ヒューマンインタフェース 92－8
（2001．1．18）

形式的仕様に基づくGUI開発環境の試作

尾崎寛和† 渋谷 雄† 辻野嘉宏†
†京都工芸繊維大学
〒606-8585 京都市左京区松ヶ崎御所海道町
e-mail：ozaki@hitms.dj.kit.ac.jp, {shibuya, tsujino}@dj.kit.ac.jp

概要

ウォーターフォールモデルのような一般的なソフトウェアの開発の手法はGUIの開発には適当ではない．なぜなら，GUIでは「使いやすさ」など設計段階では予想が困難な性質が重要なためである．そこで，GUI開発においては，しばしば，設計，プロトタイプ作成，テストを繰り返し行うラピッドプロトタイピングの手法が用いられる．しかし近年のアプリケーションのようにGUIが大規模化すると，その1サイクルに要するコストや時間を無視できなくなる．本稿では，これらのコストを総合的に小さくするため，GUIの形式的仕様に基づくGUI開発環境GUISEを提案し，その試作を行った．GUISEは，形式的仕様の作成を対話的に支援するSpec-Builder，形式的仕様に基づいていくつかの性質を自動検証しテストに要するコストを削減できるSpec-Verifier，形式的仕様からGUIプロトタイプを自動生成するProto-Generatorからなり，GUI開発をトータルに支援することができる．

GUISE：A GUI development environment based on the formal specification framework

Hirokazu OZAKI† Yu SHIBUYA† Yoshihiro TSUJINO†
†Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto-shi, 606-8585, Japan
e-mail：ozaki@hitms.dj.kit.ac.jp, {shibuya, tsujino}@dj.kit.ac.jp

Abstract

The conventional methods for the software development such as the water-fall model may not be suitable for the GUI development because some GUI properties like usability will not become clear in the design phase. Therefore, we often develop GUI based on the rapid prototyping method, in which we design GUI, implement the prototype, and test it repeatedly. But GUI becomes larger, the more cost and time are needed for each phase of the GUI development cycle. In this paper, we propose GUISE, the GUI development environment based on the formal specification framework and implement it in order to reduce the total cost of the GUI development. GUISE has Spec-Builder, the tool which supports interactively to describe a formal specification of GUI, Spec-Verifier, which is an automated verifier for some kinds of GUI properties so that it can reduce the testing cost, and Proto-Generator, which generates GUI prototype automatically based on the specification. So, GUISE can totally support the GUI development.

1. まえがき

近年では，多くのアプリケーションにグラフィカルユーザインタフェース(GUI)が使われている．GUI が使われるようになったのは計算機の能力が向上してきたことが１つの原因となっている．そのため，インタフェースにその能力を割り当てられるようになり，図形を用いた GUI を実現することができるようになった．また，最近はパーソナルコンピュータが広く普及したことにより，一般の人が計算機を使用する機会が増えている．一般の人にとって扱いに熟練を要するコマンドユーザインタフェース(CUI)よりも，図形を利用した GUI の方が直感的で使いやすいと考えられる．したがって，今日では，一般の人でも扱えるアプリケーションの作成には，GUI は欠かせないものとなっている．

一般的なソフトウェアの開発の手法は，要求定義，設計，プログラミング，テストの順に行われる [1] が，GUI にはこの開発の手法は適切でない．なぜなら，GUI は「使いやすさ」など設計の段階では予測しにくい性質を持っているためである．つまり，一度の設計で満足のいく GUI を得るのは難しい．そこで，一般的なソフトウェアの開発のように一方向に進む手法ではなく，設計，プロトタイプの作成，テストを図１のように，繰り返し行う手法が用いられる．

また，最近はアプリケーションの規模が大きくなってきているため，それにつれてその GUI の規模も大きくなってきている．そのため開発に大きな手間を要すると考えられる．したがって，大規模な GUI を図 1 のように繰り返し開発する際にその手間を減らすことが重要な問題になる．現在このサイクルのさまざまな部分に，手間を削減するために，いろいろなレベルの方法が試みられている [2].

本稿では，手間の削減方法として仕様の設計からプロトタイプの作成までの部分に焦点を当て，PGD モデル [3] に基づいた GUI の各状態を自動検証することによるテスト量の削減，形式的仕様からプロトタイプの自動生成を行うことによるプログラミング量の削減を行うことを考えた．これらを行うため GUI ダイアログの各状態についての自動検証が可能な形式的仕様の提案，形式的仕様の作成を支援するシステムの試作，形式的仕様からプロトタイプを出力するシステムの試作を行い，これらを統合した GUI 開発環境の作成を試みた．以下，２．で自動検証法，３．で自動検証可能な形式的仕様について述べ，４．で試作したシステムについて述べる．５．で試作したシステムの使用例を示し，システムの特徴について述べる．

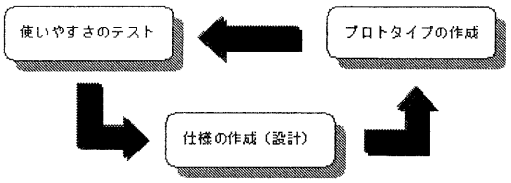

図１　GUI の開発サイクル

2. GUI の自動検証

ユーザとシステム間のダイアログを形式的に捉えたり，それに基づいてダイアログの性質を自動検証しようとするとするモデルはすでにいくつか提案されている [4] [5] [6] が，それらの多くは有限状態機械に基づくモデルである．

有限状態機械はその名の通り状態数が有限であるので，理論的にはそのモデルのすべての状態を調べ尽くすことができ，与えられた性質が成立するかどうかを判定することが原理的には可能である．しかし，有限状態機械に基づくモデルにはいくつかの共通する問題がある．

（1） 記述能力の問題

GUI などの一般的なユーザインタフェースを定数個の状態数しか持たない有限状態機械に基づくモデルで記述が可能であるかどうかという問題がある．もし，動的な要素によって実行可能なダイアログの構成要素の数がいくらでも大きくなるようなインタフェースがあれば，それを記述することは有限の状態数では不可能である．たとえば，描画ツールにおいては，ユーザが描いた図形自体に対して，移動，拡大，縮小などの操作を行える．ユーザはそのような図形を任意個描くことができるので，何個描かれたかを管理しなければならない．このような不定個の状態を有限状態機械に基づくモデルで管理することはできない．もし，このようなインタフェースを有限個の状態で近似しようとすると厳密な検証が行えないことになる．

（2） 状態爆発の問題

GUI などの一般的なユーザインタフェースが大規模である場合，構成される有限状態機械も非常に多くの状態を含むものになる．その上，GUI では一時に受け付けることのできる入力の種類が多く，各状態からの遷移数が多い．そのため，単純に GUI の形式的仕様を記述するとその記述量が膨大な量になる．また，検証を行う場合，すべての状態と遷移をしらみ潰しに調べるという方法では，いわゆる状態爆発が起こり，妥当な時間で検証を行えない．特にデッドロックフリーの問題の場合には，すべての状態の検査が必要であるので問題である．

したがって，本稿ではPGDモデルに基づく自動検証法を用いることとした．また図１でGUI の開発は設計，プロトタイプの作成，テストを繰り返し行うということを示したが，これに自動検証を含めると図２のようになる．なお図２に示すように自動検証はその対象である仕様が適合するまで行う．

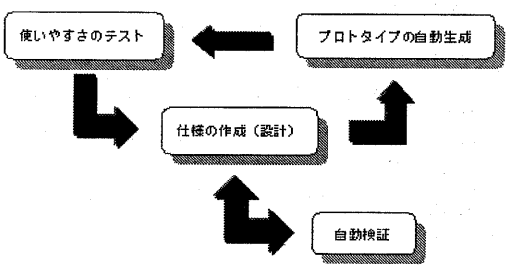

図2 自動検証を行うGUIの開発サイクル

以下では，PGD モデルとそれに基づく検証法について述べる．

2．1 PGDモデルの定義

PGD（Parallel Guarded Dialogue）モデルの定義は次のようになる．

$$PGD=(V,\ BD) \qquad (1)$$

ただし，Vは内部状態変数の集合で，各内部状態変数 vは２つの属性 $v.type$（型），$v.init$（初期値）を持つ．また，BD は基本ダイアログ d の集合であり，次のように定義される．

$$d=(vd,\ ed,\ gd,\ Td) \qquad (2)$$

ただし，vd は整数型の内部状態変数であり，この基本ダイアログを持つオブジェクトの獲得されている数を保持している．ed，gd，Tdはそれぞれ，イベント，Vの要素からなるガード式，内部状態変数への代入文の列である．基本ダイアログ dは，edが生じて gdが真のとき Tdが順に実行されることを示す．

PGD モデルは，ユーザからの入力によるイベント edとそれに対するシステム側のアクション Td，そのアクションを起こすための条件 gd の組の集合でダイアログを記述しようとするイベント駆動型のモデルである．システムの状態は各状態変数の値の組によって表現され，仕様の記述上では関係のある状態変数のみを扱えばよいので，記述量が膨大になることはない．これは１つの条件式で多数の状態を一括して扱っているとも解釈できる．また，各状態変数のとりうる値の数は有限であるとは限られていないので，有限状態機械に基づくモデルに比べて真に大きな記述能力を持つ．

定義では Td は内部状態変数への代入文のみからなるが，分岐や非再帰のサブダイアログ呼び出し（モーダル（modal）なダイアログボックスの起動）があっても等価であることが示されている[3]．

2．2 PGDモデルに基づく自動検証

ここで自動検証の対象とする問題は，到達可能性問題と到達不能性問題である．到達可能性問題とは，指定された状態から別の指定された状態（状態群）へ一連のユーザの操作で到達可能であるかどうかを判定する問題であり，ユーザの望む状態へ遷移できることを保証することを目的として検証が行われる．一方，到達不能性問題はどのようにユーザが操作しても到達できないことを示す問題であり，デッドロック状態など異常状態へ陥らないことを保証することを目的として検証が行われる．両者は理論的に同種の問題であるが，前者が到達する道を１つ見つければよいのに対して，後者は到達可能なすべての状態からの遷移をすべて調べる必要があるという違いがある．

PGD モデルに基づく検証法では，基本ダイアログの最弱事前条件に基づく検証を行う．すなわち，到達可能性を判定したい条件群を表す条件から始めて，各基本ダイアログの最弱事前条件を繰り返し計算することによって，初期状態までさかのぼることができるかどうかを調べる．このとき，有限状態機械に基づくモデルの場合の状態爆発と同様の条件式数の爆発が起こる可能性がある．しかし，そのうちの多くが削除できる（枝刈り）ことが示されている．この枝刈りによって，初期状態までさか

のぼることが不可能であることが証明されれば，到達不能性の証明となる．

ところで，現実の要求としては，到達可能であることが望ましい場合には到達できることのみで十分ではなく容易に到達できることが望まれる．言い換えると，ユーザが欲する状態に遷移するのに多くの手間を要するようなインタフェースはよいインタフェースであるとは言えないということである．

このような点を考慮して，既存のモデルである状態遷移機械に基づく検証法と本稿で採用するPGDモデルに基づく検証法を比較すると，

（1） 到達可能性問題については，前者は原理的に検証が可能であることを保証されている．後者はその保証はないが，現実の要求から考えると，数ステップの遷移のみを考慮すればいい．以上の点から考えて，両者には実質上の差はない．

（2） 到達不能性問題については，GUIの規模が大きくなれば前者はいわゆる状態爆発が起こり，実質上の検証は不可能であるが，後者は枝刈りが効果的に働く場合には検証が可能である．

ということが言え，PGDモデルに基づく検証法は従来の検証法に比べても有効であると考えられる．

3． 自動検証可能な形式的仕様

本稿では，PGDモデルの持つ検証法が利用可能な仕様の記述法を提案し，その仕様を作成支援するツールを試作する．

GUIはユーザがダイアログを行うオブジェクトを提供し，そのダイアログの管理を行う．そのダイアログの動作の間にアプリケーションの本来の機能を呼び出す必要があるので，GUIはこれも管理する．

本章では，2章で述べたPGDモデルの持つ検証法を利用することのできるGUI仕様の記法について述べる．

3．1 全体の構成

本稿で提案するGUI仕様の記法は宣言部と動作部の大きく2つの部分に分かれている．宣言部では画面に出現するオブジェクトを宣言し，動作部でそれらの動きを記述することである．自動検証を行う際には,宣言部はほとんど必要なく，動作部のみを検証するだけでよい．これらを分けることにより，検証時に動作部だけを抽出し検証を行うことができるので，スムーズに作業ができると考えられる．

3．2 宣言部

宣言部は，オブジェクトの宣言，定数の名前の宣言，制約の宣言，参照ファイルの宣言という4つの部分からなる．すでに述べたように宣言部では，画面に出現するオブジェクトを宣言する．図3に宣言部の記述例を示す．図では1つのウィンドウの中に7つのボタンのオブジェクトが存在することが定義されており各オブジェクトに対応するダイアログが適用されていることを示している．

```
Declaration{
  Object Controler:Window (TOP = 0, LEFT = 0, TITLE = Controler ){
    Object play_button:Button_Panel (TOP = 41, LEFT = 10 ){
      Apply(play);
    };
    Object stop_button:Button_Panel (TOP = 41, LEFT = 56 ){
      Apply(stop);
    };
    Object fast_forward_button:Button_Panel (TOP = 73, LEFT = 10){
      Apply(ff);
    };
    Object rewinding_button:Button_Panel (TOP = 73, LEFT = 114){
      Apply(rw);
    };
    Object pause_button:Button_Panel (TOP = 41, LEFT = 102){
      Apply(pause);
    };
    Object power_button:Button_Panel (TOP = 9, LEFT = 10){
      Apply(power);
    };
    Object record_button:Button_Panel (TOP = 41, LEFT = 155){
      Apply(record);
    };
  };
};
```

図3 仕様の宣言部の記述例

以下では，宣言部でどのようなものが宣言でき，それらがどのような機能を持っているのかについて説明する．

オブジェクト…仕様に記述できるオブジェクトには基本となる図形のオブジェクトと，集合を扱うグループオブジェクトがある．

スロット…オブジェクトはすべてスロットを持っており，そのオブジェクトの属性値を表す．また基本的なスロット以外に新しいスロットを作成しオブジェクトに追加することもできる．

継承…ある作成されたオブジェクトの属性値などをそのままコピーして別のオブジェクトに継承させて新しいオブジェクトを作成することができる．

グループオブジェクト…グループオブジェクトは，唯一 子オブジェクトを持つことのできるオブジェクトである．またグループオブジェクトを継承する場合は，オリジナルオブジェクトが持っている子オブジェクトがすべてコピーオブジェクトに継承される．

Widget…Widget とは，例えばボタン，スクロールバーなどよく使われる GUI のパーツであり，あらかじめ用意されているものである．

オブジェクトのダイアログへの付加…オブジェクトはダイアログを持つことができ，そのダイアログはある決められたイベントを起こすことによって始まる．ダイアログはオブジェクトのそれぞれに複数指定することができる．またダイアログは大きく２種類に分けられる．１つはクリックやフォーカスインなどで，どういった動作を行うかを記述できるダイアログであり，もう１つがドラッグによる移動，拡大など反応がどのオブジェクトに対しても決まっているダイアログである．後者のダイアログはあらかじめ用意されており，仕様においてオブジェクトにこのダイアログを付加するという記述を行うだけでよい．

次にどういった動作をするか記述できるダイアログについて述べる．例えばクリックイベントのダイアログをオブジェクトに付加する場合であれば，そのダイアログの名前を先程と同じように付加しておく．これはつまり，実際の動作は動作部に記述されており，オブジェクトはその名前のダイアログが適用されることを表す．宣言部にではその内容には関与しない．

定数の名前…GUI の仕様を通じて何度も現れる定数に名前を付けて，その名前が定数の代わりに使用できる．

制約…あるオブジェクトのスロットを，他のオブジェクトスロットとある一定の関係があるようにしたいときは，制約を用いる．

参照ファイル…参照ファイルとはアプリケーション本体を表したファイルのことである．必要に応じて，それらを宣言すればよい．

３．３　動作部

動作部では，ダイアログ間で使用する変数の宣言，および宣言部で宣言されたオブジェクト，定数の名前，制約を用いてダイアログを記述する．その記述例を図４に示す．図では画面上にウィンドウを表示する初期設定，動作部で使用する変数の宣言，用いられるダイアログの記述が記述されている．

```
Dialogue{
 Initial{
   Add(Screen, hex_window);
 }                                   初期状態・初期条件

 Variable{
   bool player;
   int top;
   int left;
   int p-top;
   int p-left;                       変数の宣言
   int c-top;
   int c-left;
   char board[5][5];
 }

 put_stone:click(single click){
   Set(owner.VISIBLE, true);
   if(player == BLACK){
     Set(owner.FILL_STYLE, Black);
     top = Get(owner.TATE);
     p-top = top;
     left = Get(owner.YOKO);
     p-left = left;
     user_position(board, top, left, player);
     next_position(board, WHITE);
   }
   if(player == WHITE){
     Set(owner.FILL_STYLE, White);
     top = Get(owner.TATE);
     p-top = top;
     left = Get(owner.YOKO);
     p-left = left;
     user_position(board, top, left, player);
     next_position(board, BLACK);
   }
 }                                   ダイアログ処理の記述

 press_start:click(single click){
   if(Get(radio_button.VALUE) == "First"){
     player = BLACK;
   }
   if(Get(radio_button.VALUE) == "Second"){
     player = WHITE;
     next_position(board, BLACK);
   }
 }                                   ダイアログ処理の記述

 press_wait:click(single click){
   wait(p-top, p-left, c-top, c-left);
 }                                   ダイアログ処理の記述
}
```

図４　動作部の仕様の記述例

以下に，変数宣言およびダイアログの記述方法について述べる．

変数の宣言…ここでは，先程述べたようにダイアログで使用する変数を宣言する．ここで宣言できる変数の型はスロットや制約で宣

言できる型に加えて，オブジェクトを指すポインタを宣言できるという特徴がある．こうすることで，あるオブジェクトを作成したり，あるいはオブジェクトを削除したりするという行為をダイアログで記述することが可能になる．

ダイアログ…ダイアログが持っているものは，ダイアログ名，イベント，ガード，ダイアログの処理の4つである．ダイアログ名はオブジェクトにダイアログを付加するときに必要なものである．イベントにはマウスのクリック，キーボードのボタンの押し下げの2種類がある．ガードとは，そのダイアログが実行されるための条件を表している．ダイアログの処理で記述できることには，代入文の記述，条件式を書く，アプリケーション機能の呼び出し，オブジェクトの作成，オブジェクトの削除，オブジェクトの追加，オブジェクトの除去，イベントの待機がある．

また，ダイアログの記述が表す動作の一連の流れは，そのイベントが起きたときにガードが真ならばダイアログの処理を行うというようになる．

4． GUI開発環境の試作

ここでは，先に述べた自動検証法や自動検証法を利用可能なGUI仕様の記述法を用いて試作を行ったGUI開発環境についての説明を行う．

4．1 自動検証ツール

2．で解説したPGDモデルを用いて，**3．**で提案した仕様の記述法で記述されたGUI仕様を入力とし，GUIの各状態が望ましい条件を満たしているかを自動検証するためのツール(Spec-Verifier)の試作を行った．試作したツールは，命題を格納するためのキュー，初期状態からの到達可能性判定を行う部分，最弱事前条件 [7] を計算する部分，枝刈りを行う部分，そして式の簡単化を行う部分からなる．これらを用いて検証の高速化を図っている．

4．2 GUI仕様作成支援ツール

先に述べたGUI仕様の記述法は自動検証法を利用可能であるので，そのほとんどが機械的な記述法である．したがってユーザがそれを最初から入力することはプログラムを書くのと同じくらい面倒であると考えられる．そのため仕様を記述する際に何らかの支援を行う必要があると考え図5に示すようなGUI仕様作成支援ツール(Spec-Builder)の試作を行った．図を見てわかるように画面の左側に各オブジェクトを選択するためのアイコンとその他必要な情報を入力するためのテキストボックスがあり，右側のは選択したオブジェクトを貼り付け実際にどのように表示されるかを確認するためのワークスペースがある．実際にはアイコンからオブジェクトを選択しワークスペースに貼り付け，定数などの情報をテキストボックスに入力するといったことを繰り返し行うことによって作成していく．これによってGUIの外観をユーザがアイコンなどを用いて直感的，対話的に形式的仕様を作成することができる．

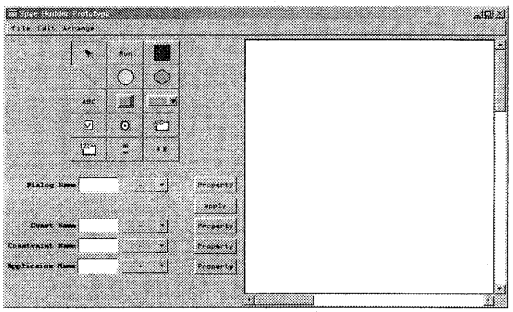

図5 Spec-Builderの全体図

4．3 プログラム自動生成ツール

3．で提案した仕様の記述法からC++プログラムソースコードを自動で生成するツール(Proto-Generator)の試作を行った．これにより，仕様を書けば自動でコードが生成されるので，プログラミングの手間が省くことができる．

4．4 GUI開発環境

以上で述べたシステムやプログラムを統合して，GUI開発環境(GUISE)の試作を行った．これにより図6に示す部分でGUI開発の手間を軽減することができると考えられる．また，実際のGUI開発の流れとしては，はじめにSpec-Builderでユーザが GUI仕様を作成し，そのGUIの各状態が望ましい条件を満たしているかであるかをSpec-Verifierで自動検証し，条件を満たしていれば次にProto-Generatorで仕様からC++プログラムソースコードを自

動生成する．満たしていなければもう一度仕様の作成の段階をやり直し，自動検証し直すといった作業を繰り返して開発を行う．

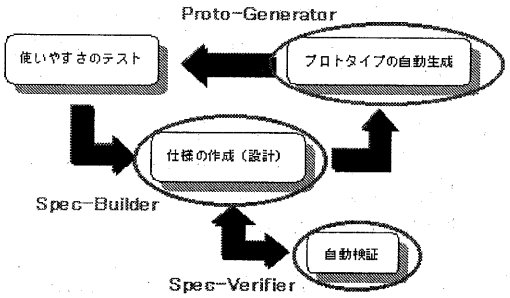

図6　GUI 開発環境による手間の軽減箇所

5．　システムの特徴

ここでは，提案した GUI 開発環境を実際に用いた作成例を示し，その特徴について述べる．

5．1　作成例

ここでは，ビデオリモコンの設計を作成例として示す．まず先に述べた Spec-Builder を用いて図7のようにリモコンの外観を作成する．これには予めボタンなどのオブジェクトが用意されているので簡単に外観を作成できる．またオブジェクト名などもデフォルトで用意されているのでユーザがいちいち名前をつける手間も省ける．次に各ボタンに付加するダイアログを図8のように作成する．またここでは予め GUI 以外のアプリケーション本体は作成されているものとする．そのようにして作成した仕様を Spec-Verifier で自動検証しすべての状態が望ましい条件を満たしているならば，Proto-Generator でプロトタイプの自動生成を行う．

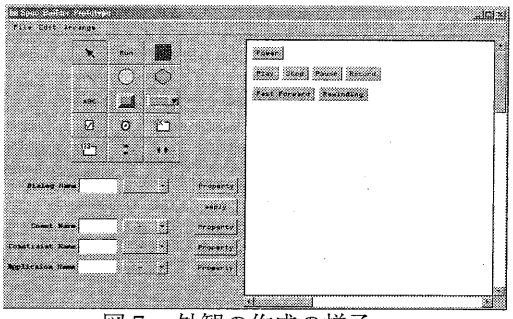

図7　外観の作成の様子

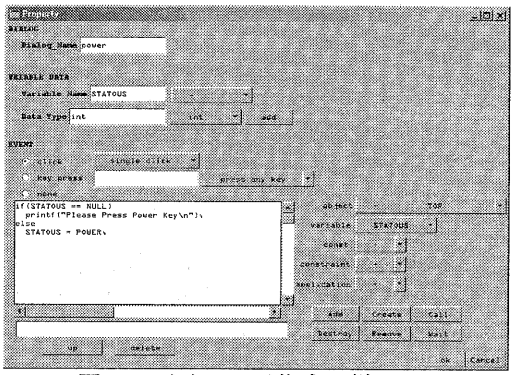

図8　ダイアログ作成の様子

次にこのリモコンにテレビリモコンの機能の追加する場合を考える．まず先に示したように外観を図9のように追加して作成し，各ボタンに付加するダイアログを同様に追加作成する．また変更を行う場合は，作成したダイアログを変更すればよい．

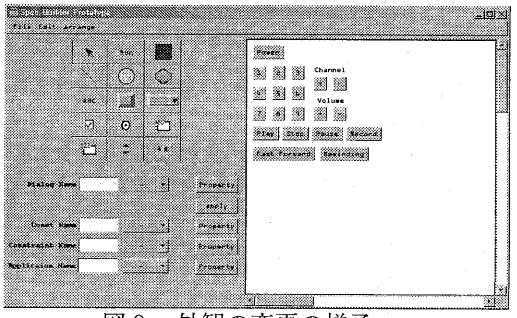

図9　外観の変更の様子

5．2　システムの特徴

作成したシステムの特徴として以下のことが挙げられる．

- ボタンなどのオブジェクトが予め用意されているので仕様の外観を簡単に作成できる
- オブジェクト名などがデフォルトで用意されておりユーザが名前をつける手間がいらない
- 仕様の変更を行う場合付加するダイアログの内容のみを変えるだけでよい．また追加を行う場合も追加するオブジェクトとそれに付加するダイアログを追加してやればよい

また GUI 開発環境での特徴として次のようなことが挙げられる.

- 上記で示したように容易に形式的な GUI 仕様を作成することが可能である
- 形式的仕様を自動検証することにより各状態のテストを行う量を減らすことができる
- 仕様からプロトタイプを自動で生成できるので仕様ができればプログラミングをする必要がなくなる

6. むすび

GUI のダイアログのモデルである PGD モデルに基づく検証法が利用可能な GUI 仕様の記述法を提案し, その仕様の作成を支援するツール(Spec-Builder)及び自動検証ツール(Spec-Verifier), GUI 仕様から C++プログラムソースコードを自動生成するツール(Proto-Generator)の試作を行い, それらを統合した GUI 開発環境(GUISE)の試作を行った. これにより GUI 開発サイクルにおける手間を軽減することが可能である. 問題点として, ダイアログの処理の記述を行う際にユーザがある程度キーボード入力を行わなければならないので, この部分での入力負担を軽減する方法を考える必要がある. また今回提案した以外の GUI 仕様の記述法 (例えばアニメーションのある GUI などの仕様の記述)を研究し, それらも本システムで用いることが出来るようにする必要がある.

参考文献

[1] 藤野, 花田:"ソフトウェア生産技術", 電子情報通信学会, pp.11-14 (1985)

[2] Geolf Lee : " Object-Oriented GUI Application Development " ,PTR Prentice_Hall, Inc., pp.147-167 (1993)

[3] 辻野 嘉宏:"GUI ダイアログの自動検証への試み", 電子情報通信学会 論文誌 D-I, Vol. J82-D-I No.10, pp.1286-1294 (1999)

[4] D. R. Olsen, A. F. Monk and M. B. Curry : " Algorithms for Automatic Dialogue Analysis Using Propositional Production Systems", HumanComputer Interaction, Vol.10, pp.39-78 (1995)

[5] D. Harel : " Statecharts: A Visual Formaslism for Complex Systems " , Science of Computer Programming, Vol.8, pp.231-274 (1987)

[6] G. D. Abowd and L. Ton : "Automated Verification of Temporal Dialogue Properies" , SIGCHI Bulletin, Vol.28, No.2, pp.50-52 (1996)

[7] D. Gries and F.B. Schneider : "A Logical Approach to Discrete Math" Springer Verlag, (1993)