拡張可能なC++ソースコード・ブラウザ
—グラフィカルユーザインタフェース—

大平 剛, 三ッ井 欽一
日本アイ・ビー・エム(株) 東京基礎研究所

我々はC++ソースコード・ブラウザを開発している。C++プログラミングを効果的に支援するための多様な機能は、必ずしも全て最初から組み込むことができるわけではない。このような要求を満たすための最も重要な要素は、ブラウザが拡張性を持つことである。また、このようなツールを効率良く開発するためには、開発中の変更が容易に行なえる必要がある。そこで我々は、ウィンドウ・オブジェクトを制御するための、インタプリタ言語UIDLを導入した。ただし全ての制御をインタプリタで行うのではなく、柔軟性を求められる部分、例えばウィンドウレイアウト、メニューの追加／変更、コールバック処理、ダイアログボックスとに対話などはUIDLで記述し、実行効率が求められる部分、例えばデータベースへの問い合わせ結果の表示(リスト、グラフ)等はC++で記述している。

Extensible C++ Source Code Browser
- Graphical User Interface -

Tsuyoshi Ohira, Kin'ichi Mitsui
IBM Research, Tokyo Research Laboratory
5-19, Sanban-cho, Chiyoda-ku, Tokyo 102, Japan

This paper describes design and implementation of an extensible C++ source code browser. This browser consists of graphical user interface(GUI) components written in C++, user interface description witten in User Interface Definition Language(UIDL), and UIDL interpreter. Window layouts, menu definitions, and uses of dialog boxes are often updated during the prototyping or development of an application with a GUI. Extensibility is therefore important for prototyping and development. To achieve it, interpretation is better than compilation; however, interpreting whole processes reduces the performance. Therefore, portions that require run-time performance are written in C++ and portions that require flexibility are written in UIDL.

1 はじめに

C++は多機能で複雑な言語であり、C++プログラミングを効果的にサポートするツールの必要性が高まっている。我々はC++ソースコード・ブラウザを開発している。C++プログラミングを効果的に支援するための多様な機能は、必ずしも全て最初から組み込むことができるわけではなく、いくつかの機能は、本質的にアプリケーションごとに特化されるものと予想される。このような要求を満たすための最も重要な要素は、ブラウザが拡張性を持つことである。

ブラウザ開発中、特にグラフィカルユーザインタフェース (GUI) 開発中に使用変更がたびたび起こることがある。GUIの仕様は設計の段階では最終的な決定を下すことは困難である。GUIは直接ユーザとの対話を行う部分であるため、実際に操作してはじめて評価できるものである。そのため最終的な仕様を決定するために何度か使用感を評価することが必要となる。特にメニュー、ウィンドウレイアウト、ダイアログボックス等が変更の対象になることが多い。

GUI部分を開発中にウィンドウレイアウトの変更や機能の変更がたびたび必要な場合には、そのつどコンパイルとリンクを行うことは、開発効率が悪くなる。その場合にはインタプリタ機能で変更後すぐにテストを行えると開発効率が上がる。

我々は以上の2つ、開発後の拡張性と、開発中の変更容易性を考慮に入れ、ウィンドウ・オブジェクトを制御するための、インタプリタ言語UIDLを導入した。ただし全ての制御をインタプリタで行うのではなく、柔軟性を求められる部分、例えばウィンドウレイアウト、メニューの追加／変更、コールバック処理、ダイアログボックスとに対話などはUIDLで記述し、実行効率が求められる部分、例えばデータベースへの問い合わせ結果の表示(リスト、グラフ)等はC++で記述している。

以下では、拡張可能なC++ソースコード・ブラウザの概要と、GUIに関する特徴と実装、その拡張性について述べる。

2 ブラウザの概要

2.1 ブラウザの構成

ブラウザに求められる一般的な機能は、データベースの中のデータを、ユーザの要求に合わせて様々な形式で表示するものであると考えられる。その様々な形式は、リストであったり、グラフであったり、テキストであったりする。また表示されたものの中から選択したものをパラメータとして、新たにデータベースへ問い合わせを行える必要がある。

C++ソースコード・ブラウザは、図1のようにプログラム情報がはいったプログラム・データベース[1],[2]と、ユーザとの対話を受け持ちプログラム・データベースへの問い合わせを行い、その結果を表示するGUI[3]によって構成される。プログラム情報はC++コンパイラによって生成され、C++ソースコード・ブラウザによって読み込まれる。プログラム・データベースはPrologインタプリタによって構成され、プログラム情報はPrologのファクトとルールの形式で保存される。GUI部分はユーザの要求により、問い合わせのためのパターンを生成し、プログラム・データベースへ問い合わせを行う。結果はプログラム・データベースからGUIに送られ、ユーザの要求に応じた形式で表示される。

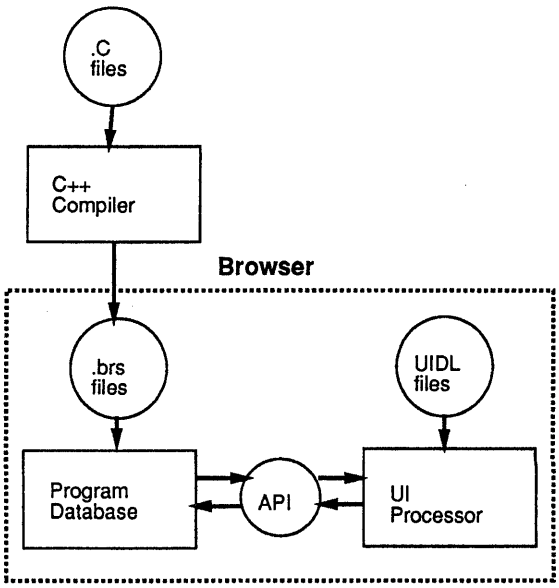

図1ブラウザの構成図

2.2 ウィンドウの構成

C++ソースコード・ブラウザのメインウィンドウは図2のようにメニューバー、複数のサブウィンドウをタイリングすることができる領域、メッセージ等を表示するための領域を持つ。各サブウィンドウは2つのコントロールバーと呼ばれる、ボタンやメニューから成る領域と、表示領域を持つ。最初のコントロールバーは図2のように各サブウィンドウに共通で、左からアイコン、搬入ボタン、自動搬入設定ボタン、フォーカス・オプションメニュー、フォーカス表示領域から成る。各機能は後述す

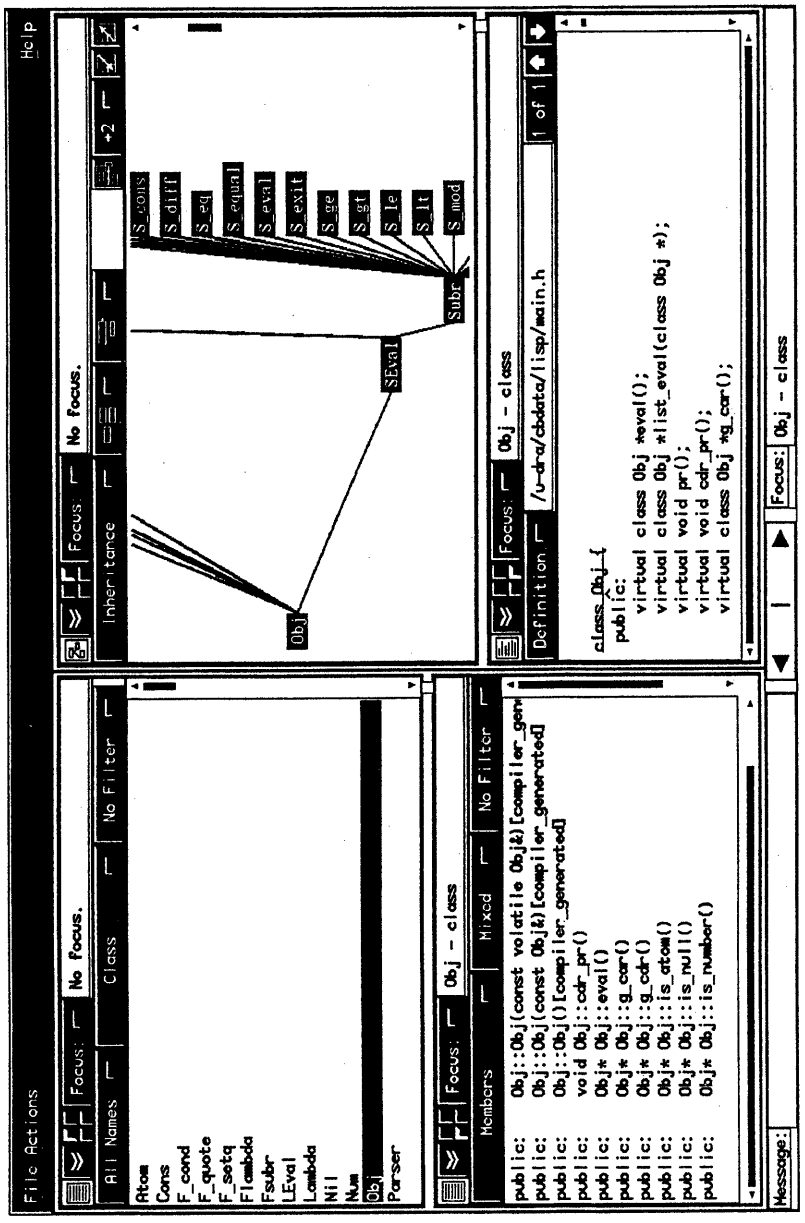

図2ブラウザのメインウィンドウ

る。次のコントロールバーは、問い合わせを作成するためのオプションメニュー群を持ち、サブウィンドウのタイプによって異なる。また各タイプに特有の機能を実行するためのボタンを持つ。

我々が2または4個のサブウィンドウによるタイリングを考えているのは、複数のウィンドウを同時に参照しながらブラウザを利用することが多いと考えられるからである。もちろん、オーバラップして任意の個数のウィンドウを開くこともできる。オーバラップのみを用意することも考えられるが、ウィンドウ間の関連付けを行なう手間、例えば一方のウィンドウで選択したオブジェクトの詳しい情報をもう一方のウィンドウで表示するなど、複数のウィンドウを利用するのが典型的であるならタイリングのほうが有利である。だだし、これは我々がブラウザを製品化するにあたって下した判断であり、あるユーザにとっては、オーバラップ方式を要求することも十分考えられる。そこで我々はユーザが、タイリングまたはオーバラップどちらでも選択できるように対応している。

2.3 サブウィンドウの機能

サブウィンドウは3つの表示形式を持ち、実行時にこの表示形式を変更することができる。起動時の表示形式は、リソース・ファイルで指定することができる。この表示形式をビューと呼び、それぞれ次のような機能を持つ。

リスト 問い合わせの結果をリスト表示する。

グラフ 問い合わせの結果をグラフ表示する。各ノードは選択可能であり、指定したノード以降のサブグラフを拡張／省略表示する。グラフ全体を拡張／縮小表示する。ノードの色指定、ラインの色／スタイルを指定する。

テキスト 問い合わせの結果をテキスト表示し、目的の場所にカーソルを移動する。任意の部分文字列を選択可能であり、その文字列をフォーカスにしたり、それがプログラム中の何であるかをデータベースに問い合わせる。エディタとしての編集機能を持つ。

各サブウィンドウの問い合わせ作成の操作は、そのビューに関係なく一般性を持っている。問い合わせは、特定のプログラム・オブジェクト（ファイル、クラス、関数、変数など）をパラメータとして指定できるものと、そのようなパラメータを持たないものに分けられる。例えば、前者の例は、全てのクラス。後者の例は、あるクラスのメンバ関数。このパラメータのことをフォーカスと呼び、各ビューは、次の2つの状態を持つ。

- フォーカスを持たない
- フォーカスを持つ

いずれのビューのサブウィンドウでも、特定のオブジェクトを表示している矩形領域を選択することができる。例えば、グラフではノード、テキストでは任意の部分文字列がこれに相当する。選択されたオブジェクトは、後の問い合わせのパラメータとして利用することができる。

各サブウィンドウは、問い合わせを作成するためのメニュー群（オプションメニュー、ポップアップメニュー）を持っている。メニュー選択の組み合わせにより、Prologで記述されたデータベースへの問い合わせ要求が生成され、実行される。

各サブウィンドウに付属する搬入ボタンを押すと、前に選択されたオブジェクトがそのサブウィンドウのフォーカスとなり、問い合わせ作成のためのパラメータとして利用することができる。このように一連の問い合わせ動作は、サブウィンドウのビューやメインウィンドウ上の場所に独立なので、ユーザにとって理解しやすい。

オプションメニューの組み合わせはフォーカスの有無やフォーカスのタイプによって、自動的に切り替わる。ポップアップメニューはサブウィンドウ上での選択の有無や、選択されたオブジェクトのタイプによって自動的に切り替わる。これらの機能により、サブウィンドウは搬入されたフォーカスのタイプに合わせて、ブラウズ機能を動的に変化させる。

例えば、リスト・ビューにクラスが搬入された場合には、メンバ関数の一覧が表示され、メニューは以下のようにメンバ関数特有のものになる。

・Public/Protected/Private の各属性を持つメンバを表示。

・ベースクラスを含めてメンバを表示。

・Virtual/Pure Virtual/Non Virtual の各属性を持つものを表示。

メンバ関数が搬入された場合には、その関数が呼び出している関数が表示され、メニューは以下のように切り替わる。

・呼びだしている関数の表示

・呼ばれている関数の表示

各ビューの表示されるものも、フォーカスのタイプによって切り替わる。

例えば、グラフ・ビューにクラスが搬入された場合には、インヘリタンス・グラフが表示され、関数が搬入された場合には、コール・グラフが表示される。

このブラウザの特徴は、特定のオブジェクトを表示するためのウィンドウがそれぞれ用意されているのではなく、1つのウィンドウが、フォーカスとして指定されたオブジェクトのタイプによって、自動的に変化すること

である。この機能により、1つのウィンドウを動的に、目的別に使い分けることができる。

2.4 ウィンドウ間の関連

各サブウィンドウは搬入ボタンを押すことでフォーカスを設定するが、ボタンを押さずにフォーカスを指定することもできる。2つのウィンドウをリンクして、一方で選択されたオブジェクトを自動的に他のサブウィンドウに搬入する機能を持つ。例えばリスト表示されたクラスを選択たびに、他のウィンドウにそのクラスのメンバ関数が表示される。

各サブウィンドウはメインウィンドウ上のサブウィンドウの位置を表す相似形のボタン群を持ち、これを設定することで、自動的な搬入を実現する。このボタンを自動搬入設定ボタンと呼ぶ。図1では、メインウィンドウ上には4つのサブウィンドウが、2 x 2の配列で置かれている。各サブウィンドウのコントロールバーには、この位置関係を表す2 x 2の自動搬入設定ボタンがある。各サブウィンドウ上で選択されたオブジェクトは、この自動搬入設定ボタンで指定されたサブウィンドウへ搬入される。この設定は各サブウィンドウ上で、独立に設定できる。この機能により、独立していたサブウィンドウ間に、主従関係を持たせることができる。

3 実装

C++ソースコード・ブラウザは、OSF/MotifをC++のクラスでカプセル化したクラス・ライブラリと、それらのクラスのオブジェクトを生成し制御するインタプリタ部とから構成される。オブジェクトの生成や制御のコードは、ユーザインタフェース記述言語UIDLで記述される。

ユーザ自信がブラウザのGUIを容易に変更できるようにする。例えば、新しい問い合わせのパターンを考案した場合、対応するメニューを付け加え、メニューの選択に対して適当な問い合わせを実際にデータベースに行い、結果をサブウィンドウに表示する、という記述をユーザが行うことができる。この記述は実行時にシステムに読み込まれ実行される。従って、もとのシステムを変更すること無しにGUIの挙動を変えることができる。

よく言われるように、GUIはオブジェクト指向に基づく設計が適している。我々のブラウザでも、サブウィンドウ、メニュー、ダイアログボックスといった基本部品はC++のオブジェクトとして実装されている。UIDLでは、このC++オブジェクトを実際に生成し、それらのオブジェクトの必要な機能をC++の外から呼び出すことを可能にする。言い替えると、ユーザは、基本機能を持った部品としてのオブジェクトをUIDLで組み合わせることによりGUIを構築する。

3.1 ユーザインタフェース記述言語

ユーザインタフェース記述言語UIDLは基本的にはS式で記述される。言語上の基本機能は、

- クラスの定義
- インスタンスの生成
- メッセージ送信

といったオブジェクト指向の基本機能を持つ。またUIDLで扱えるタイプには以下のものがある。

・アトム
・文字列
・数値
・オブジェクト
・リスト

文法規則は、ここでは重要ではないので、例を参照することで省略する。

例. クラス定義とインスタンス生成とメッセージ送信

```
%% class definition %%
(class sub-window (super window)
  % instance variables
  (attribute
    (var1 nil)
  )
  % method definitions
  (action
    (method1 (p1 p2) (body))
  )
)
%% instantiation %%
(setq obj (new sub-window))
%% message sending %%
(send obj 'method1 a1 a2)
```

クラス定義のなかでは、attribute以下でインスタンス変数を宣言し、action以下でメソッドを定義する。newはインスタンスを生成するための関数。snedはオブジェクトにメッセージを送信する関数。

UIDLはC++のオブジェクトを制御するための特徴的機能として、

- C++のクラスをUIDLのクラスと結合
- C++オブジェクトのメソッドを呼び出し

- C++側から UIDL 側のオブジェクトへメッセージを送信

を持つ。

特徴的機能を実現するために、UIDL インタプリタとの対話を司る C++のクラス U_object を用意した。UIDL から制御するクラスはこのクラスをベース・クラスとして継承する必要がある。このクラスは以下のプロトコルを用意し、C++クラスの作成者は UIDL から制御できるように、これらを定義する必要がある。

- オブジェクトを生成するための static メンバ関数 New。UIDL 上でのクラス名とこの New を結合し、登録しておく。

- C++のメンバ関数をインタプリタから呼び出すために使われる仮想メンバ関数 call。メッセージのディスパッチを行なう。

オブジェクトの生成は UIDL 上であるクラスに対して new が呼ばれると、そのクラス名に結合している C++クラスの New 関数が呼ばれる。以後この UIDL オブジェクトを通して C++のオブジェクトが制御される。

UIDL のメソッドを呼び出すと、対応する C++クラスのメンバ関数 call が呼び出さる。call はパラメータとしてメッセージ名とパラメータのリストを持ち、C++クラスの作成者はメッセージの振りわけを行ない、対応するメンバ関数を呼び出すように call を実現する。

イベント処理に関しては、あらかじめ C++クラスが用意しているコールバックに UIDL で利用可能な名前が割り当てられており、イベントが発生すると対応する名前が、メッセージとして UIDL のオブジェクトに送信される。従って、UIDL プログラマは必要なコールバック名と同じ名前のメソッドを定義することで、イベント処理を実現できる。

例えば、グラフ・ビューはノードのクリックのコールバックを用意している。

```
(class graph-view
  ...
  (action
    ...
    (node-click (id) (body))
  )
)
```

あるノードがクリックされると node-click が呼び出され、ノードの ID が引数として渡され、ボディが実行される。

UIDL には組み込みのクラスがあり、次の節で述べる C++クラスと結合されている。これらのうちビューはデータベースへの問い合わせが可能で、その結果をそれぞれのビューに表示する。

例えば、リスト表示には list-items メッセージをリスト・ビューのオブジェクトの送る。

```
(setq goal
     ''class_decl(ID,Name,Kind,Access)'')
(setq format ''Name'')
(setq arrt '(ID Name Kind))
(send a-list-view 'list-items
                  goal format attr)
```

goal が Prolog に対する問い合わせであり、大文字で始まる名前は Prolog の変数に対応する。format はリスト項目の表示形式の指定、attr はリスト項目に登録するアトリビュートの指定である。このアトリビュートは項目選択のコールバックに引数として渡される。

グラフ表示には、add-arc,layout メッセージをグラフ・ビューに送る。

```
(setq goal ''inherit(PID,PName,CID,CName)'')
(setq parent '(PID PName))
(setq child  '(CID CName))
(setq parent-attr '(PID))
(setq child-attr  '(CID))
(send a-graph-view 'add-arc
    goal parent child parent-attr child-attr)
(send a-graph-view 'layout)
```

リストと同様に goal は Prolog に対する問い合わせ、parent,child でノードのリンク情報を渡す。add-arc はノード情報を蓄えるだけで、layout でグラフのレイアウトを決定し表示する。

メニューの定義は、例えばメインウィンドウは add-menu メソッドを用意してる。

```
(setq a-menu
  '((pulldown action-menu ''Action''
      (menu open-menu ''Open'')
      (separator)
      (menu close-menu ''Close''))
    (pulldown help-menu ''Help''
      (menu help-on-context ''On Context'')
      (menu help-on-help ''On Help''))))
(send a-main-window 'add-menu a-menu)
```

メニュー定義の中の pulldown, menu, separator はキーワードとして使われ、キーワードの次の名前は各メニュー

のIDとして使われ、メニューのコールバックで引数として渡される。

3.2 GUIライブラリ

ライブラリは大きく分けて次に2つの部分に分けられる。

- OSF/Motifのウィジェットを C++のクラスでカプセル化
- これらのクラスを利用して、高機能なグラフやテキストのビューを実現した部分

OSF/Motifの widgets を C++のクラスでカプセル化する話しは [5] でも述べられているが、プリミティブなウィンドウクラスを用意するだけでは十分ではない。良く利用されるような高機能なクラスもライブラリに用意すべきである。我々の場合は、ブラウザにとって有効なクラスを用意した。

高機能な部品としては、以下のものを用意した。

メインウィンドウ サブウィンドウをタイリングするための機能、メニューバーを作成する機能、メッセージ領域にメッセージを表示する機能。

サブウィンドウ コントロールバーにオプションメニューを登録する機能、オプションメニューを切り替える機能、ポップアップメニューを登録する機能、ポップアップメニューを切り替える機能、ビューを切り替える機能。

リスト・ビュー 問い合わせ結果をリスト表示する機能。

グラフ・ビュー 問い合わせ結果をグラフ表示する機能、グラフのズーム機能、グラフのオーバービュー表示機能。

テキスト・ビュー 問い合わせ結果をテキスト表示する機能、テキスト上の任意の位置にカーソルを移動する機能、エディタとしての編集機能。

各種ダイアログボックス メッセージ・ダイアログボックス、ファイル選択機能、テキスト入力機能。

3.3 UIDLとC++との協調

この実現方法の特徴は、実行効率の良さを求められる部分は、C++ クラスとして実装し、その制御はUIDLインタプリタ部で行うことである。例えば、グラフ・ビューのようにグラフのレイアウトやズーム機能はC++で実現されている。これは実行効率を求められる部分だからであり、メニューと組み合わせて、問い合わせを作成し結果を受け渡す部分は、UIDL で記述される。従って、C++のコードとUIDLのコードが協調することで、効率性と柔軟性を実現できる。

Tcl[4] のようにプリミティブなウィンドウ・オブジェクトだけ用意して、すべてを Tcl で記述するようなアプローチでは、高機能を記述するには限界があり、効率面でも問題があると思われる。

4 拡張性

C++ソースコード・ブラウザとしてエンド・ユーザに可能な変更は、サブウィンドウの数の指定、つまりオーバラップ方式ならサブウィンドウは1つであり、タイリング方式ならサブウィンドウは2つ、または4つ。各サブウィンドウのデフォルトのビューを指定できる。またビューは実行時に切り替えることができる。自動搬入ボタンを設定することで、サブウィンドウ間の関連付けを変更することができる。この機能により Smalltalk のようなブラウザにすることも可能である。

4.1 UIDLでの変更

UIDL レベルで可能な変更には以下のものがある。

ウィンドウレイアウトの変更 任意の個数のサブウィンドウをタイリングできる。また任意の個数のウィンドウをオープンできる。

メニューと問い合わせの変更 メニューを定義することができ、Prolog で記述された任意の問い合わせと結合することができる。これにより、新しい問い合わせを容易に追加できる。

ウィンドウ間の関連付けの変更 任意にウィンドウ間の関連付けを記述できる。フォーカスの搬入方法を定義できる。

コールバック処理の変更 コールバック処理を定義できる。ノードやリストの項目をクリック、ダブルクリックした場合の振舞いを変更できる。

4.2 GUIライブラリの拡張

U_object クラスを継承することで、UIDL から制御可能な任意のクラスを作成することができる。これにより新しいビューを考案した場合、それを実現した C++クラスを容易に UIDL から利用可能にすることができる。現在は、call 関数の定義は全てユーザが行なうが、ある程度の自動化が可能と思われる。

4.3 その他の拡張性

ブラウザはPrologで記述された問い合わせと、データベースの内容を変更することで、C++以外のブラウザに変身することができる。従って、ジェネリック・ブラウザとしての利用が可能である。また、デバッガーとの連携により、デバッガーのGUIとしても利用可能である。

5 評価（考察・議論）

ユーザは各サブウィンドウのビューを動的に変化させたり、フォーカスのタイプのよってブラウズする対象を変化させられることは、新しくブラウズ・ウィンドウをオープンする手間を省略することができ便利だと思われる。その反面、同じウィンドウを使い回すので、以前に表示していたものを見たいとき、それを表示するために行なった操作をやり直す必要がある。ヒストリ機能が求められる。

問い合わせメニューの追加／変更、あるいはshellの別コマンドを呼び出すなど、このブラウザのGUIの基本的な枠組みを崩さない程度のカスタマイズならユーザにとって十分容易に可能であると考えられる。我々にとって、多くの変更がこの記述言語で行なえたので、コンパイル・リンクの回数を減らし、開発のサイクルを速める上で有効であった。

サブウィンドウの数を変えたり、ボタンの位置を変えたり等のレイアウトに関して高い柔軟性を持たせた場合には、比較的多くの記述を必要とする。我々の記述言語は、任意のウィンドウ・レイアウトが可能であるほど完全ではない。インタフェース・ビルダを利用してGUIを構築したり、その評価をする方法がある。インタフェース・ビルダは静的なウィンドウレイアウトを作成するのには便利であるが、動的にウィンドウレイアルトが変化するような場合には、十分対応しているとは言えない。その点インタプリタ方式では動的変化に対応できる。イベント処理はテスト・モードである程度実行することができるが、最終的にはコンパイル言語のプログラムを記述しなければならない場合が多い。インタフェース・ビルダとインタプリタの組合せは、GUI構築、評価にとっては強力であり、さらにコンパイル言語プログラムへの変換、逆変換機能があればかなり強力だと考えられる。

我々のブラウザは、C++以外にも、Fortran、COBといった他の言語のブラウザとしても利用されている。また実行時のオブジェクト・ツリーのブラウザとしても利用されている。このことは、ジェネリック・ブラウザとしての可能性も示唆していると思われる。

参考文献

[1] 三ッ井欽一, 久世和資, Shahram Javey: 拡張可能なC++ソースコードブラウザ - 基本設計, 第45回情報処理全国大会, 7Q-05, 1992.

[2] 中村宏明, 安田和, 三ッ井欽一, Shahram Javey: 拡張可能なC++ソースコードブラウザ - プログラムデータベース, 第45回情報処理全国大会, 7Q-06, 1992.

[3] 大平 剛, 三ッ井 欽一, Shahram Javey: 拡張可能なC++ソースコードブラウザ - ユーザインタフェース, 第45回情報処理全国大会, 7Q-07, 1992.

[4] John K. Ousterhout: An X11 Toolkit Based on the Tcl Language, USENIX - Winter '91 - Dallas, TX

[5] Douglas A. Young: Object-Oriented Programming with C++ and OSF/Motif, PRENTICE HALL