GUIビジュアルプログラミング環境UIBT/RAD

3N－1

○二木　誠司　橘　昌宏　仲野　隆行　湯浦　克彦　（(株) 日立製作所）
久保　昭一　久保　一郎　（日立超LSIエンジニアリング(株)）

1. はじめに

UIBT/RAD はOSF/Motifをベースとした GUI構築ツールである。原型となっているUIBT[1]はGUI外観の設計機能が中心である。UIBT/RAD では外観を構成するボタンなどのUI要素をユーザが操作した際のアクションをビジュアルにプログラミングする機能を付加している。

ここでのビジュアルプログラミングは２つの意味を持っている。１つはマウスなどによる直接操作を最大限活用したプログラミングインタフェースである。そしてもう１つはコード自体を図式化して了解性と保守性を高めるものである。本稿ではこれらの方式の設計を中心に述べる。

2. UIBT/RAD 概要

UIBT/RAD の構成を図１に示す。このうちUIBTのサポート部分はUIBT C ライブラリと外観編集系である。UIBT/RAD ではライブラリをC++言語対応に拡張し、外観編集系をこれに対応させている。またコードを図式で入力し編集する機能（コード図式管理系）と、図式を専用のスクリプト言語に変換して対話実行する機能（スクリプト言語処理系）が付加された。

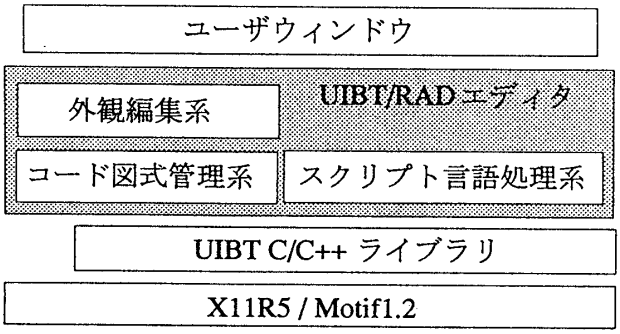


図1. UIBT/RAD の構成

GUIを有するアプリケーション（特に大規模なもの）では、まずウィンドウの外観を記述し、次にその間の画面遷移を記述し、その後本体部分を記述することが多い。UIBT/RAD ではこの開発手順を前提に、このうちの画面遷移の記述までをサポートする開発環境である。アプリケーションの記述はC++言語を用いる。

ここでプログラミングの対象となるのはユーザの作成するウィンドウである。UIBT/RAD では外観編集系で作成されたウィンドウをユーザ定義の新規クラスとしている。ここでウィンドウ内のUI要素のアクションは、このクラスのメンバ関数として定義される。各UI要素はこのクラスのメンバとして定義される。こうした定義は外観編集系での操作によって自動的に行われる。

3. ビジュアルプログラミング方式

3.1 記述内容

ビジュアルプログラミングの対象となるオブジェクトとこれへのアクセス方法と操作を次表にまとめる。

表１. ビジュアルプログラミングの記述対象

ｵﾌﾞｼﾞｪｸﾄ	ｱｸｾｽ方法	操作
UI要素	メンバ	表示、消去 表示属性の設定、変更
表示属性: 色、表示文字列、ﾌｫﾝﾄ、ﾊﾟﾀｰﾝ	UI要素から取得 作成	作成、破壊 基本データ型による設定 基本データ型への変換
ﾕｰｻﾞｳｨﾝﾄﾞｳ	外部変数	表示、消去、外観の設定
ｲﾍﾞﾝﾄﾃﾞｰﾀ	メンバ関数の引数	アクション発生時のﾃﾞｰﾀ所得
基本データ型：整数,浮動小数,真偽値,文字列	リテラル入力、表示属性から取得	四則演算 比較論理演算

ここで示した操作は画面遷移の記述に必要なものに限定している。この限定とアクセス方法を固定することで、メンバ関数の記述を定型化することができる。C++言語によるコーディングと比べて自由度は損なわれるが、コードの図式化にあたってはその意味を明確にすることができる。

Visual Programming Environment for the Construction of GUI Applications

Seiji Futatsugi[1], Masahiro Tachibana[1], Takayuki Nakano[1], Katsuhiko Yuura[1], Shoichi Kubo[2], Ichiro Kubo[2]
[1] Hitachi, Ltd., [2] Hitachi VLSI Engineering Corp.

3.2 プログラミングインタフェース

プログラミング時の操作の流れを図2に示す（網掛け部がユーザ操作）。

UIBT/RAD 環境では記述対象となるオブジェクトは視覚表現を備えている。視覚表現は時にはエディタ内に備えられたリスト内の文字列表示であり、ユーザウィンドウ内の外観(UI要素のみ)であり、コード図式内の表現である。

ユーザが各オブジェクトの視覚表現を選択すると、これに対して指示可能なメンバ関数の一覧がエディタ内に提示される。これを選択することでプログラムを構成する１ステップが入力される。

この後メンバ関数の引数の指定や、実行順序の変更、制御構造の記述などの編集を行う。

編集が終われば、入力された図式からC++ソースコードが生成される。図式からは専用のスクリプト言語も生成され、これを用いて対話実行によって画面遷移のテストを行うこともできる。

この方式により、ユーザはコーディングの負担が軽減され、コンパイルやリンク時間を節減することができる。

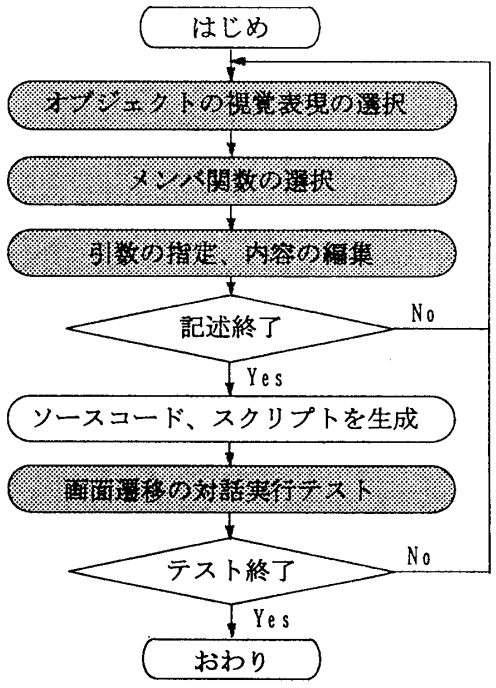


図2. プログラミングにおけるユーザ操作

3.3 コードの図式化

オブジェクトとメンバ関数、これへの引数を併せて１つの図式が作成される(図3)。個々のメンバ関数(メソッド)をデータフローに基づいて図式化するシステムとしてPrograph[2] が知られているが、本システムの方式はC++言語との親和性を考慮してコードの一文単位で図式化している。ここでメンバ関数名表示域以外はそれぞれ視覚表現であり、これらを選択することができる。またメンバ関数間の変数の共用はポート間の結線で表現する。

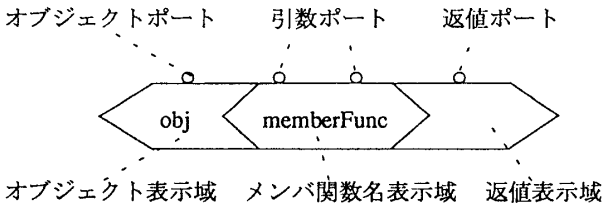
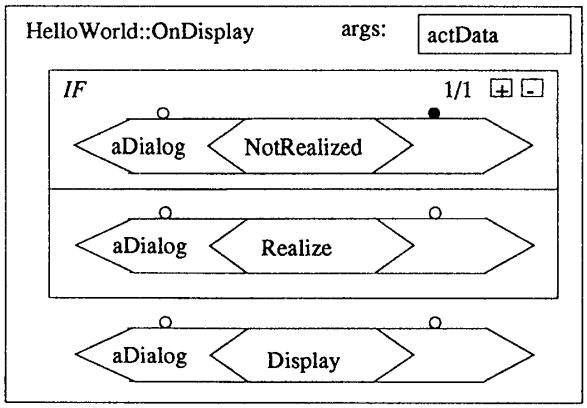


図3. コード図式化(２引数の場合)

条件文、繰り返し文などの制御構造も独自の図式で表現される。図4.にこれらを含んだメンバ関数の図式表示例と、これに等価なC++コードを示す。ここでIFと表示されたブロックの上部が条件部であり、この中の文の返値が真の値であれば、ブロック下部内の文が実行される。




```
HelloWorld::OnDisplay( Event* actData ){
    if ( aDialog -> NotRealized() )
        aDialog -> Realize();
    aDialog -> Display();
}
```

図4. コード図式化例

4. おわりに

次々と提示される視覚表現に適用するメソッドを選択していくことで、図式化されたプログラムを作成できる。ここでGUIプログラムの記述は定型化されているため、図式の意味は明確である。このためC++言語を熟知していないプログラマでも早期にウィンドウ間の画面遷移を記述できる。

5. 参考文献

[1] UIBT ﾘﾌｧﾚﾝｽ: 日立製作所 (1993)

[2] P.T.Cox, T.Pietrzykowski, Using a Pictorial Representation to Combine Dataflow and Object-Orientation in a Language Independent Programming Mechanism, Proceedings International Computer Science Conference, pp.695~704 (1988)