2G−10

開発過程記述言語ＰＤＬによるＰＤＬ処理系管理スクリプトの記述

岡田　世志彦　飯田　元　井上　克郎　鳥居　宏次
大阪大学　基礎工学部

1．　はじめに

近年，ソフトウェアの開発過程を形式的に記述しようという試みが盛んに行われている．開発過程を形式的に記述することにより，開発工程の管理を容易に行ったり，ソフトウェアの品質をある水準以上に保つことが期待される．

我々は，開発過程記述言語ＰＤＬ (Process Description Language)[5] を設計し，また，ＰＤＬによって記述された開発過程を実行するための，ＰＤＬインタプリタなどの処理系を作成した．

今回，Ｃ言語で書かれたＰＤＬ処理系の保守や改良を既存のツールを統合して用いることで効率よく行うための支援系をＰＤＬ自身で記述した．この記述（スクリプト）を実行すると，各ソースプログラムの任意のバージョンを取り出して，エディットやコンパイルを行うことができる．また，行うべき適切な作業をメニューから選択できる．このスクリプトはＰＤＬの実際的な応用例のひとつで，ＰＤＬの有効性を示すひとつの実例といえよう．本稿では，このＰＤＬ処理系管理スクリプトの記述方法とその内容について述べる．

2．　ＰＤＬとその処理系の概要

2.1　開発過程記述言語ＰＤＬ

ＰＤＬ (Process Description Language) は，代数的仕様記述言語ＡＳＬ[2] に基づく関数型言語であり，ＡＳＬの

・言語の意味が簡明に定義されている
・様々な抽象化レベルでの抽象化ができる
・どの抽象どの記述も，同一の意味定義の枠内で記述できる
・記述の正しさを検証することが比較的容易である

といった性質を引き継いでいる．また，段階的詳細化によるプログラム開発を関数定義の追加という形で容易にすすめることができる．

さらに，ＰＤＬは基本関数としてツールの起動，ウィンドウのオープン，クローズなどの操作等を行う基本関数を備えている．これらの関数を組み合わせて開発過程の支援プログラム（以後，スクリプトと呼ぶ）を記述できる．

2.2 ＰＤＬの処理系

ＰＤＬの処理系としてＰＤＬインタプリタ[1][3][4]が存在する．このインタプリタには抽象的な記述から得られた未定義関数を含むようなＰＤＬスクリプトでも実行できる機能を有する．インタプリタはスクリプトの実行が未定義関数の評価に及んだ場合，実行を一次中断して，ユーザが，その関数の評価値を与え実行を再開したり，その関数を定義し，その後実行を再開するなどの動作を選択することができる．

また，デバッグ機能として，ブレーク関数による中断機能やトレース機能，ウィンドウシステムを用いたデバッグインターフェースなどの多数の機能が用意されている．

3．　ＰＤＬ処理系の管理スクリプトの作成

3.1　管理機能の必要性

生成物の管理はソフトウェア開発支援系の果たすべき大きな役割である．必要な生成物を保管場所から取り出したり，その編集の履歴を保存する，あるいは，他の生成物の更新に伴った作成，更新を行うといった作業は開発過程において，重要な意味を持つことが多いと思われる．

そこで，今回は管理機能として，

・ＰＤＬ処理系のソースに対するバージョン管理
・ソースの更新に伴う新しいＰＤＬ処理系の生成

という２つのことに重点をおいて考えた．これらの個々の機能は，これまでにツールとして実現されている．しかし，これらのツールを用いるためには，

・そのようなツールの存在を知っていなければならない
・正しい使用法を学ばなければならない

といった問題がある．そこで，これらのツールの正しい使用手順をＰＤＬスクリプトとして記述しておけば，細かい知識のない初心者でも，有効にツールを活用できる．

また，これらの管理を効率よく行うために，バージョンの取り出しや，エディット，コンパイルなどといった作業の選択，および各作業で対象とする生成物（ファイルなど）の選択をマウスで操作するメニューによって，行えるようにした．

3.2　ＰＤＬ処理系のソースに対するバージョン管理

バージョン管理を行うツールとしてＵＮＩＸではＳＣＣＳ (Source Code Control System) やＲＣＳ (Revision Control System) が提供されている．ここではＳＣＣＳを用いた．

ＰＤＬではＵＮＩＸのツールを関数ｅｘｅｃを用いて利用することができる．本スクリプトでは，この関数ｅｘｅｃを用いてＳＣＣＳコマンド群を操作し，ＳＣＣＳコマンドを特に知らなくても，それを扱えるようなインターフェースを実現した．さらに，ＳＣＣＳで提供されている様々な機能のなかからメニューを用いて必要な機能の選択を行えるようにした．メニューを用いることで，開発者はどのような機能が用意されているかを一目で知ることができ，また，各機能を正しく操作できる．

Description of Maintenance Script for PDL System
Yoshihiko OKADA, Hajimu IIDA, Katsuro INOUE, and Koji TORII
Faculty of Engineering Science, Osaka University

3.3　ソースの更新にともなう新しいＰＤＬ処理系の生成

あるソースプログラムの内容を変更，更新した場合，次の段階として考えられるのは，それをコンパイルして新しい実行ファイルを生成することである．ＰＤＬの処理系についていえば，そのことは新しいバージョンのＰＤＬの処理系の生成を意味する．

単一のソースからなるプログラムであれば，それを単にコンパイルすれば問題はないが，一般に大きなプログラムを作成する場合には，それを小さなプログラムに分割してコンパイルし，そのそれぞれをリンクし，実行形式のプログラムを生成しなければならない．

このような作業を効率よく行うツールとして，ＵＮＩＸには，make というコマンドが用意されている．

make は，makefile 中に記述されたファイル同士の関係定義に基づいて効率的なコンパイルを行い，生成されたオブジェクトファイルをリンクして，実行形式のプログラムを生成する．

具体的には，オブジェクトファイルとそのソースファイルのタイムスタンプを比較し，ソースファイルのほうがオブジェクトファイルよりも新しければ，ソースファイルが更新されたものとみなして，そのソースをコンパイルする．また，makefile 中で定義された依存関係に基づき，あるヘッダファイルが変更されれば，それを含んでいるソースの再コンパイルを自動的に行う．

今回は，このような機能を make を用いずにＰＤＬのスクリプトで実現した．このスクリプトでは自動的にファイルの依存関係を調べているので，makefile のようなものを記述する必要はない．

さらに，前節で述べたバージョン管理の機能を組み合わせることで，コンパイル時のソースのバージョン指定を可能にする働きを持つスクリプトを作成した．（図１）また，スクリプトの一部を（図２）に示す．

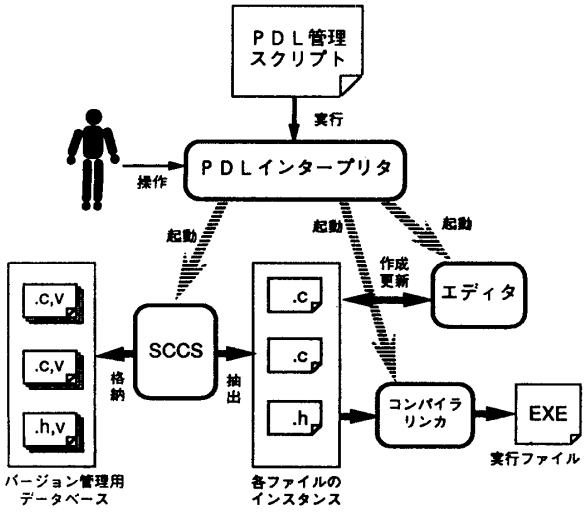

図１　管理機能の概要

```
sccsedit(clist, ilist, flist, S) ==
if menu(Title2, clist, S):t = "newfile" then
        sccsedit(clist, ilist,
                element1(newfile(flist, S):u), element2(u))
else if t = "getedit" then
        sccsedit(clist, ilist, flist, getedit(flist, S))
else if t = "edit" then
        sccsedit(clist, ilist, flist, edit(flist, S))
else if t = "save" then
        sccsedit(clist, ilist, flist, save(flist, S))
else if t = "unsave" then
        sccsedit(clist, ilist, flist, unsave(flist, S))
else if t = "rmver" then
        sccsedit(clist, ilist, flist, rmver(flist, S))
else if t = "chcomment" then
        sccsedit(clist, ilist, flist, chcomment(flist, S))
else if t = "rmall" then
        sccsedit(clist, ilist,
                element1(rmall(flist, S):v), element2(v))
else if t = "infomation" then
        sccsedit(clist, ilist, flist,
                infomation(ilist, flist, S))
else write("EXIT SCCS\n", S);

mcompile(flist, S) ==
if flist = []
then S
else if element2(head(flist)) = TRUE
  then if time_stamp(element1(head(flist)), S)
 > time_stamp(basename(element1(head(flist))) + ".o", S)
    then if status(Compiler(element1(head(flist)), S):t) = 0
        then mcompile(tail(flist), t)
        else t
    else mcompile(tail(flist), S)
  else mcompile(tail(flist), S);

mklist(str, bool, S) ==
        [stol(stripnl(element1(
        execstr(str, S):t)), bool, 1, element2(t):u), u];
```

図２　管理スクリプトの一部

４．　おわりに

ＰＤＬで記述されたＰＤＬ処理系の管理スクリプトについて説明した．このスクリプトは約５００行あり，ＰＤＬの関数約６０個で構成されている．これにより，ＰＤＬ処理系の保守・改良が，以前より容易，かつ効率的にできるようになり，また，ＰＤＬの実用性について，ひとつの実例を示すことができた．

今後は，より一般的なシステムの保守・改良についてもＰＤＬによる記述で行いたい．

［参考文献］

[1] 飯田，西村，荻原，新田，井上，鳥居：“関数型言語に基づくプロセスプログラミングシステム”，電子情報通信学会１９８９年秋期全国大会(1989).

[2] 井上，関，谷口，嵩：“関数型言語ＡＳＬ／Ｆとその最適化コンパイラ”，電子通信学会論文誌(D), Vol.J67-D, No.4, pp.458-465(1984).

[3] 荻原，飯田，新田，井上，鳥居：“ソフトウェア開発環境記述用関数型　言語の設計と処理系の試作”，電子情報通信学会技術報告，SS88-35(1989).

[4] 荻原，井上，鳥居：“ソフトウェア開発を支援するツール起動自動制御システム”，電子情報通信学会論文誌(D-I)，Vol.J72-D-I, No.10, pp.742-749(1989).

[5] K.Inoue, T.Ogihara, T.Kikuno, and K.Torii:“ A Formal Adaptation Method for Process Descriptions”,Proceedings of the 11th International Conference on Software Engineering, pp.145-153 (1989).