生成・カスタマイズ方式によるGUI構築手法の提案とクラスライブラリGhostHouseによる実現

北村操代† 杉本明†

本論文では，アプリケーションプログラム(AP)から自動生成されたGUIに対して，APを実行しながらGUIを編集していくことにより最終ソフトウェアを作成する，生成・カスタマイズ方式を提案する．本方式によれば，GUIを備えたソフトウェア開発においてGUIプログラミングの知識を全く必要としない．また，エンドユーザによるプログラム実行時のGUIの柔軟な変更も可能となる．本論文では，本方式を実現するため筆者らが開発した，C++言語によるクラスライブラリGhostHouseについても述べる．APとGUIの結合を維持しながらの柔軟なカスタマイズを可能とする際の問題点を論じ，データ表現の標準化とリンクオブジェクトの導入を特徴とする部品間の参照方式を示す．GhostHouseの部品群の枠組として，GUI自動生成機能を持つ部品とAP実行時のGUI編集機能を持つ部品を備える．迅速なGUI編集のためドラッグ&ドロップ操作を拡張する．これらにより，部品間参照関係の変更操作を含んだ迅速なGUI修正が可能となる．本論文では，GhostHouseを用いたAPの作成方法を簡単な例を用いて説明しており，実際の適用例を通して有用性の考察を行っている．

GhostHouse: A Class Library for Generating Customizable Graphical User Interfaces

MISAYO KITAMURA † and AKIRA SUGIMOTO †

This paper describes a new method to construct GUI (Graphical User Interface) software. First, a default GUI is generated from data and functions included in an application program (AP). Then the default GUI is customized interactively using a mouse while running the AP. This method allows for application programmers to create GUI software without knowledge of GUI programming. To realize this method, we have developed a C++ class library called GhostHouse. In order to enable flexible customization which preserves the generated connection between the AP and GUI, we introduce a new software structure using a link object, and normalize the representation of data. We also propose a rapid modification method using drag and drop, which changes not only the layout of the GUI but also the relationship and appearance of components in the GUI. This paper shows how to create GUI software with GhostHouse, using a simple example. Results of an evaluation are also shown.

1. はじめに

近年，グラフィカルユーザインタフェース(GUI)を実現するためのソフトウェア開発コストが増大している．GUI構築用のツールキットとしては，Xt(Xツールキット)[1]上のOSF/Motifウィジェット[2]や，C++言語によるクラスライブラリであるInterViews[3]等が提供されている．また，Tcl/Tk[4]のようにこれらのツールキットを操作できるスクリプト言語も開発されている．しかし，プログラミングを通してツールキットを利用するためツールキットの学習が必要であり，一般的なプログラマにとって大きな負担となっていた[5]．また，このため，プログラムの作成に慣れていないヒューマンファクタや応用分野の専門家は，GUIの構築に直接参加できなかった．

GUIの知識を持たずに視覚的にプログラムを作成できる方式として，Hi-Visual[6]やIntelligentPad[7]等が提案されている．これらでは，提供された部品の中にアプリケーションとしてのロジックとGUI機能が組み込まれ，これをビルディングブロックとしてソフトウェア全体を組み立てていくことが可能である．

しかし，監視制御システムやシミュレーションシステムなどでは，従来のUIMS(例えば文献8))が利点としていたGUI部と応用プログラム本体との分離が有効な分野も依然として残されている．このような分野

† 三菱電機株式会社中央研究所
Central Research Laboratory, Mitsubishi Electric Corporation

では，GUI構築を容易にするために，従来二つの方式が研究されてきた[9]．一つはグラフィック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，それらの属性を編集することによりGUIを作成する「エディタ方式」[10)~13)]である．もう一つは，事前にGUI部品の選択，配置や属性設定方法をルールで記述しておき，応用プログラムの仕様記述をもとにGUIを自動生成する「ルール記述方式」[14)~19)]である．

エディタ方式では，ツールキットの詳しい知識がなくても画面の作成が可能となり，開発効率が改善された．しかしエディタにより作成されたGUIプログラムと，応用プログラム本体（以下，APと略す）とを結合するためには，結合部のプログラムを作成する必要がある．この作業には依然としてツールキットの知識を必要とする．例えば，AP内の変数の値の変化をGUI側に反映させるためには，AP内でツールキットのライブラリ関数を用いなければならない．

一方，ルール記述方式では，APの仕様記述からGUIを自動生成する．初期の研究[14)~16)]では，仕様記述としてAPに含まれる関数の仕様を用いていたが，最近の研究[17)~19)]では，APに含まれるデータモデルの記述を中心とした仕様が用いられている．ルール記述方式では自動生成の際にGUIとAPとの結合も行うため，ツールキットの知識を必要としないという利点がある．しかし，適切なGUIは各応用プログラムごと，あるいはユーザごとに異なるため，一般的なルールから所望のGUIを生成することは困難である．またルールの記述も，誰にでもできるわけではない．

以上のような両方式の問題点に対処するために，本論文ではルールを用いてAPからGUIを自動生成した上で，画面上において対話的にGUIを修正していく方式を提案する[20),21)]．本方式を筆者らは生成・カスタマイズ方式と呼んでいる．生成・カスタマイズ方式では，自動生成の段階でAP，GUI間の結合を行うので，エディタ方式のように結合のためのプログラムを書く必要がない．また，GUIの対話的修正をAPとの結合を保ったまま行うので，ルールの再記述を行わなくても，自動生成された暫定的なGUIを所望のGUIに視覚的に変形していくことができる．

筆者らは，生成・カスタマイズ方式を実現するため，C++言語によるクラスライブラリGhostHouseを試作した．本論文では生成・カスタマイズ方式を実現する場合の問題点を明らかにすると共に，GhostHouseで用いた，オブジェクト指向方式による実現手法について述べる．

オブジェクト指向方式では，ソフトウェアはオブジェクト部品から構成される．APを構成する部品をモデル部品，GUIを構成する部品をビュー部品と呼ぶものとする．AP，GUI間の結合は，モデル部品，ビュー部品間の参照関係で表される．**図1**にに生成・カスタマイズ方式の模式図を示す．まず，AP開発者はAP内のモデル部品を定義する．この定義を用いてビュー部品の生成とモデル部品との結合が行われ，暫定的なGUIを備えたソフトウェアが自動生成される．これに対して，APとGUIの結合を維持したままGUIをカスタマイズすることにより，最終的なGUIを持つソフトウェアが完成する．図の開発手順では，開発者は利用するモデル部品の指定と，視覚的なGUIの編集を実施するだけでよい．AP開発者は，ビュー部品のプログラミング知識や，モデル部品とビュー部品との結合方法に関する知識を習得する必要がない．さらに本論文では，利用するモデル部品をC言語により記述されたAPから抽出する方法について述べる．

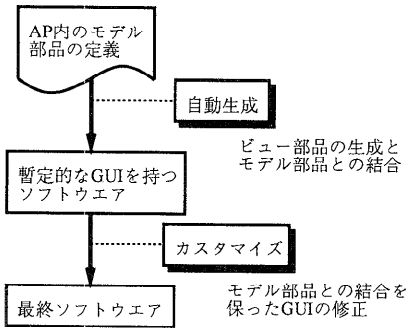


図1 生成・カスタマイズ方式
Fig. 1 Generation and customization.

生成・カスタマイズ方式を実現するため，GhostHouseでは，まず，

1 GUI自動生成機能を持つモデル部品
2 AP実行時のGUI編集機能を持つビュー部品

を備えた．

AP，GUI間の最初の参照関係の設定は，モデル部品によるGUI自動生成時，すなわちモデル部品と対応したビュー部品を自動生成する時に行われる．GUIのカスタマイズフェーズにおいては，ビュー部品の持つGUI編集機能により，GUIを構成するビュー部品が変化する．APとGUIの結合を維持しながらのカスタマイズを可能とするためには，部品間の関係をGUI変更操作に対して柔軟なものとする必要がある．本論文ではデータ表現の標準化とリンクオブジェクトの導入を特徴とする部品間参照方式を示す．

また生成・カスタマイズ方式を有効なものとするた

めには，迅速なGUI修正手段も必要となる．本論文では，従来からデスクトップ操作で用いられているドラッグ&ドロップ操作により，ビュー部品とモデル部品との参照関係の変更を実施する手法も提案する．

以下，2章では，柔軟なカスタマイズを可能とするAPとGUIの結合の実現法を示す．3章では，生成・カスタマイズ方式を実現したクラスライブラリGhostHouseについて述べる．クラスライブラリの概要を述べた後，自動生成のメカニズム，迅速なGUIのカスタマイズ方法，GUIの保存，復元法を説明する．4章では，生成・カスタマイズ方式によるAP作成方法を実例を挙げて説明し，上記のGUI構築方法が提供できることを示す．また，エディタ方式やルール記述方式との比較を通じて，本手法の有効性を明らかにする．5章では結論を述べる．

2. カスタマイズを考慮したAPとGUIの結合の実現

オブジェクト指向クラスライブラリでは，継承を利用して新たな個別部品が作成されていく．前節で述べたようなカスタマイズ操作を統一的に実現するためには，それに適した部品実現の統一的な枠組みを設計しておく必要がある．

生成・カスタマイズ方式のカスタマイズでは，GUI生成の対象となった応用プログラムの表示や操作方法を変更することが求められる．したがって，カスタマイズ機能としては，

1) ビュー部品の属性変更，
2) メニューやボタンによる画面展開などのビュー部品間の連携動作の設定，
3) 新たなレイアウト用部品の追加といった親子関係の再設定をも許すレイアウト変更，
4) ビュー部品の種類の変更（置き換え操作），

が考えられる．生成・カスタマイズ方式では，カスタマイズ時にアプリケーションに新たな機能を実現したり，そのためのGUIを追加したりすることはしない．

生成・カスタマイズ方式では，APとGUIの結合を維持しながらのカスタマイズ操作の実現が必要となる．そこで本章では，上記の1)から4)の中で，特に結合の維持が必要となるビュー部品の置き換え操作を例に詳細を述べ，これを可能とするための，部品間参照関係の実現手法を示す．

2.1 部品間の参照関係とGUIのカスタマイズ

本論文ではAPを構成するオブジェクトをモデル部品，GUIを構成するオブジェクトをビュー部品と呼んでいる．オブジェクト指向によるGUIの構築の枠組みとしては，SmallTalk-80で提案されたMVC(モデル・ビュー・コントローラ)[22]やその変形が用いられてきた．モデル部品はMVCにおけるモデルに対応し，APで用いられるデータや手続きをカプセル化したものである．またビュー部品はMVCにおけるビューとコントローラに対応し，GUI上への表示のほか，ユーザからの操作イベントを取り扱う．

特定のモデル部品とビュー部品の間にはAPとGUIの結合に基づく参照関係が存在する．例えばあるモデル部品内のデータが変更された時には，所定のビュー部品の表示を変更しなければならない．またユーザがあるビュー部品を通じてデータの入力を行った場合には，所定のモデル部品内のデータを変更すると共にその関数手続きを実行する必要がある．逆にモデル部品とビュー部品の間のこのような参照関係により，APとGUIは結合される．

最初の参照関係の設定はモデル部品によるGUI自動生成時に行われる．モデル部品内のデータの表示のために，様々な種類のビュー部品が用意されている．例えば数値データの表示には，数値を文字列で表すフィールド部品やメータの針の位置で表すメータ部品などが用意される．モデル部品によるGUI自動生成時には，表示に用いるビュー部品の種類が選択され，そのビュー部品を生成した後，モデル部品との参照関係が設定される．

GUIのカスタマイズ時にはビュー部品の種類の変更を許さなければならない．例えば数値データの表示を，フィールド部品からメータ部品へ置き換えるといった操作をユーザが行える必要がある．この時，APとGUIの結合を維持するためには，モデル部品とビュー部品の間の参照関係を再設定しなければならない．

図2のGUI例をもとに，置き換え操作による参照関係の再設定の問題を明らかにする．図でモデル部品mはデータとして三つの整数からなる構造体の配列を持つ．vlはレイアウト用部品である．vlの中に，複合デ

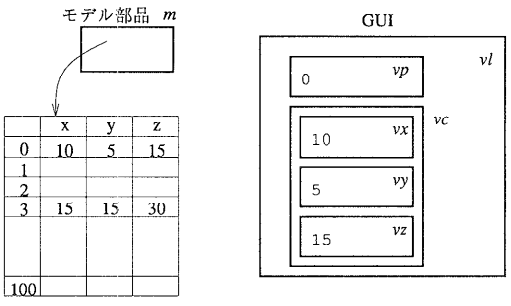


図2 モデル部品とそのGUI
Fig. 2 Graphical User Interface and its data.

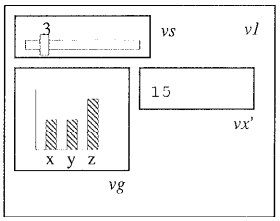


図3 カスタマイズ後のGUI
Fig. 3 An example of GUI after customization.

ータを参照できる部品（以下，複合表示部品と呼ぶ）であるvcとフィールド部品vpがある．さらにvcには，三つのフィールド部品vx,vy,vzが内部に含まれる．vx,vy,vzはモデル部品の持つ配列データの中で，vpの表す数値を配列の添字とする構造体データの，それぞれx,y,zのメンバ変数の持つ部分データを表示する．例えばvpに3を入力すると，vx,vy,vzの表示値は3番目のデータのそれぞれx,y,zの値に変更される．

このGUIに対して，例えば次のような変更操作をカスタマイズ時に実施する．

（1）フィールド部品vpをスケール部品vsに置き換える．
（2）vxをコピーしたvx′をレイアウト部品vlに配置する．
（3）vcを複合表示部品であるグラフ部品vgに置き換える．

結果としてできるGUIを**図3**に示す．これらの変更操作を行った後にも，スケール部品のスライダをユーザが操作して表示値を変更すると，その値を配列の添字とするモデル部品のデータがグラフ部品のそれぞれの棒の長さとして表示されるよう，参照関係の再設定がなされなければならない．また，vx′もスケール部品の値を配列の添字とする構造体データのx部分の表示を継続する必要がある．

また，GUIによってはビュー部品間の参照関係も存在する．例えば，あるビュー部品の属性（フォントや背景色）を属性シート部品を用いて変更するようなGUIでは，属性が変更されるビュー部品と属性シート部品の間に参照関係が設定される．このような場合も，属性シート部品内の個々の属性の表示部品を他の種類のものに置き換えることが可能でなければならない．

2.2 リンクオブジェクト

ビュー部品の統一的な置換操作を可能とするためには，モデル部品とビュー部品，ビュー部品とビュー部品の間の参照関係の実現を再設定容易なものとする必要がある．このためには，部品間の参照関係を部品の動作定義（クラスのメソッド定義）とは独立したものとし，参照関係だけを容易に変更できるソフトウェア構造としなければならない．

GhostHouseでは，まず，部品オブジェクト間の（後に述べる親子関係以外の）参照関係を部品オブジェクトの属性間値伝播関係として一律に表現した．ある部品オブジェクトの属性値が変更されると，設定された属性間値伝播関係により，他の部品の属性値が変更されると共に，属性値の変更がその部品に通知される．部品間の直接的な参照を許さず，属性間値伝播関係により部品間の連係を表現することで，クラスのメソッド定義の局所性が保たれる．属性間値伝播関係を実現する機構としては，active valueによる方式[23]や制約解決による方式[24]がすでに提案されている．また，部品間の参照関係（連係）を属性間伝播関係に統一したとしても，広範囲な機能を持つGUIが構成できることも既に示されている[23]．

GhostHouseの特徴は，参照関係の再設定を容易なものとするため，属性間の値伝播関係を表すリンクと呼ぶオブジェクトを導入したことである．リンクオブジェクトは複数の接続端子を持ち，その接続端子と部品オブジェクトの属性を接続する．ある部品の属性値を変更すると，その属性に接続されたリンクオブジェクトを通じて他の部品の属性値が変更される．部品オブジェクトを他の部品オブジェクトで置換した場合には，元の部品オブジェクトの属性に接続されていたリンクオブジェクトの接続端子を，置き換えた部品オブジェクトの属性に再接続するだけでよい．

リンクオブジェクトで接続された部品オブジェクトの挙動を**図4**を例として述べる．リンクオブジェクトは部品オブジェクトのクラス継承階層とは独立したクラス継承階層を持ち，そこで様々な属性間値伝播関係が定義される．図4では，その中の同値リンクと呼ぶ単純なデータ共有を実現するリンクオブジェクトにより，二つの部品が接続されている．リンクオブジェクトは2種類の接続端子を持つ．一つの端子は仮想データ端子，もう一つは実データ端子である．部品オブジェクトの属性は値を持つか，リンクオブジェクトの仮想データ端子と接続される．図4では，同値リンクL

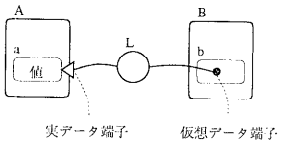
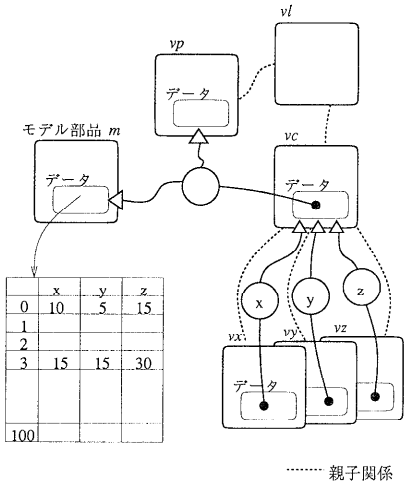
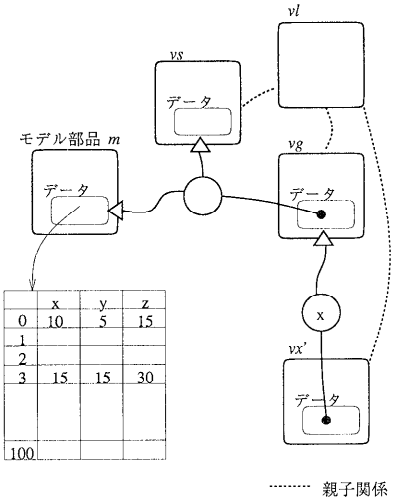


図4 リンクオブジェクト
Fig. 4 A link object.



	x	y	z
0	10	5	15
1			
2			
3	15	15	30
100			

図5 モデル部品とビュー部品のリンクによる接続
Fig. 5 A linking structure of Fig. 1.



	x	y	z
0	10	5	15
1			
2			
3	15	15	30
100			

図6 カスタマイズ後のリンク構造
Fig. 6 A linking structure after customization.

の実データ端子が部品オブジェクトAの属性aと接続され，仮想データ端子が部品オブジェクトBの属性bと接続されている．また，Aの属性aには実際のデータが格納されている．

部品オブジェクトは仮想データ端子が属性と接続されている場合にも，あたかもデータ値を属性値として持つかのように属性にアクセスできる．また，部品オブジェクトの属性値が変化した時，一様に属性変更通知メッセージを受ける．図4では，部品オブジェクトAは属性aの値を直接読み出す．一方，部品オブジェクトBが属性bの値を読み出すと，同値リンクLを通してbの値であるかのようにAの属性aの値を受け取る．部品オブジェクトAが属性aのデータ値を変更した場合，Aは属性aの値が変更されたとの通知を受ける．その時，同時に，部品オブジェクトBは同値リンクLを通して自分の属性bが変更されたとの通知を受ける．部品オブジェクトBが属性bの値を変更しようとする場合は，同値リンクLを通して属性aの値の変更要求がAに伝えられる．その結果Aがaの値を変更した場合には，上記と同様にAとBがそれぞれ自分の属性値の変更通知を受け取る．

図5に図2に対応する部品間のリンク構造を示す．図5ではそれぞれの部品にデータと呼ぶ属性を設定し，それらの属性間の伝播関係で部品間の参照関係を実現している．図では複合表示部品であるvcもデータ属性を持ち，m,vpとvx,vy,vzのそれぞれのデータ属性の中継点としての役割を持つ．モデル部品mと，vc,vpのデータ属性に三つの端子がそれぞれ接続されたリンクオブジェクトにより，vcはデータとしてモデル部品mの配列データの中で，vpの値を配列の添字とする部分データを仮想的に持つ．一方，vx,vy,vzのデータ属性は，それぞれx,y,zの成分に対応したデータフィルタ機能を持つリンクオブジェクトとの接続により，vcのデータ属性値のそれぞれの成分に対応した値を仮想的に持つ．

以上のように本手法によれば，部品間の参照関係をすべてリンクオブジェクトで表現し，個々の部品オブジェクトの実現では部品間の参照関係に依存しない動作定義となる．したがって，リンクオブジェクトとの接続を変更することで，APとGUIの結合を維持した置換操作が可能となる．前節で述べたカスタマイズ操作の結果としてできる部品間のリンク構造を**図6**に示す．

2.3 データ表現の標準化

柔軟なカスタマイズ操作を可能とするためには，データの表現方法の違いを吸収した広範囲な部品の置き換えが可能でなければならない．例えば，図2のモデル部品mの持つ配列データが，x,y,zの属性を持つモデル部品のリスト構造で保持されており，しかも，それぞれの数値の型が実数に変わった場合にも，図2のGUIで表示され上記のような変更操作が可能であることが望ましい．逆に部品の置き換えによる参照関係の再設定ができないような組み合わせの場合には，APとGUIの結合を維持する必要から，ビュー部品の置換を試みた時に，置換ができないことをユーザに通知しなければならない．

GhostHouseでは，まず部品オブジェクト自体のデータ操作の柔軟性を増すため，リンクを流れるデータ

に型情報を持たせた．ソフトウェア実行時に部品オブジェクトはデータの型を識別し，適切な型変換を行う．例えば，フィールド部品は表示データがストリング型の時はそのまま表示するが，整数型の時はその整数を表すストリングに変換してから表示する．

次に，部品オブジェクトの属性やリンクオブジェクトの接続端子には，取り扱うことのできる型の集合（許容型集合）を定義するものとした．部品オブジェクトの属性がリンクオブジェクトの端子と接続できるかどうかは，それぞれが取り扱う型集合の包含関係により決定される．具体的には，実データ端子が属性と接続する場合には，実データ端子の許容型集合が属性の許容型集合を含むことが実行時に検査される．また，仮想データ端子が属性と接続する場合には，属性の許容型集合が仮想データ端子の許容型集合を含むことが検査される．

以上のような手法で柔軟かつ安全な置換操作が可能となるが，データ型の種類が多いとその実現は煩雑なものとなる．これを解決するため，オブジェクト指向方式の持つ情報隠蔽機能を利用し，型情報をオブジェクトにより表現した．リンクを流れるデータは型を表現するタイプオブジェクトを持ち，部品オブジェクトやリンクオブジェクトは，データに付加されたタイプオブジェクトのメソッドを通じてデータをアクセスする．従って，データの実際の表現形式や格納手段は，タイプオブジェクトのメソッドインタフェースにより隠蔽される．例えば，図2において，複合データが実際に配列形式で格納されていてもリスト形式で格納されていたとしても，リンクオブジェクトからは一様なアクセス手段により操作できる．

3. 生成・カスタマイズ方式の表現

本章では，GhostHouseにおける生成・カスタマイズ方式の実現手法について述べる．まず，GhostHouseの概要について述べ，次に，GUI自動生成，実行時のGUIカスタマイズ，GUIの保存，復元手法について順次手順を追って，その特徴を示す．

3.1 GhostHouseクラスライブラリの概要

GhostHouseは生成・カスタマイズ方式を実現した，C++言語によるクラスライブラリである．現在はXウィンドウシステムを対象としている．GhostHouseの設計においては，実際のシステム開発に適用できるよう，既存のソフトウェア部品との共存を考慮した．

システム開発用にはすでに多くの部品が作成され，提供されている．例えば，Xウィンドウシステムを用いたシステム開発用には，Xツールキットや，それを用いたOSF/Motifのウィジェット（GUI部品）などが提供されている．また，アプリケーション本体の作成用部品もソフトウェア資産として，すでに多くの蓄積がなされている．これらの既存ソフトウェア部品を利用しながらも，2章で述べたようなカスタマイズ可能なソフトウェア構造を実現するため，既存部品を操作するGhostと呼ぶクラスと，データを操作するGTypeと呼ぶクラスを導入した．本節ではこれらを中心にクラスライブラリの構成と概要を述べる．

3.1.1 Ghostクラス

GhostHouseクラスライブラリのモデル部品，ビュー部品の最上位クラスはGhostである．Ghostはウィジェットなどの既存部品を利用するため，既存部品に個々に添付され，既存部品を操作し，既存部品間のインタフェースを提供するクラスである．Ghostクラスには，AP内で用いられるデータや関数に対応したモデルゴーストと呼ぶサブクラスと，画面上の対話部品に対応したビューゴーストと呼ぶサブクラスがある．Ghostクラスの主なサブクラスを図7に示す．

図8はモデルゴースト（GModel）とビューゴースト（GView）の役割を示したものである．AP内で用いられるデータの設定や取得，AP関数の呼び出しは

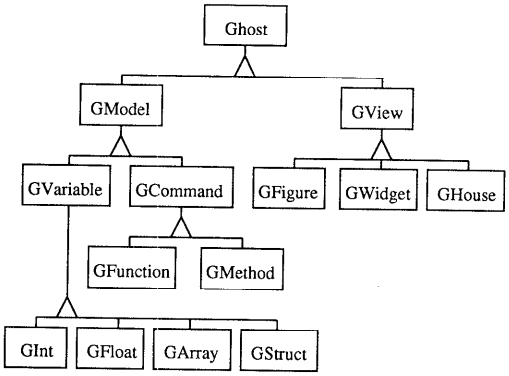


図7 クラス階層図
Fig. 7 Inheritance hierarchy.

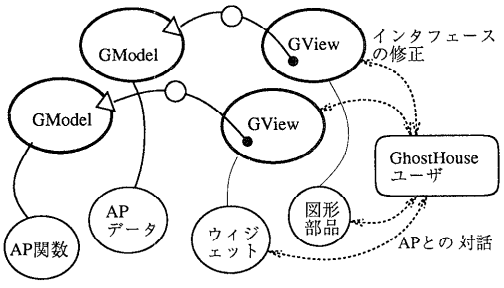


図8 Ghostクラスの役割
Fig. 8 A role of Ghost classes.

GModelを通じて行われ，GModelはGUI自動生成機能を持つ．一方，GViewは既存のGUI部品（OSF/Motifのウィジェットや別途用意している図形部品）の生成，配置を司り，GUI編集機能を持つ．共に，実データあるいは操作対象へのポインタをオブジェクト内部に持つ．

AP通常実行時には，GModelとGViewはAP内のデータや関数とGUI部品との間を，前章で述べたリンクオブジェクトを用いて介在する働きを行う．AP内のデータ値の変更はGModelを通じて行われる．GModelはその副作用として，接続されているリンクオブジェクトを介してGViewにデータ変更を通知する．その結果，GViewは操作対象のウィジェットや図形部品の表示内容を変更する．逆にウィジェットや図形部品がユーザによる対話的な操作を受けると，その操作はGViewを通じて接続しているGModelに通知され，AP内のデータ値の変更や関数の起動が行われる．

GModelのサブクラスには，AP内の変数データを操作するGVariableと関数を操作するGCommandがある．GVariableのサブクラスには，C言語の各基本型に対応するサブクラスを用意している．例えばGIntクラスはC言語のint型データを，GFloatクラスはfloat型データを操作する．また配列や構造体を用いた複雑なデータ構造に対応するために，GArray，GStructが用意されている．一方，GCommandはAP関数を操作対象とする．GCommandに対して実行依頼が発行された時に，操作対象のAP関数が実行される．GCommandには，一般的な関数を呼ぶGFunctionとオブジェクトのメンバ関数を呼ぶGMethodがサブクラスとして用意されている．

また，GViewにはXツールキットのウィジェットを生成，操作するGWidgetと，ウィジェットを補完する目的でGhostHouseで別途用意された任意図形部品GShapeを生成，操作するGFigureがある．GWidgetのサブクラスには，OSF/Motifの各ウィジェットに対応するクラスを提供している．

Ghostのインスタンスには親子関係が設定される．子供のインスタンスは親のインスタンスに所属し，親のインスタンスが削除されると，子供のインスタンスも削除される．GViewの場合には親子関係による入れ子構造により画面上の配置が決定される．インスタンス間の親子関係の最上位に位置するのはgHouseと呼ぶオブジェクトである．したがって，すべてのGhostのインスタンスはgHouseを起点とするツリー状の階層構造の中に位置付けられる．また，Ghostは親子関係の中で一意な名称を持ち，すべてのGhostはgHouseを起点として親子関係を辿る際の名称列（パス名）で特定できる．

3.1.2 タイプクラス

2.3節で述べたような実行時のタイプ情報の提供や，データアクセス手段の標準化のため，GhostHouseではタイプクラスを導入している．リンクオブジェクトを流れるデータはタイプオブジェクトと実際の値（あるいは値へのポインタ）の組で表現される．

タイプオブジェクトは型変換のメソッドや，配列やリスト，構造体等の複合データの内部要素をアクセスするメソッドを持つ．また，配列の個数や列挙型データにおける候補データ集合等の，データに関する補助情報を取得するためのメソッドも備えている．タイプオブジェクトは実際のデータの表現形式を隠蔽する．例えば，配列として実装されたデータとリスト形式により実装されたデータには，違う種類のタイプオブジェクトが組み合わされるが，配列形式のデータに付加されたタイプオブジェクトへのメソッドインタフェースは，リスト形式のデータのタイプオブジェクトにも有効である．

逆に，部品オブジェクトやリンクオブジェクトがその属性や端子に設定されたデータを扱えるかは，そのデータに付加されたタイプオブジェクトの持つ有効なメソッドインタフェースの種類により決定される．Smalltalk-80の実装に関するCookの研究の例[25]と同様に，タイプオブジェクトを実装するためのクラス階層と，タイプオブジェクトの持つ有効なメソッドインタフェースの包含関係から構成されるタイプオブジェクト間の階層関係が独立となるため，2.3節で述べた属性や端子の取り扱えることのできる型の識別には，有効なメソッドインタフェースの種類を使用している．

3.1.3 リンククラス

リンクオブジェクトのためのクラスは，その接続端子の情報を持つ．個々の接続端子の情報は端子名，許容型集合，仮想データ端子か実データ端子かの区別，接続されている部品オブジェクトへのポインタとその属性名から構成される．

部品オブジェクトの属性とリンクオブジェクトの仮想データ端子が接続される場合には，部品オブジェクトの属性値として，リンククラス用のタイプオブジェクトとリンクオブジェクトへのポインタの組が格納される．部品オブジェクトがその属性にある値を設定したり，属性値を参照すると，タイプオブジェクトを通じてリンクオブジェクトの値の変更メソッドや参照メ

ソッドが端子名を引数として起動される．また部品オブジェクトの属性とリンクオブジェクトの実データ端子が接続される場合には，リンクオブジェクトへの参照が，Ghostのクラスにメンバ変数として定義されている部品オブジェクト内のリンクリストに登録される．部品オブジェクトが属性値を変更した場合には，リンクリストに登録されているリンクオブジェクトを検索し，その属性に実データ端子が接続されているリンクオブジェクトの値変更通知メソッドを，端子名を引数として起動する．

3.2 モデルゴーストによるGUI自動生成

GhostHouseはモデル部品を用いて記述されたAPから，その実行時にデフォルトGUIを自動生成する機能を持つ．このことにより，アプリケーションプログラマはGUIを意識せずにプログラムを作成することが可能となる．本節ではAPからデフォルトGUIを生成する方法を述べる．

デフォルトGUIの自動生成は次の段階を追って行われる．

1) プログラマによるAPの作成
2) コンパイルとリンク
3) AP実行時のGUI自動生成
   3-1) AP中で定義されたGModelの検索
   3-2) 発見されたGModelに対し，それぞれ対応するGViewを生成し，リンクで接続

第一段階でアプリケーションプログラマはGModelの各サブクラスを用いてC++言語によりAPを作成する．可視化したいAPの変数について，例えばintと記述する代わりにGIntと記述するだけなので，この作業は特別なGUIの知識を持たずに行うことができる．

C言語を用いる場合は，画面上に視覚化したい変数や関数の宣言の前に修飾子visualをつけてプログラムを作成する．例えば，

```
visual int x;
visual void foo() {…}
```

のように変数や関数を宣言する．この形式で作成されたC言語のプログラムは言語変換システムOPTEC[26]を用いて，GModelが付加されたC++言語のプログラムに変換される．

第二段階では，GhostHouseライブラリをリンクして実行ファイルを作成する．

第三段階のAP実行時の自動生成では，次のような処理が行われる．AP中で宣言されたGModel(のインスタンス）をすべてgHouseに登録する．この登録は各GModelのコンストラクタ中で行われる．実行プログラムが入力イベント待ち状態に入る直前に，gHouseに登録されたGModelそれぞれに対し，対応するGViewを生成して当該GModelと属性間をリンクで接続する．各GModelに対応するGViewを**表1**に示した．現状の生成ルールは表のような単純なものに留まっている．生成されたGViewは順次一つのスクロールバー付きの配置部品中に配置され，表示される．

表1 GModelと自動生成されるGView
Table 1 GModel classes and generated GView classes.

GModel	対象	自動生成されるGView
GInt	int型変数	GField
GFloat	float型変数	GField
GString	文字列型変数	GField
GEnum	enum型変数	GOptionMenu
GBool	2値型変数	GCheckButton
GArray	配列	GMenu 但し子がGStructならGTable
GStruct	構造体	GCard 但し親がGArrayならGMenu
GFunction	関数	GButton

以上のようにして生成されたGUIがデフォルトGUIとなる．モデルとビューの結合が確立されているので，APは正常に動作する．しかし一般に，生成されたデフォルトGUIは利用しやすいとは言えない．GUIの編集機能を利用して，さらにGUIを洗練することが必要である．

3.3 AP実行時のGUI編集機能

GUIの編集機能はGViewに備えられており，すべてのサブクラスに継承される．本節では，GViewに備わったGUI編集機能について述べる．

3.3.1 GUI編集機能

従来のGUIビルダは一つの編集用ウィンドウを用意し，その中にGUI編集コマンド群やGUI編集エリアを配置している．しかし，GhostHouseではAP実行時における迅速なGUI修正を可能とするため，すべての編集は画面上のビュー部品に対して以下のような方法で直接行われる．

1 ドラッグ&ドロップによる編集
2 プロパティボードによる属性の設定
3 ポップアップメニューによる編集

通常の対話操作と区別するため，これらの編集操作ではメタ操作が用いられる．以下ではメタ操作として，キーボード上のコントロールキーを押しながらのマウスボタン入力を用いる．例えば，コントロールキーを押しながらマウスの左ボタンをクリックすることで

GUI部品のドラッグ&ドロップ操作を開始し，中央ボタンではプロパティボードを表示し，マウスの右ボタンを押すことによって，編集用ポップアップメニューが表示される．

3.3.2 ドラッグ&ドロップ操作の種類

GhostHouseではGUI編集操作法にドラッグ&ドロップを採用している．画面上でビュー部品を選択し，ボタンを押しながらマウスを動かすと，GViewのメソッドによってその部品のドラッグが開始される．そして別のビュー部品上でマウスボタンを離すと，それに対してドラッグした部品がドロップされる．

ドラッグ&ドロップ操作は従来からファイルマネージャのデスクトップ操作などで用いられている．その機能には，位置の移動やファイルのコピー，削除（ゴミ箱アイコン上にファイルのアイコンをドロップ）などが挙げられる．GhostHouseでは迅速なGUI編集の手段として，従来の機能に加えて，ビュー部品とモデル部品との接続関係の変更，表示画面設定などの機能を導入した．ここではこれらの機能を説明する．以下では，ドラッグしたオブジェクトを上側部品と呼び，ドロップの対象となったオブジェクトを下側部品と呼ぶ．

a) 置換と吸収

置換操作と吸収操作は，リンクによる接続を変更する操作である．

置換操作は2章で述べたように画面上のビュー部品の種類を変更するために用いる．すなわち，単に下側部品を上側部品に取り換えるだけでなく，下側部品の属性に接続されていたリンクは上側部品の同一属性名を持つ属性に引き継いで接続される．例えば**図9**(1)では，下側部品Lに接続していた二つのリンクは上側部品Uに設定される．図では単純化のために属性を示していないが，GViewが共通に持つdataと呼ぶ属性への接続が引き継がれている．また，下側部品の親子関係と表示位置も上側部品に引き継がれる．画面上に表示されていない種類の部品は，ポップアップメニューにより表示された部品箱からドラッグすることにより追加される．

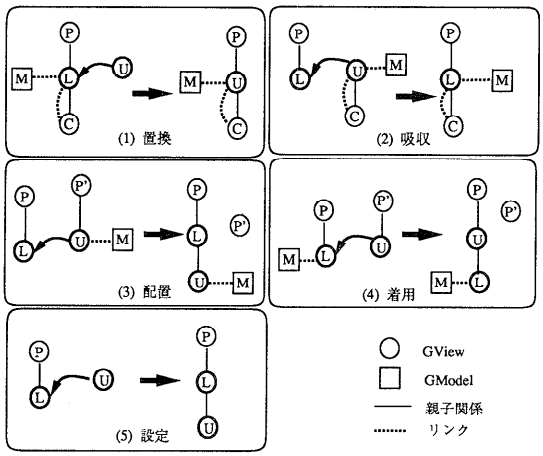


図9 ドラッグ&ドロップ操作
Fig. 9 Drag & Drop operation.

一方，吸収操作は上側部品のリンク接続を下側部品に再設定するために用いられる．上側部品の属性に接続しているリンクを下側部品の同一属性名を持つ属性に付加し，上側部品は消去される．図9(2)では，Uの二つのリンクがLに渡される．

b) 配置と着用

配置，着用操作では，リンクは一切変更されず，画面上のレイアウト変更のみが行われる．

配置操作では，表示位置や大きさの変更を行う．この際に下側部品が上側部品の親でなければ，部品の親の変更も自動的に行われ，下側部品は上側部品の新しい親となる．図9(3)では配置を行っている．Uの新しい親はLである．また，Uのリンクは変更されない．

着用操作は下側部品の親と下側部品の間に上側部品を挟み込む．すなわち，下側部品の新しい親が上側部品となり，上側部品の親は以前の下側部品の親となる．この操作により，例えば元の画面上の位置を保ったままGViewer(スクロールバー付きウィンドウ部品)の中に下側部品を入れることができる．図9(4)では，リンクを変更せずにPとLの間にUを挟み込んでいる．

c) ダイアログやメニューの設定

設定では上側部品と下側部品との画面表示順序の関係づけを行う．例えば，AP実行時に下側部品をマウス操作によりピックした時に，一時的な対話を行うためのダイアログウィンドウやポップアップメニューとして，上側部品が表示されるようにドラッグ&ドロップ操作で設定できる．図9(5)のように，画面表示順序は親子関係の一種として設定される．

d) ドラッグ時の複写

GhostHouseのドラッグ&ドロップ操作には，ドラッグ開始時に上側部品を複写する方法が提供されている．複写操作は，モデルとの接続を保ったコピーを作成する．複写操作を用いて複数のコピーを作成した後に，置換操作を利用して一つのデータと接続された様々なビューを画面上に同時に表示することができる．

3.3.3 ドロップ時の操作の決定

ドラッグ&ドロップ操作により，多様な編集を可能としている．しかし，上側部品と下側部品のクラスにより，実行できる編集の種類は限られる．また，置換や吸収操作は双方のGViewが接続しているリンクに依存する．

ドラッグ&ドロップ操作時には，まず，上側部品と下側部品の二つのクラスと，それぞれの部品が接続しているリンクに応じて可能な操作を列挙する．そして複数の操作が可能ならば，ユーザに対話形式で実際に行われる編集操作を選択させる．可能な編集操作が存在しない場合には，警告が表示される．

3.4 GUIの保存復元方式

本方式では，プログラム稼働中にオブジェクト間の関係を変更することによりGUIの変更を行っている．この変更を次回の起動時にも有効とするには，変更を保存する必要がある．本節では，まず保存方式の概要を述べる．また，自動生成されたGUIを修正した後に再びAPの修正を行っても，実施したGUI修正が有効であることが望ましい．これを可能とするための手法についても述べる．

3.4.1 保存と復元

まず，部品箱や各種設定ツール以外の，存在するGViewをすべてファイルに保存する．保存ファイルはC++言語形式であり，各GViewのコンストラクタ(生成関数)および各属性の設定関数が記録される．コンストラクタの中にはGView間の親子関係も保存される．次にすべてのリンクオブジェクトを保存する．このとき接続関係が保存される．接続関係の保存には，端子に接続されているGhostのパス名とその属性の名称が用いられる．

復元時には保存ファイルをインタプリタ実行する．まずGViewのインスタンスを構築し，次にリンクを再構築する．リンクの再構築では，保存されたパス名と属性名を用いて接続するGhostとその属性を検索する．

保存ファイルはC++言語形式であるため，最終的に実行速度を向上したいときには，保存ファイルをコンパイルしてAPとリンクすることもできる．また，GUI編集機能のみを取り除いたライブラリと結合することにより，エンドユーザにGUI編集を許さない実行プログラムを作成することも可能である．

3.4.2 GUI編集後のAP変更

保存から復元までにAP内のモデル部品をプログラミング作業により変更した場合にも，実施したGUIの修正が有効となることが望ましい．このため，GhostHouseではAPを変更した場合にも，上記と同様に保存ファイルに基づいてビュー部品とリンクを再構築し，その上で，復元後にビュー部品とのリンクが存在しないモデル部品について，3.2節で示したビュー部品の自動生成を行っている．

AP変更の結果，モデル部品がもはや同一リンクに接続できない場合，もしくは，モデル部品が削除されている場合，リンクの再構築に失敗する．この場合にはそのリンクに属性が接続されていたビュー部品の背景色を警告色に変えることで，ユーザに注意を促す．一方，APに新たにモデル部品が導入された場合には，デフォルトのビュー部品が画面上に表示される．

APの変更がモデル部品の実装方法の変更であるような場合には，リンクの再構築にパス名を用いていることから，同一名称を使用している限り，支障なくモデル部品とビュー部品の間のリンクを再構築できる．また，モデル部品の名称や親子関係による階層が変更された場合には，リンク接続されたデフォルトビュー部品が新たに生成される．吸収操作により，デフォルトビュー部品の接続を，以前にリンク接続されていたビュー部品（背景色が警告色となっている）に移すことにより，編集後のGUIを有効とすることができる．

4. 対話型ソフトウェアの作成例

本方式によれば，ルールに基づきAPからGUIを自動生成した上で，生成された画面上で対話的にGUIを修正していくことによりプログラム全体が作成される．一方，あらかじめ対話的に画面を作成しておき，別途作成されたAPプログラムから生成されたGUIと対話操作で融合する方法も可能となる．本章ではGhostHouseによる対話型ソフトウェアの作成手順について具体例を述べ，これら二つのGUI作成方式が可能であることを示す．また，適用例を通して本方式の有効性を明らかにする．

4.1 作 成 例

図10に，摂氏の温度を華氏に変換するAPのC言語による例を示す．このAPでは，エンドユーザが摂

```
                                        <変数群定義部>
visual float celsius = 0.0;
visual float fahr = 32.0;

-------------------------------------------------------
                                        <関数群定義部>
visual static void transform () {
    fahr = 9.0 / 5.0 * celsius + 32.0;
}
```

図10 簡単なAPの例
Fig. 10 A simple example of application program.

氏の温度を入力し，入力された温度の変換を行い，計算結果を表示することを想定している．GUIに表示すべきAP内のデータは摂氏の温度の変数と華氏の温度の変数である．また変換関数も可視化する必要がある．図10では，これらの変数，関数の宣言時にvisualをつけて宣言している．

このプログラムを前処理しコンパイル，リンクする．これを実行すると，実行開始時に**図11**(a)に示すデフォルトGUIが作成される．二つの変数はフィールド部品で，関数はボタン部品で可視化されている．この状態で摂氏のフィールドに数値を入力し，その後に変換のボタンを押すと，華氏のフィールドに変換後の数値が出力される．

通常，このGUIは所望のものでないため，GViewに備わっているGUI編集機能により，GUIのカスタマイズを行う．他の種類のGUI部品を利用できるよう，部品箱を表示させておく．

図11(b)は，スケール部品を部品箱から図11(a)のフィールド部品の上にドロップすることにより，置換操作を行った後の画面である．この時，フィールド部品のAPとの結合はスケール部品に引き継がれるので，スケールを動かして変換ボタンを押すと，変換関数がスケール部品の示す値を用いて実行される．

図11(b)のボタン部品をドラッグしてスケール部品上にドロップすると，ボタン部品と結合している関数がスケール部品に吸収される(図11(c))．これにより，スケール部品のつまみを動かすと，変数値が変更されるとともに，変換関数が起動されるようになる．

また，メインウィンドウ部品を図11(c)のボックス部品の上にドロップして着用を行うと，スクロールバーによるパニング操作が可能となる（図11(d)）．

さらに，図11(e)に示すメニュー部品を別途作成し，メインウィンドウ部品上にドロップして設定を行うと，メインウィンドウのメニューとして設定される(図11(f))．また，図11(e)と同様のメニューを作成しFileボタン部品上にドロップすると，そのメニューはFileのプルダウンメニューとして設定される．

このようにしてGUI編集を加えていくことにより，最終的なGUIが作成される．この方法を用いれば，GUIに関する知識をほとんど必要とせずに，しかも，AP実行時にAP動作を確認しながら，GUIを作成することができる．

一方，生成・カスタマイズ方式においても，GUIとAPを分離して独立に開発することも可能である．例えば，まず最初に部品箱を表示し，配置操作によって最終画面（図11(f)）を作成して，保存する．これとは別に，図10のAPを作成しておく．APをコンパイル，リンク後実行すると，図11(a)と図11(f)の二つのウィンドウが画面上に表示される．これを元に，図11(a)の各部品を図11(f)の部品にそれぞれドロップして，リンクとの接続を吸収させる．その結果，図11(f)の画面を持ち，かつ，APと結合したプログラムが完成する．この場合にも，GUIとAPを結合するプログラミング作業を必要としない．

4.2 評価と考察

本節では，適用例を通して本方式の有効性を明らかにする．題材としては，OSF/Motifのデモプログラムであるmotifburgerを取り上げた．

本実験でも，C言語を用いたプログラムを作成した．これからC++言語への変換時に抽出されたモデルゴーストは14個であり，内訳はGVariableが12個，GCommandが2個であった．また，motifburgerのC言語のプログラムは846行に渡ったが，本実験で作成したプログラムは113行に減少した．motifburgerでは，GUI画面の定義はUIL(GUI記述言語）を用いて記述されているため，C言語プログラム中には，AP本来のロジックと，APとGUIの結合のためのプログラムが含まれていた．一方，本実験のプログラムでは，APとGUIの結合の記述が不要であり，この結合記述部分だけ行数が減少した．

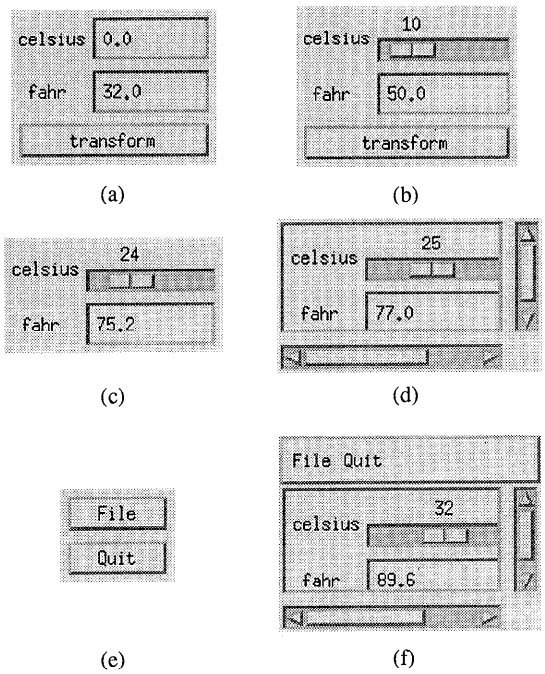


図11 GUIのカスタマイズ画面例
Fig. 11 An example of a customizing flow.

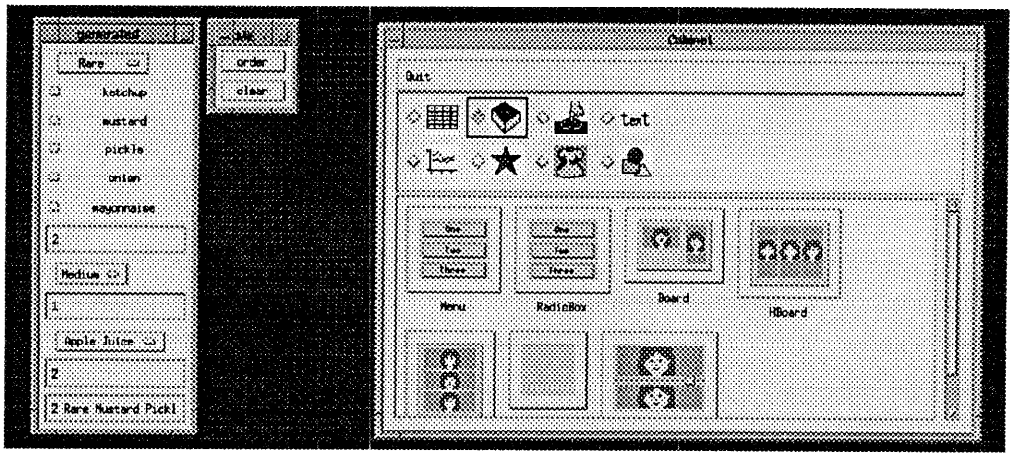


図12 デフォルト画面と部品箱の例
Fig. 12 An example of a default GUI and a parts box.

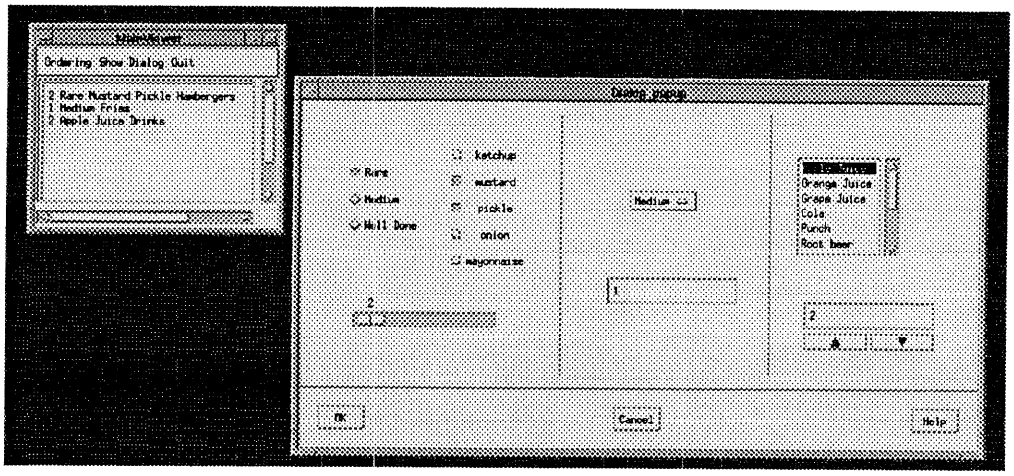


図13 最終画面例
Fig. 13 An example of a final GUI.

図12は自動生成されたGUIと編集に用いる部品箱の画面であり，**図13**は修正後の完成したGUIの画面である．デフォルトGUIから変形して，この完成GUIを作成するために必要な操作とその回数は，使用したドラッグ&ドロップ操作は計31回，内訳は配置が最も多く19回，着用が5回，設定が4回，置換が3回，吸収は0回であった．ドラッグ&ドロップ操作のほか，ラベル名を変更するなどのためにプロパティボードを4回使い，ポップアップメニューでは部品箱の表示や保存を用いた．約5分で完成GUIを作成することができた．

次に，予めGUI画面を作成しておき，同一APから自動生成されたビュー部品を吸収させて完成GUIを作成する手法について，必要な操作とその回数を求めた．まず，GUI画面を作成する段階で，ドラッグ&ドロップ操作は計24回行った．内訳は配置が20回，着用が2回，設定が2回であった．このほか，ラベル名の設定等でプロパティボードを12回用いた．次に，データ吸収を行う段階で，ドラッグ&ドロップ操作を11回用いた．これはすべて吸収操作であった．

エディタ方式で上記の例を構築した場合，同様な時間で図13のようなレイアウトは完成できる．しかしエディタ方式では，GUIとAPの関連付けを，そのエディタの利用するツールキットの関数を記述することにより行わなければならない．この作業にはエディタ方式で提供しているプログラミング環境を利用することもできるが，ツールキットの関数の知識を必要とする．例えばNEXTSTEPのInterface Buillder[12]で作

成した場合，レイアウトやメッセージを送る対象のオブジェクトを視覚的に定義でき，Objective Cのヘッダファイルと，メッセージに応答するメソッドのテンプレートを含んだソースファイルが生成される．これをテキストエディタで編集してメソッドの本体を記述することによりAP全体が構築される．このメソッドの実現の段階で，GUI用のツールキットの知識が必要となる．一方，生成・カスタマイズ方式によれば，自動生成時に結合が確立され，カスタマイズ時にもその結合は維持されるので，改めて結合をプログラミングする必要はない．また，エディタ方式では提供されているプログラミング環境で通常記述するビュー部品間の連携動作（画面展開）は，ドラッグ&ドロップ操作により設定できるようにしている．従って，GhostHouseでは，エディタ方式に見られるようなプログラミング環境は提供していない．

ルール記述方式で上記の例を構築した場合，APからGUIを自動的に生成するため，GUIを利用するための知識は必要としない．しかし，GUIを特定のアプリケーションに適したものにしようとすれば，そのアプリケーションの開発者による詳細なルール記述が必要となる．Baarの研究[17]やWiechaの研究[18]では，アプリケーション内部のデータを定義する時に，特別に用意された記述形式により，GUI生成用の付加的な属性（部品選択やレイアウト上のヒント）を記述していくことで，一般的なルールのもとでアプリケーションに特化したGUIを生成することが可能となる．一方，本論文の方式では，生成されたGUIの直接操作によるカスタマイズが必要であるものの，C言語で応用プログラムを記述する際に3.2節で述べたようにvisualの指定だけをすればよいので，レイアウトのための特別な記述形式の習得を必要としない．

さらに，生成・カスタマイズ方式では，一旦完成されたソフトウェアのGUIをエンドユーザがさらに容易にカスタマイズできるという特徴がある．エンドユーザによってカスタマイズ可能なGUIとしては，GUI部品の属性値の変更を許す研究がある．例えば，Xのリソースファイル[1]では，GUI部品の属性を別ファイルに記述でき，これを各ユーザが変更することで属性値を変更できる．また，editresコマンド[27]によりAP実行時に属性値を変更することもできる．TAE Plus[28]では，属性としてスタイルを持ち，これを変更することで部品に対して定義されている中から画面上の表現形式を選択することができる．一方，生成・カスタマイズ方式の実現には，以上のような部品の属性変更に加え，APとの結合を維持したGUI部品間の連携動作の設定，GUI部品間の親子関係や部品の種類の変更，といったカスタマイズ機能が必要であり，GhostHouseではこれを実現している．

5. 結　論

本論文では生成・カスタマイズ方式とその実現手法について述べた．生成・カスタマイズ方式は，APから自動生成されたGUIに対して，APを実行しながらGUIを編集していくことにより最終ソフトウェアを作成する方式である．これにより，GUIプログラミングの知識を全く持たないプログラマがGUIとのデータやメッセージの受渡しを必要とするAPを作成できる．あるいは，APとGUIを分離して作成した上で，画面上で視覚的にAPとGUIを統合することも可能となる．さらには，エンドユーザがプログラムの実行中にGUIを柔軟に変更することも許す．

筆者らはC++言語によるクラスライブラリGhostHouseに生成・カスタマイズ方式を実現した．まずGhostHouseに，GUI自動生成機能を持つモデルゴーストと，AP実行時のGUI編集機能を持つビューゴーストを備えた．また，APとGUIの結合を維持しながらの柔軟なカスタマイズを可能とするため，データ表現の標準化とリンクオブジェクトの導入を特徴とする部品間の参照方式を備えた．これにより，参照関係の構造を変更する操作を含んだGUIの変更も可能となった．さらに，GUI編集を迅速に実施するため，ドラッグ&ドロップ操作を拡張し，ビューゴーストにその機能を備えた．これにより，部品の種類の変更，部品間の参照関係の変更を素早く行えるようになった．GUI修正後のAP変更に対応するために，名前によるモデルの検索を用い，APを変更しても名前が変更しない範囲で以前のGUI修正が有効であることも明らかにした．

現在，GhostHouseではグローバルなAP変数のみを扱っている．AP中でデータが動的に作成される場合については今後の課題として残されている．また，取り扱えるAP変数は，表1にあげた型のみを取り扱っており，一般的なポインタ型などは対象としていない．

本論文ではC言語からの生成手法について述べたが，アプリケーション分野に応じた生成ルールを記述することにより，よりカスタマイズ量が小さい暫定的なソフトウェアを生成することもできる．筆者らはプラント監視制御システムの監視GUIや帳票GUIを本方式により開発する環境を構築している[21]．それらでは，プラント監視に用いるデータの定義から暫定的な

GUIを生成する．この生成ルールを記述する作業には，本論文で述べたモデル部品，ビュー部品，リンクオブジェクトの取り扱いについて熟知する必要がある．また，アプリケーション分野に応じて新たに部品を追加する作業にもこれらの知識は必要である．しかし，生成ルールが実現された構築環境を利用するユーザは，モデル部品やビュー部品について一切知る必要はない．

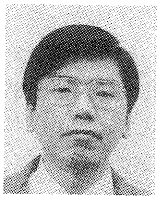
参考文献

1) Nye, A. et al.: *X Toolkit Intrinsics Programming Manual (OSF/Motif 1.2 Edition)*, O'Reilly (1992).
2) Open Software Foundation: *OSF/Motif Programmer's Guide*, Prentice-Hall, New Jersey (1991).
3) Linton, M. et al.: Composing User Interfaces with InterViews, *Computer*, Vol. 22, No. 2, pp. 8-22 (1989).
4) Ousterhout, J. K.: *Tcl and the Tk Toolkit*, Addison Wesley, Massachusetts (1994).
5) Beshers, C. M. et al.: Scope: Automated Generation of Graphical Interfaces, *Proc. ACM SIGGRAPH Symp. on User Interface Software and Technology*, pp. 76-85, ACM, New York (1989).
6) Hirakawa, M., Iwata, S., Yoshimoto, I., Tanaka, M. and Ichikawa, T.: HI-Visual Iconic Programming, *Proc. IEEE Workshop Visual Languages*, pp. 305-314 (1987).
7) 長崎 祥，田中 譲：シンセティック・メディアシステム：IntelligentPad，コンピュータソフトウェア，Vol. 11, No. 1, pp. 36-48 (1994).
8) Schulert, A. J. et al.: ADM—A Dialog Manager, Human Factors in Computing Systems, *Proc. ACM SIGCHI '85*, pp. 177-183 (1985).
9) 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).
10) Cardelli, L.: Building User Interfaces by Direct Manipulation, *Proc. ACM SIGGRAPH Symp. User Interface Software*, pp. 152-166, ACM, New York (1988).
11) DEC: *DEC VUIT User's Guide*, pp. 1-1-1-5, DEC, Massachusetts (1991).
12) Webstar, B.: *The NeXT Book*, Addison-Wesley (1989).
13) 杉山高広ほか：鼎（かなえ）インタフェースビルダ「ゆず」の構築，第45回情報処理学会全国大会論文集（分冊5），pp. 99-100 (1992).
14) Fisher, G. L. et al.: A Control Panel Interface for Graphics and Image Proccssing Applications, *Proc. SIGCHI+GI '87*, pp. 285-290 (1987).
15) Olsen, D. R., Jr.: MIKE: The Menu Interaction Kontrol Environment, *ACM Trans. Graphics*, Vol. 5, No. 4, pp. 318-344 (1986).
16) Foley, J.: Transformations on a Formal Specification of User-Computer Interfaces, *Computer Graphics*, Vol. 21, No. 2, pp. 109-113 (1987).
17) de Baar, D. et al.: Coupling Application Design and User Interface Design, *Proc. ACM Conf. on Computer-Human Interaction*, pp. 259-266 (1992).
18) Wiecha, C. et al.: ITS: A Tool for Rapidly Developing Interactive Applications, *ACM Trans. Information Systems*, Vol. 8, No. 3, pp. 204-236 (1990).
19) Szekely, P. et al.: Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design, *Proc. ACM Conf. on Computer-Human Interaction*, pp. 507-515 (1992).
20) 北村操代，杉本 明：GUI生成・編集機能を持つクラスライブラリGhostHouse，情報処理学会研究報告（ソフトウェア工学），Vol. 92, No. 59, pp. 27-34 (1992).
21) 杉本，北村，中田，川岸，小島：対話型システム視覚的構築用クラスライブラリ：GhostHouse（I），（II），（III），第46回情報処理学会全国大会論文集（分冊6），pp. 269-274 (1993).
22) Goldberg, A.: Information Models, Views, and Controllers, *Dr. Dobb's Journal*, pp. 54-61, 106, 107 (July 1990).
23) Stefik, M. J., Bobrow, D. G. and Kahn, K. M.: Integrating Access-Oriented Programming into a Multiparadigm Environment, *IEEE Software*, Vol. 3, No. 1, pp. 10-18 (1986).
24) Borning, A. H.: *Thinglab — A Constraint-oriented Simulation Laboratory*, Doctorial dissertation, Dept. of Computer Science, Stanford (1979).
25) Cook, W. R.: Interfaces and Specifications for the Smalltalk-80 Collection Classes, *OOPSLA'92*, pp. 1-15 (1992).
26) 杉本，小島，阿部：C++言語のシステム向き拡張ツール：OPTEC，情報処理学会論文誌，Vol. 33, No. 5, pp. 681-690 (1992).
27) Quercia, V. and O'Reilly, T.: *X Window System User's Guide (OSF/Motif 1.2 Edition)*, pp. 331-340, O'Reilly (1993).
28) Szczur, M. R. and Sheppard, S. B.: TAE Plus: Transportable Applications Environment Plus: A User Interface Development Environment, *ACM Trans. Information Systems*, Vol. 11, No. 1, pp. 76-101 (1993).

（平成6年4月 4 日受付）
（平成7年1月12日採録）

北村　操代（正会員）

平成元年大阪大学理学部数学科卒業．同年三菱電機(株)入社．以来同社中央研究所にて，幾何情報システム，オブジェクト指向システムなどの研究に従事．



杉本　　明（正会員）

昭和52年京都大学理学部卒業．昭和54年同大学院工学研究科(数理工学)修士課程修了．同年三菱電機(株)入社．以来同社中央研究所にて，設計支援システム，オブジェクト指向言語，分散システムなどの研究に従事．工学博士．電子情報通信学会，ACM各会員．