ソフトウェア工学 86－5
（1992．7．21）

GUI生成・編集機能を持つクラスライブラリ GhostHouse

北村 操代　杉本 明
三菱電機株式会社 中央研究所

本稿ではGUI生成・編集機能を持つC++のクラスライブラリGhostHouseについて述べる。GhostHouseの設計ではアプリケーションプログラムからデフォルトのGUIを自動生成し、プログラムを実行し動作確認を行いながらGUIを修正していく、という枠組の実現を目指している。そのため、アプリケーション中のオブジェクトに対応したGModelクラスと、GUI編集機能を持つGUI部品であるGViewクラスをアブストラクトクラスとして提供している。本稿ではそれらの部品によるデフォルトGUIの生成や、Drag & Drop操作による、モデルとの結合を維持したままでのビュー部品によるGUI編集機能など、GhostHouseの特長を述べている。

GhostHouse: A Class Library for Generating and Editing Grapghical User Interfaces

Misayo Kitamura and Akira Sugimoto
Mitsubishi Electric Corporation　Central Research Laboratory

This paper describes the C++ class library called GhostHouse. GhostHouse provides two kinds of classes, models and views. Models are used for developing application programs, and views are components of graphical user interface. The novel features of GhostHouse includes the following; (1) models can generate views as a default GUI, and (2) views can be edited for refining the GUI while preserving connections between models and views. This paper also shows an extended drag and drop method for modifying GUI easily at run-time. The method can replace and merge views which are connected to models.

1 はじめに

EWS上の対話型プログラムの作成において、グラフィカルユーザインタフェース(以下、GUIと呼ぶ)構築を容易に行うために、Xツールキット[1]やOSF/Motifのウィジェットセット[2]など、さまざまなツールキットが整備されてきている。しかしツールキットを活用するためには、ツールキットについて熟知しなければならない。このことは一般的なアプリケーションプログラマにとって大きな負担となっている。

このため、ツールキット部品を画面上で描画・構築することによって画面配置部分のプログラムを生成するGUIビルダが開発されている。GUIビルダとして例えばVUIT[3]が挙げられる。GUIビルダによって画面作成は容易となったものの、アプリケーションプログラム(以下、APと呼ぶ)の動きに応じてGUI側の表示や操作の制御を行う場合には、依然としてツールキットに関する知識を必要とする。

GhostHouseの目的は、テキスト端末に対する入出力を用いた対話型プログラムと同等の容易さで、洗練されたGUIを持つAPの作成を可能とすることである。このため、次の様な枠組を採用した。

1. **デフォルトGUIの自動生成**
   AP中に表れるデータや関数から、デフォルトのGUIを自動生成する。
2. **AP実行時におけるGUIの修正**
   AP実行時に動作確認を行いながら、画面上における対話的な修正を通じて、生成されたGUIを最終的なGUIに変形していく。

上記の枠組を実現するため、GhostHouseはC++のクラスライブラリとして次の様な特長を持つ部品を提供している。

1. **GUI自動生成機能を持つモデル部品**
   Smalltalk[4]やObjective-C[5]におけるGUI用クラスライブラリと同様に、GUI部分とAP部分を分離するため、GhostHouseでは部品をモデル部品とビュー部品に分割している。モデル部品はAP内で用いるデータや関数に対応する。画面上に表示されるビュー部品はモデル部品と互いに結合し、モデル部品の内容表示や対話的操作を担当する。GhostHouseの特長の1つは、モデル部品が対応するGUI部品をデフォルトGUIとして自動的に作成することである。
2. **AP実行時のGUI編集機能を持つビュー部品**
   ビュー部品はGUI部品としての通常の機能に加えGUI編集機能を持つ。しかもその編集機能として、APと連結したままのGUI変形に適した操作を提供している。例えば編集時のDrag & Drop機能としてGhostHouseでは置換操作を用意しており、置換されるビュー部品のモデル部品との連結は置換後のビュー部品に引き継がれる。

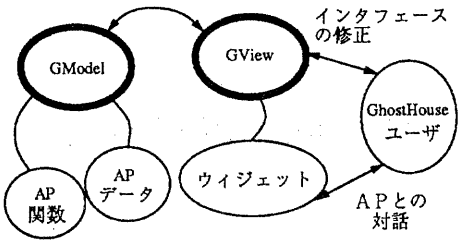

図1: Ghostの役割

第2章以下の構成は次のようになっている。第2章ではGhostHouseクラスライブラリのモデル部品とビュー部品について述べる。第3章では、APからデフォルトGUIを生成するメカニズムを記述している。第4章ではGhostHouseでのGUI編集機能の詳細を述べる。特に、GhostHouseで用いているDrag & Dropによる編集機能についても述べる。第5章では第3章と第4章で述べたアーキテクチャを具体例を通じて説明する。第6章ではAPからGUIを自動生成し、これを編集することによってGUIを作成するという枠組に関して他の研究との比較をしている。第7章で結論を述べている。

2 GhostHouseクラスライブラリ

GhostHouseはC++のクラスライブラリである。本章ではGhostHouseが提供しているモデル部品とビュー部品の概要を述べる。

先に述べたように、一般的なビュー部品としては、既にOSF/Motifなどの GUI部品が広く提供されている。また、AP用のクラスライブラリも分野によっては既に開発されている。GhostHouseのクラス設計においてはこれら既存の部品を最大限に利用できることを考慮した。即ち、クラスライブラリとしては、1章で述べた機能を実現するために既存の部品にとりつき、これらを操作するオブジェクトを用意した。GhostHouseではこれらのオブジェクトはGhostと呼ぶ抽象クラスのサブクラスとして実現されている。

自動生成機能を持つモデル部品はGModelクラスに、GUI編集機能を持つビュー部品はGViewクラスに、

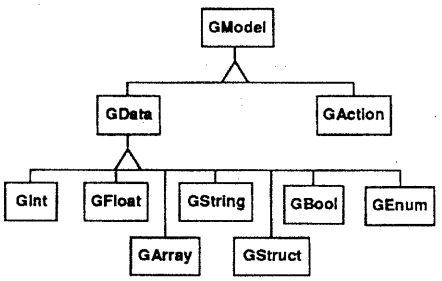

図 2: GModel クラス階層図

共に Ghost のサブクラスとして提供されている。図1は GhostHouse における GModel と GView の役割を示したものである。GModel は AP 内で用いられるデータや関数を操作する。一方、GView は通常の GUI 部品(ここでは OSF/Motif のウィジェット)を操作する。GView に対しての入力操作はインタフェース修正入力となり、GView に備えられた GUI ビルダ機能を使用することができる。また、GView がとりついている GUI 部品に対しての入力によって、AP との対話も可能である。AP に対する対話入力やインタフェース修正入力があった場合、および、AP 内でデータが変更された場合には GModel と GView が互いに通知をし、AP 内のデータや GUI 部品に変更を反映させる。

以下では GModel および GView クラスの詳細と、GModel と GView の対話方法を述べる。

2.1 GModel クラス

GhostHouse では GModel は AP 内のデータと関数にとりつく。これらをそれぞれ GData クラスと GAction クラスとした。図2に GModel クラスのサブクラスのクラス階層を示す。

GData 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 型にとりつき、GFloat クラスは float 型にとりつく。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。C++ を用いて AP 内で定義されたクラスに対しては、GStruct のサブクラスが対応する。一方、GAction は AP 関数にとりつく。GAction に対して実行依頼が発行された時に、GAction がとりついている AP 関数が実行される。

2.2 GView クラス

GhostHouse は X ウィンドゥシステムを対象としており、リソース管理やイベント管理は Xt、標準 GUI 部品には OSF/Motif のウィジェットを利用している。各 GUI 部品にとりつく GView のサブクラスを図3に示す。図3中のイタリック体の記載は、その GView に対応する Motif のウィジェットである。GView のクラス構成はウィジェットの機能に追加する部分に着目して構成したので、Motif のウィジェットのクラス階層とは異なっている。

図3で、GWidget は Xt のウィジェットを操作し、GFigure は図形部品を操作する。Motif などのツールキットでは GUI を作成する際に階層的な構造を用いる。即ち、各々のウィジェットには親ウィジェットが存在する。親ウィジェットは当該ウィジェットを画面上で包含し、その位置や表示状態を管理するウィジェットである。GWidget のサブクラスの中で、子供のビュー部品を持つ複合オブジェクトは、GBox と GClothes に分けられている。GBox は複数の部品の配置のために用いられるが、GClothes はほかの GView に付属しその GView のビューの設定やビューの修飾をするために用いられる。

GView は Motif のウィジェットと1対1対応をするとは限らない。例えば図中の GButton クラスのように、1つの GView に2つ以上のウィジェットを対応させている場合がある。この場合には1つの GView 内で様々な表示形態を取ることができ、この表示形態をスタイルと呼ぶ。また逆に、1つのウィジェットに対して2つ以上の GView クラスが対応することもある。

2.3 GModel と GView のインタフェース

AP 実行時に AP 内のデータを GUI に反映させ、GUI 入力を AP に反映させるために、GModel と GView は互いに変更を通知する。通知を行うためにはモデルとビューの結合が必要となり、GhostHouse では GModel と GView に双方向のリンクを持たせている。

また GhostHouse では二重に値を持たずにすむように、GView はモデルの値を持たない。GView はモデルの値を参照する必要が生じるたびに GModel から値を入手する。そこで、GModel と GView の間の対話プロトコルは以下の様にした。GModel は GView に対して (1) 値の変更通知のみを行う。GView は GModel に対して (1) 値の設定依頼 (2) 値の入手を行う。

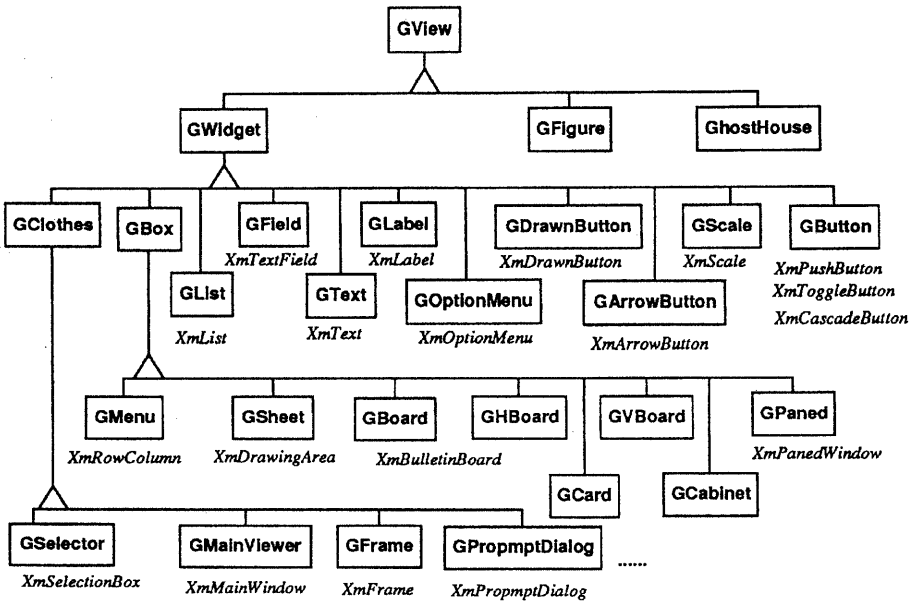

図 3: GView クラス階層図

3 デフォルト GUI の自動生成

GhostHouse はビュー部品を用いて AP を記述することにより、AP 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。この章では 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
   (a) AP 中で定義された GModel の検索
   (b) 発見された GModel に対し、それぞれ対応する GView を生成し結合

第三段階において、もし既に AP を起動して GUI 修正を行い、かつ、その結果がファイルに保存されていれば、自動生成は行われない。保存ファイルが存在しない時にのみ、自動生成が行われる。

表 1: GModel と自動生成される GView の対照表

GModel	C 言語での型	自動生成されるデフォルト GUI 部品
GInt	int	GField
GFloat	float	GField
GString	char[]	GField
GEnum	enum	GOptionMenu
GBoolean		GButton CheckButton スタイル
GArray	[]	GMenu
GStruct	struct	GCard
GAction	関数	GButton

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 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 知識を持たずに行うことができる。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、次の様な処

理が行われる。AP 中で宣言された GModel をすべて GhostHouse 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。GhostHouse が入力イベント待ち状態に入る直前に、GhostHouse 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 と結合する。各 GModel に対応する GView を表 1に示した。さらに、生成された GView は順次 1 つのスクロールバー付き GMenu (RowColumn ウィジェット) 中に配置され、表示される。

以上のようにしてデフォルト GUI が生成される。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。

4 AP 実行時の GUI 編集機能

従来の GUI ビルダは 1 つの編集用ウィンドウを用意し、その中に GUI 編集機能コマンド群や GUI 編集エリアを配置している。しかし、GhostHouse では AP 実行時におけるすばやい 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。

- Drag & Drop による編集
- プロパティボードによるスタイルや属性の設定
- 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、これらの編集操作は Ctrl キーを押しながらのマウスボタン入力により開始される。すなわち、Ctrl 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 Drag & Drop 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。

これらの GUI 編集機能は、アブストラクトクラスの GView クラスに備えられており、全ての GView のサブクラスに継承されている。本章では、AP 実行時における Drag & Drop による編集機能について詳しく述べる。

4.1 Drag & Drop 操作の種類

GhostHouse では GUI 編集操作法に Drag & Drop を採用している。画面上で GView のサブクラスのオブジェクトを選択し、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オブジェクトのドラッグの描画が開始される。そして別のオブジェクト上でマウスボタンを離すと、それに対してオブジェクトがドロップされる。

Drag & Drop 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 における Drag & Drop では従来の機能に加えて、ビュー部品の置換、吸収、メニューやダイアログの設定などの機能も備えている。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サブジェクトと呼び、ドロップの対象となったオブジェクトをターゲットと呼ぶ。

4.1.1 置換と吸収

GhostHouse は AP 実行時に対話的な修正を加えることによって GUI 編集を行うことを目指して設計された。このため、プロパティボードによる AP との結合設定を繰り返す手間を省く為に、置換や吸収という操作を導入した。

置換操作は画面上のビュー部品の種類を変更するために用いる。この時、ターゲットをサブジェクトに取り換えるだけでなく、ターゲットに結合されていた GModel はサブジェクトに引き継いで結合される。現在の画面上に表示されている以外の種類の部品は、GCabinet と呼ばれる部品箱をポップアップメニューにより表示させ、そこからドラッグする。

一方吸収操作は、サブジェクトに結合している GModel をターゲットに付加し、サブジェクトは消去される。この操作は 2 つの部品のモデルとの結合を 1 つの部品に融合するために用いられる。

4.1.2 配置と着用

配置操作では、表示位置や大きさの変更を行うだけでなく、この際にターゲットがサブジェクトの親でなければ、リペアレントも自動的に行われる。即ち、そのターゲットをサブジェクトの新しい親とする。X ツールキットにはリペアレントの機能がなく、リペアレントは GView によって実現されている。

X ウインドウシステムのルートウインドウに対応する部品は GhostHouse クラスである。GhostHouse 上に GView がドロップされると、新しいトップレベルのウインドウがその位置に配置される。

着用操作はターゲットの親とターゲットの間にサブジェクトをはさみこむ。即ち、ターゲットの新しい親がサブジェクトとなり、サブジェクトの親は以前のターゲッ

トの親となる。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インドウ) の中にターゲットを入れることができる。

4.1.3 ダイアログやメニューの設定

設定ではターゲットとサブジェクトとの関係づけを行う。例えば、AP実行時にターゲット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インドウやポップアップメニューとして、サブジェクトが表示されるようにDrag & Dropで設定できる。

Xツールキットではこのような機能はGUI部品間のコールバックとして設定される。そのため、従来のGUIビルダではプロパティボードを用いて設定を行っていたが、GhostHouseではDrag & Drop操作によって容易に設定できる。

4.1.4 ドラッグ時の複写

なお、GhostHouseのDrag & Drop操作には、サブジェクトを複写してドラッグを開始する方法が提供されている。複写操作は、モデルとの結合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1つのデータと結合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。

4.2 Drag & Drop 操作の機構

ドロップするサブジェクトのクラスとターゲットのクラスによって、ドロップによって実行できる操作が限られる。例えば、GBoxは他のGViewを配置することができるが、GButtonには配置できない。また、実行できる操作はクラスのみで決まるわけではない。両方のGViewの状態に依存する。本節ではDrag & Drop操作の機構について述べる。

Drag & Drop操作は二段階で行われる。まず操作の決定を行い、次に調節を行う。第一フェーズの操作の決定では、ドロップしたオブジェクトとドロップされたターゲットの2つのクラスと、それぞれのオブジェクトが結合しているモデルに応じて可能な操作を列挙する。この判断規準の一例は、

- ターゲットがデータが結合していれば、そのデータとの結合が可能であるか、
- ターゲットやサブジェクトがGBoxやGClothesであれば、配置や着用が可能か、
- ターゲットがGButtonであれば、サブジェクトをダイアログとして設定できるか、
- サブジェクトがGMenuであれば、ポップアップメニューとして設定できるか、

などである。このような規準により可能な操作を全ての操作の中から選びだした後、ユーザに対話形式で選択させる。これにより、実際に行われる操作が決定する。可能な操作が存在しなかった場合には、警告が表示される。

次に第二フェーズで調節を行う。この段階では、モデルとビューの結合状態の更新、および、ドロップしたGViewのスタイル (ウィジェット) の変更が行われる。

GModelとGViewの結合状態の更新は、置換や吸収が選択された時に実行される。一方、スタイルの変更はMotifの制限を吸収するために導入した。Motifについて詳しくないユーザは、Motifで設定できないことを試みる場合が多々ある。これに対して警告を与えることは、親切ではなく、本来ユーザがやりたかったことに近いことを行えるべきである、と考えたからである。例えばGButtonをラジオボックスタイプのGMenuにドロップしたとする。Motifの制限によって、ラジオボックス中のウィジェットはトグルボタンでなければならない。そこでスタイルの変更が行われ、GButtonのスタイルがプッシュボタンからトグルボタンに変更される。

5 例

この章では、摂氏の温度を華氏に変換する簡単なAPを対象にしてGhostHouseでのGUI作成手順を述べる。このAPでは、エンドユーザが摂氏の温度を入力し、入力された温度の変換を行い、計算結果を表示する。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。また変換開始のアクションも必要である。

プログラマはGhostHouseを用いて図4左上のようなAPを記述する。記述するAPは大きく3つの部分に分けることができる。1つめはGModelを定義する部分、2つめはAPで用いる関数を実現する部分、3つめはmain関数で、GhostHouseのインスタンスを生成して、そのメンバ関数Runを起動する。この例では第1のGModelの宣言部分で、摂氏と華氏の温度をそれぞれfloatと宣言するかわりにGFloatとして宣言している。また、第2の関数定義部分では、変換関数を定義している。

作成されたプログラムをコンパイルして実行すると、実行開始時にデフォルトGUIが作成される。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。アプリケーションは正常に実行されている。

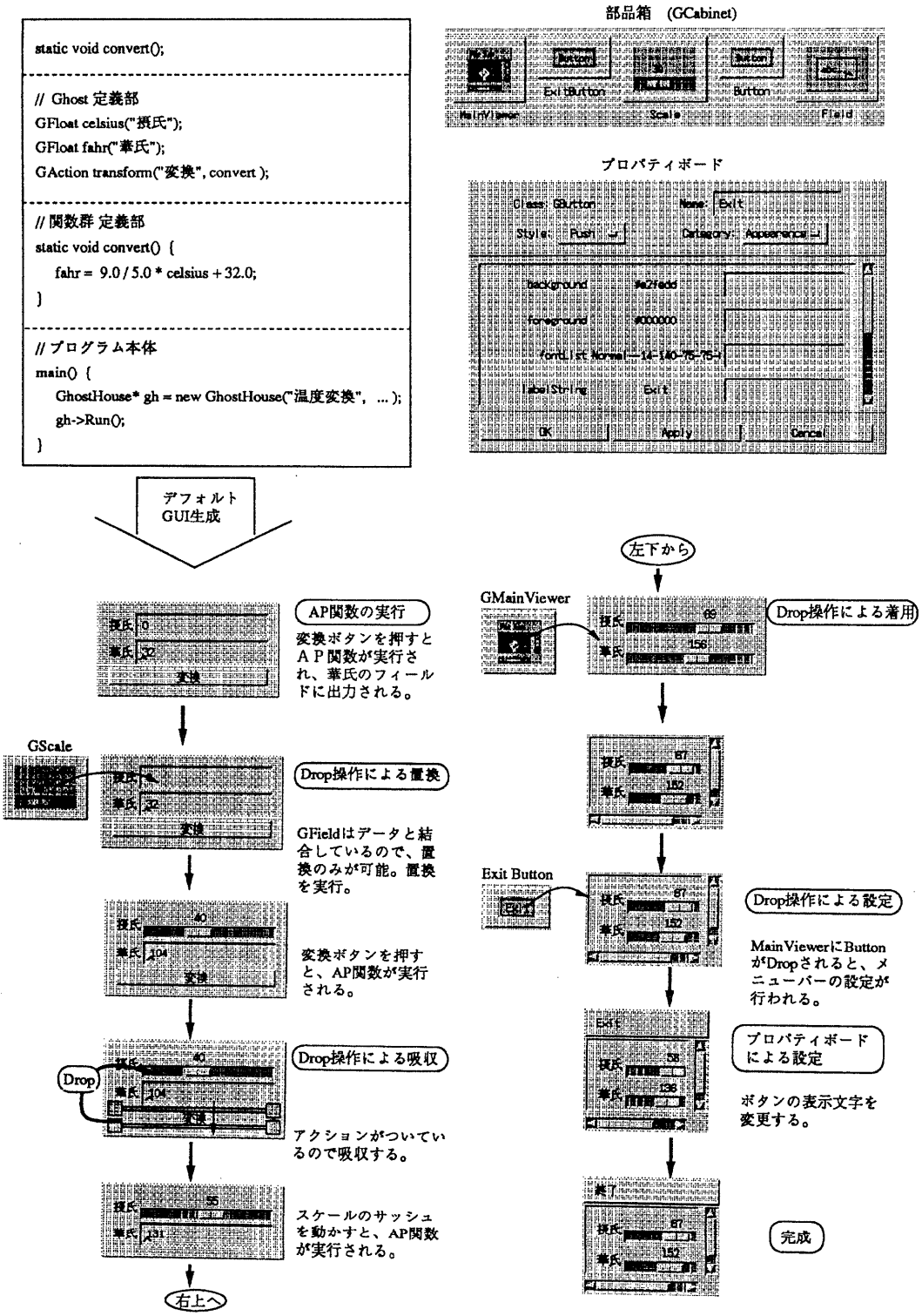

図４： 簡単なAPの作成例

通常はデフォルトGUIは所望のGUIではなく、GUI部品に備えられたビルダ機能により、編集を行う。この様子を図4下に示している。図4では、Drag & Dropによる置換、吸収、メニューバーの設定、着用が行われ、さらに、プロパティボードによる属性設定を行っている。自動生成されなかった各種部品は、部品箱(GCabinet)からドラッグして来ることによって利用できる。

GUIが完成すると、保存する。再びAPのソースを書換えたとしても、データの名前さえ一致していれば、保存ファイルは有効となる。

このようにして図4右下のようなGUIが作成される。GhostHouseでは、APの動作を確かめながら以上の様なGUIの修正を連続的に行い、GUIを作成することができる。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。

6 Related Works

従来のGUIの作成方式として、ツールキットを用いたビルダが挙げられる。ビルダを用いることにより、GUI部品の構成部分のプログラムを作成することができる。この場合でもAPと連結するためにはプログラミングが必要となる。GhostHouseでは、AP実行時にアプリケーションデータとの結合を保ったままGUI編集を行うことができるので、連結のためのプログラミングが一切不要となる。また従来のGUIビルダでは、AP部分を統合化した段階でのインタフェース確認をその場で行うことができない。この確認のためにはGUI部分のソースコードの生成やAP部分との結合といった時間のかかる操作が必要であった。しかしGhostHouseではAP実行時にインタフェースの動的側面の確認、修正もその場で行うことができるので、GUI開発期間の大幅な削減が期待できる。

GhostHouseでは、APからのデフォルトGUIの自動生成を行う。自動生成という点でも、従来から研究がなされている。例えば、MIKE[6]はまずコマンド群を定義し、これに基づきデフォルトのインタフェースを作成する。また、Humanoid[7]ではルールによるアプリケーションからの GUI 自動生成という方法が提案されている。しかしながらこれらの研究では生成されるものがUIMSの入力となるユーザインタフェース仕様であり、生成されたインタフェースの変更はその仕様に対して行わなければならない。これに対してGhostHouseでは、Drag & Drop機能により、APと連結されたまま直接操作によりGUIを変更できる。

7 結論

本稿ではGhostHouseクラスライブラリについて述べた。GhostHouseはAPからデフォルトのGUIをAP実行時に自動生成し、APを実行しながらGUIを編集していく、という枠組に基づいて設計された。その枠組を実現するため、GhostHouseでは、APオブジェクトに対応したGModelクラスと、GUI編集機能を持つGUI部品であるGViewクラスをアブストラクトクラスとして提供している。本稿ではそれらの部品によるデフォルトGUIの生成や、Drag & Drop操作による、APとの結合を維持したままでのビュー部品によるGUI編集機能など、GhostHouseの特長を述べた。

既にクラスライブラリの主な部品の試作は完了している。本稿では述べていないが、図形部品も用意している。今後、帳票用部品やグラフ用部品などのビュークラスを拡充していく予定である。

参考文献

[1] J.McComack et al.: "X Toolkit Intrinsics", X11R4 online documentation

[2] OSF: "OSF/Motif™ Programmer's Guide"

[3] DEC: "DEC VUIT User's Guide", 1991

[4] Adele Goldberg: "Information Models, Views, and Controllers ", Dr.Dobb's Journal, July 1990 pp. 54 – 62, 106, 107

[5] Brad J. Cox: "Object Oriented Programming", Addison Wesley Publishing Company, 1987

[6] Dan R. Olsen Jr.: "MIKE: The Menu Interaction Kontrol Environment", ACM Transactions on Graphics Vol. 5, No, 4, pp. 318 – 344 (1986)

[7] Pedro Szekely: "Templete-Based Mapping of Application Data to Interactive Displays", In Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, pp 1 – 9, October 1990