オブジェクト指向言語 COOL とその開発環境の構築†

1K－5

杉山 潤，千葉 幸男，越田 一郎†
東京工科大学†

1 はじめに

本研究は，バグが少なく品質の高いソフトウェアを，従来よりも短時間で開発するプログラミング言語と，開発を支援するツールの作成を目的としている．本稿では，言語 COOL(Cool is Object Oriented Language) の特徴と開発ツールの構築方法について述べる．

- 品質の高いソフトウェアの開発方法の検討
- プログラムの表記の品質への影響の考察
- 柔軟性の高い開発環境の構築
- COOL 開発環境の設計・構築

2 従来の開発環境におけるツール構成

従来の開発環境では，Lisp など一部の処理系を除いて，以下のように各ツールが分散して存在していた．

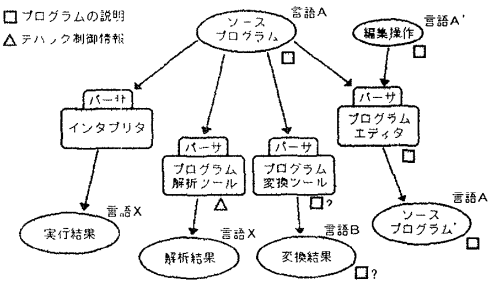

図 1: 一般的な開発環境におけるツール構成

このように，開発ツールが独立して存在している構成では，以下のような欠点が考えられる．

- ツールごとにパーサや内部表現を実装しなければならない．
- ツール間の連携が不十分であり，情報の授受に一貫性がない．
- コメントの利用方法に統一性がないため，コメントの内容が保証されない．
- デバッグの制御情報が保存されない．

3 COOL 開発環境の構成

前節で挙げられた問題点を解決するために，図2のような構成の開発環境について，必要な言語機能やツールの実装方法の検討を行った．

†An object oriented language COOL and the development environment for COOL

†Jun Sugiyama , Yukio Chiba , Ichiro Koshida
†Tokyo Engineering University,
1404-1, Katakura, Hachioji, Tokyo, Japan

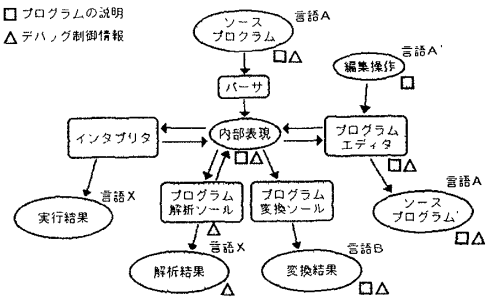

図 2: COOL 開発ツールの構成

3.1 あらかじめ用意する開発ツール

言語処理系 COOL で用意された機能を最大限に活用するため，プログラムの整形出力や，デバッグのサポートを行う開発ツール群を，Java 上で実装する．

- プログラム整形出力ツール
  COOL プログラムを標準のスタイルで出力するツールと，Java のソースに変換するツールも用意する．後者のツールは，バインドされたコメントを，javadoc で処理できる形式に変換し，信頼性の高いドキュメントの生成を行う際にも利用する．
- プログラムエディタ
  プログラマが入力した編集操作に対し，プログラムの内部表現を直接変更する．入力時には，関数や変数などの識別子を補完する機能も，実装している．ファイルに書き込む際は，プログラム整形ツールを利用する．
- デバッグ支援の機能
  デバッグ情報の埋め込み機能を利用し，デバッグ情報出力を細かく制御する．そのためのデバッグクラスを，処理系側で用意する．

なお，コメントのバインドやデバッグ情報の埋め込みについては，後で詳しく述べる．

3.2 内部表現を用いた方法の利点

図2のようにツール群を構成することにより，以下のような利点が考えられる．

- パーサの共有により，ツールの実装が楽になる．
- 内部表現を介することで，ツール間の情報伝達が確実になる．
- デバッグ情報やコメントなどの付加的な情報を，処理系側で制御できるようになる．

4 COOLインタプリタの実装

前節で述べた形のツール構成を実現するためには，補助的な情報を記述するための構文や，内部表現へのアクセスを，言語処理系側でサポートする必要がある．

4.1 拡張された言語機能

COOL の文法は，Java とほぼ同じであるが，機能拡張を行なった部分については，それを記述するための構文が追加されている．

コメント指示子

特定のプログラムの構成要素に対して，コメントを結び付ける (バインド) ための構文であり，以下のように記述する．

<コメント対象>@ (<コメント>)

この構文を利用することにより，コメントのスタイルが統一されるため，その意味も分かりやすいものとなる．そして，Java のドキュメンテーションコメントよりも，コメントの内容が信頼できるものとなる．

デバッグ情報指示子

デバッグを制御するための情報や，デバッグ時にのみ実行するコードを記述するための構文である．

<デバッグ対象>#(<条件>, <デバッグコード>)

デバッグ情報指示子により，デバッグのためのコードを，通常のコードと明確に区別することが可能となるだけでなく，インタプリタが必要に応じて，それらのコードの制御を行うことができる．

4.2 インタプリタの動作

アプリケーションプログラムは，ソースプログラムの双方向的な読み書きが行えるインタプリタ[1]によって動作する．COOL で記述されたプログラムは，COOL インタプリタによって内部表現に変換され，その内部表現を元に実行や情報のやり取りを行い，必要に応じてソースファイルの形式で出力することもできる．

4.3 内部表現がもつ情報

プログラムの内部表現は，大きく分けて 7 種類のオブジェクトによって構成される．COOL インタプリタによって，プログラムが内部表現に変換されても，開発ツール間で情報を共有するためには，ソースプログラム内に記述された情報を，内部表現においても保持しなければならない．

一般に，プログラムの構成要素ごとに，管理すべきデータが変わってくるので，COOL インタプリタでは，それらの情報を表 1 のように管理している．

[1] インタプリタという表現は適切でないかも知れない

表 1: 構文単位ごとに管理される情報

管理される情報	構文単位 (宣言や定義)				
	クラス	メソッド	変数	文	式
クラス	○	×	×	×	×
メソッド	○	○	×	×	×
変数	○	○	○	×	×
文	×	○	×	○	×
式	×	×	○	○	○
デバッグ	○	○	○	○	○
コメント	○	○	○	○	○

○は情報が保存されるが，×は保存されない．

5 まとめ

Java を拡張した言語や開発環境は多数存在するが，ドキュメント自動生成やデバッグ支援のための機能については，プログラムの動作に直接結び付かないためか，十分なサポートが行われているとは言えない．

本システムでは，開発ツール間の情報の受け渡しメカニズムが用意され，ツール間の連携が取りやすくなったと言える．これにより，ソフトウェアの品質に大きく関わる，コメントやデバッグ制御などの情報を，有効に活用することが可能となった．

また，開発ツールを COOL で記述することにより，ツール自身の内部表現の操作を行うことで，柔軟なカスタマイズが可能となる．しかし，内部表現に対するアクセスを，制限するための構文が用意されてないため，カスタマイズすべきでない部分の変更ができてしまう，などといった問題点も残されており，そのための機能を実装する必要もあると言える．

参考文献

[1] 杉山, 越田: 制御情報の埋め込みが可能な言語の作成とビジュアルプログラミングへの応用, 情報処理学会第 51 回全国大会 論文集, 1995

[2] 杉山, 越田: バインダによるモジュール間アクセスの制御とその応用, 情報処理学会第 52 回全国大会論文集, 1996

[3] Java API Documentation 1.0.2, Sun Microsystems, 1996

[4] The Java Language Specification Version 1.0, Sun Microsystems, 1996