グラフィクスとCAD 28－7
（1987・8・20・21）

ユーザインタフェースにおける対話記述の実現

今宮　淳美　　関村　勉
山梨大学　計算機科学科

G－Systemは現在著者らが開発中の知識ベースUIMSである。G－Systemはユーザインタフェース設計者がユーザインタフェースを記述する対話記述（Dialog Description）機能を持つ。本論文ではグラフィックスを用いたインタフェースの対話記述機能の設計と構造について述べる。また、Dialog Description生成システム実現におけるyaccとlexの利用について述べる。

An Implementation of a Dialog Description on the User Interface

IMAMIYA Atsumi , SEKIMURA Tsutomu
Dept. of Computer Science, Yamanashi University.
Takeda-4, Kofu, Japan, 400

We are now developing a Knowledge-based user interface management system (UIMS) G-System. The G-System contains a facility for user interface designer to specify user interface (human computer dialog description) declaratively. This paper describes the design and the mechanism of facility for specifying graphical interface based on abstracted interface definitions in some detail. We use the translator writing systems yacc and lex for Dialog Description generation system.

1．はじめに

ユーザインタフェースの設計および実現を支援するツール、すなわちユーザインタフェース管理システム（UIMS）は、広く適用するのにどんな応用からも独立でなければならない。G－Systemは現在著者らが開発中の知識ベースUIMSである[Imam86]。（図1）G－Systemでは、ユーザインタフェース設計者がマルチウィンドゥ上のユーザインタフェースを記述できる。そのユーザインタフェースを対話記述言語（DDL）を用いて宣言的に記述する。本論文では、G－Systemにおけるユーザコンピュータ相互作用のための指定と設計のツールである対話記述機構についてその設計および試作の現状について述べる。私たちの作成アプローチは、パーサジェネレータとオブジェクト指向言語概念に依っている。

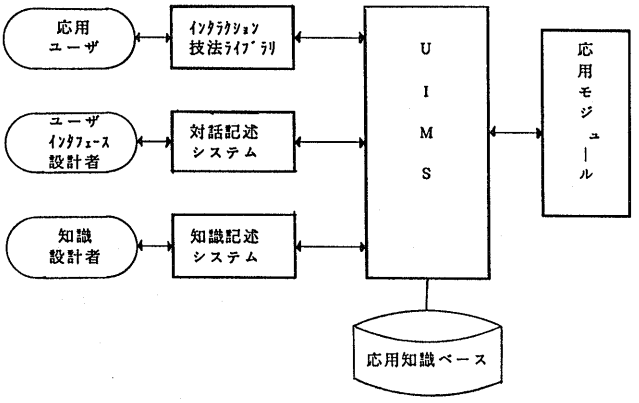

図1　ユーザインタフェース管理システム　G－System

2．対話記述

2．1　対話記述の概念

対話記述機能（DDF）は、ユーザコンピュータ相互作用を指定および設計するためのツールである。対話記述（DD）は応用とウィンドゥマネージャ間のインタフェースの指定、およびメニュー、カーソルなどのインタラクション技法を持つ。

たとえばメニューを使用する場合、以下の情報を必要とする。

- メニューを表示する位置
- メニューの大きさ
- 枠線の太さと色
- メニュー項目の数
- メニューの表示形式とその内容
- 前景と背景の色
- メニューの動作
- マウスボタンの操作により何を実行するか
- マウスカーソルの形状と色
- メニューウィンドゥの開始と終了のタイミング

対話記述はこれらの情報をすべて保持し、マルチウィンドゥにおけるメニューウィンドゥの開始、終了、そしてメニューの描画と動作をすべておこなう。

対話記述を用いることにより、応用プログラムとインタフェースツールの分離を明確にし、システム開発において以下の利点を得ることができる。[Schu85]

①メニュー指向のプログラムを簡単に作れる
上記のメニューに関する情報を対話記述言語（第３節参照）で宣言的に記述することにより、インタフェースツールを容易に作成できる。ユーザインタフェース設計者はグラフィックスに関する専門知識を必要としない。

②異なるアプリケーションに同一のインタフェース
設計者は開発するすべての応用プログラムに一貫した表示様式と使い方に統一することができる。これにより異なる応用プログラムでも同一のインタフェース様式でオペレータに対応することができる（操作環境の統一）。

③ひとつのアプリケーションに複数のインタフェース
初心者、熟練者、左利き、外国語などのために、応用プログラムをまったく変更することなく異なるインタフェースを持たせることができる。

④応用プログラムの可搬性
対話記述を用いることにより、インタフェースと応用プログラムとのモジュール界面が明確になる。こうして作られたプログラムは高い移植性と可搬性を持つ。また応用プログラム作成者とユーザインタフェース作成者の役割分担が明確になる。

⑤ユーザインタフェースのプロトタイピングとシミュレーション
①よりユーザインタフェースのプロトタイピングやシミュレーションを比較的簡単におこなえる。簡単に対話機能が実現できることでユーザインタフェースの設計－実現－評価の開発期間が短縮でき、設計者はインタフェースの性能向上に時間を費やすことができる。

２．２　Dialog Descriptionの構成

応用プログラムにウィンドゥにおける対話機能を提供するDialog Description - DD -は次の４つの部分からなる。（図２）

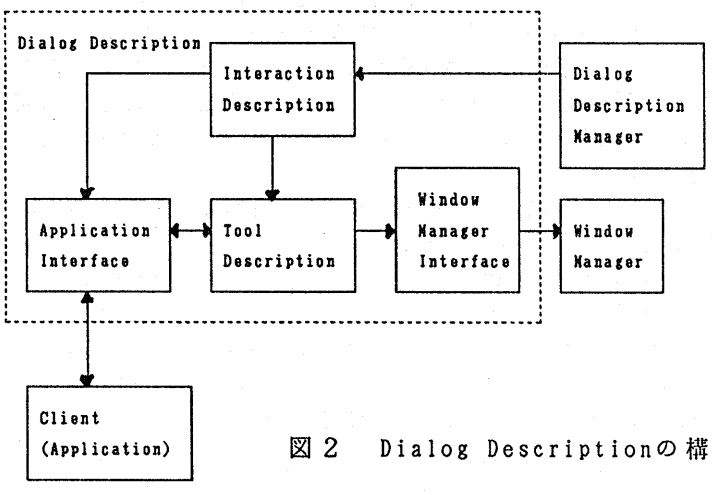

図２　Dialog Descriptionの構成

・Interaction Description
メニューやスクロールバーを始めにどこに配置して、オペレータの入力装置の操作によりどのような動作をおこなうか、などのDDにおける操作の流れに関する記述。
・Tool Description
ツール表示のためのパラメータ設定，動作,入出力タイミングに関する記述。ツールとは、メニュー,スクロールバー,カーソル,アイコンなどのDDの対話機能である。図３に対話記述を用いて使用できるツールライブラリを挙げる。
・Application Interface
DDから起動する応用プログラムの機能を外部関数定義として記述する。
・Window Manager Interface
ウィンドゥマネージャとの通信機能を持ち、DDからウィンドゥマネージャを起動する。

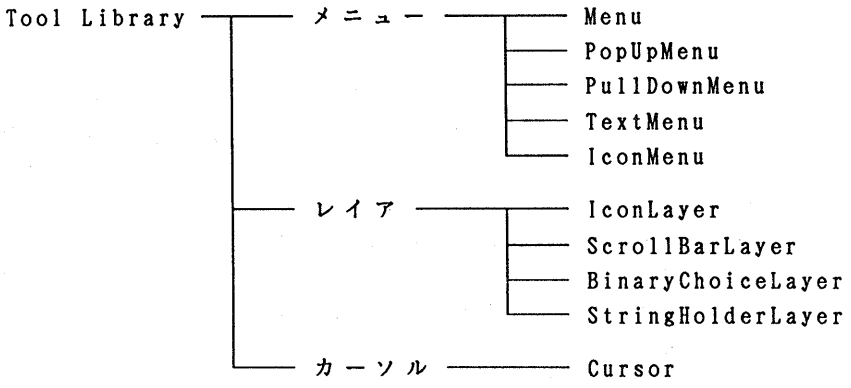

図３　ツールライブラリ

３．対話記述言語　－ＤＤＬ－

対話記述言語Dialog Description Language - DDL - により対話記述を記述する。DDLはユーザインタフェース記述の言語である。DDLで記述することによりユーザインタフェース設計者の負担を軽減し、ツールとしての対話記述の可搬性を高めることができる。言語仕様はツールに関する記述を宣言的に記述できるようになっている。以下、現在のプロトタイプのDDLについて述べる。DDL文法に関しては付録Ａを参照されたい。

DDLによる記述はApplication Interface,Interaction Description,Tool Descriptionの３つの部分から成る。ユーザインタフェース設計者がWindow Manager Interfaceについて意識する必要はない。なぜならば、Window Manager Interfaceはツールウィンドゥの生成、削除、移動、大きさの変更、プライオリティの変更、透過属性の変更、ツールを図形や文字に分解し描画するなどのツールに関する命令をウィンドウマネージャに伝える。したがって各ウィンドゥマネージャの仕様に合わせて一度作成すれば、後は同じ機能を使える。

Application Interfaceは、DDから起動する応用プログラムの外部関数定義であ

る。Interaction Descriptionは、open active closeの３つの関数からなる。open 関数は起動時のウィンドゥのオープンを行い、応用プログラムの関数を呼ぶこともある。active 関数は、DDが使用するウィンドゥ内にカーソルが入ると実行される１ステップの処理ルーチンである。close 関数は、ウィンドゥのクローズを行いDDの実行を終了する。

Tool Descriptionは、open active closeの３つの関数とツール表示のためのパラメータの設定からなる。１つの対話記述において複数個のツールに関する記述が可能である。 open関数は、ツールウィンドゥとツールインスタンスを生成する。active関数はツールの動きに関する記述である。 close関数はツールインスタンスとツールウィンドゥを消去する。

４．対話記述トランスレータ　－ＤＤＴ－

ユーザインタフェース設計者が対話記述言語DDLで作成する対話記述は対話記述トランスレータDialog Description Translator - DDT -によってObjective-Cソースファイルに変換される。これがObjective-Cコンパイラによりコンパイルされ Dialog Descriptionとして機能しマルチウィンドゥにおける対話機能を応用プログラムに提供する。

対話記述トランスレータの作成には、UNIXのプログラム開発ツールYacc[Step79]とLex[Lesk79]を用いる。これらのツールはテキスト処理用プログラムを作成するためのフロントエンドプログラムとして適しているので、対話記述言語DDLの仕様を拡張、変更する場合でも比較的簡単にDDTの修正がおこなえる。すなわち、ユーザインタフェースのプロトタイピングに有効である。Lexにより字句解析を行うプログラムを作製し、Yaccにより構文解析を行うプログラムを作成する[Bria84]。構文解析の後、それぞれの構文規則に合わせてObjective-Cソースコードを出力する。Cプログラムがこのコード出力をおこなう。（図４）付録Ｂに入力コードと出力コード例を示す。

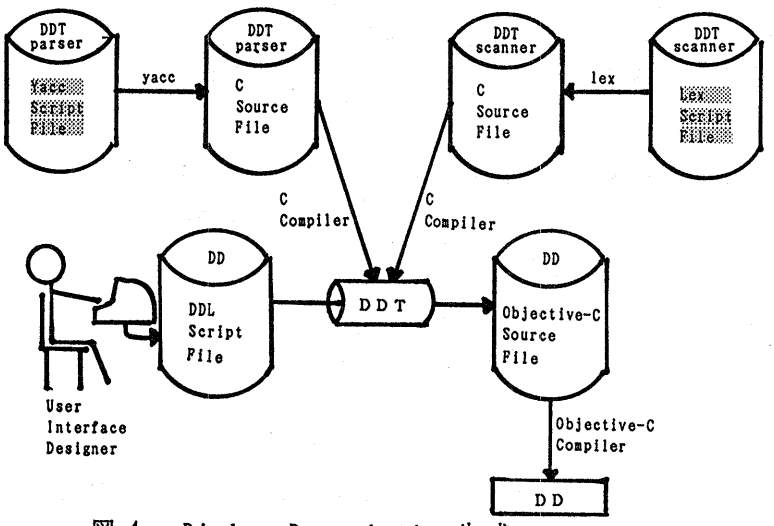

図４　Dialog Description生成システム

5．オブジェクト指向

Dialog DescriptionをObjective-C [PPI84]で実現することにより、メニュー、スクロールバー、カーソルなどのユーザインタフェースの要素機能をオブジェクトとして記述することができ、能率的なプログラミングが期待できる。またDDLに、以下のようなオブジェクト指向の利点を活用できる。

オブジェクト指向言語では、データの操作をオブジェクトにメッセージを送ることでおこなう。すなわちオブジェクトをブラックボックス化し、各オブジェクトへのアクセスを統一して記述できる。（モジュラリティの向上）

動的結合を用いることにより、DDL記述者はデータの型を考慮する必要がない。オブジェクトに対して抽象化したコマンド —メッセージ— を送ることでデータの操作をおこない、レシーバオブジェクトの型を考慮しなくてもよい。たとえば、メニュー、カーソル、スクロールバーを描画したい場合、menu,cursor,scrollbarなどのオブジェクトに対して、drawというメッセージ送出によっておこなう。menuやcursor,scrollbarのオブジェクトがどのようなデータ構造かを考える必要はない。（動的結合の利用）

オブジェクト指向言語では、クラス定義しているツールを作るのはインスタンス生成で可能である。たとえば、ポップアップ・メニューを使用したい場合クラスpopupmenuにおけるファクトリ・オブジェクト[PPI84]でインスタンス生成をおこなうことにより、複数のpopupmenuインスタンスを利用できる。言語処理系が、popupmenuで必要なデータを格納するメモリ領域の獲得を自動的におこなう。（インスタンス生成の利用）

クラスの階層化によりスーパークラスで定義するメソッドやインスタンス変数をサブクラスで使用できる（図５）。たとえば、windowクラスで定義するウィンドゥの位置、大きさ、プライオリティ、透過属性、前景色と背景色などのウィンドゥオブジェクトに必要な変数をwindowクラスで定義する。これによりサブクラスmenu,scrollbarとなどのクラスや、さらにmenuクラスのサブクラスであるpopupmenu,pulldownmenuクラスでもメソッドやインスタンス変数が使用可能である。ライブラリに登録されているツールに関して、DDL記述者はあらかじめ用意されているメソッドやインスタンス変数を使用できる。（継承の利用）

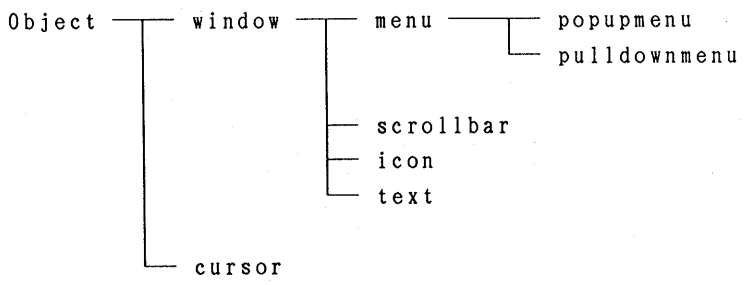

図５　Dialog Descriptionにおけるクラス階層

6．今後の課題

対話記述が有用なシステムであるかどうかは、対話記述言語 DDLの記述能力に依る。ツールを複数個同時に使用する場合、ツールの組合せ方でユーザインタフェースに特徴的な機能を持たせることができる。複雑な組合せの記述を、簡単に記述できる文法（表現法）を持たなければならない。また、シミュレーションやプロトタイピングをおこなうには、豊富なライブラリを持つことや、対話型であることが必要である。したがって、ユーザインタフェース設計者が、より効率的に開発やシミュレーションを行える言語（DDL）とトランスレータ（DDT）を含むシステムの設計に関する研究が重要である。さらに設計者の記述を容易にするために知識の利用も考えられる。

参考文献

[Bria84] Brian W.Kerningham,Rob Pike: The UNIX Programming Environment, Bell Telephone Laboratories,Incorporated, 1984 （邦訳）石田晴久監訳: UNIXプログラミング環境, アスキー出版, 1985

[Imam86] Imamiya,A.,Kondoh,A.,Miyatake,A.: An Artificial Intelligence Approach to the Modelling of the User-Computer Communications, Proc. Applications of AI in Engineering Problems, Apr.1986, pp1139-1151

[Imam87] Imamiya,A.,Maki,K.,Sekimura,T.: DEFINING A USER INTERFACE IN THE OBJECT-ORIENTED DIALOG DESCRIPTION FACILITY, IMACS INT. SYMP. ON EXPERT SYSTEMS AND LANGUAGES IN MODELLING AND SIMULATION, Jun.1987, pp81-89

[Lesk79] M.E.Lesk and E.Schmidt: Lex - A Lexical Analyzer Generator, UNIX PROGRAMMER'S MANUAL, Bell Telephone Laboratories,Incorporated, Jan.1979

[牧86] 牧喜代司,今宮淳美: オブジェクト指向言語によるウィンドゥ管理システムの実現,「グラフィクスとCAD」シンポジウム, 情報処理学会, Nov.1987, pp101-109

[牧87] 牧喜代司: マルチウィンドゥ管理システムに関する研究, 山梨大学大学院工学研究科計算機科学専攻修士論文, 1987

[PPI84] Productivity Product International,Inc.: Objective-Cリファレンス・マニュアル, 1984

[Schu85] Schulert,A.J.,Rogers,G.T.,Hamilton,J.A.: ADM -A dialog manager, Proc.CHI'85, Apr.1985, pp177-183

[Step79] Stephan C.Jhonson: Yacc:Yet Another Compiler-Compiler, UNIX PROGRAMMER'S MANUAL, Bell Telephone Laboratories,Incorporated, Jan.1979

付録A　ＤＤＬ構文

```
〈DDL〉::=DialogDescription 〈Identifier〉
        { ApplicationInterface '{' 〈Application Interface definition〉 '}' |
          ToolDescription '{' 〈Tool Description definition〉 '}' |
          InteractionDescription '{' 〈Interaction Description definition〉 '}'     }
〈Application Interface definition〉::={ 〈Application Interface definition_unit〉 }
〈Application Interface definition_unit〉::=extern 〈type specifier in Objective-C〉
        〈declarator in Objective-C〉 ;
〈Tool Description definition〉::={ 〈Tool Description definition_unit〉 }
〈Interaction Description definition〉::=〈Open definition〉
                                        〈Active definition〉
                                        〈Close definition〉
〈Open definition〉::=Open ( ) '{' 〈declaration_list in Objective-C〉
        〈statement_list in Objective-C〉 '}'
〈Active definition〉::=Active ( ) '{' 〈declaration_list in Objective-C〉
        〈statement_list in Objective-C〉 '}'
〈Close definition〉::=Close ( ) '{' 〈declaration_list in Objective-C〉
        〈statement_list in Objective-C〉 '}'
〈Tool Description definition_unit〉::=Tool 〈Identifier〉 : 〈Identifier〉
        '{' 〈Open definition〉 〈Active definition〉 〈Close definition〉
          { 〈argument name〉 '(' 〈variable-length argument_list in Objective-C〉 ')' } '}'
〈argument name〉::=Area | BackColor | ForeColor | Item | Bitmap | Message |
        Help | BorderLineWidth | BorderLineColor | ScrollBuffer |
        ScrollBufferSize
```

付録B　ＤＤＴ入出力例

入力コード例(DDL)

```
DialogDescription       Example

ApplicationInterface
{
  extern  int     ap_Load();
  extern  int     ap_Save();
}
// end of Application Interface

InteractionDescription
{
  open()
  { [menu1 open]; }
  active()
  { [menu1 active]; }
  close()
  { [menu1 close]; }
}
// end of Interaction Description

ToolDescription
{
  tool menu1 : PopUpMenu
  {
     open()
     {
        [item1 open];
        [item2 open];
     }
     active()
     {
        [self untransparency];
        while([self containMouse]
        ) {
           [item1 active];
           [item2 active];
        }
     }
     close()
     {
        [item1 close];
        [item2 close];
     }
     area(100,100,200,200);
     forecolor(blue);
     item(item1,item2);
     help("help_file_pass");
  }
  /* menu1 */

  tool item1 : TextMenu
  {
     open()
     {
        [self transparency];
        [self draw];
     }
     active()
     {
        [self reverse];
        while([self containMouse])
           if ([button on])
              ap_Load();
     }
     close()
     { }
     message("LOAD");
  }
  /* item1 */

  tool item2 : TextMenu
  {
     open()
     {
        [self transparency];
        [self draw];
     }
     active()
     {
        [self reverse];
        while([self containMouse])
           if ([button on])
              ap_Save();
     }
     close()
     { }
     message("SAVE");
  }
  /* item2 */
}
// end of Tool Description
```

出力コード例 (Objective-C)

```
extern  int     ap_Load();
extern  int     ap_Save();
=InteractionDescription:Object (DD,
Primitive)
-open   {
  [menu1 open];
}
-active {
  [menu1 active];
}
-close  {
  [menu1 close];
}

= menu1:PopUpMenu (DD,Primitive)
+create { return [[self new] initia
lize]; }
-initialize {
  area_x0 = 100;
  area_y0 = 100;
  area_x1 = 200;
  area_y1 = 200;
  forecolor = blue;
  item[0] = item1;
  item[1] = item2;
  help = "help_file_pass";
  return self; }
-open   {
  [item1 open];
  [item2 open];
}

-active {
  [self untransparency];
  while([self containMouse]) {
     [item1 active];
     [item2 active];
  }
}
-close  {
  [item1 close];
  [item2 close];
}

= item1:TextMenu (DD,Primitive)
+create { return [[self new] initia
lize]; }
-initialize {
  message = "LOAD";
  return self; }
-open   {
  [self transparency];
  [self draw];
}
-active {
  [self reverse];
  while([self containMouse])
  if ([button on])
     ap_Load();
}
-close  { }

= item2:TextMenu (DD,Primitive)
+create { return [[self new] initia
lize]; }
-initialize {
  message = "SAVE";
  return self; }
-open   {
  [self transparency];
  [self draw];
}
-active {
  [self reverse];
  while([self containMouse])
  if ([button on])
     ap_Save();
}
-close  { }
```

討論

7．ユーザインタフェースにおける対話記述の実現　　関村（山梨大）

坂下：Window Manager InterfaceはWindow Managerが１つ決まってしまえば書き直す必要はないという話でしたが、複数のWindow Managerに対応したことはあるのですか。

関村：まだ完全にできていないので、複数のWindow Managerには対応していません。現在はU-stationの上で開発していて、Window Managerはオリジナルのものを使っています。

川合：ウィンドウシステムは標準化の動きもあるようですが、簡単に移植できるのですか。

関村：ええ、X-window のようなものなら簡単にできそうです。
方法としては、Window Manager Interfaceのところに各Window Manager用のプロトコルを用意しておけば良いと思います。

坂下：ウィンドウシステムに独立というのは実際問題としてかなり大変だと思います。それぞれのウィンドウシステムにかなり癖があるからです。ユーザインタフェースの統一というのは良いのですが、逆に言うと、それぞれのウィンドウシステムの特徴を活かせなくなるという問題があると思います。

村上：ＡｐｐｏｌｏやＳｕｎのウィンドウシステムも同じように宣言的に記述するようになっていると思いますが、それらと比べてこのシステムの特徴は何かありますか。

関村：複数のメニューを使う場合、それらの組合せによる記述をしようと考えています。

川越：ライブラリー方式とこのような宣言的な言語を用意するアプローチでは、どのような違いがあるのですか。

関村：メニューのネスティングとかメニューの複雑な組合せの記述をもたすためにこのような言語を用意しました。

川合：ＵＩＭＳを簡単に作るための言語ということですね。

坂下：この言語の記述能力はどうですか。難しいかもしれませんが。

守屋：応用はどんなものを考えていますか。

関村：まだ特定なものは考えていませが、図形、知的エディタなどが考えられると思います。

林：　ＵＩＭＳが参照する応用知識ベースとはどんな知識で、どういう用途に使うのですか。

関村：知識の利用は今後の課題で、図形生成との関係がおもしろいと考えていますが、これからどう利用して行くか考えて行きます。

村上：付録ＢのＤＤＴの出力を見ると、ＹａｃｃとＬｅｘを使うまでもないと思いますが。

関村：ＹａｃｃとＬｅｘは、トランスレータの変更が楽になるということで使いました。プロトタイプの作成変更が比較的容易という点です。