オブジェクト指向型ビジュアルプログラミング環境の構築

6W−03

真柄喬史† 武田正之‡

‡東京理科大学
理工学部情報科学科

1 はじめに

本研究はGUIによるオブジェクト指向型スクリプトインタプリタ環境の構築を目的とし、スクリプトの記述及び実行と、これをシェルとして利用するための機能を、統一されたユーザインタフェースで提供する。この環境上ではオブジェクトはアイコンとして表示され、これに対するユーザーの操作からダイレクトに構文木が構築される。実装はJavaで行い、言語機能をJava言語のサブセットとする。

2 スクリプトインタプリタについて

一般にスクリプトインタプリタには以下のような利用形態が存在する。

- 小規模のプログラムの記述
- コンポーネント同士を結びつける接着剤(Glue)としての利用
- 既存のプログラム内から呼び出しての利用 これによってプログラムに柔軟性、拡張性を与えることができる。
- シェルとしての利用 ユーザーの対話的な操作により行われる。

Javaで記述され、JavaのAPIを利用することのできるスクリプトインタプリタとして、pnuts [1]、BeanShell [2]等がある。双方とも文法は独自のものであり、記述を簡潔に行うためのsyntax sugerをいくつか持っている。

2.1 シェルとしての利用

現在、”ファイルと外部プログラムに関する操作を主体としたGUI及びCUIのシェル”が広く利用されているが、これに対し上記のpnuts/BeanShellをシェルとして見たときに、これは”Javaオブジェクトの操作を主体としたCUIベースのシェル”であると言うことができる。

本稿では”Javaオブジェクトの操作を主体としたGUIのスクリプトインタプリタ”を提案する。GUIとすることにより得られる利点として記述の簡易化、対話性の強化があげられる。具体的には、

- 変数の識別子としての文字列が不要になる。これにより 変数名を与える必要がなくなる。(変数の自動命名化)
- 使用するクラス、メソッド等のメニューからの選択が可能になる。
- ローカル変数の値を常に表示しておくことが可能になる。

等である。

3 内部構造

言語機能はJava言語のサブセットとする。また、記述したコードのJavaコードへの変換をサポートする。これによりJavaプログラミングの経験者にとって習得が容易になり、また記述したコードをJava言語のコード資産として利用することが可能になる。

3.1 構文木の仕様

Java言語（フル仕様）の構文解析木を扱うAPIとしては、javacの構文木パッケージ(sun.tools.tree)があるが、今回作成した構文木の仕様は以下の点を除けばjavacの構文木と大きな違いのない、一般的な構造に従っている。

- インタープリティングの機能を持つ。
- バイトコードは出力せずJava言語のコードを出力する。
- Java言語のコードからの生成ではなく、ユーザーの操作からのダイレクトな生成を前提にしている。

Construction of Object-Oriented Visual Programming Environment
Takafumi MAGARA†, Masayuki TAKEDA‡
‡Tokyo University of Sceience

3.2 構文木要素

構文木を構成するノードとして以下のものを実装している。

- 変数アクセス式（ローカル、インスタンス、クラス変数の参照及びこれに対する値の代入）
- メソッド、クラスメソッド、コンストラクタの呼び出し式
- プリミティブ型変数に関する演算式
- **コードブロック文** 下位にノードを追加していくことが出来るノード。新たなノードはすべてこの下に追加されていく形になる。
- **制御文 (if, for, while)** スクリプト記述モードでのみ利用できる。（後述） 下位に条件式（型がbooleanである式ノード）等と、条件に応じて実行されるコードブロックを（if-then-elseであれば2つ）持つ。

4 ユーザーインタフェース

シェルモード及びスクリプト記述モードの2つのモードを設ける。シェルモードではユーザーの操作ごとにその処理（式の評価）を実際に行い、結果（式を評価した値）を表示する。スクリプト記述モードではユーザーの操作から構文木が構築されるのみで、実際の処理は行われない。また、このスクリプト記述モードではif、whileなどの制御文を利用することが可能である。

GUIベースとすることで、テキスト主体の環境と比べて記述したコードの見通しが立ちにくくなるという問題がある。これに対する解決策として、この環境上では構文木の視覚的な表示、及びこれのJava言語への変換・表示を行う。以下で全体のUIの構成要素とその役割について述べる。

- **ローカル変数パネル** UIの中心部分となる。ローカル変数がアイコンとして表示され、これに対する操作がメニューで提供される。具体的にはメソッドの呼び出し、フィールドの参照、フィールドへの代入などである。メソッドの引数の指定はマウスのドラッグ&ドロップで行う。実行結果に対して、自動的に新たなローカル変数が宣言され代入が行われる。基本的な操作はシェルモードとスクリプト記述モードで共通である。
- **構文木パネル（図1）** 記述中の構文木を視覚的に表示を行う。構文木要素が追加されれば、即時にこの表示に反映される。要素は上から評価が行われる順で表示されることになる。また、コード内での移動（ノードの挿入位置の変更）もこのパネル上で行うことが出来る。

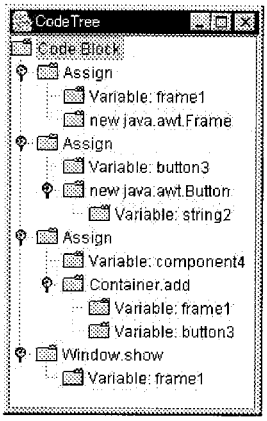

図 1: 構文木パネル画面

- **Javaコードパネル** Javaコードへの変換結果を表示する。ここでも構文木要素の変更は常に表示に反映される。

5 おわりに

本稿ではこのスクリプト環境の概要を述べた。今後、この環境に対して以下のような機能追加を行うことを検討している。

- **構文木の編集** 現時点では構文木へのノードの挿入のみが可能であり、ノードに対する変更や削除（余分なローカル変数の削除など）を行うことができない。
- **Javaコード出力及びクラスの記述機能の強化** 現在のJavaコードの出力は断片的なものであり、新規のクラスのクラスメソッドとしての形をとっている。今後インスタンス変数の宣言、及びインスタンスメソッドの記述等をサポートしたい。

参考文献

[1] http://javacenter.sun.co.jp/pnuts/

[2] http://www.beanshell.org/