

2001 年度 修士論文

GUI 部品の変更時のプログラム自動変更支援に  
関する研究

2002 年 2 月 5 日 (火) 提出

指導：深澤良彰教授

早稲田大学 大学院 理工学研究科 情報科学専攻

学籍番号：600P068-3

渡部 宏志

# 目次

<b>1</b>	<b>はじめに</b>	<b>1</b>
<b>2</b>	<b>ユーザビリティの高い GUI 構築と従来開発の問題点</b>	<b>3</b>
2.1	ユーザビリティを高める開発条件	3
2.2	ユーザビリティ向上のためのウィジェットの変更	3
2.3	従来の開発の問題点	4
<b>3</b>	<b>本研究の特徴</b>	<b>5</b>
<b>4</b>	<b>本研究の概要</b>	<b>6</b>
4.1	GUI の役割によるウィジェットの分類	6
4.1.1	GUI の役割	7
4.1.2	役割を果たすための手続き	7
4.2	GUI 構築の流れ	8
4.2.1	タスクの分析	9
4.2.2	分析結果に基づく GUI 構築	10
4.3	ウィジェット変換表	10
<b>5</b>	<b>GUI 変更指針</b>	<b>13</b>
5.1	ウィジェットの持つデータ特性	13
5.2	変換の対象とする変数の型	13
5.3	メソッド変換	14
<b>6</b>	<b>ウィジェット変換表</b>	<b>17</b>
6.1	ウィジェット変換表の XML 記述	17
6.2	ウィジェット変換表記述ツール	18
<b>7</b>	<b>本システムの構成</b>	<b>22</b>
7.1	タスクの分析	23
7.2	GUI 構築	23
7.2.1	ウィジェットの配置	23
7.2.2	役割・役割名の設定	23
7.2.3	GUI 処理部の構築	24
7.2.4	GUI 処理部とアプリケーション処理部の結合	25
7.3	GUI 変更	26

<b>8</b>	<b>評価</b>	<b>29</b>
8.1	本システムによる GUI 構築時の評価 . . . . .	29
8.2	本システムの GUI 変更時の評価 . . . . .	31
<b>9</b>	<b>考察</b>	<b>35</b>
9.1	役割名の設定における優位性 . . . . .	35
9.2	手動変更の要因 . . . . .	35
9.2.1	対象外の手続きを変換する場合 . . . . .	35
9.2.2	機能の追加と削除 . . . . .	36
9.2.3	機能の重複 . . . . .	36
<b>10</b>	<b>関連研究</b>	<b>39</b>
<b>11</b>	<b>おわりに</b>	<b>41</b>

## 1 はじめに

コンピュータが登場してきた当時、ソフトウェアを扱う人達は、ほとんどがコンピュータの専門家だった。しかし近年ではコンピュータの普及により、コンピュータにあまり精通していない多くの人達もソフトウェアを扱うようになってきた [1]。そこで、これらの人々でも扱いやすいように、近年のソフトウェアは、機能 (ユーティリティ) だけではなく、使い勝手 (ユーザビリティ) も重視される傾向になってきている。

これに伴い、ヒューマンインターフェース、認知心理学、人間工学等の分野に長けた人々が GUI(Graphical User Interface) デザイナーとして求められるようになってきた。しかし、一般的に優れた GUI デザイナーと優れたプログラマの素養は異なるため、GUI デザイナーは、紙などでプログラマに自分の意図する GUI を伝えることを余儀なくされている。これでは、GUI デザイナーの意図を正確にプログラマに伝えるために多大な労力が必要となり、また、時間のロスにもつながる。一方、エンドユーザの手によって GUI を試行・評価し、その結果をアプリケーションに反映して改良することも高いユーザビリティを実現する上で効果的である [2]。この際、エンドユーザが求める明確な GUI をエンドユーザ自身が提示する場合は、プログラマの手によって提示された GUI を構築すればよいが、実際のアプリケーション使用に近い環境下で再構築した GUI を通して操作を行ってみるまで、その GUI が本当にユーザビリティの高いものであるかは分からない。また、エンドユーザが明確な GUI を提示することができないため、エンドユーザの評価を元に GUI デザイナーやプログラマが GUI を再構成し、エンドユーザに再評価してもらうという過程を繰り返す場合も数多くある。そこで、ユーザビリティを高めるためには、変更を前提として GUI を構築することが重要となる。

また GUI は、ユーザビリティ上の問題からアプリケーションの開発段階での変更はもちろん、実際の運用段階でも、変更の要求が生じる場合がある。ここでは、開発段階、あるいは、運用されているアプリケーションの GUI を変更する必要がある。

以上より、ユーザビリティの向上には GUI の変更容易性が重要となる。本研究では、この変更容易性を実現することを目指し、GUI アプリケーションを生成し、その GUI を変更する手法について提案する。具体的には、GUI が果たす役割ごとにウィジェットを分類し、共通の役割を持つウィジェット同士の振舞いを考慮し、メソッド等を自動的に変換することで、GUI 変更時のプログラムの自動修正を行う。

本研究では、Java 言語を対象としている。例として挙げられるウィジェットは、Java の Swing パッケージの GUI 部品である。

本論文の構成を述べる。第 2 章では、ユーザビリティの高い GUI の構築について触れ、それに対する従来手法の問題点を明らかにする。第 3 章は、本研究がどのような特徴を持っており、第 2 章で述べた問題点に対してどのような解決を行っているかを述べる。第 4 章では、本研究の概要について説明し、第 5 章、6 章で本手法によるウィジェット変更のアーキテクチャを述べる。第 7 章では本システムによる GUI 構築とウィジェットの変更を解説し、第 8 章で従来の 2 つの

手法と本手法による変更結果を比較し、第 9 章で考察を行い、第 10 章で関連研究について問題点を説明し、第 11 章で今後の課題を述べる。

## 2 ユーザビリティの高い GUI 構築と従来開発の問題点

本研究の特徴を述べる前に、ユーザビリティの高い GUI を構築するための条件を挙げ、従来の GUI 開発と比較して、その問題点を挙げる。

### 2.1 ユーザビリティを高める開発条件

ユーザビリティを高めるためには、GUI の専門家である GUI デザイナが GUI をデザインして、そのデザインを元にプログラマがプログラミングをすることが重要である。しかし、実際のエンドユーザが望む GUI と GUI デザイナが考える GUI が一致するとは限らないため、エンドユーザによるテストを行い、その評価を繁殖して GUI を再構築することも必要となる。また、アプリケーションが運用されている間も、そこから得られるユーザからの評価をアプリケーションに反映させることができるならば、よりユーザビリティの高い GUI を構築することが可能である。

### 2.2 ユーザビリティ向上のためのウィジェットの変更

ユーザのテスト結果を反映して GUI を変更する場合、レイアウトの変更以外にもウィジェットの変更が考えられる。例えば JComboBox に設定する項目の適切な数や、JLabel に設定する文字列の適切な長さのような、ウィジェットの使用上の指針は存在しない。従って実際のアプリケーションを使う状況やユーザの好みなどにより、GUI デザイナが適切であると考えられるウィジェットをユーザによる評価次第で変更する可能性は十分考えられる。

図 1(a) の例は、JComboBox により削除するファイル名を選択して、JButton の「OK」を押すと選択されたファイル名のファイルが削除される GUI である。図 1(b) は、与えられた項目名(ファイル 1 ~ 4) をファイルイメージと共に表示し、それらをごみ箱イメージを表示した部分にドラッグ&ドロップした瞬間にアクション要求(選択されたファイルを削除する)を発生し、その項目名(選択されたファイル名)を取得する、DnDSelect というウィジェットを用いた GUI である。対象となるファイル数が多い場合には (a) の GUI の方が適しているが、少数の場合には、(b) の方が一覧性の点で優れている。

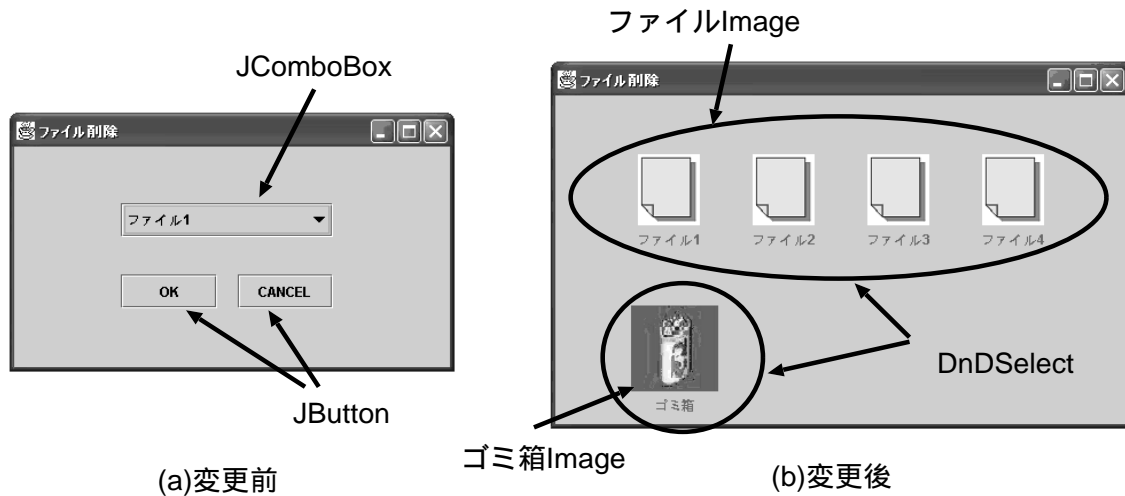


図 1: GUI 変更例

### 2.3 従来の開発の問題点

一方、GUIの変更が必要となるのは、開発中のアプリケーションのエンドユーザの試行結果による評価を反映する場合や、アプリケーション運用中での、使い勝手の悪さ・ユーザの嗜好の変化等の場合である。しかしこれらの状況でのアプリケーションでは、GUI部分を変更するとそのGUI部に関連したアプリケーション処理部分との結合等も変更しなければならない。この際、複雑な構造のGUIを持つアプリケーションから変更箇所を見つけるのは困難であり、バグが生じないように正しく修正するように注意しなければならない。また簡単な構造のGUIであっても、純粋に修正箇所が多いと、変更箇所の発見・修正に時間がかかるだけでなく、バグも生じやすくなり、それらのバグの修正もさらに困難になる。プログラムの知識をあまり持たないGUIデザイナーがこれらの修正をするのは難しいので、プログラマが作業することになるが、再びGUIデザイナーの意図するGUIをプログラマに正確に伝えなければならなくなり、ウィジェットの変更ごとに多大な労力と時間のロスを生むことになる。

以上のように従来の開発では、ユーザビリティ向上のためのGUIの変更の際に多くの問題があり、GUI変更を支援する手法が必要である。

### 3 本研究の特徴

本研究ではユーザの評価を反映させた GUI への変更を簡単にするために、GUI 処理部とアプリケーション処理部との結合を保ったまま、ウィジェットの変更による GUI の変更支援を行う。本研究の大きな特徴として、以下の 3 つが挙げられる。

1. GUI 変更に伴うプログラムの自動修正
2. GUI デザイナやエンドユーザの手による GUI 構築
3. ライブラリに用意されていない新しい変更対象となるウィジェットの追加

本研究では、1. により GUI 変更に伴う修正箇所の発見のための労力やプログラム修正時に生じる恐れのあるバグの大部分を無くしている。また、2. により、プログラムの知識をあまり持たない GUI デザイナやエンドユーザなど、GUI を構築するのに適していると思われる人が直接 GUI を構築することができるため、ウィジェットの変更ごとに GUI デザイナやエンドユーザの意図をプログラマに正確に伝えるための多大な労力と時間のロスが減らしている。そして、ウィジェットは同じ機能を持つものであっても見た目や操作感に合わせていろいろな種類のウィジェットが用意されている。それらのウィジェットの組み合わせにより変更できる種類は増えるので、3. により、様々なウィジェットを GUI 変更の対象とすることで、ユーザビリティの向上を図ることができる。

本研究では、簡単に GUI を構築するためにシステムを構築した。本システムを使うことによって、GUI のレイアウトを RAD ツールのように生成できるだけでなく、GUI 部分とアプリケーション処理部の結合を手動で書き換えたり、連結しなおすことなく、簡単な操作でウィジェットを交換することができる。また、一連のタスクを実行するための GUI があった場合、それらのウィジェットをコピーして、他のウィジェットに交換することで、ユーザの多様なタスク実行操作方法を追加のプログラミングをすることなく実現できる。本研究ではタスクを、アプリケーションが実行する、一連の処理単位と定義する。例えば、図 1 のように、ファイルを選択して削除を実行する、というものである。



#### 4.1.1 GUIの役割

ウィジェットの役割は、

- 表示
- 選択
- 入力（選択による入力）
- アクション要求
- コンテナ

という5つに大きく分類することができる。「表示」は、アプリケーション処理部での処理結果などのデータをユーザに対して表示する役割である。「選択」は、ユーザに提示した条件をユーザが選択したかどうかをアプリケーション処理部に伝える役割である。「入力（選択による入力）」は、アプリケーション処理部にユーザからの入力（選択による入力）を伝える役割である。「アクション要求」は、ユーザの操作により、何らかの処理を行うよう要求を発する役割であり、「コンテナ」は、GUI部品を配置する敷物の役割である。これらの役割を果たすために、ウィジェットには共通の処理が存在する。例えば、「入力（選択による入力）」ならば、「入力されているデータを取得する」という処理により、ユーザの入力した値が、入力値として取得されることになる。

これらの役割に分類される具体的なウィジェットの例としては、「表示」、「選択」、「入力（選択による入力）」、「アクション要求」、「コンテナ」の役割に対してそれぞれ、JLabel、JRadioButton、JTextField（JComboBox）、JButton、JPanel等がある。

#### 4.1.2 役割を果たすための手続き

GUIとしての機能を果たすために、ウィジェットには手続きが多数用意されている。本研究ではこれらの手続きのうち、5つの役割（入力（選択による入力）、選択、表示、アクション、コンテナ）を果たすために用意されたもののみを扱うこととする。その結果、5つの役割を果たすために必要になる処理は、以下の通りである。

- 表示
  - 表示するデータをセットする
  - 表示するデータを削除する
  - 表示しているデータを取得する

- 選択
  - 選択用のデータをセットする
  - 選択用のデータを削除する
  - 選択用のデータを取得する
  - 選択されているかどうかを取得する
- 入力（選択による入力）
  - 入力（選択）されたデータを取得する
  - 入力されたデータを削除する
  - (選択用のデータをセットする)
- アクション要求
  - アクションに反応して呼び出されるメソッドを持つオブジェクトを登録する
  - アクションに反応して呼び出されるメソッドを持つオブジェクトを登録から削除する
- コンテナ
  - GUI 部品を追加する
  - GUI 部品を削除する

本研究では、これらの処理に対応するウィジェットのメソッドをあらかじめウィジェット変換表に記述しておく。例えば、「表示」の役割を持つ JLabel ならば、「表示するデータをセットする」に対応するメソッドは、「void setText()」となる。

## 4.2 GUI 構築の流れ

本研究では、最初にデザイナーとプログラマーがシナリオ等を参考にして、タスクの分析を行う。次に、本システムを使ってデザイナーが GUI を構築し、GUI としての動作を抽象的な手続きとして組み合わせることで実現する。プログラマーは、GUI とアプリケーション本体の結合を行う。その後の GUI 変更は、デザイナーの担当となり、プログラマーは修正を行う必要はない。

以上により、デザイナーが意図する GUI をプログラマーに伝える労力は減少する。また、ユーザの評価の際にユーザが望む明確な GUI が提示されない場合でも、デザイナーによって GUI を再構築するだけでよい。

GUI のプログラミングが必要となる状況としては、以下の 3 つが挙げられる。

### 1. 初期設定

## 2. ユーザによるイベント発生

## 3. アプリケーション処理部による GUI の構築操作

これらのうち、1. の GUI は、タスクの分析によって必要とされるウィジェットを本システムを使用して RAD ツールのように配置し、初期設定として必要となるデータも本システム上で設定することとする。2.、3. の時に新しいウィンドウを生成して表示をする場合などは、本システムの別ウィンドウ上で GUI の構築を行い、イベント発生時やアプリケーション処理部による GUI 構築操作の際に呼出されるメソッド中で、その構築した GUI を生成することを本システムに通知することで、GUI のプログラミングを行う。

### 4.2.1 タスクの分析

本研究では、GUI 構築の観点からタスクの分析を行う。シナリオ等から GUI が必要となるタスクを調べ、GUI 処理部とアプリケーション処理部に共通に使用されるデータをピックアップする。次に、それらのデータがどのような意味を持つものなのかを分析する。その後、GUI 処理部とアプリケーション処理部に分かれて、それぞれの観点から分析を進める。アプリケーション処理部は、どのようなデータを利用して処理を行うかということと、その際に呼び出すメソッドを決定する。GUI 処理部は、GUI 処理部に必要なデータに対して、ユーザがどのように関わるかによって、必要となる GUI の役割を決め、その GUI の役割を表す抽象的な役割名を付ける。また、それらの役割を果たすためにどのような手続きが必要となるかも同時に決める。タスクの分析によって必要となる GUI としての機能が決定したら、次にデータに対して決められた GUI の役割ごとにウィジェットを割当てる。

例として時計の時刻設定のタスクを挙げて説明する。まず、このタスクを詳しく記述すると、「ユーザによる時刻の入力値を時刻設定の処理を行う手続きに引数として渡す」という具合になる。次に、このタスクの中で GUI 処理部とアプリケーション処理部に共通に使用されるデータをピックアップすると、「設定するための時刻の入力値」が挙げられるので、それらのデータを分析し、時刻の入力値として、時 (24 時) ・分 (60 分) ・秒 (60 秒) の 3 つのデータとすることを決定する。プログラマは、これら 3 つのデータを考慮に入れ、アプリケーションの処理として呼出すメソッドを、「setTime(int hour,int minute,int second)」という具合に決め、実装を行う。一方 GUI 処理部は、時・分・秒の 3 つのデータをユーザが入力することになるので、入力のためのウィジェットが 3 つと、ユーザによる決定でこのタスクを実行するので、アクション要求のウィジェットが 1 つ必要となる。そこで、デザイナーはそれぞれのウィジェットに対して、「時の入力」、「分の入力」、「秒の入力」と、「時刻設定アクション」という役割名を設定する。また、タスクを処理するためには、それぞれの入力ウィジェットに対して、「入力値を取得する」という操作が必要となることがわかる。

#### 4.2.2 分析結果に基づく GUI 構築

タスクの分析に基づき、デザイナーは本システムによってウィジェットを配置し、プログラムを記述する代わりに、本システムを通して抽象的な役割名とその手続きという形でプログラムを構築する。その後、プログラマがアプリケーション処理部のメソッド呼出しや GUI 処理部との結合をプログラミングする。ウィジェットを変更する場合、デザイナーが本システムを通して変更するウィジェットの役割名を指定し、変更後のウィジェットに役割名を付け替えることで実現する（図 3）。

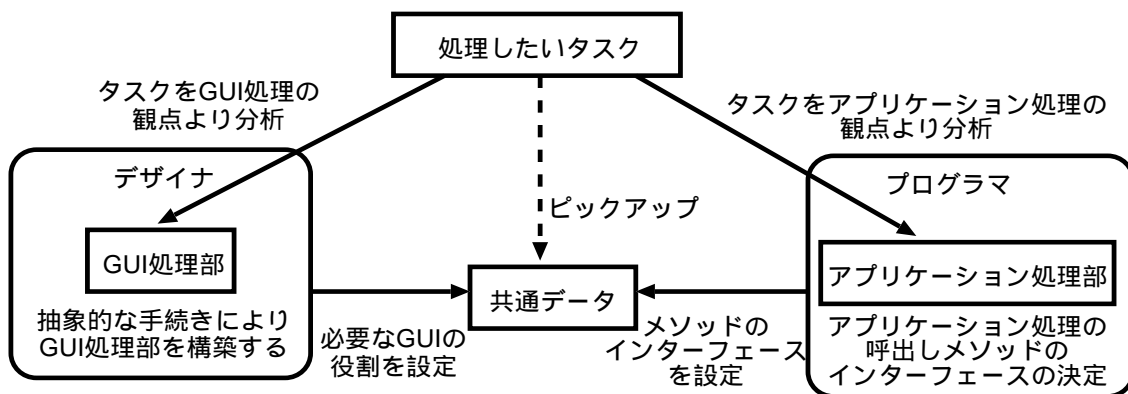


図 3: タスク分析の流れ

時計の時刻設定タスクの例では、タスクの分析に従って、デザイナーは入力用のウィジェット A を 3 つ、アクション用のウィジェット B を 1 つ配置し、それぞれに役割と役割名を与えることで必要となる GUI を構築する。そして、タスクを実行するために必要となる手続きを〈秒の入力.入力値を取得する〉といった具合に抽象的な手続き名で構築する。プログラマは、〈秒の入力.入力値を取得する〉といった GUI 処理部の手続きを通常の方法呼び出しと同等に扱い、この手続きの戻り値を受け取り、アプリケーション処理部のメソッド呼び出しの際にこの戻り値を引数として与えることで、アプリケーション処理部の手続きを結びつける。その後、「時の入力」に割り当てられたウィジェット A をウィジェット C に変更する場合は、デザイナーがウィジェット C を配置し、役割と役割名をウィジェット A からウィジェット C に付け替えることで実現する。

#### 4.3 ウィジェット変換表

ウィジェット変換表とは、GUI としての役割ごとにウィジェットを分類した結果を記述したものである。GUI の機能を果たす際に必要となる手続きに対応するウィジェットのメソッドと、それらのメソッドのインターフェース（引数の数、引数の型、戻り値の型）も併せて記述してあ

る。本研究では、ウィジェット変換表を用いることによって、役割名とその抽象的な手続きとして構築してある GUI 処理部をプログラムに変換する。図 4は、役割名 1 と役割名 2 にクラス 1 のウィジェットを割り当てている場合の変換である。

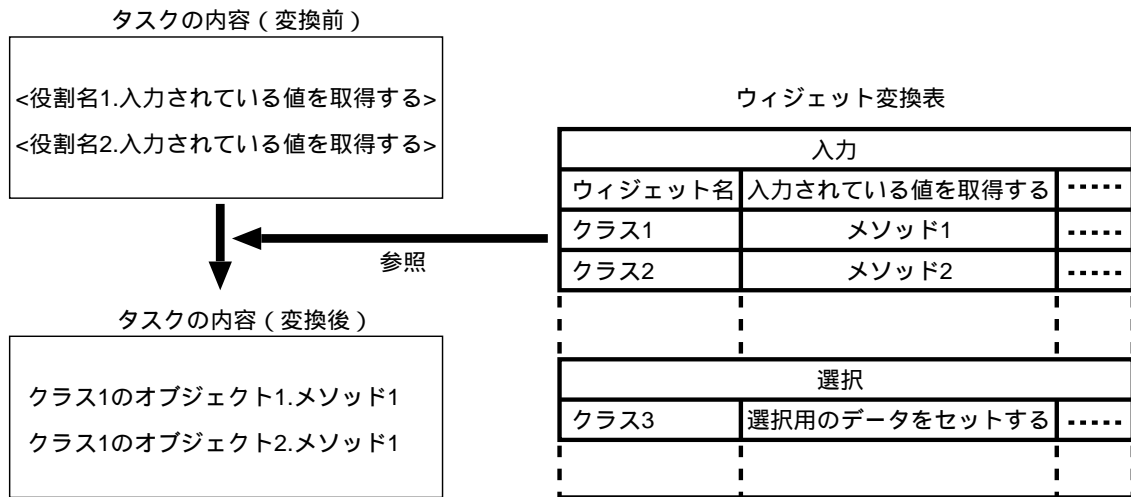


図 4: ウィジェット変換表による変換

従って、本システムにウィジェットの交換を通知し、ウィジェット変換表を参照することで、GUI を構成するウィジェットとそのメソッドを間接的に別のウィジェットとメソッドに交換することとなる。図 5は、役割名 1 を、クラス 1 のウィジェットからクラス 2 のウィジェットに変更した場合の変換である。

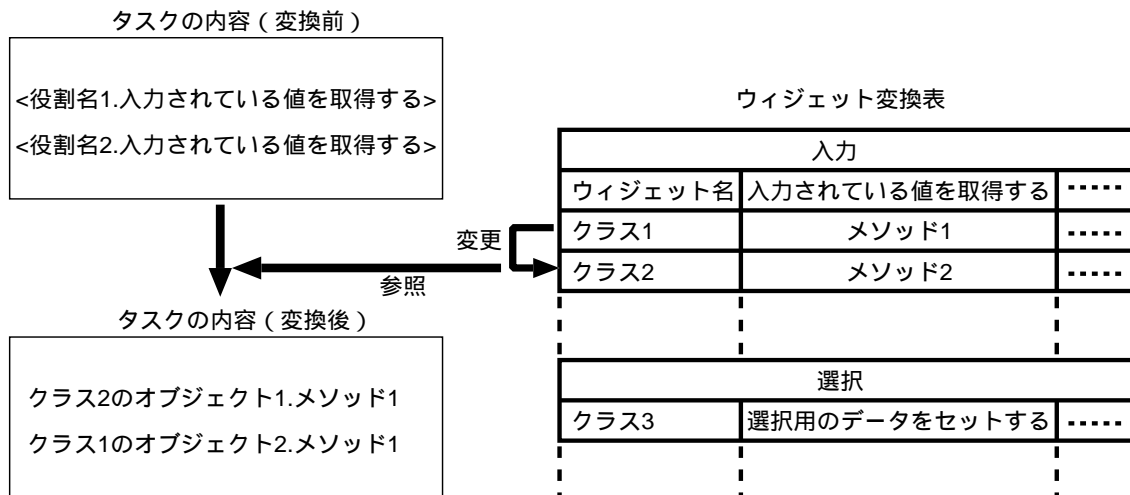


図 5: ウィジェット変更後のウィジェット変換表による変換

ウィジェット変換表は、事前に役割ごとに分類したウィジェットとそのメソッド名を、XML形式のファイルを読み込ませることで設定する。ライブラリに用意されているウィジェットは、あらかじめウィジェット変換表に登録済みであるため、プログラマが登録する必要はない。ライブラリに用意されていないウィジェットを本システムで扱いたい場合、プログラマはウィジェットの仕様に従って分類を行い、ウィジェット変換表に追加することで簡単に実現することができる。

## 5 GUI 変更指針

ウィジェット変換表を用いて抽象的な手続きを割当てられたウィジェットのメソッドに変換する場合、ウィジェットの仕様によっては通常では交換できない場合がある。メソッドの自動交換を行う場合、交換するウィジェットのメソッド同士の引数の型と数、戻り値の型が違つと、適切な形に変更しなければならない。本研究では、ウィジェットが役割を果たすために持つデータに着目し、ウィジェット交換時に生じるメソッド間の変更を行う。

### 5.1 ウィジェットの持つデータ特性

ウィジェットは、役割を果たすためにそれぞれデータを持っている。ウィジェットが GUI の役割を果たすための処理は、これらのデータにアクセスすることで実現できる。これらのアクセス対象となるデータは、単一データの場合と配列データの場合に大別できる。本研究では、前者を単一データウィジェット、後者を配列データウィジェットと呼ぶ。

例えば、JLabel は「表示」のためのデータを 1 つ持っており、これにアクセスするメソッドは、「表示するデータをセットする」に対応する「void setText(String text)」、「表示しているデータを取得する」に対応する「String getText()」などがある。一方、JList は JLabel と同様に「表示」としての役割を果たすことができるが、JLabel と違い、「表示」の役割を果たすためのデータを配列の形で持っている。従つて、これらのデータにアクセスするメソッドは、「表示するデータをセットする」に対応する「void setListData(Object[] listData)」、「表示しているデータを取得する」に対応する

「Object getModel().getElementAt(int index)」のように、配列としてのデータにアクセスするために、複数のデータを一度に設定したり、データの順番を指定して操作するものになる。

本研究では、この 2 つのデータウィジェットのどちらかのウィジェットと、これらの組合せのウィジェットを対象とする。この 2 つのデータウィジェット以外のデータウィジェットで構成されるウィジェット（複数のデータに 1 つのアクセスメソッドしか対応していない場合）の自動変換はユーザに注意を喚起するだけで特別なサポートはしないものとする。これは、本研究が対象としている Java の Swing に属するウィジェットの大部分が両者のデータウィジェットに分類できるためである。また、新規に追加するウィジェットとして、JavaBeans が利用される可能性が高いが、こちらのアクセスメソッドに対する構成規則も該当するためである。

### 5.2 変換の対象とする変数の型

本研究では、本研究が対象としている Java の Swing パッケージのウィジェットの分析と、ウィジェットの汎用性の観点から、対象言語である Java の基本データ型 (byte,short,int,long,float,double,char,b) 文字列型 (String)、オブジェクト型 (Object) とそれらの配列型、可変長配列型 (Vector) を変換の際に扱う型とする。

### 5.3 メソッド変換

ウィジェットを交換する場合、自動変更する必要があるプログラム部分は、ウィジェットの生成と役割を果たすための手続き呼出しになる。ウィジェットの生成部分は、交換するウィジェットのクラス名が分かれば自動的に変更することができるので、メソッドの変更方法について述べる。

メソッドの自動変換を行う際には、メソッドの「引数の数」、「引数の型」、「戻り値の型」を変更する必要がある。本研究では、単一データウィジェット間や配列データウィジェット間の変更であれば、引数の数の違いに関しては、役割を果たすためにウィジェットが持っているデータに対するアクセス方法は類似しているものと仮定する。引数の型に関しては、対象としているプログラミング言語で用意されている変換用のクラスを使うことで、自動的に変更を行う。

次に、単一データウィジェットと配列データウィジェットの間で変更する場合の、同じデータウィジェット間での変更との相違点とその自動変換について述べる。単一データウィジェットと配列データウィジェットを交換する場合、基本的には N:1 の交換となる。つまり、ある役割の単一データウィジェット N 個に対し、同じ役割の配列データウィジェット 1 個が対応するので、それぞれの数が違うことになる。

そこで本研究では、配列データウィジェットと同様に、複数の単一データウィジェットをグループ化することで一連のデータを 1 つのメソッドで扱えるように単一データのメソッドを変換する。また、本システムでウィジェットを選択し、グループ化を指定する時に、グループ化された単一データウィジェットが持つそれぞれのデータに配列のように番号を与えることで、配列データウィジェットと交換する際に本システムで自動的に判別できるようにしている (図 6)。

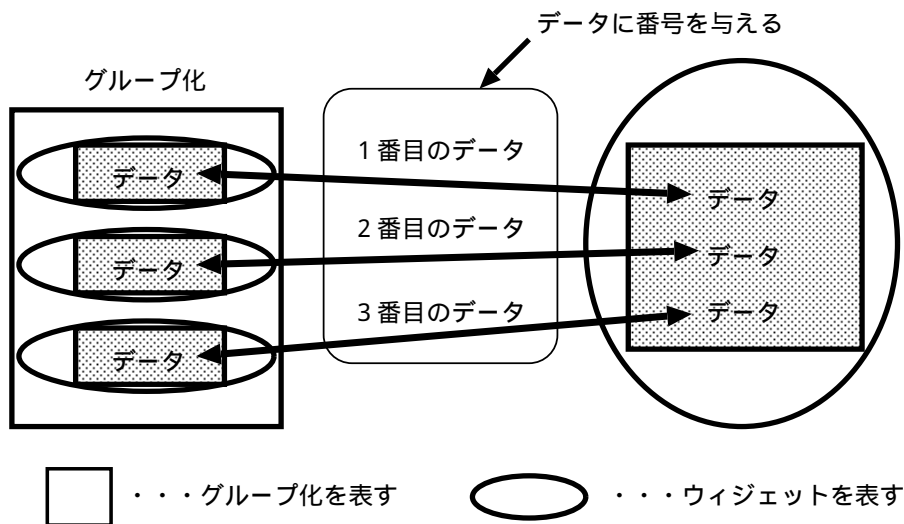


図 6: グループ化とデータの対応

配列データウィジェットや、単一データウィジェットをグループ化して扱うことで、本システムでは、以下の手続きもそれぞれの役割を果たすものとして扱うこととする。

- 表示
  - 指定された表示データをセットする
  - 指定された表示データを削除する
  - 指定された表示データを取得する
  - 全ての表示データをセットする
  - 全ての表示データを削除する
  - 全ての表示データを取得する
- 選択
  - 指定された項目の選択用データをセットする
  - 指定された項目の選択用のデータを削除する
  - 指定された項目の選択用のデータを取得する
  - 指定された項目の選択状況を返す
  - 選択されているデータを取得する
  - 全ての項目の選択用データをセットする
  - 全ての項目の選択用のデータを削除する
  - 全ての項目の選択用のデータを取得する
  - 選択されている全てのデータを取得する
- 入力（選択による入力）
  - 指定された項目の入力されているデータを取得する
  - 指定された項目の入力されているデータを削除する
  - (指定された項目の選択用のデータをセットする)
  - 全ての項目の入力されているデータを取得する
  - 全ての項目の入力されているデータを削除する
  - (全ての項目の選択用のデータをセットする)
- アクション要求

- アクションに反応して呼び出されるメソッドを持つオブジェクトを登録する
- アクションに反応して呼び出されるメソッドを持つオブジェクトを登録から削除する

- コンテナ

- GUI 部品を追加する
- GUI 部品を削除する

ここでは例として、「選択」や「入力（選択による入力）」の役割を持つウィジェット同士 (JRadioButton と JList) の変更を例として述べる。

「選択」や「入力（選択による入力）」の役割を果たすためには、「選択用のデータをセットする」や「選択されたデータを取得する」という手続きがある。JRadioButton の場合、単一データウィジェットなので、「選択用のデータをセットする」に対応するメソッドも

「void setText(String text)」となり、引数は設定したいデータそのものを与えるという形になる。そのため、複数の選択項目を持つ GUI として単一データウィジェットを使うと、選択項目の数だけウィジェットを生成・設定する必要が生じる。また、「選択されたデータを取得する」は、単一データウィジェットの場合、対応する 1 つのメソッドが用意されていることは稀であると考えられるので、通常は、「その項目を持つウィジェットが選択されている」ならば「セットされているデータを取得する」という 2 つの手続きを使わなければならない。

一方、配列データウィジェットである JList の場合、「選択用のデータをセットする」に対応するメソッドは、「void setListData(Object[] listData)」となり、引数が複数のデータを表す配列型となる。そのため、複数の選択項目を持つ GUI として配列データウィジェットを使う場合、生成・設定も 1 つのメソッドでできる。また、「選択されたデータを取得する」という手続きに対するメソッドも用意されており、ユーザがウィジェットを選択状態にしていると、その選択状態にされた項目を返すことができる。JList の場合、

「Object[] getSelectedValues()」が対応するメソッドになる。

## 6 ウィジェット変換表

本研究では、GUI 変更指針を元に、ウィジェット変換表に必要となるウィジェットの情報を記述し、これを元にウィジェットの変更に伴う修正部分を自動的に変更する。変換表に追加するためのウィジェットの情報は XML 形式で記述されており、XML で定義したタグにより、ウィジェットの役割とそれを果たす手続きに該当するメソッド、単一データウィジェットか配列データウィジェットかを分類している。以下に XML タグと分類の例を示す。

### 6.1 ウィジェット変換表の XML 記述

XML タグとしては、1つのウィジェットの情報を表す  $\langle Widget \rangle$  があり、その中にウィジェットのクラス名である  $\langle Name \rangle$  と役割とその振る舞いを表す  $\langle Behavior \rangle$  が記述されている。 $\langle Behavior \rangle$  タグの中は、表示・入力・選択・アクション要求・コンテナの役割を表す  $\langle View \rangle$ 、 $\langle Input \rangle$ 、 $\langle Select \rangle$ 、 $\langle Action \rangle$ 、 $\langle Container \rangle$  のうち、そのウィジェットが果たすことができる役割のタグで構成されている。これらの役割を果たすタグは、属性「plurality」を持っており、「plurality = "true"」なら配列データウィジェット、「plurality = "false"」なら単一データウィジェットを表す。各役割のタグの中には、役割を果たすための手続きを表すタグが含まれており、入力の役割を果たすための手続きである、「入力されているデータを取得する」の場合、 $\langle get\_data \rangle$  というタグが対応する。これらの手続きを表すタグはメソッド名の他に、引数を表す  $\langle argument \rangle$ 、返り値の型を表す  $\langle return \rangle$  で構成されている。 $\langle argument \rangle$  は、引数の順番と型を属性として持ち、 $\langle argumentno = "1" type = "String" \rangle$  のように記述する。これらの手続きの他に、 $\langle Action \rangle$  のタグ中にはユーザのイベントが発生した時に呼び出されるリスナと呼ばれるクラスと、実際に呼び出されるそのリスナの持っているメソッドをそれぞれ、 $\langle listener \rangle$ 、 $\langle action\_method \rangle$  というタグで記述している。また、 $\langle Container \rangle$  のタグ中では、コンテナの中に追加することのできるウィジェットのクラスを  $\langle component\_type \rangle$  タグで定義している。図 7は、JRadioButton というウィジェットを「選択」の役割を持つものとしてウィジェット変換表に登録するための XML 記述の例である。

```
<Widget>
<Name>javax.swing.JRadioButton</Name>
<Behavior>
<Select plurality="false">
<!-- public void setText(String t) -->
<set_data>setText
<argument no="1" type="java.lang.String" />
<return>void</return>
</set_data>
<!-- public void setText(new String("")) -->
<delete_data>setText
<argument no="1" type="java.lang.String">new String("")</argument>
<return>void</return>
</delete_data>
<!-- public String getText() -->
<get_data>getText
<argument no="1" type="" />
<return>String</return>
</get_data>
<!-- public String getText() -->
<get_selected_data>getText
<argument no="1" type="" />
<return>String</return>
</get_selected_data>
</Select>
</Behavior>
</Widget>
```

図 7: ウィジェット変換表に追加するための XML 記述例

## 6.2 ウィジェット変換表記述ツール

本研究では、ウィジェット変更表へウィジェットを追加する際の XML 記述を簡単に行うためのツールを提供する(図8)。

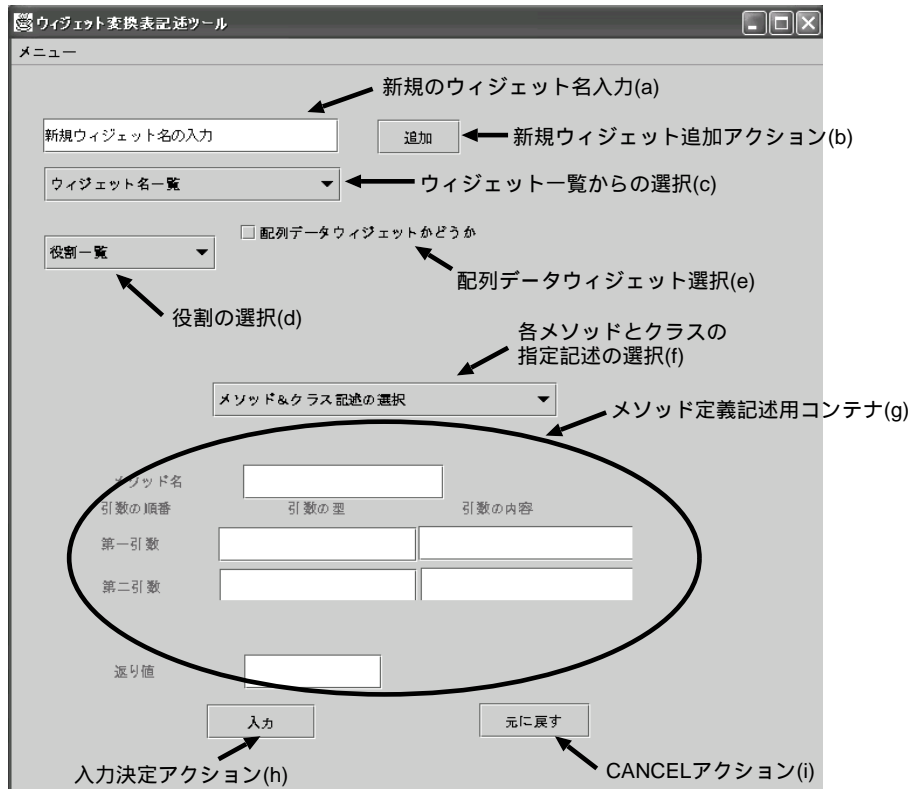


図 8: ウィジェット変換表記ツール

以下、簡単な使用法を説明する。新規のウィジェットを変換表に追加したい場合、図の (a) よりウィジェット名を記入し、(b) の追加ボタンにより (c) の一覧に追加する。新規ウィジェットの追加を続ける場合、(d) で役割を選択すると、それにあわせて (f) や (g) が適切な形式に変わる (図 9)。 (c) のウィジェット一覧から選択したウィジェットが変換表に記述されているものであれば、(d) ~ (g) にそのウィジェットの情報が反映され、必要に応じて修正が可能である。

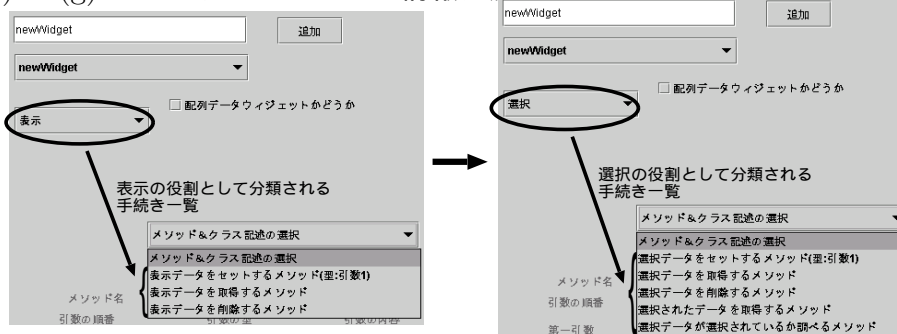
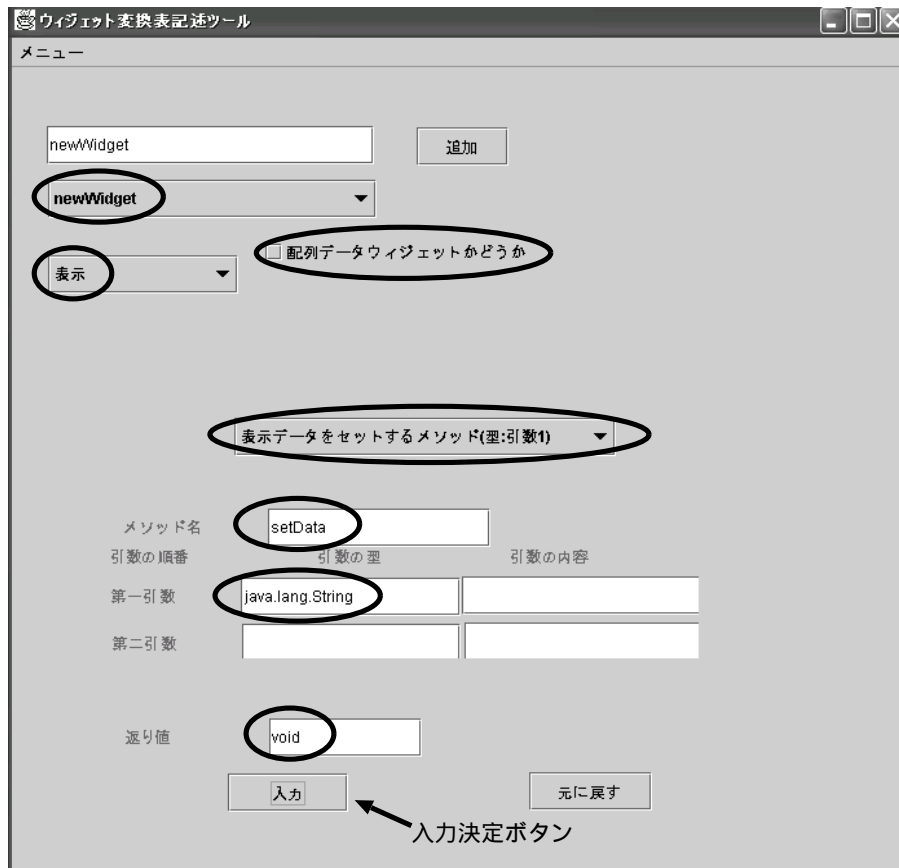


図 9: 追加項目に合わせた GUI の変化

例として newWidget というクラス名を持つウィジェットを分類する。newWidget は単一データウィジェットで、「表示」の役割を持ち、「表示データをセットするメソッド」に対して、「void setData()」というメソッドが対応する。以上のウィジェットに対する情報を図 8(c),(d),(e),(f),(g) に対して、それぞれ入力又は選択する。図 10 のように入力されると、結果の XML として図 11 が出力される。このような XML を本システムに読み込ませ、クラスファイルの指定により newWidget クラスをインポートすることで、newWidget を本システムが変換対象とするウィジェットとして登録する。



○ ...ウィジェットの情報。入力または選択により記述

図 10: ツール記述例

```
<Widget>
<Name>newWidget</Name>
<Behavior>
<View plurality="false">
<!-- public void setData(String t) -->
<set_data>setData
<argument no="1" type="java.lang.String" />
<return>void</return>
</set_data>
</View>
</Behavior>
</Widget>
```

図 11: newWidget の XML 出力結果

## 7 本システムの構成

本システムの構成を図 12 に示す。以下においては、ファイルを選択して削除する例 (図 1) を用いて GUI の構築からウィジェット変更までの手順は以下の通りである。

1. タスクの分析
2. GUI 構築
  - 2.2 ウィジェットの配置
  - 2.3 役割・役割名の設定
  - 2.4 GUI 処理部の構築
  - 2.5 GUI 処理部とアプリケーション処理部の結合
3. GUI 変更

以下においてそれぞれの手順について詳しく述べる。

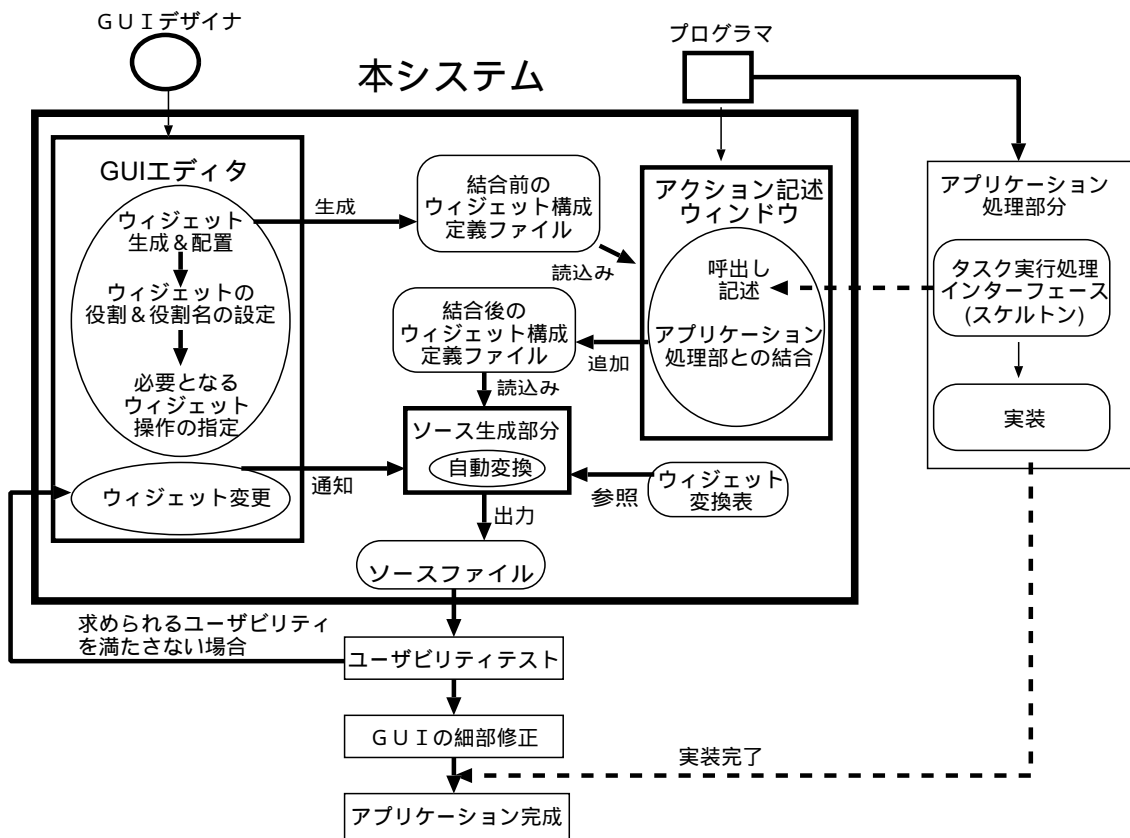


図 12: 本システムの構成図

## 7.1 タスクの分析

まず、デザイナーとプログラマが、「ファイルを選択して削除する」というタスクを分析する。この例の場合、共通のデータとして、「ファイル名」、「ファイル内容」というものが必要となる。これに対し、GUI 処理部では、「選択による入力」の役割を持つ「ファイル名選択」、入力の役割を持つ「ファイル内容の入力」、ユーザによるタスク実行を促すための「アクション要求」の役割を持つ「削除アクション」というそれぞれの役割と役割名と、タスクを実行するために、「選択されたファイル名を取得する」、「入力されたファイル内容を取得する」という手続きが必要となる。一方、アプリケーション処理部では、「ファイル名」と「ファイル内容」というデータを引数とする「ファイル入力メソッド」が必要となる。

## 7.2 GUI 構築

### 7.2.1 ウィジェットの配置

デザイナーはタスク分析の結果をもとにして、本システムの GUI エディタを用い、生成したいウィジェットを一覧から選択し、RAD ツールのようにウィジェットを配置する。この際、GUI の初期設定として必要なデータを本システムの設定ウィンドウで設定する（図 13）。例では JButton を生成し、「OK」という文字をデータとして設定している。



図 13: ウィジェットの配置とデータの初期設定

### 7.2.2 役割・役割名の設定

ウィジェットを配置したら、5つの役割からそのウィジェットが表す役割を選択し、そのウィジェットが果たす役割名を入力する。この操作により、ウィジェットと GUI 処理部で記述する

抽象的な手続きが結びつけられる。例では、JComboBox というウィジェットの役割名を「ファイル選択」、役割を「選択による入力」と設定している (図 14)。

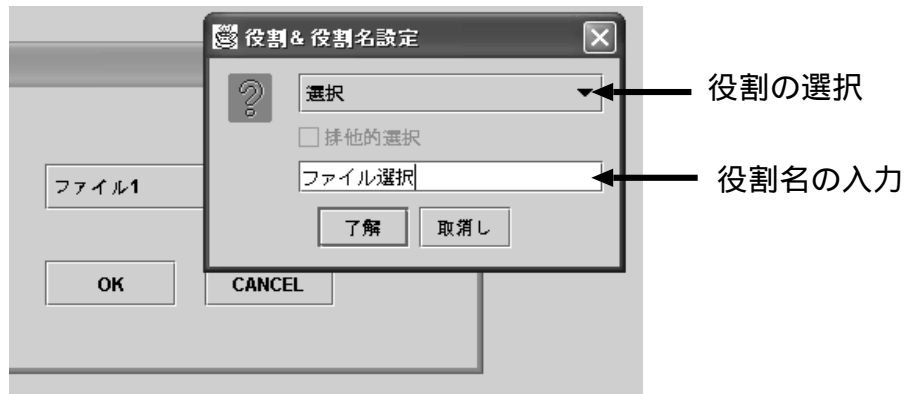


図 14: 役割と役割名の設定

### 7.2.3 GUI 処理部の構築

ウィジェットの配置と役割・役割名の設定を行った後、デザイナーは GUI 処理部の構築を行う。デザイナーはプログラムを書く代わりに、ウィジェットに対する処理を抽象的な手続きとして扱い、これらの手続きを組み合わせることで GUI の処理の流れを構築する。

例の場合、ユーザによって選択されたファイル名を取得し、ユーザによって入力された内容を取得する、という GUI の処理が必要となる。ユーザからのイベントを処理する場合、このイベントが起こった時に GUI としての処理が必要となるウィジェットに付けられた役割名を、アクション要求の役割を持つウィジェット (このイベントによって呼出されるメソッドを持ち、そのメソッドの中でタスクの処理を行うウィジェット) と関連付ける。例では、「ファイル選択」を「削除アクション」に関連付けている。デザイナーが、関連付けされたウィジェットの処理を記述する時は、関連付けされたウィジェットの役割である「選択による入力」の中から「選択されている項目を返す」という処理を選択する (図 15)。この処理の選択により「選択されたファイル名」を取得できることになる。本システムの現状では、アクションの発生時に呼び出されるメソッドを指定する形になっている。そのため、アクション要求によって呼び出されるメソッドの名前を知っていなければならないので、デザイナーにも若干プログラムの知識が必要とされる。

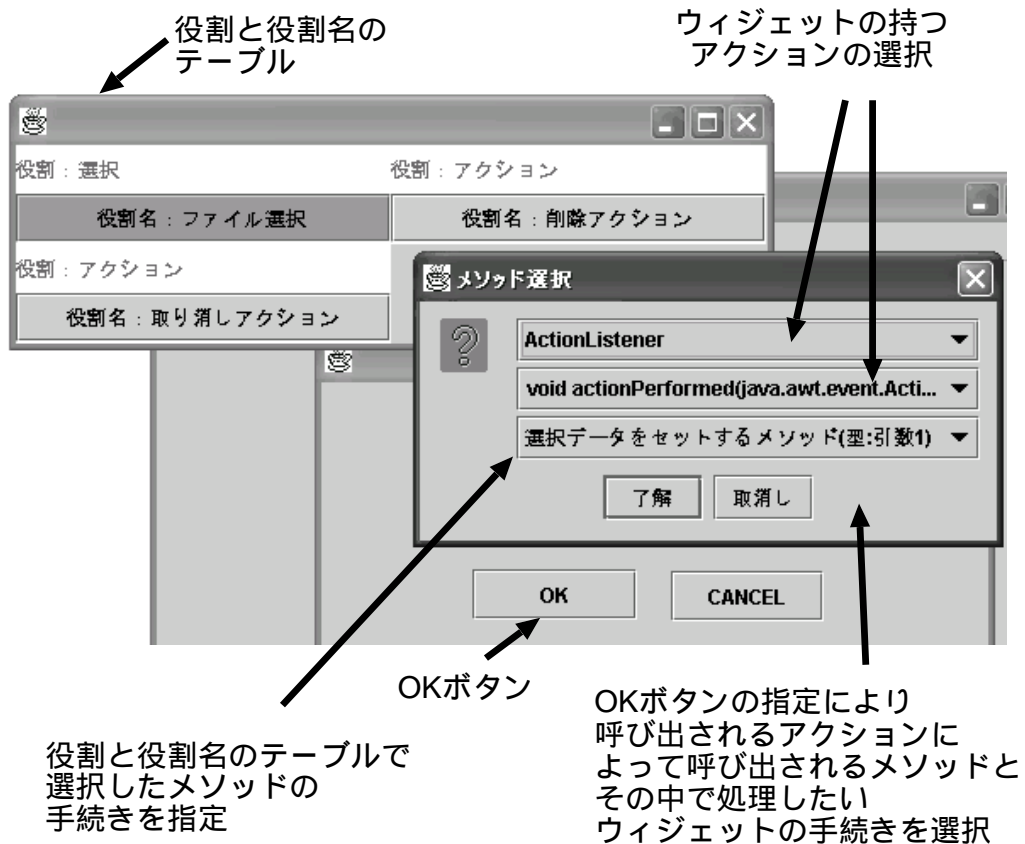


図 15: 要求される役割の手続きの関連付け

#### 7.2.4 GUI 処理部とアプリケーション処理部の結合

一方、プログラマは本システムのアクション記述ウインドウを通して、通常の処理メソッドの呼出し記述の他に、デザイナーが選択したウィジェットの処理記述を具体的なメソッドと同じように使い、アクションに対する処理を完成させる(図 16)。この際、引数が必要となる時は、プログラマが本システムで決められた形式に従って記述する。戻り値の型も同様とする。この内容がウィジェット構成定義ファイルに追加される。これによりデザイナーがプログラミングすることなく、意図したウィジェットとアプリケーション処理との結合が可能となる。本システムでは、ウィジェットのレイアウト情報などもウィジェット構成定義ファイル中に記述する。

以上の情報を元にして GUI アプリケーションの GUI 部分に関するソースコードが自動生成される。ウィジェット構成定義ファイルは、GUI のレイアウトに関する情報、ウィジェットに設定された役割と役割名、本システムにより構築される GUI を表すクラスのコンストラクタ中で実行するための処理(初期化時の処理)、ユーザイベントにより呼び出されるリスナのメソッド定義、アプリケーション処理部からのイベント通知の際に呼び出されるリスナのメソッド定義な

どの情報で構成される。

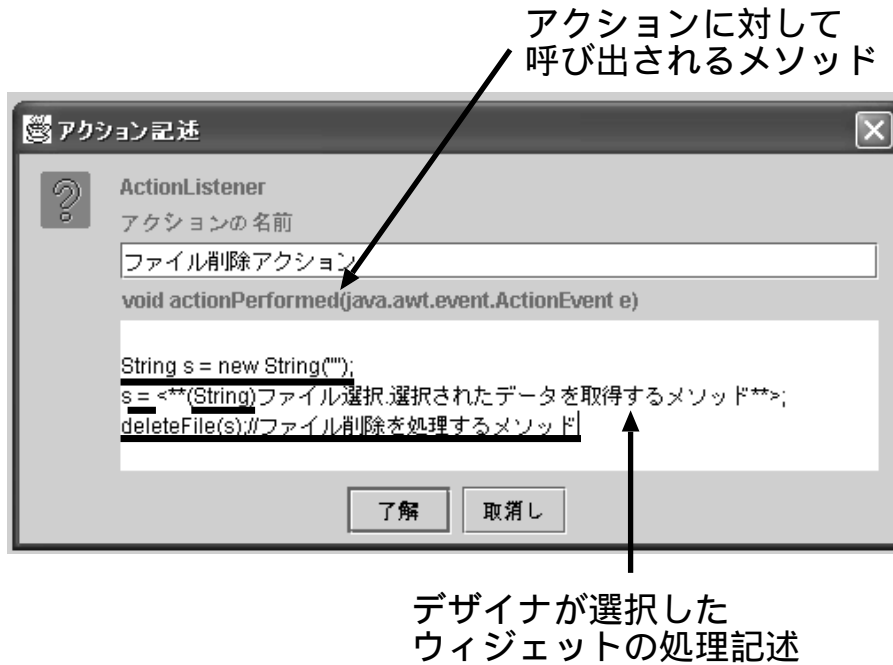


図 16: アクションに対するプログラマー記述例

### 7.3 GUI 変更

ウィジェットを交換する際には、そのウィジェットに指定された役割をもとにして本システムにより交換することのできるウィジェットの一覧が表示される。デザイナーはその一覧の中から、同じ役割を果たす別のウィジェットを選択することによりウィジェットを交換することができる。図 1 の変更例では、「JComboBox」に関連付けられている「ファイル選択」という役割名をもとに、同じ「選択」の役割を持つウィジェット候補の一覧から「DnDSelect」を選択することで、付け替えを行っている (図 17)。

## ウィジェット候補の一覧

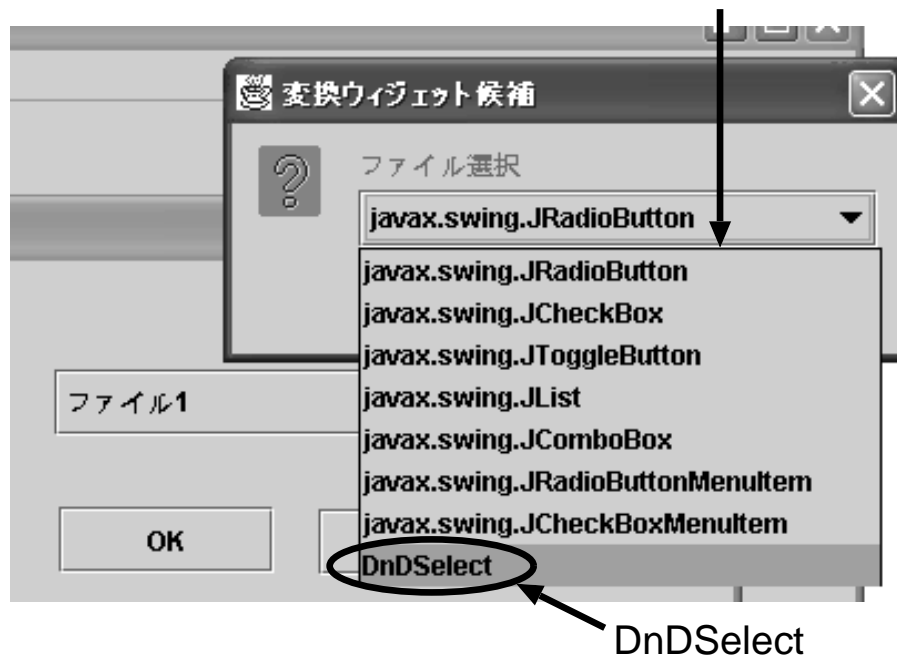


図 17: 変換候補の選択

システムは、ウィジェット変換表 (図 18) を利用することで、ウィジェット構成定義ファイルに記述してあるウィジェットの処理を自動的に変換する。図 18 の例の「JComboBox」と「DnDSelect」は、予め同じ「選択による入力」の役割に分類され、その役割の「選択されている項目を返す」というウィジェットの処理に対して、「getSelectedItem」と「getDroppedText」というメソッドを対応するものとして分類している。戻り値の型は、アクションに対するプログラマ記述 (図 16) の時にプログラマが指定された形式に自動変換する。DnDSelect ウィジェットを使うとき、ファイルとごみ箱イメージはデザイナーが手動で設定する必要があるが、選択項目名を設定する等のウィジェットの処理は自動的に変換される。

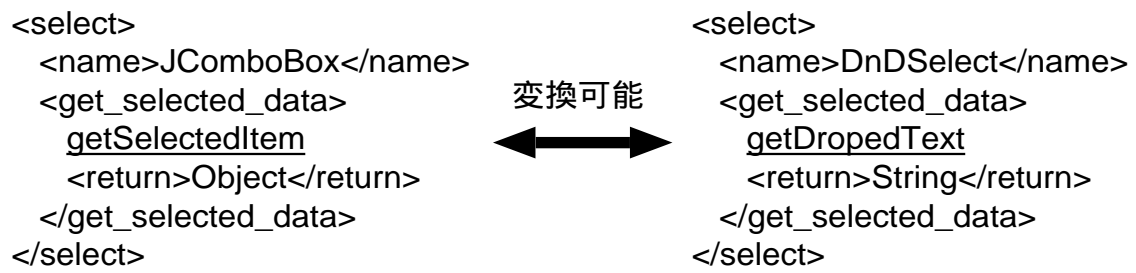


図 18: ウィジェット変換表 (一部)

## 8 評価

本システムによる GUI 構築に対する評価と、本システム・一からのプログラミング・RAD ツール (Forte for JAVA) による、GUI 変更時の評価を説明する。

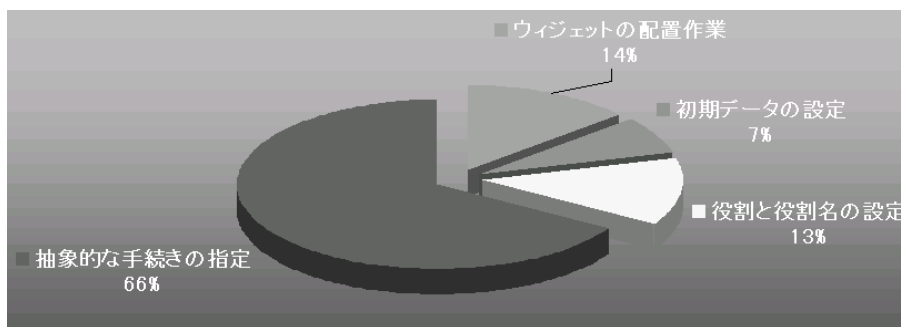
### 8.1 本システムによる GUI 構築時の評価

第 6.2 章で説明したウィジェット変換表記述ツールを本システムを使用して構築した。その際の GUI 構築の作業工程を、一からのプログラミングした場合と、RAD ツールを使用した場合に対して比較・分析した。また、抽象的な手続きによる GUI 構築の処理内容を分析した。

#### GUI 構築作業の分析

表 1 は、ウィジェット変換表記述ツールを本システムで構築した際にデザイナーが本システムを通じて行った操作の内訳である。それぞれ、ウィジェット 1 つに対して行われる作業を 1 回として数えている。

表 1: 本システムによる GUI 構築作業の分析



上記の結果より、ウィジェットの配置や初期データの生成は、従来の RAD ツールと同様に行われる操作であり、特に作業量、作業内容共に大差はない。抽象的な手続きの指定は、ウィジェットの抽象的な手続きを埋め込んでいき、この組み合わせによって GUI 処理部を構築するものなのである。従って、作業の量としては RAD ツールや一からプログラミングした場合と同等である。むしろ、本システムを通してビジュアルに構築することを考えれば、作業の内容としては両者よりも簡単であると言える。そこで、通常他の手法では行われない作業として、役割と役割名の設定が残るが、今回の表 1 では 13% と、ウィジェットの配置作業の割合とほぼ変わらない。本研究のタスクの分析は、本来 GUI を構築する際に必要となる作業を明確に行ったものなので、役割名の設定は、一からのプログラミング

や RAD ツールの使用と比べても、ウィジェットを配置する際に設定されるデータが 1 つ増えるという手間になるだけである。

#### ウィジェットの役割名設定数

ウィジェット変換表記述ツールを本システムで構築した際、使用したウィジェットの数は 28 個であり、そのうち役割名を設定していないウィジェットは 4 個であった。これは、今回のアプリケーションでは抽象的な手続きを必要としないウィジェットである。初期設定時にコンテナの役割として配置のみに使用したウィジェットと、同じく初期設定時にデータを設定したラベルがこれに該当する。また、設定した役割名は 26 個で、そのうち 2 つのウィジェットを 2 つの役割と役割名を設定した。本研究では、これらの役割の重複もシステムを通じて変更するため、一からのプログラミングや RAD ツールのようにウィジェットを生成し、役割を果たすための機能を再構築する必要がない。

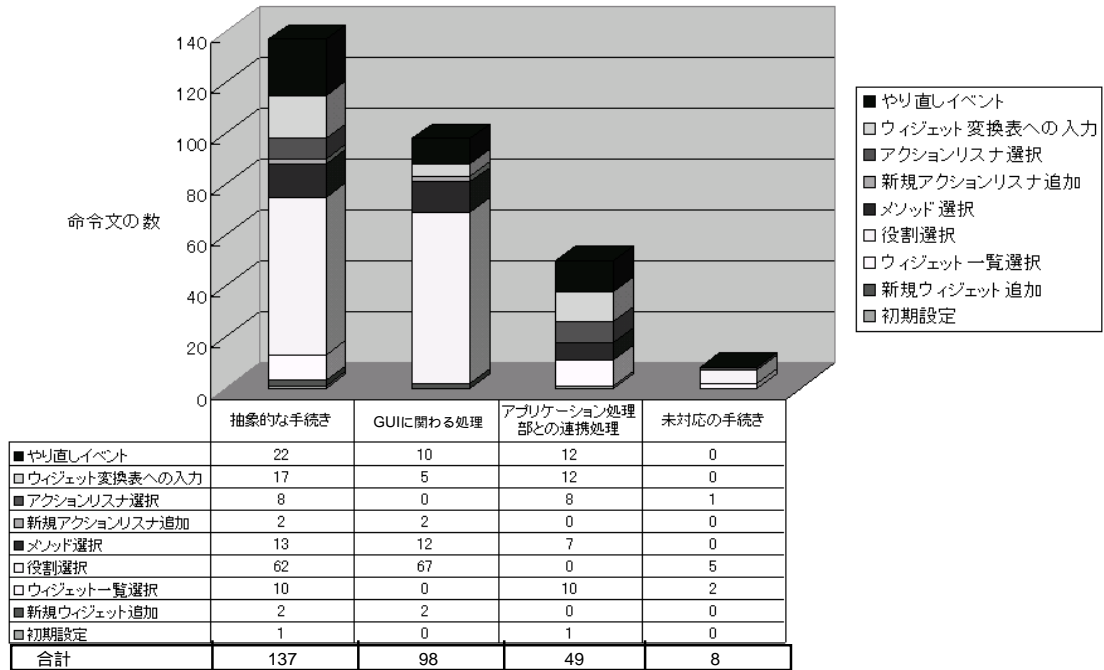
#### GUI 処理の内容

次に、GUI 処理部を抽象的な手続きを中心に分析した。表 2 の縦軸は、GUI の処理が必要となる部分を表しており、ウィジェット変換表記述ツール中で起こるユーザイベントと GUI の初期設定を指す。横軸は、命令文の数を表す。それぞれの項目の意味は、以下になる。

- 抽象的な手続き  
本システムによって生成した抽象的な手続きの数。
- GUI に関わる処理  
抽象的な手続きのうち、ウィジェット間の処理に関わる。ユーザからの入力値をウィジェットに設定してそのままユーザに表示する場合等が該当する。
- アプリケーション処理部との連携処理  
抽象的な手続きのうち、アプリケーション処理部とのデータの受け渡しに関わる。アプリケーション処理部で処理されたデータを表示する場合等が該当する。
- 未対応の手続き  
現時点で本システムの抽象的な手続きとして対応していない処理。

GUI に関わる処理とアプリケーション処理部との連携処理が一部重複しているが、入力ウィジェットにより入力値を取得して、GUI のレイアウトを変更し、その入力値をアプリケーション処理部に引き渡す、等の場合が該当する。

表 2: 本システムによる GUI 処理の内容



上記の結果より、GUI に関わる処理がアプリケーション処理部との連携処理の 2 倍であることが分かる。GUI に関わる処理が多ければ、デザイナーが本システムを用いてアプリケーション処理部と直接関係のない GUI の処理部分を構築することで、プログラマによるアプリケーション処理部との結合を待たずに、試行錯誤により、動的な GUI のレイアウトの処理（入力値によってレイアウトを変更する等）をより多く構築することが可能となる。ただし、現時点での本システムでは、動的な GUI のレイアウトの処理を完全に記述するには、若干のプログラムの知識を必要とする。例えば、条件分岐命令やループ処理が必要になる場合等である。

### 8.2 本システムの GUI 変更時の評価

また、本システムを使用した場合と、一からプログラミングした場合、RAD ツールを使用した場合に対して、アプリケーションの GUI を構築し（図 19）、それぞれの方法で GUI を変更した（図 20）。このアプリケーションは、カレンダーになっており、日付ごとに一行メモを記入して参照するものである。図 19は、月の表示と1ヶ月ごとの切り替えに JTabbedPane という排他的選択の要素を持つウィジェットを使用しており、JTabbedPane をダブルクリックすることで1ヶ月ごとのメモの一覧を表示できる。図 20は、月の表示を 12 個の JCheckBox に変更し、これ

らを多重選択を可能とすることでそれぞれの月のカレンダーを JFrame というウィンドウごとに表示させるようにし、メモの一覧を表示させるためのユーザイベントを、カレンダーである JFrame ウィンドウの年月表示をダブルクリックすることで実現するように変更したものである。尚、今回用いたアプリケーションは、文献 [3] に掲載されているサンプルプログラムであり、これを元に変更前と変更後の GUI を比較した。また、RAD ツールは、Forte for Java[4] を使用した。

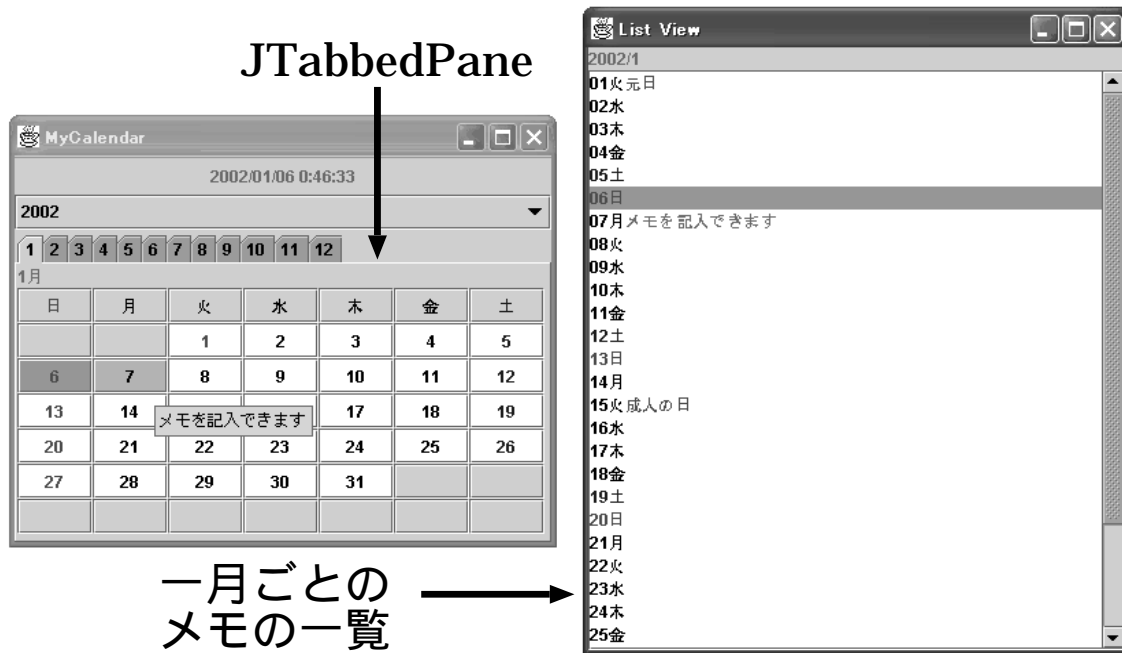


図 19: カレンダーアプリケーション変更前

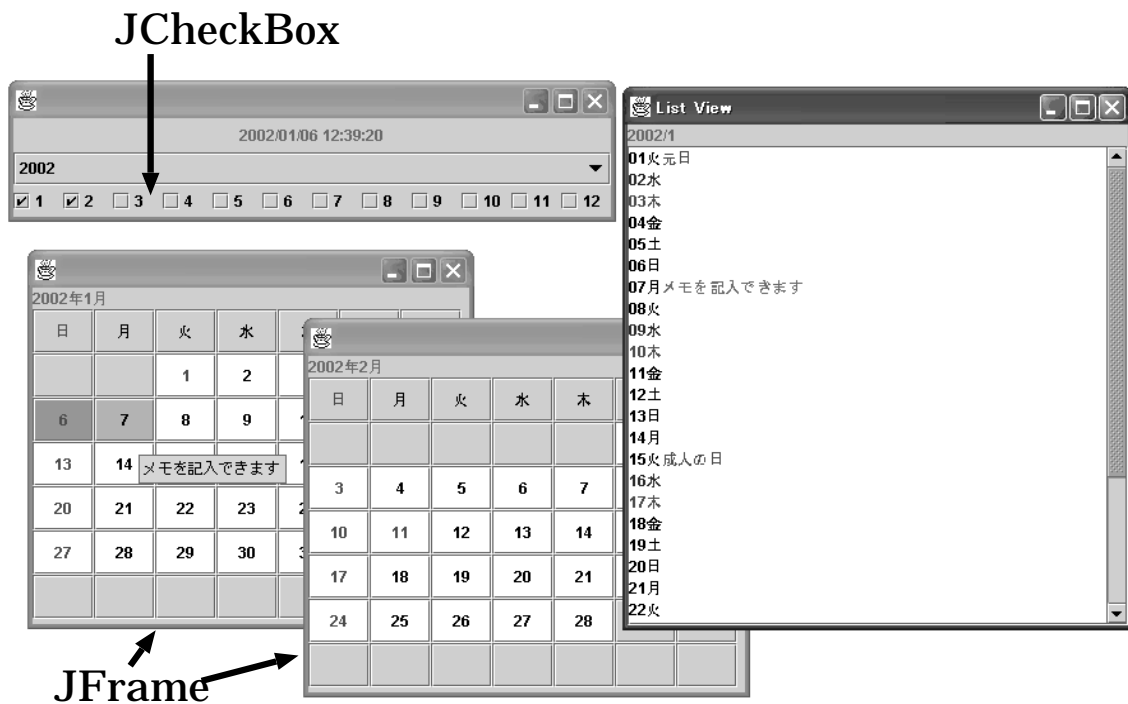


図 20: カレンダーアプリケーション変更後

比較の結果を表 3 に示す。表 3 は、「1 からのプログラミング」、「RAD ツール」、「本システム」による変更に関して、「削除（削除された命令文の数）」、「追加（追加された命令文の数）」、それらの変更箇所が「自動修正」と「手動変更」のどちらに該当するか、自動修正をさせるためのユーザのシステムに対する「操作回数」を記したものである。「操作回数」は、クリック等の実際のユーザの操作を指すものではなく、RAD ツールの場合、「表示させる文字を入力する」や本システムの場合、「役割名をつける」などの回数を数えるものとした。

表 3: GUI 変更作業の比較表

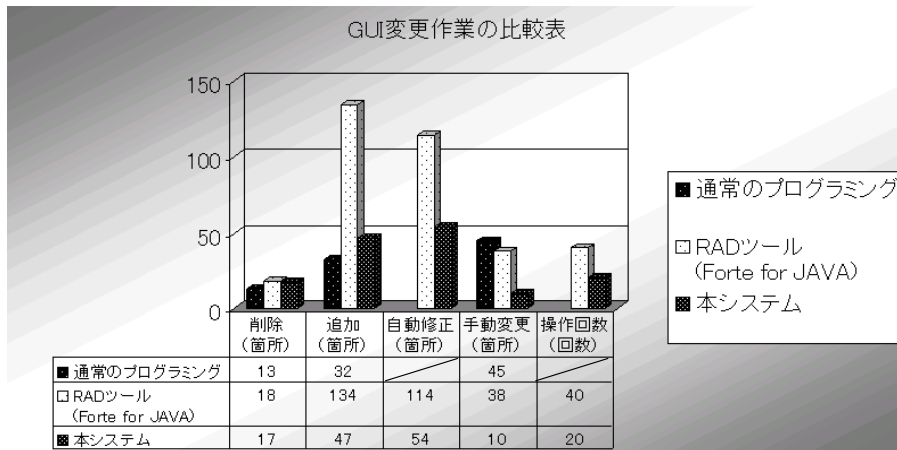


表 3の結果より、GUI を変更した時に単に変更部分が少くないのは「削除」と「追加」の合計が 45 個の「1 からのプログラミング」であるが、本システムでは、システムに対して操作することで変更部分のうち、約 84 % を自動修正することができる。その結果、手動の修正箇所は 10 箇所となり、3 つの中で最小となる。手動で修正を行った箇所は、現時点の本システムではサポートしていなかったウィジェットの操作の部分と、JTabbedPane のような排他的選択しかできないウィジェットから JCheckBox の集合体のように多重選択できるものへの変換の部分である。また、操作方法等が異なるため一概には比較することはできないが、操作回数も RAD ツールと比べて約半分なので、本システムでは RAD ツールよりも GUI の変更の労力は少ないと考えられる。「追加」の項目において、本システムと同様にグラフィカルな配置によってウィジェットの生成を行う RAD ツールに比べて、本システムの変更数が約 35 % に収まっているのは、関連のある役割を果たすウィジェットのグループ化により、これらのグループをループ命令を使って配列のように扱うことで、従来の RAD ツールだとウィジェットの個数分だけ、それぞれ命令が必要であるところを、数行で処理を行うことができるためである。また、本システムで作成・変更を使用したウィジェットは、単一データウィジェット、配列データウィジェットのどちらかに分類できるものであった。

以上より、本システムでは、1 からプログラミングする、あるいは、RAD ツールを利用するよりも容易にウィジェットの変更とそれに伴うアプリケーション本体の修正を行うことができる。

## 9 考察

本システムによる GUI 構築結果において、見られた特徴を挙げ、その原因と対策について考察する。

### 9.1 役割名の設定における優位性

本研究では、GUI の役割を果たすためのウィジェットにそれぞれ固有の名前を設定した。本システムでは、ウィジェットを配置し、抽象的な手続きをプログラムとして埋め込む作業を行うことで GUI を構築するので、静的なレイアウト及び、動的なレイアウトを構築することが可能である。この際、ウィジェットがどのような役割を果たすものであるかがユーザビリティの高い GUI を構築する際に必要となる。例えば、アプリケーションの関連した機能を実現するウィジェットの操作などは、選択の役割を果たすウィジェットにまとめて、一連の選択肢としてユーザに提示したほうが便利である。本システムでは、最初に役割名を設定することで、この役割名とウィジェットの対応を本システム上で確認しつつ GUI を構築することができる。これは、上記の理由により、高いユーザビリティを持つ GUI 構築の観点において優位な点である。

### 9.2 手動変更の要因

本システムでは、単一データウィジェット、配列データウィジェットといった、ウィジェットの特性を考慮した自動変換により、GUI 変更の支援を行うことができた。しかし、ウィジェットの変更時に手動で修正を加える必要が生じる場面がある。手動変更の箇所が生じる原因としては、以下の 3 つが理由として考えられる。

- 本システムで対象としていないウィジェットの手続きを自動変換できなかった場合
- GUI の処理を追加、削除する場合
- 変更後のウィジェットの機能とアプリケーション処理部で実現している機能が重なった場合

#### 9.2.1 対象外の手続きを変換する場合

GUI の役割を果たすための手続きとして、対象外の手続きを本システムで扱うためには、ウィジェットの使用例や API の仕様を参照して対応することが考えられる。例えば、現時点では、選択の役割を持つ配列データウィジェットの「指定したデータが選択された状態にする」という手続きが対象外であるが、選択の役割を果たす手続きとして対応可能な手続きである。

しかし、本研究が対象とする 5 つの GUI の役割以外の、限定された役割を果たすための手続き間の自動変更の全てを、予め予測してウィジェット変換表に記述しておくことはほぼ不可能である。そこで、同じ限定された役割を果たす複数のウィジェットがあった場合、それらの役割と

役割を果たすための抽象的な手続きを本システムに追加し、ウィジェット変換表で分類できるように対応する必要がある。

### 9.2.2 機能の追加と削除

ウィジェットの変更で新たな GUI としての機能を追加することになる場合、その機能が単純にウィジェットの外観の変更によるものであったり、操作感の変更であるならば、手動によるプログラム変更はほとんど必要ないが、交換したウィジェットが持つ独自の機能を使用する場合、また、交換前のウィジェットが持つ独自の機能を使用していた場合は、それぞれ追加・削除の作業が必要となる。

ただし、基本的な役割を果たすための手続きは変更しなくてもよいので、ウィジェット変更に伴う手動による修正作業は軽減される。

### 9.2.3 機能の重複

ウィジェットの変更次第では、プログラマーがアプリケーション処理部で対応していた処理が無意味なものになることや、ウィジェットにあわせて修正が必要となることがある。例として、4.2.1 章の時刻設定をするための時の入力の GUI を考える。時の入力用に通常の JTextField を使用していた場合、文字列が入力された場合に不具合が生じる。そこで、1 から 24 の数字以外を受け取った時の処理をするための変更を行うとする。図 21 は、JTextField、WidgetA、JComboBox の入力の役割を持つ GUI である。WidgetA は JTextField を継承し、数字以外が入力された場合に、アクションを発生して外部に通知する機能を持ち、入力値を null(何も無い値) に変換して返すウィジェットである。

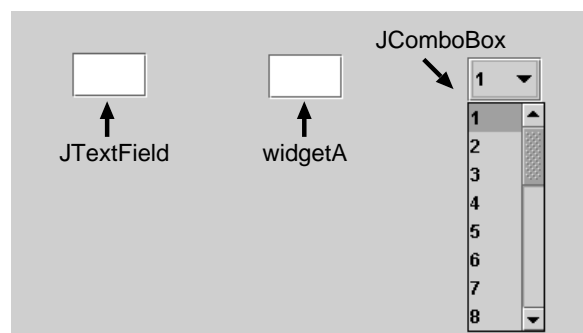


図 21: 時の入力用 GUI の変更例のウィジェット

プログラマーが意図していない入力値をユーザが入力するのを防ぐ場合、誤った値が入力された時にユーザに注意を喚起し、その入力値を受け取らない処理を行う対応と、選択による入力によ

て、正しい値しか入力できなくする対応が一般的に考えられる。前者を対応 A、後者を対応 B とする。ユーザからの入力として、ウィジェットで入力値の型を限定しない場合、対応 B をとることはできないので、入力値として文字列型で受け取り、改めてアプリケーションの処理として値の判定を行い、対応 A をとるようにプログラマがプログラミングすることが考えられる（図 22）。

```
String s = (String)<時の入力用GUI.入力されている値を取得する>;
try{
    Integer hourInt = new Integer(s);
    int hour = hourInt.intValue();
    if(hour>0&&hour<=24){
        /*入力値として採用*/
    }else{
        /*入力が不可な値であることを通知*/
    }
}catch(NumberFormatException error){
    /*入力が不可な値であることを通知*/
}
```

数値型に変換。変換不可の場合は NumberFormatException エラーが発生する ... (b)  
 入力値が 1 ~ 24 の間かどうか判別して処理を行う  
 NumberFormatException エラーが発生した場合、... (a) 強制的にこの部分に処理が移る

図 22: ウィジェットで入力値を限定しない場合のプログラム

JTextField を使う場合、通常では入力値を正しい値のみに限定して取得することはできないので、アプリケーション処理部分によって図 22 のように対応した場合、手動による変更を行う必要はない。また、WidgetA に変更した場合、数値型以外が入力された場合にアクションを発生するので、それを受け取るリスナの登録と呼び出されるメソッド中で、ユーザに注意を喚起するプログラムを構築すれば、図 22 の (a) の部分は重複して必要なくなる。しかし、そのままウィジェットの変更を行ってもプログラムにバグが生じることはなく、正常に動作する。JComboBox に変更した場合、対応 B となるので、図 22 の (b) 以外は余分な処理となるが、WidgetA の場合と同様に正常に動作する。逆に、ウィジェットで入力値の判定を行うと、その機能を持っていないウィジェットを使用した場合、アプリケーション処理部のプログラムを修正する必要がある。これは、ウィジェットの持っていた GUI の機能を一旦削除し、その機能をアプリケーション処理部で実現したことになるので、9.2 章の「機能の追加と削除」の特別な形の変更である。よって、全てのウィジェットとアプリケーション処理部との関係に当てはまるとは限らないが、ウィジェットとのデータの受け渡しを数値等に限定しない文字列型で行い、アプリケーション処理部で機能を実現した場合、アプリケーション処理部と同じ機能を実現しているウィジェットに変更を行った場合でも、機能の重複はあってもプログラムが正常に動かなくなることは少ないと推測される。

以上より、本システムを用いてプログラムを構築する際のこれらの問題点の対策としては、ウィジェット交換時にウィジェット固有の機能追加、削除が必要となった場合の支援を行うことが必

要となる。

## 10 関連研究

関連研究として、以下の4つに分類されるものを挙げ、その特徴と問題点について説明する。

### アプリケーション開発環境

アプリケーションの開発環境に関しては、VisualAge[5] や JBuilder[6] などの種々のツールが開発され、製品化されている。これらの製品では、WYSIWYG (What You See Is What You Get) が実現されており、GUI に関しても、ビジュアルに GUI の構成要素 (ウィジェット) を配置し、それぞれの色や大きさ等の属性もビジュアルに設定することができる。しかし、これらの開発環境は、依然としてプログラミング言語に依存した知識を必要とするため、GUI デザイナーが利用することは困難である。また、ウィジェットの交換による GUI の変更に関しては、特に考慮されていないので、GUI の処理部分とアプリケーション処理部分との結合は、ウィジェット変更ごとに再構築が必要となる。

### GUI プログラム自動生成

ユースケースのシナリオとしてコラボレーション図を記述し、そこからアプリケーションの GUI 部のプロトタイプを生成するという研究 [7] や E-R モデルでアプリケーション内のデータ間の関係や各データの属性などを記述し、ウィンドウ遷移などの GUI の制御構造をペトリネットで記述することにより、GUI を生成するという研究 [8] が提案されている。これらは、GUI 部のプログラムの生成のための特別な記述を付加したり用意しなければならず、開発者に大きな負担をかけると考えられる。また、生成されたプログラムにはアプリケーションの全体的な GUI の制御構造が含まれていないものもあり、実用性に欠けると考えられる。また、これらの研究は GUI 自動生成のみに注目しており、ウィジェットの変更に関しては注意が払われていない。

アプリケーション内のユーザイベントや処理を、タスクと呼ぶ小さな単位でモデル化し、そこから GUI 部のプログラムの生成を行うということが提案されている [9][10]。しかしこれらも、GUI 自動生成のみに注目しており、ウィジェットの変更に関しては注意が払われていない。

### GUI の自動変更

GUI の自動変更の関連研究として、アプリケーションのソースコードから GUI を生成し、その GUI をカスタマイズする機能を提供するツールも研究されている [11]。このシステムでは、視覚化したいデータをプログラム中で記述し、それに対してウィジェットが割り当てられ、初期の GUI を変更したい時は、用意されたライブラリからウィジェットを選択し変更する。視覚化するデータと結合されたウィジェットが入力を受け付けるものであれば、しかし、ウィジェットを使用するのにシステムに特化したクラスライブラリを使用するため、ライブラリに用意されていないウィジェットなどを追加することは難しい。また、ア

アプリケーション処理部のデータを指定してウィジェットを生成しているため、レイアウトやウィジェットの動的な変更などのプログラミングができないため、ユーザビリティ向上のための変更を十分にサポートしているとは言い難い。

## 11 おわりに

本研究では、GUIの変更に伴うウィジェット交換時のアプリケーション本体との依存性の自動再結合を実現する手法を提案した。同じ役割を果たす単体同士のウィジェット間の変更に伴う自動修正だけでなく、ウィジェットのグループ化により、複数のウィジェット間で変更した際の自動修正も実現した。

今後の課題としては、

- GUI 変更の際にサポートできるウィジェット操作の範囲を広げること
- ウィジェット固有の機能のためプログラムを変更しなければならない時の具体的な支援方法の提案
- 本システムで対象としていないウィジェットの操作を追加してプログラムを変更した後の、本システムでの再構築
- ユーザに対するタスクの情報提示による GUI のレイアウト変更支援

などが挙げられる。

## 謝辞

本研究を進めるにあたり、御指導頂いた深澤良彰教授と博士の白銀さんに深く感謝致します。また、多大なる御助言、御助力を下さった深澤研究室研究員の皆様に感謝致します。

## 参考文献

- [1] X. Ferre, N. Juristo, H. Windl and L. Constantine: Usability Basics for Software Developers, IEEE Software Vol.18, No.19, pp.22-29 (2001)
- [2] ヤコブ・ニールセン著, 篠原稔和監訳, 三好かおる訳: ユーザビリティ・エンジニアリング原論, 株式会社トッパン (1999).
- [3] 大村忠史著 Swing による JavaGUI プログラミング, 株式会社カットシステム (1998) .
- [4] Forte for Java: <http://www.sun.co.jp/forte/ffj/>
- [5] VisualAge: <http://www-6.ibm.com/jp/software/ad/vajava/>
- [6] JBuilder: <http://www.borland.co.jp/jbuilder/>
- [7] M. Elkoutbi, I. Khriss and R. K. Keller: *Generating User Interface Prototypes from Scenarios*, IEEE International Symposium on Requirements Engineering(RE'99) (1999).
- [8] C. Janssen, A. Weisbecker and J. Ziegler: *Generating User Interfaces from Data Models and Dialogue Net Specifications*, Proc. of the Conference on Human Factors in Computing Systems(CHI'93) (1993).
- [9] P. P. Silva, To. Griffiths and N. W. Paton: *Generating User Interface Code in a Model Based User Interface Development Environment*, Proc. of Advanced Visual Interfaces(AVI2000) (2000).
- [10] M. Ikeda, Y. Takata and H. Seki: *Formal Specification and Implementation Using a Task Flow Diagram in Interactive System Design*, Proc. of The 5th World Multi-Conference on Systemics, Cybernetics and Informatics(SCE2001) (2001)
- [11] 北村操代, 杉本明: 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, 情報処理学会論文誌, Vol36, No.4, pp.944-957 (1995).