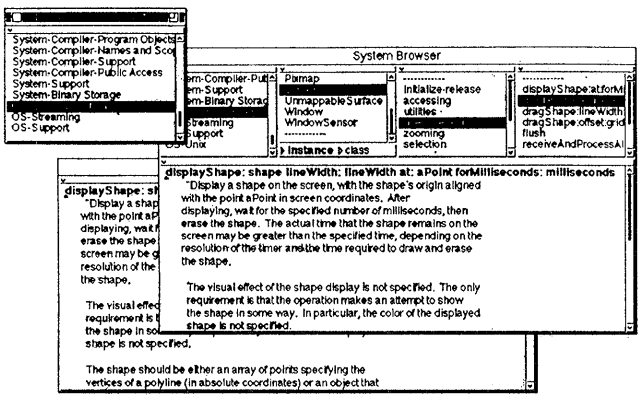


図 13 ビューを別ウィンドウとして開く機能
Fig. 13 The function where a view can be opened as another window.

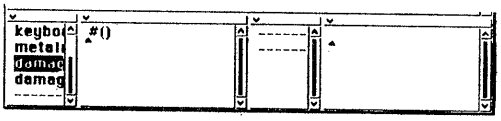


図 14 変更前の横スクロールバーのないリスト
Fig. 14 A list view without horizontal scrollbar.

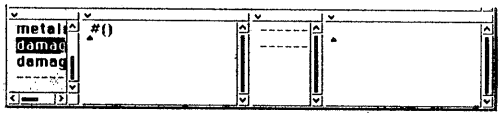


図 15 スクロールバーをつける機能
Fig. 15 A function of adding horizontal scrollbar.

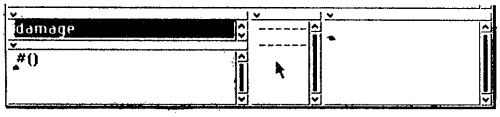


図 16 ビューの並び方を変更する機能
Fig. 16 The function where views can be rearranged either horizontally or vertically.

とにより元の状態に戻る．Metamer がビューの元の領域を辞書によって管理し，レイアウトの変更は DEVO が行う．**図 11** にその様子を示す．

図 12 にその実行例を示す．また，元に戻すこともできる．Metamer の管理辞書によって，DEVO を利用してビューの領域を復元する．

また，指定したビューを別のウィンドウとして開く機能として spawn を用意している．

図 13 では，左上の Class category ビューと下の Code ビューを別ウィンドウとして開いた例である．

5.3 ビューにスクロールバーをつける機能

本機能はスクロールバーのないビューにスクロールバーをつける機能である．**図 14** は，横スクロールバーのないリストの例である．Smalltalk では，通常，リストには縦方向のスクロールバーしか付属していない．

図 15 では，横スクロールバーのないリストに横スクロールバーを付加した例を示す．

5.4 ビューの並び方を変更する機能

本機能は，ビューの並び方を変更できる機能である．**図 16** に示すように，複数のビューの並ぶ方向を横から縦または縦から

横に変更する機能である．Metamer の複数選択機能を利用し，DEVOのグループ化機能と並びの変更機能を利用することによってビューの並びの変更を行う．

5.5 ユーザ作業ログの採取

レイアウト以外の支援として，ユーザ作業ログの採取機能がある．ユーザが Metamer の機能を利用した場合に，その対象と操作をログとして記録できる．これによりアンドゥ操作や，後でユーザがカスタマイズしたレイアウトの反映を行うために利用する[20]．また，ユーザのレイアウト変更作業の解析に利用する予定である．

6. レイアウト変更機能の評価

ユーザの主観的な評価を得るための評価実験として，図9に示すブラウザを利用した3種類の検索実験を行った[21]．3種類の実験とも，ブラウザのサイズは一般的なパーソナルコンピュータのディスプレイサイズである640×480ピクセル固定とした．

6.1 ユーザに機能を利用してもらい意見を調査する

3.2節で述べたように，実際に表示を必要とする領域よりも表示が許された領域の方が小さいという状況がブラウザを利用した検索時にも発生する．実験1に利用した機能は領域を動的に変更する機能（以下サッシ機能と表す）およびビューをウィンドウいっぱいに広げる機能（以下 expand 機能と表す）である．標準のブラウザおよび「明」のブラウザ（あらかじめ横スクロールバーがついているもの）を比較対象とした．被験者はウィンドウ環境の利用経験がある13名である．数回の練習の後，それぞれについて10回の検索を行った．実験終了後，各機能や操作性について被験者の意見を言ってもらった．

1. 標準のブラウザ
   - 操作が簡単で使いやすい
   - エディタ部で確認する時に表示が見づらい
   - 1つ1つ選択してエディタ部で項目名の確認をするのが面倒
2. 横スクローバー付き
   - 前半の部分に同じものがたくさんあるときに便利
   - ずらし過ぎると項目名の前半部が見づらくなって検索しにくい
   - スクロールするのが面倒
3. サッシ機能の利用
   - 領域をちょうどいい大きさにできれば便利
   - 操作が面倒
4. expand 機能の利用
   - 慣れれば一番やりやすい
   - ポインティングが大雑把でよい
   - 1回の操作で項目名全部が見られるのがよい
   - 操作が面倒

標準のブラウザや横スクロールバーがついているものは検索操作が簡単であるが，検索項目の表示が見にくく，サッシや expand はレイアウト変更操作は複雑だが表示は見やすくできるという意見が得られた．操作の複雑さはキーボードとマウスの併用をしなければならないという点が指摘された．GUI の変更を行うためのユーザインタフェースの検討を行う必要がある．実験1のように，一時的に検索項目を見渡したい

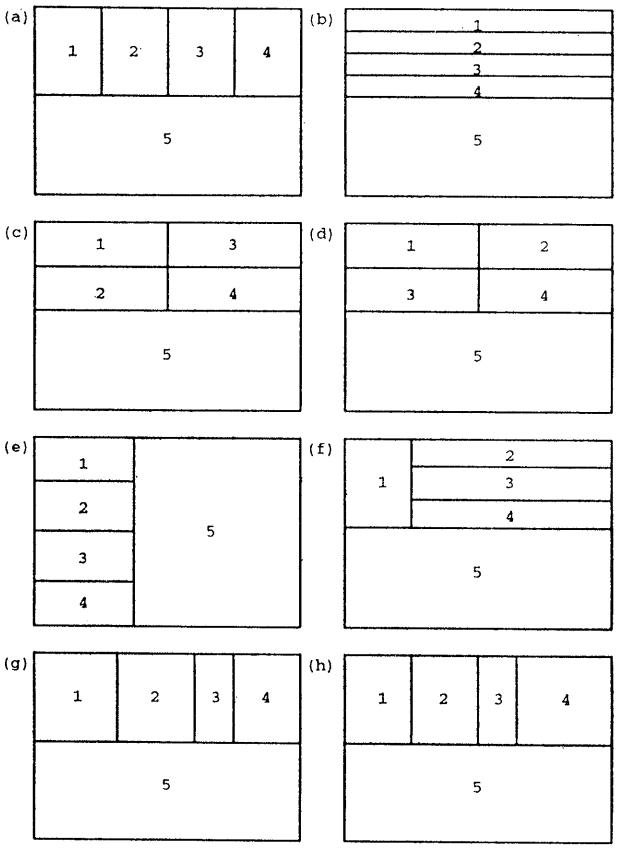


図 17 用意したレイアウトパターン
Fig. 17 Prepared layout patterns for experimentation 2.

状況ではサッシ機能よりも expand 機能が好まれた．

6.2 同一のアプリケーションに対して異なったレイアウトを用意する

実験2では，同一のアプリケーションに対して，異なったレイアウトを複数用意し，レイアウトに関する意見を調査した．ここではブラウザに対してあらかじめ標準のものを含めて8種類のレイアウトを用意した．Smalltalk の知識がまったくない実験担当者が30分程度の独習で操作法を習得し，標準以外の七つのレイアウトパターンを約45分で作成することができた．そのレイアウトパターンを**図17**に示す．図17(a)が標準のブラウザのパターンであり，図17(b)～(h)が用意したパターンである．図17の矩形内の1から4までの数字の順に検索用のリストが並び，5に検索したメソッドが表示される．

被験者はウィンドウ環境の利用経験がある11名である．それぞれについて与えられたレイアウトのまま5回の検索を行い，実験終了後，良いまたは悪いと思ったレイアウトのものを二つずつ選んでもらった．結果を**図18**に示す．

パターン(b)は項目名がすべて見えるが縦方向のスクロールを頻繁に行う必要があり，全員が良くないパターンであると答えた．標準のもの(a)および(a)の並びの縦横を変えたもの(e)を悪いと評価した被験者はいなかった．

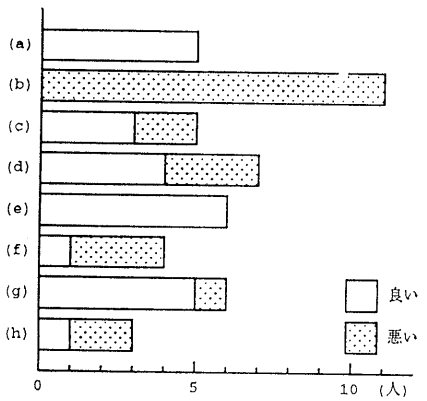


図 18 レイアウトの好みの結果
Fig. 18 User's taste for layout.

6.3 ユーザによるレイアウトの変更

実験3では，同一のアプリケーションに対して，被験者自身にレイアウトパターンを変更してもらった．標準のブラウザのレイアウトから被験者の好みにあったレイアウトパターンに変更してもらった．被験者はウィンドウ環境の利用経験がある6名である．

利用された機能としては，サッシ機能が最も多く，次に縦から横または横から縦に並びを変更する機能が続いた．今回の被験者の中には横スクロールバーを付ける機能を利用したものはいなかった．実験1であらかじめ横スクロールバーが付いていた場合にはスクロールバーが利用されたが，実験3では被験者自身がスクロールバーを付けることはなく，スクロールの必要がなるべくないようにレイアウトが変更されている．

被験者の変更したパターンの例を**図19**に紹介する．

実験後，実験2のレイアウトパターンも含めて，「最も良いと思うレイアウトパターンはどれですか？」の問いに，全員が「自分で変更したレイアウトのものが一番良い」と答え，本研究の目的である，ユーザの主観的な満足度を上げることができたと考えられる．

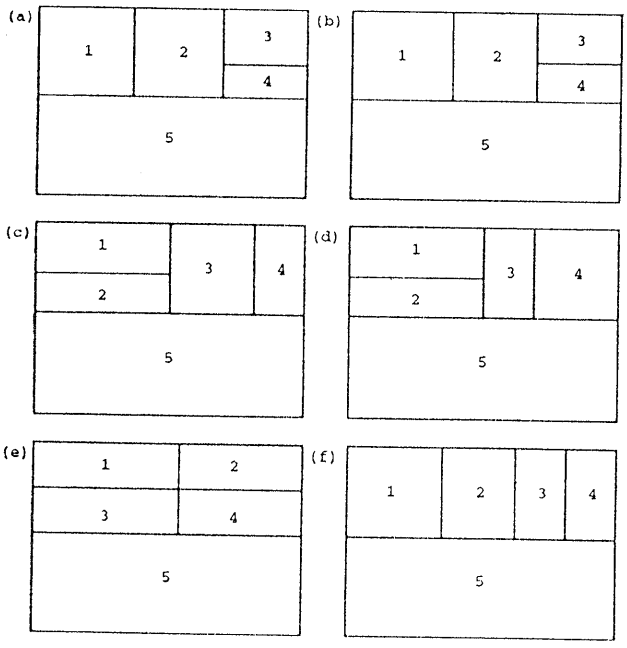


図 19 被験者が変更したパターンの例
Fig. 19 Examples of modified layout pattern by subjects.

7. おわりに

実行時 GUI レイアウト変更機能によって，既存のアプリケーションに手を加えずに GUI 部品のレイアウト変更が可能となる．この機能によって，アプリケーション起動後にユーザが視覚的に GUI のレイアウト変更を行うことができるため，GUI のカスタマイズ機能として利用できる．また，既存のアプリケーションに対してこの機能を利用することによって，GUI のレイアウトを変更し，レイアウトが与えるユーザへの影響を試すことが可能になる．

現在，Smalltalk にあらかじめ用意されているツール群および，明クラスライブラリに用意されているツール群でレイアウト変更機能を利用可能であることを確認した．先の調査で実際のユーザから挙げられた要求に対して，様々な GUI レイアウト変更を行うことが可能となった．これによって，今までソースコードを変更することによってレイアウトを変更していたユーザに対してプログラミングをすることなく希望する変更を可能にし，GUI レイアウトに関するプログラミングの知識がないユーザにも利用可能となった．

今後は，残されたユーザインタフェース設計の要素である，構成要素の選択および対話の変更を実現する予定である[22]．ビュー階層中の任意のビューを選択できるため，Metamer のメニュー項目に機能を追加していく形で対応できる．これによって，ユーザのカスタマイズを視覚的に支援し，さらに同一のアプリケーションに対して，GUI 部品やレイアウトが異なる GUI がどのようにユーザの作業に影響を与えるかを実験し，検討していく予定である．

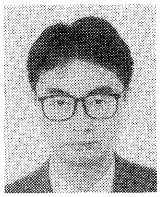
最後に，本論文をまとめるにあたって有益なアドバイスをして頂いた三菱電機株式会社情報システム研究所の宮崎一哉氏，株式会社 SRA プラクティカルソフトウェア工学研究所の渡邊克宏氏ならびに東京電機大学工学部情報通信工学科の守屋慎次教授に感謝いたします．

参考文献

1) 宮崎一哉：ユーザインタフェースに対する要求，情報処理学会夏のシンポジウム—ヒューマンフレンドリーなシステム，pp. 267-274 (1986).
2) 武岡 元，後藤斉衣子，落合 勲，野呂影勇：交流分析によるマン・マシン・コミュニケーションの研究〜交流パターンの分析〜，日本人間工学会第21回関東支部大会，B 1-09 (1991).
3) 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).
4) Open Software Foundation, Inc: *OSF/Motif Style Guide Release 1.1*, Prentice-Hall, New Jersey (1991).
邦訳 株式会社日立製作所ソフトウェア開発本部訳：OSF/Motif スタイルガイドリリース 1.1，トッパン，東京 (1991).
5) 増田英孝，笠原 宏：GUI レイアウト変更要求調査と Metamer の実装，第 47 回情報処理学会全国大会論文集，5 K-4 (1993).
6) Open Software Foundation, Inc: *OSF/Motif User's Guide Release 1.1*, Prentice-Hall, New Jersey (1991).
邦訳 株式会社日立製作所ソフトウェア開発本部訳：OSF/Motif ユーザーズガイドリリース 1.1，トッパン，東京 (1991).
7) 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，共立出版 (1992).
8) ParcPlace Systems, Inc.: *Objectworks\Smalltalk Release 4.1 User's Guide* (1992).
9) 青木 淳：Smalltalk ソフトウェア開発，*Super-ASCII*, Vol. 2, No. 6 - Vol. 3, No. 5 (1991-1992).
10) 渡邊克宏：Objectworks\Smalltalk クラスライブラリ明，UserTalk グループ「春のフォーラム '93」(1993).
11) Marcus, A.: *Graphic Desigh for Electronic Documents and User Interfaces*, Addison-Wesley, New York (1992).
邦訳 小川俊二訳：見せるユーザー・インタフェース・デザイン，日経 BP 社，東京 (1993).
12) 増田英孝，笠原 宏：DEVO (Dynamically Extended View Objects)：UIMS のためのフレームワーク，第 45 回情報処理学会全国大会論文集，2 T-5 (1992).
13) 増田英孝，菊池 肇，笠原 宏：DEVO：GUI プレゼンテーション・フレームワーク，第 47 回情報処理学会ヒューマンインタフェース研究会，47-8 (1993).
14) 増田英孝，笠原 宏：アプリケーション実行時 GUI 変更を可能とするカスタマイズ環境，第 46 回情報処理学会全国大会論文集，7 H-4 (1993).
15) Alexander, J. H.: Painless Panes for Smalltalk Windows, *OOPSLA '87 Proceedings*, pp. 287-294 (1987).
16) 杉本 明，北村操代，中田秀男，川岸元彦，小島泰三：対話型システム視覚的構築用クラスライブラリ：GhostHouse (1)—設計方針と概要—，第 46 回情報処理学会全国大会論文集，6 R-1 (1993).
17) Shan, Y. P.: MoDE: A UIMS for Smalltalk, *ECOOP/OOPSLA '90 Proceedings*, pp. 258-268 (1990).
18) 桑原修二，藤村 茂，富田昭司：MVC の拡張，

MVCG とそれに基づく UIMS, Talkie の実装, コンピュータソフトウェア, Vol. 9, No. 1, pp. 27-41 (1992).
19) Epstein, D. and LaLonde, W. R.: A Smalltalk Window System Based On Constraints, *OOPSLA '88 Proceedings*, pp. 83-94 (1988).
20) 増田英孝, 笠原 宏: Metamer のユーザ履歴管理, 第 48 回情報処理学会全国大会論文集, 2 J-1 (1994).
21) 山下順子, 増田英孝, 笠原 宏: 動的なレイアウト変更を可能とした GUI の評価, 第 47 回情報処理学会全国大会論文集, 5 K-6 (1993).
22) 佐藤博之, 増田英孝, 笠原 宏: GUI 部品の交換によるデザイン変更のためのプラグインアダプタ, 第 48 回情報処理学会全国大会論文集, 2 J-2 (1994).

(平成 5 年 11 月 18 日受付)
(平成 6 年 5 月 12 日採録)



増田 英孝（学生会員）

昭和 40 年生. 昭和 62 年東京電機大学工学部電気工学科卒業. 同年東京電機大学大学院工学研究科電気工学専攻修士課程入学. 平成 2 年同大学院前期課程修了. 同年三菱電機株式会社入社. 平成 4 年同退職. 同年東京電機大学大学院工学研究科電気工学専攻博士後期課程入学. 在学中. オブジェクト指向とグラフィカルユーザインタフェースに興味を持つ. ACM, 日本ソフトウェア科学会, UserTalk 各会員.

笠原 宏（正会員）

昭和 15 年生. 昭和 39 年東京電機大学工学部電気工学科卒業. 昭和 45 年同大学大学院博士課程満期退学. 工学博士. 同大学助手, 講師, 助教授を経て, 現在工学部電気工学科教授. パワーエレクトロニクス, 計算機制御, 制御用分散処理システム, 循環動態計測, オブジェクト指向システム設計法の研究に従事. IEEE, ACM, 日本ソフトウェア科学会, 電子情報通信学会, 電気学会各会員.