論　文

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

非会員　荻原　剛志†　　正　員　井上　克郎†　　正　員　鳥居　宏次†

An Automated Tool Control System for Software Development

Takeshi OGIHARA †, *Nonmember*, Katsuro INOUE † *and* Koji TORII †, *Members*

あらまし　ソフトウェア開発では，どのような作業を，どのようなツールを用いて，どのような順序で進めるかということが重要である．本論文は，ソフトウェア開発に伴うツールの起動やメッセージの表示などを制御するために新たに設計した関数型言語 PDL (Process Description Language) と，そのインタプリタについて述べる．PDL はツール起動やウィンドウ操作，メッセージ表示などの機能を備えている．PDL では開発作業の流れを抽象的に記述でき，更にツールなどに関する具体的な情報を付加して詳細化することが容易にできる．PDL インタプリタはスクリプトを実行するだけでなく，スクリプトの記述や詳細化を容易にするためのさまざまな機能をもつ．実際に PDL を用いて，C プログラムの開発作業やジャクソンシステム開発法 (JSD) などを記述し，実行した．

1. まえがき

ワークステーション上で1人のソフトウェア開発者がソフトウェアを開発する状況を考える．開発者はエディタやコンパイラなどのいろいろなユーティリティプログラム (ツールと呼ぶ) を利用して，仕様書の作成，要求分析，設計書の作成，コーディング，コンパイル，デバッグなどの作業を繰り返し行い，要求されるプログラムやドキュメントを作成する．要求されるソフトウェアを効率良く作成するには，どのような作業を，どのようなツールを用いて，どのような順序で行うかなどが非常に重要である．

通常，開発者は自分自身で作業内容を考え，ツールの選択を行い，ツールを繰り返し起動しながら作業を進めなければならない．これらを適切に行うには，作業内容や種々のツールの機能とそれを起動するコマンド等に関する多くの知識が必要であり，特に経験の少ない開発者にとっては大きな負担となる．

もし，作業内容に合ったツールを順次起動し，行うべき作業に対する指針を開発者に提供するようなシステムがあれば，それを使用することにより開発者の負担は軽減されるであろう．開発者は与えられた指針に従って作業に専念でき，一連の作業が効率良く正確に行えるようになる．

従来，作業に合わせてツールを自動的に起動する方法やそのシステムについての研究はあまり行われていない．しかし，開発の方法や手順はソフトウェアの生産性や品質に大きく影響する．このようなシステムは開発者の負担の軽減と共に，ソフトウェアの生産性，品質の向上にも効果があると考えられる．

ここでは仮に，このような1人のソフトウェア開発者が UNIX が稼働しているワークステーション上で行うソフトウェア開発の一連の作業系列を開発過程と呼ぶ．

本論文では，開発過程の中で生じるツールの起動や指針となるメッセージの表示等を制御するために設計した言語 PDL (Process Description Language) およびそのインタプリタについて述べる．PDL インタプリタは現在，いくつかのワークステーション上で稼働中であり，実際に PDL のスクリプト* を作成，実行することができる．これまでにソフトウェア開発法である JSD の記述[1]などを行い，開発過程を容易に進めることができることを確認している．

以下では，PDL の設計方針，構文と意味および機能

† 大阪大学基礎工学部情報工学科，豊中市
Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan

* ツールの起動などを行うコマンド列の記述は，習慣的にプログラムと呼ばず「スクリプト」と呼ぶ．本論文でも PDL のプログラムをスクリプトと呼ぶ．

について述べ，次いでPDLインタプリタの機能を説明する．更にPDLスクリプトの作成例を示し，開発支援の有効性について論じる．

2. PDLの設計方針

2.1 詳細化の容易さ

ここでは，ツール起動などの制御を記述する言語の設計方針について検討する．但し，1.で述べたようなソフトウェア開発過程を記述の対象とし，チームによる複数の計算機を用いた開発や，発注者との話合いなどの人的側面については考慮しない．

開発過程の支援を目的としたスクリプトは，抽象度の高い記述を順次詳細化するという方法で作成することが多い．例えば開発作業を，仕様書の作成，要求分析，設計書の作成，コーディング，コンパイル，デバッグといったいくつかの作業段階に分け，これらの系列としてスクリプトを記述しておき，次第にツールや入出力ファイルなどに関する具体的な情報を付加して詳細化してゆく．

また，詳細化によって付加される情報には，開発対象や使用する計算機環境に依存して変化するものも多い．このため，スクリプトをそのような情報に依存しない形で記述しておき，目的や環境に応じて個別に詳細化して使用することがある．同様に，詳細化の際に開発者（以下，ユーザと呼ぶ）の使いやすいツールを選ぶことができれば，各ユーザごとに使いやすいシステムを構築することができる．

このようにツール起動のための言語には，さまざまな詳細化のレベルで記述できることや，詳細化が容易に行えることが求められている．

我々は従来ソフトウェアの仕様記述を代数的言語ASL[(2)]の枠組みで試みている．ASLは以下に示す特徴をもっている．

① 言語の意味が簡明に定義されている．
② さまざまな詳細化のレベルで記述が行える．
③ ある記述がもとの記述の詳細化になっているかなどの検証を比較的容易に行える．

これらの特徴はスクリプトの詳細化を進めるのに適している．そこで，このASLの意味定義をもとにした関数型言語PDLを設計した．

2.2 ツールの起動

このような言語では，ツールの起動やメッセージの表示，ファイルの操作など，計算機システムの機能が利用できなければならない．また，ウィンドウシステム上で効率的な開発を行うために，ウィンドウの操作や複数の作業の並列実行を記述する機能も必要である．PDLではこれらの機能を特別な組込み関数として用意した．

しかし，計算機システムに対するこのような操作は，ファイルシステムの状態や画面の表示などを変化させてしまう．一般に関数型言語でこのような状態変化を記述するのは困難である．また，関数型言語では普通複数ある引数の評価順序が任意であるために，ツールの起動などの具体的な実行順序が記述しにくいという問題もある．

これらの問題を解決するため，PDLではシステム状態という概念を導入した（**3.2**で述べる）．これによって関数型言語のもつ意味の簡明さを損なわずにツールの起動順序を決定し，計算機システムの状態変化を表現することができる．

3. PDLの構文と意味

3.1 PDLスクリプトの概要

PDLスクリプトは以下の要素から構成される．

（1） 関数定義
（2） マクロ定義（**3.4**で述べる）
（3） インタプリタへの指示（**4.5**）
（4） 評価すべき式

具体的なスクリプトの例を図1に示す（この例については**6.**で述べる）．①はインタプリタへの指示，②～④はマクロ定義，⑤～⑰は関数定義，⑱は評価すべき式である．

関数定義は以下のように，定義する関数名と仮引数リストを'＝＝'記号の左辺に，定義本体の式を右辺に記述する．定義は複数行にわたってもよい．

関数名（仮引数1，仮引数2，…）＝＝式；

式は組込み関数，定義関数，定数および左辺に現れる仮引数（変数）から構成される．表1にPDLの組込み関数の一部を示す．このうち，四則演算，比較演算などは中置記法で用いることができる．また，if関数は，

if　条件式　then　式1 else　式2

という形式をしており，条件が真の場合式1，偽の場合式2をこのif関数の値とする．

PDLでは以下のデータ型を使用できる．

整数　　文字列　　論理
システム状態　　タプル（並び）

タプルはいくつかのデータの組で，

［式1，式2，…］

```
#include /usr/users/pdl/strings                          ①
#let EDITOR: "vi "                                      ②
#let MANUAL: "man "                                     ③
#let CC(src,obj,S): exec("cc -o "+obj+" "+src,S)        ④

C_proc(src,S) ==
    Execute(src,
        element1(Compile(src,Create(src,S)):C),
        element2(C));                                   ⑤
Create(src,S) == Edit(src,S);                           ⑥
Correct(src,S) ==
    S @ wclose(Edit(src,wopen(S)));
Compile(src,S) ==                                       ⑦
    if element1(CCompile(src,Obj(src),S):T)
    then [Obj(src),element2(T)]
    else Compile(src,Correct(src,element2(T)));         ⑧
Execute(src,obj,S) ==
    if element1(Check(Run(obj,S)):T)
    then element2(T):T2
    else Execute(src,
        element1(
            Compile(src,Correct(src,T2)):C),
        element2(C));                                   ⑨
Edit(src,S) == exec(EDITOR+src,S);                      ⑩
CCompile(src,obj,S) ==
    [status(
        CC(src,obj,write("*compile*",S)):T)
     = 0, T];                                           ⑪
Obj(src) == substr(src,0,strlen(src)-2);                ⑫
Run(obj,S) == exec(obj,write("*run*",S));               ⑬
Check(S) == [read("OK?",S)="y", S];                     ⑭

C_dev(src,S) == wclose(C_proc(src,wopen(S)))
              @ wclose(Refer(wopen(S)));                ⑮
Refer(S) == if read("manual> ",S):K="q" then S
            else Refer(Manual(K,S));                    ⑯
Manual(key,S) == exec(MANUAL+key,S);                    ⑰

C_dev("test.c", S);                                     ⑱
```

図1 Cプログラム開発用PDLスクリプト
Fig. 1 A PDL script for development of C program.

表1 PDLの主な組込み関数

四則演算：+, −, *, /	
比較演算：=, <>, <, >, <=, >=	
論理演算：&, |, !	
並列演算：@	
条件判定：if then else	
Cシェルコマンドの実行：	exec
Cシェルの条件コードを得る：	status
ウィンドウを開く：	wopen
ウィンドウを閉じる：	wclose
文字列の読込み：	read
文字列の書出し：	write
ユーザ設定ブレーク：	break
タプルの先頭要素を取り出す：	element1

のように表現する．システム状態については**3.2**で述べる．

PDLでは記述のしやすさのため，各定義関数の引数や結果のデータ型を宣言しない．PDLの意味定義上，それらのデータ型は固定されていなければならず，場合によって異なるデータ型を返すような関数は(表記上可能であるが)定義してはならない．

3.2 PDL関数の意味

関数の意味は式の合同関係を用いて定義される[3]．組込み関数の定義(例えば1+1==2, 1+2==3, …)および定義関数の定義は，左辺と右辺の式が関数の適用という演算で閉じた合同関係を表すものとする．この合同関係から得られる最小の同値類分割を考え，評価すべき式の同値類中の構成子項(定数)をその関数の意味(値)と定義する．

PDLはデータ型の一つとしてシステム状態をもつ．システム状態とは，ファイルシステムやその他の計算機資源の状態すべてを表すもので，実行中のどの時点においてもある値が一つだけ存在する．組込み関数(ツールの起動やウィンドウ操作など)は，あるシステム状態を引数として受け取り，その値を更新し，次の関数に渡す．図1の例では，Sという名前の仮引数はすべてシステム状態型である．

3.3 状態を更新する組込み関数

ここではPDL特有の，システム状態を更新する組込み関数について述べる．

(1) システム機能の呼出しexec

UNIX上でツールを呼び出すには，Cシェルというコマンドインタプリタにコマンドの文字列を与えるのが容易な方法である(本論文では4.2, 4.3 BSDあるいはこれらと同等なBSD系のUNIXについて考える)．PDLからCシェルの機能を利用するには組込み関数execを用いる．execは引数として与えられた文字列をCシェルにそのまま渡して実行させる．

図1⑩の関数Editは，マクロで定義した実際のツール名"vi"と引数のファイル名，例えば"test.c"を連結して文字列"vi test.c"をCシェルに渡す．ここでは，演算子+は文字列の連結である．

(2) ウィンドウの操作 wopen, wclose

PDLではウィンドウをいくつか開いてその中でツールを起動し，作業を効率良く行うことができる．ウィンドウを操作するための関数としてwopen, wcloseがある．wopenは新しいウィンドウを開き，wcloseはウィンドウを閉じる．

(3) 並列演算子 @

ソフトウェア開発では複数の作業を並列に行いたいことがある．例えば，別のプログラムやオンラインマニュアルを参照しながらプログラムの作成を行うような状況が頻繁に生じる．

このような並列した作業を記述するために，PDLでは新たに組込み関数 @（並列演算子と呼ぶ）を導入した．この関数は二つの作業を並列に開始させ，最後に双方の同期をとる．引数は二つのシステム状態の値であり，新しい一つのシステム状態の値を返す．

図1⑮の関数C_devを起動するとウィンドウが二つ開き，一方でC_procが，他方でReferが起動され，別々の作業を並列に行うことができる．両方の作業が終了するとウィンドウは閉じられ，C_devは新しいシステム状態の値を返して終了する．

関数C_devの値は，引数Sのシステム状態に対して@の左側の式で得られる値と右側の式で得られる値の“合成”と考える．ここでいう合成されたシステム状態とは，@の左側の関数（wopen, C_proc, wclose）と右側の関数（wopen, Refer, wclose）によるシステム状態の更新が，互いに重なり合って起こった結果のシステム状態である．重なり方は引数のシステム状態S，すなわち，そのときの計算機のハードウェア，ソフトウェア，およびユーザの操作に依存する．

並列演算子は並列した作業間の通信や共通の資源へのアクセス制御などの機能は備えていない．しかし，多くの開発過程の作業は逐次的に進行するため，この並列演算子で簡明に表現できる．

3.4 マクロ定義

PDLではグローバルマクロとローカルマクロの2種類のマクロ記法を用いることができる．

グローバルマクロの定義を行うと，関数定義内に現れるマクロ名は定義した文字列で置換される．グローバルマクロには図1②，③のように引数をもたないもの，④のように引数をもつものがある．グローバルマクロはスクリプトの段階的な詳細化に有効である．例えば図1の例のように，はじめはEDITORやMANUALといった抽象的なツール名を用いて記述を行い，後でグローバルマクロを用いてviやmanという具体的なツール名を与えることができる．

ローカルマクロは通常の関数定義の記述の一部として定義され，その関数定義内でのみ有効である．ローカルマクロは式の記述の中で以下のように定義する．

因子：マクロ名

因子とは仮引数，関数呼出し，または括弧でくくられた式のことである．関数定義内部のマクロ名は定義した因子に置き換わる．例えば，図1⑤の定義式はマクロ名Cの置換によって以下のようになる．

```
C_proc(src, S) ==
    Execute(src,
      element1(Compile(src, Create(src, S))),
      element2(Compile(src, Create(src, S))));
```

PDLは関数型言語であり，手続き型言語でいう変数の概念をもたない．このような言語では同一の式の値を複数箇所で利用する場合が多く，それぞれの箇所にその式全体を記述したり，別の関数にしたりする必要がある．PDLではローカルマクロを利用することによってこのような煩雑さを取り除くことができ，わかりやすい記述を行うことが可能である．

4. PDLインタプリタの機能

PDLインタプリタは端末，あるいはファイルからスクリプトを入力し，これらを解釈実行する．このインタプリタは単にスクリプトを実行するだけではなく，スクリプトの記述や詳細化を容易にするようなさまざまな機能を備えている．

4.1 式の評価

インタプリタは式の記述を直接与えられるとそれを評価し，結果の値を返す．式は，関数定義の内側，左側から順次評価される．関数の引数には実引数を評価した値が渡される（call by value）†．

関数の定義式中に同一の部分式が複数現れた場合，その部分式は一度だけ評価され，保存される．その値が再び必要になったときには保存された値が参照される．すなわち，部分式の評価はたかだか1回である．また，組込み関数に渡される値の型検査を実行中に行う．

4.2 システム状態の取扱い

システム状態型の値は整数型などの他のデータ型の値と異なり，値のコピー，保存ができない．

いま，下の(1)の関数について考える．これを実行すると，まずif関数の条件判定部で組込み関数execによってシステム状態が更新される．その結果によってthen部，else部いずれかが実行されるが，いずれの場合も，writeの引数のシステム状態は条件判定部で更新される前の値Sを指定しており，このとおり実行することはできない．

一方，(2)は，更新されたあとのシステム状態の値S1に対してwriteを実行することを記述しており，実行可能である．通常，プログラマは(2)のように，常に最

† この評価方法は容易に実現でき，実行効率も良いが，結果の値が存在するのに評価が停止しない場合が原理上起こり得る．しかし，我々が試みた通常の開発過程の記述ではそのようなことは起こらず，実用上問題がなかった．

新のシステム状態の値に対して関数を施すように記述する必要がある．(1)の例のように，誤ったシステム状態が指定された場合には，インタプリタが実行時に発見し，ユーザに警告を出すことができる．

```
Compile(src, S) ==
  if status(exec("cc "+src, S))=0
    then write("success", S)
    else write("error", S);                    (1)
Compile(src, S) ==
  if status(exec("cc "+src, S): S1)=0
    then write("success", S1)
    else write("error", S1);                   (2)
```

4.3 未定義関数の処理

組込み関数以外で，その定義が与えられていない関数を未定義関数と言う．スクリプトを実行中に未定義関数の値が必要になったとき，PDL インタプリタは実行をそこで一時中断してブレークモードに入る．そこでユーザは次のいずれかの動作を選択することができる．

（a） その関数値を一時的に与え，実行を再開する．

（b） その関数の定義を与え，実行を再開する．

（c） 実行を中断する．

ブレークモードとは，実行が中断しているときのインタプリタの状態を言う．ユーザはブレークモード内でも通常と同様に式を評価したり，インタプリタに指示を与えたりすることができる．

この未定義関数処理機能により，未定義関数を含むスクリプトでも，ユーザが実行中に値を決めたり関数定義を行いながら実行を進めることができる．この機能はスクリプトを実際に実行しながら詳細化を進める場合に有効である．

4.4 デバッグ

ブレークモードに入るには以下の三つの場合がある．

（a） 未定義関数を評価した場合，

（b） 端末から中断文字（通常 control-C）が入力された場合，

（c） 組込み関数 breakが実行された場合．

ブレークモードでは，中断後の引き続く実行を関数単位で行ったり，実行した関数名と値を表示することなどができる．これらの機能により，スクリプトのデバッグを容易に行うことができる．また，使い勝手を向上させるため，ウィンドウを活用したデバッグ専用のインタフェースを用意している．

4.5 インタプリタへの指示

インタプリタに指示（ディレクティブ）を与えることによってさまざまな機能が利用できる．指示にはスクリプト内部に含めることができるものもある．

インタプリタには，あらかじめスクリプトを記述したファイルを読み込む機能がある．この指示はスクリプト内に含めることができる（図1①）．この指示を利用すればスクリプトをモジュール化して記述できる．また，頻繁に使われる関数定義や OS などに依存する記述をファイルにまとめてライブラリとして利用すれば，スクリプトの記述をわかりやすく容易に行うことができる．

また，あるマクロ名が定義されているかどうかを条件として，スクリプトの一部を解釈させるかどうかを指定する指示もある．

このほか，関数定義，マクロ定義を表示させたりファイルに出力させる機能，端末にエディタを呼び出して定義の編集を行う機能，以前の指示の履歴を用いる機能などが用意されており，PDL スクリプトを効率良く作成できるようにしている．

5. PDL インタプリタの実現方法

PDL インタプリタの基本的な実行方式は，ASL/F などの関数型言語における関数評価方法[4]と同様である．ここでは，システム状態を更新する組込み関数の実現方法について簡単に触れる[5]．

PDL インタプリタは自由にウィンドウを開き，その中でツールを起動できる必要がある．このためには，PDL インタプリタからウィンドウプログラムを起動し，そのウィンドウプログラムから C シェルを起動する．そして，PDL インタプリタからこの C シェルにコマンドの文字列を次々に渡さなければならない．

通常 UNIX では，プロセス† 間通信の方法としてパイプやソケットという機構が用いられる．しかし，パイプでは直接起動したプログラム以外との通信が困難である．また，ソケットもソケットで通信できるように作られたプログラム間でしか利用できない．

そこで，ここでは PDL インタプリタと C シェルとの通信に UNIX の擬似端末（pseudo terminal）機能を利用した．擬似端末とは仮想的な端末装置（デバイス）であり，実際の端末装置と同様の振舞いをするが，それへの入出力はプログラムによって制御することが可能

† UNIX における，プログラム実行の単位．

である．UNIXでは装置も一種のファイルであると考えているため，擬似端末をCシェルのスクリプトファイルとして渡せば，Cシェルは擬似端末からコマンドを読み込んで実行を行うことができる（図2）．この実現方法はウィンドウプログラムにもCシェルにも変更を加えず，しかも移植性が高い．

PDLインタプリタではまた，組込み関数 @ の引数が表す作業をそれぞれ並列に実行しなければならないが，複数の作業を一つのインタプリタで同時に制御するのは大変困難である．そこで，並列に動作する作業のそれぞれについてインタプリタのプロセスを一つずつ作って作業を制御させるという方法を用いている．

これらの実現方法はUNIXの標準の機能のみを用いており，更に，特定のウィンドウシステムにも依存していない．

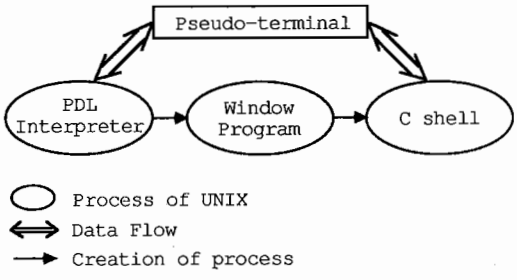

図2　擬似端末によるプロセス間通信
Fig. 2　Inter-process communication with a pseudo-terminal.

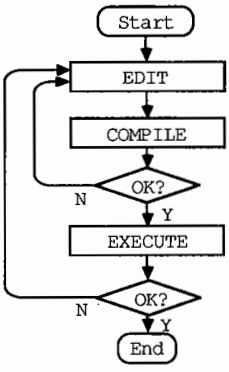

図3　Cプログラム開発作業の流れ
Fig. 3　Development processes of C program.

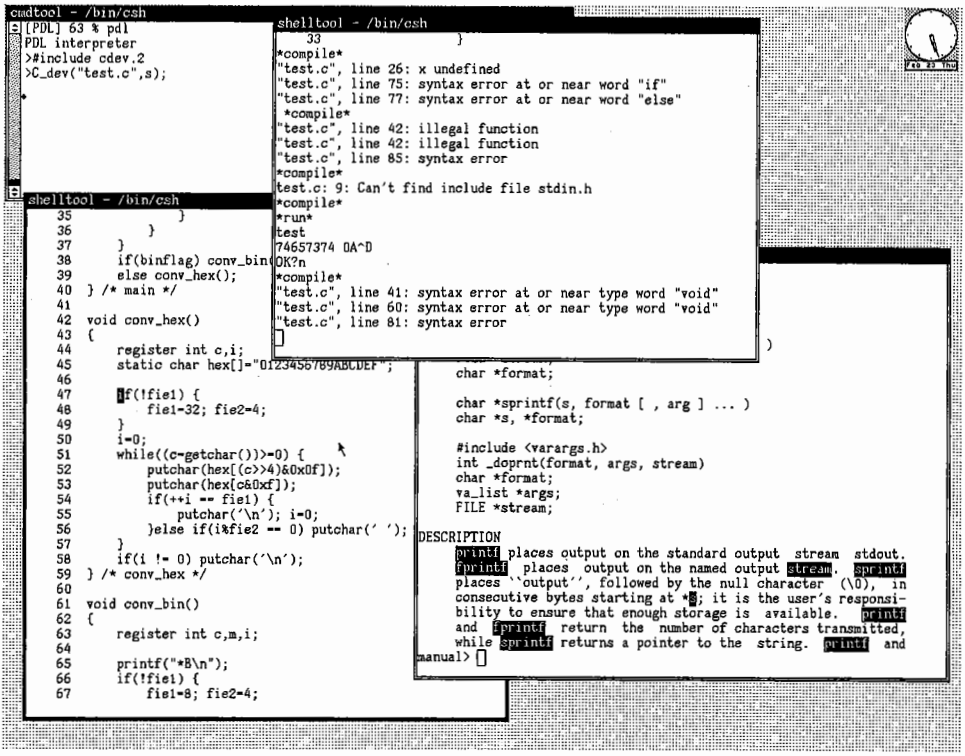

図4　PDLスクリプトの実行例
Fig. 4　An example of execution of PDL script.

6. 詳細化によるPDLスクリプトの作成方法

ここでは，図1に示した簡単なCプログラム開発過程のスクリプトの記述，詳細化の概略と実行例を示す．

（1） 作業の流れの記述　作業の流れの概要は図3のように表される．最初にプログラムを作成し，次にコンパイルを行う．コンパイルが成功すればオブジェクトプログラムの実行を試みるが，失敗すればプログラムの修正を行う．実行の結果が要求を満たしていれば開発作業は終了するが，そうでなければプログラムの修正に戻る．

この開発過程をPDLで記述するため，プログラムの作成，コンパイル，修正，実行のそれぞれの開発段階を，Create, Compile, Correct, Executeという関数で表す．開発過程全体を表す関数をC_procとすれば，これは図1⑤のように定義できる．

（2） 記述の詳細化　次に，テキストエディタの起動を表す関数EditやCコンパイラの起動を表す関数CCompileなどを用いると，各開発段階を表す関数は⑥～⑨のように定義できる．更に，Edit, CCompileなどはPDLの組込み関数を使って⑩～⑭のように定義できる．

以上のような段階的な詳細化によって定義した関数C_procは図3の手順通りに実行することができる．

（3） 実行例　図1では更に⑮～⑱の定義を加え，関数C_procの実行と並列に，オンラインマニュアルを参照するための関数Referが起動できるようにした．

このスクリプトを実際にPDLインタプリタで実行したときの画面の例を図4に挙げる．左上のウィンドウから関数C_devを起動して実行を開始する．残り三つのウィンドウは左からそれぞれ，エディタ用，コンパイル用，オンラインマニュアル用のウィンドウである（なお，この画面はSun3上で実行した例である）．

7. 議　論

図1のスクリプトでは，⑤～⑨がEdit, CCompileなどの抽象的なツールの概念を用いた記述であり，⑩～⑭がそのツールの概念を具体的なツールに対応させて定義した記述になっている．PDLを用いると，このように詳細化のレベルを段階的に分けて，開発過程をわかりやすく記述，詳細化することができ，またスクリプトの変更も容易になる．

このスクリプトを実行すると，開発作業を図3の手順どおりに進めることができる．ツールの起動を手作業で行う場合のような煩わしさなしに開発を進められ，ツールの具体的な起動方法などについての知識も必要としない．

PDLによる実用的なスクリプトの例としては，ソフトウェア開発法であるJSPおよびJSDに対するものがある[1]．これらのスクリプトも上の例のように段階的に詳細化を進めて作成したものである．これらをインタプリタ上で実行することによって，その開発法に従った作業を支援することができる．

PDLのように特別に設計した言語でなく，OSの提供するコマンド言語を用いても，ツールの起動やメッセージの表示などを行うことはできる．しかし，このようなコマンド言語は形式的な意味定義をもたず，さまざまな抽象度でスクリプトを記述することができない．また，特定のOSのみに依存してしまい，移植性，汎用性が乏しい．

PDLと同様の目的からソフトウェア開発過程におけるツールの起動などを記述するため，他にいくつかの方法および言語が提案されている．

Osterweil[6]は，ソフトウェア開発における手順の記述自体をソフトウェアとみなすことができると論じている．この考え方をプロセスプログラミングと呼ぶ．OsterweilはPascal風の言語を用いてさまざまなソフトウェア開発の手順を記述しているが，今のところこの記述を計算機上で実行して開発を行ったという報告はない．

Williams[7]は開発過程を，前提条件，完了条件および人間がすべき作業の三つに分けて記述するSPM (Software Process Model) というモデルを提案し，これに基づいて開発過程の記述を行っている．しかしこの記述自体は自然語であり，計算機上で実行することはできない．

Kaiserら[8]は開発過程をルールに基づいて記述する専用言語Marvelを設計し，これを用いて開発過程の記述を行っている．Marvelは実際に計算機上で実行可能であるが，ルールを漏れなく記述しなければ作業を適切に進めることができず，また，さまざまな詳細化の記述を行うことも難しい．

8. むすび

本論文では，ソフトウェア開発過程におけるツール起動やメッセージ表示を制御するための言語PDLを提案し，言語とインタプリタの特徴について述べた．

PDLインタプリタは，約6人月の作業量をかけて作成された．使用言語はCで，ソースコードはおよそ8500行である．ツール起動やウィンドウ操作，メッセージの表示などの目的には十分な実行速度をもつ．

PDLインタプリタはUNIX上のウィンドウシステムを用いて実行する．PDLインタプリタは現在，IBM RT/PC (ACIS) およびSony NEWSのX-Window上，Sun3のSunViewおよびX-Window上で稼働している．

現状ではPDLスクリプトの作成は人間がすべて手作業で行わなければならない．そこで，図的表現などから実用的に実行可能なスクリプトを導出する方法の検討，および支援システムの作成を進めている．

また，**2.2**でも述べたが，本システムでは開発作業はすべて一人のユーザが行うと考えている．チームによる開発が行えるようにするには，ユーザ相互の通信などの機能が必要になるため，その記述方法および実現方法を検討している．

謝辞 本研究への適切な助言，協力を頂いたSRAの新田稔氏に感謝します．処理系の作成に協力して頂いた本学大学院修士課程稲田良造氏（現ダイキン工業），飯田元氏に感謝します．

文　献

(1) 稲田良造，荻原剛志，井上克郎，菊野　享，鳥居宏次：“ジャクソン開発法の形式的記述の詳細化とその実行”，信学技報，SS88-36 (1988-12).
(2) 東野輝夫，関　浩之，谷口健一：“代数的仕様から関数型プログラムの導出とその実行”，情報処理，**29**, 8, pp. 881-896 (昭63-08).
(3) 嵩　忠雄，谷口健一，杉山裕二：“代数的言語の設計と処理系”，榎本編，ソフトウェア工学ハンドブック，pp. 1066-1073, オーム社 (昭61).
(4) 井上克郎，関　浩之，谷口健一，嵩　忠雄：“関数型言語ASL/Fとその最適化コンパイラ”，信学論(D)，**J67-D**, 4, pp. 458-465 (昭59-04).
(5) 荻原剛志，飯田　元，新田　稔，井上克郎，鳥居宏次：“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”，信学技報，SS88-35 (1988-12).
(6) L. Osterweil : “Software processes are software too”, Proc. 9th ICSE, pp. 2-13 (1987).
(7) L. G. Williams : “Software process modeling : A behavioral approach”, Proc. 10th ICSE, pp. 174-186 (1988).
(8) G. E. Kaiser and P. H. Feiler : “An architecture for intelligent assistance in software development”, Proc. 9th ICSE, pp. 180-188 (1987).

（平成元年3月2日受付）

荻原　剛志

昭60山梨大・工・計算機科学卒．昭62同大大学院修士課程了．現在，阪大・基礎工・大学院博士課程在学中．ソフトウェア工学の研究に従事．情報処理学会，ソフトウェア科学会各会員．

井上　克郎

昭54阪大・基礎工・情報卒．昭59同大大学院博士課程了．同年同大・基礎工・情報・講師．昭59～61ハワイ大助教授．関数型言語の処理系等の研究に従事．工博．情報処理学会，ACM，IEEE各会員．

鳥居　宏次

昭37阪大・工・通信卒．昭42同大大学院博士課程了．工博．同年電気試験所（現電子技術総合研究所）入所．昭50ソフトウェア部言語処理研究室室長．昭59阪大・基礎工・情報・教授．ソフトウェア工学の研究に従事．