OUKA
The University of Osaka
Institutional Knowledge Archive

Title	プロセスの性質記述に基づく実行モデルの提案
Author(s)	飯田, 元; 荻原, 剛志; 井上, 克郎 他
Citation	情報処理学会研究報告. ソフトウェア工学研究会報告. 1990, 1990-SE-072(47), p. 1-7
Version Type	VoR
URL	https://hdl.handle.net/11094/50174
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

ソフトウェア工学　72－5
（１９９０．　５．　２９）

プロセスの性質記述に基づく実行モデルの提案

飯田元　荻原剛志　井上克郎　鳥居宏次

大阪大学基礎工学部

内容梗概

ソフトウェア開発過程をツールの起動などによって表される基本作業の系列と考え，この系列の文法を定めることで開発過程の定義を与える。このプロセスの性質記述を基に，インタラクティブな開発支援環境を実現するための実行モデルＲＡＩＳＥ(Restricted ActivIty Selection and Execution)を提案する。ＲＡＩＳＥモデルでは，適用するプロダクションルールの選択を属性付メニューにより決定する．

さらに，ＲＡＩＳＥモデルに基づいた開発支援環境を開発過程記述言語ＰＤＬ(Process Description Language)により記述／構築する方法について述べ，実例を示す。

A MENU-BASED MODEL FOR SOFTWARE PROCESS ENACTION

Hajimu Iida Takeshi Ogihara Katsuro Inoue Koji Torii

Department of Information and Computer Sciences,

Facluty of Engineering Science,Osaka University

1-1 Machikaneyama,Toyonaka,Osaka 560,Japan

Software development process can be assumed as a sequence of activities such as tool enaction and human activity. In this paper, we propose a method to define software processes using regular expressions and context free grammers. We also propose a model named RAISE(Restricted ActivIty Selection and Execution) to enact the processes defined by this method. RAISE consists of a set of menues and preconditions associated by the menus.

Using RAISE model, software process may be interactively controlled by software developers. We show an example of RAISE model applied to the process of C program development. RAISE model is actually enacted by using the system of functional language PDL(Process Description Language).

1．はじめに

ソフトウェア開発プロセス（以下ではプロセスと記述する）を形式的に記述し，その記述に基づいた開発支援環境を作成してソフトウェア開発の作業効率と信頼性を高めようとする研究が進められている．

プロセスの記述には，これまでに様々な方法が提案されている．Osterweilら[5] はAdaの拡張言語**Appl/A**を用いてプロセスを記述しようと試みている．**Appl/A**はAdaを下敷にすることによって，プロセスを記述する上での様々な要求に応える能力を備え，更にプロダクトや資源の間の関係を記述できるような拡張が加えられている．この言語によって，作業の手順とプロダクトの構造を手続き的に記述する．

Kaiserらは，あらかじめ記述したルールに従って自動的にツールを起動したり，プロダクトに対する操作を行なうことができる開発環境**Marvel**を設計した[4]．**Marvel** はプロダクトやツールを管理するためのオブジェクト群と，作業の手順を表現する拡張可能なルール群から構成されている．ルールは前提条件，完了条件，及び開発作業からなり，あるルールの条件が満たされたとき，そこに記述された開発作業を表すツールが起動される．

これらの方法は，プロセスを構成する各作業単位（ツールの起動や人間の行なう作業）を計算機言語における基本構成要素（例えば手続き型言語における演算子や関数呼び出し，論理型言語における述語等）と見立て，その言語が持つ様々な制御機構を用いて目的とする作業の系列（の集合）を得ようとしている．ソフトウェアの開発において，作業の進行状況を把握したり，作業に関するデータの収集を行なう等といった，いわゆるプロセス・マネジメントを効果的に行なうためには，このような作業の流れを制御する必要があろう．

その一方で，現実の作業手順は複雑で，それを詳細にそれらの言語であらかじめ記述するのは困難であろう．また過度の制御を行なうと開発者のの自由度を奪い，開発効率が低下する恐れがある．従って，記述を簡単にし，開発者に自由度を与えるため，個々の基本的な作業単位をそれぞれ必要最小限の制約下でインタラクティブに選択／実行するような支援系が望ましい．

そこで我々は，プロセスの作業単位を制御する方法を考えるのではなく，許されうる可能な作業の集合を与え，その集合中から開発作業者の好みやそのときの条件によって一つの適当な系列を動的に選ぶ方法について考えた．

本稿では作業の集合を表すものとしては，正規文法，及び，文脈自由文法を用いた．また，これらの文法が与える作業系列の集合から任意の一つを選択する為の方法として，開発者が選択するメニュー及びそのメニューの各項目が選択可能かを決める制約条件から構成される**RAISE**（ Restricted Activity Selection and Execution ）モデルを提案する．

さらに，RAISE モデルに基づいた開発支援系を開発過程記述言語**ＰＤＬ**（Process Description Language ）により記述／構築する方法についても述べ，実例を示す．

本稿では開発プロセスが基本作業単位の系列で表しうる，という仮定に基づいて，許される作業の集合を正規文法，文脈自由文法等で表現したが，今後，並行作業等も表現しうるよう拡張する予定である．

以降２章では開発作業系列の文法による定義について，３章では**RAISE**モデルについて，４章では言語ＰＤＬによる**RAISE**モデルの記述／実行について述べたのち，５章で若干の議論を行なう．

2．開発作業系列のモデル化

2.1 作業系列の構文則による定義

本稿ではソフトウェア開発過程を一人のソフトウェア開発者が一台のワークステーション上でソフトウェア開発を行なう一連の作業系列とする．さらに，開発作業は系列で表しうるという仮定をおく．本稿では，系列が正規言語の場

合と文脈自由言語の２つの場合についてのモデルをそれぞれ示す．

2.2 正規言語に属する系列

まず，系列が正規表現に属する場合を考えてみる．例えば，一般的なウォーターフォールモデルでは，各段階の作業が逐次的に行われることになっているが，実際には，個々の作業のやり直しや，上流の作業に戻ってのやり直しなどが行われる．従って，一般に

P : begin → a → b → c → d → end

のように捉えられているプロセスは正規表現

$(((a^* b^*)^* c^*)^* d^*)^*$

のような作業系列を持つ．

例としてエディット／コンパイル／リンクといった作業の流れを考える．各作業を表すシンボルを

E = Edit
C = Compile
L = Link

のように定める．このとき，コンパイル／リンク単独でのやり直しは考慮しなくてよいので，作業系列は正規表現： $((E^* C)^* L)^*$ で表される．従って，たとえば系列

ECL, EECECL

などは正しい作業系列であるが，系列

CEL, LEC, LCE, ECCL

などは不正な作業系列である．

2.3 文脈自由言語に属する系列

次に，作業系列が文脈自由文法によって生成される場合を考える．例えば，複数のファイルに対してエディット(:E)を行った後，それらのファイルをコンパイル(:C)し，それらのあとでリンク(:L)を行なうプロセス系列の集合

$P = \{E^n C^n L \mid n \geq 1\}$

を考える．このような系列は文脈自由文法

P → EACL
A → EAC | ε

によって生成される．

3．RAISEモデルによるプロセス定義

3.1 メニューを用いた実行系列の制御

正規文法及び文脈自由文法で定義された作業系列に従った開発を行なう場合，一つの非終端記号からのプロダクションルールが複数存在する事がある．これは，ある時点で複数の作業を行なうことが出来ることを意味する．このような場合には，どのルールを適用するのか － つまり，どの作業を行なうのかを決定しなければならない．

このような場合にはメニューによって実行可能な作業を表示し，開発者がそれを選択する．図１は作業選択メニューの一例である．

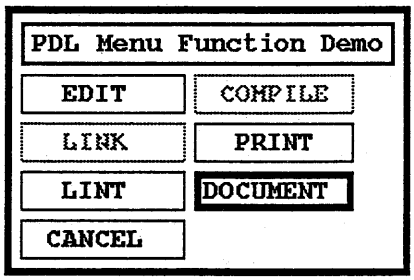


図１　作業選択メニューの例

メニューには複数の候補の中から選択できるという効果の他に，メニューに挙げられた候補からしか選ばせないという制限の効果もある．また，ユーザーインターフェースとしても優れている．

RAISE モデルは開発作業系列の構文則に対して

(1) 終端記号（各作業）および非終端記号に対する制約条件（precondition）
(2) 各作業の内容（body）

を定義する．プロセスの実行（＝開発作業）は制約条件と構文則によって生成された属性付メニューを逐次，選択することによって進行する．実際にはメニューに頼らずに，条件判定によって自動的に行なう作業を決定することのできる部分も存在するが，ここでは省略する．

3.2 属性付メニューの定義

まず属性付メニューによる実行メカニズムを形式的に定義し，それを用いたプロセス定義を示す．プロセスPが作業 A_1 A_2 A_3 ... A_n によって構成される場合，各作業には属性＝制約条件 C_1 C_2 C_3 ... C_n がそれぞれ定義される．RAISE モデルでは大域的な状態（変数やスタックなど）を持ち，作業の選択時にはこの状態を用いて各制約条件が評価される．その結果，各作業について制約条件の満たされているもののみがメニューから選択出来る．

開発者はメニューによって制約条件の満たされている作業のうち一つを選択し，実行する．つまり，構文則

P → A_1 | A_2 | ... | A_n

は

P → <A_1:C_1> | <A_2:C_2> | ... | <A_n:C_n>

のように拡張される．

この動作は，プロセスP，及び，Pで行う各作業Ai（1≦i≦n）を状態遷移関数の形で記述することで，以下のように定義される．なお，変数 S は抽象的状態である．

リスト1　属性付メニューの動作定義

```
P(S) == if      menu(S) = 1  then A1(S)
        else if menu(S) = 2  then A2(S)
  ...
        else if menu(S) = n  then An(S);

where menu(S) ==
display_and_get_responce( "A1 A2 ... An",
        [C1(S),C2(S),...,Cn(S)],S)
```

以後，属性付メニューによる選択実行による状態遷移を

```
SELECT_EXEC( <C1(S),A1(S)>,
             <C2(S),A2(S)>,
             ...
             <Cn(S),An(S)>,
)
```

の様に略記する．

3.3 正規言語の実行

正規言語に属する系列の実行は，構文則を用いることなく，次の２つの操作で単純に実行が可能である．

(1) X^* の部分は，終了条件を満たすまでループする．

(2) (A + B + C) の部分は属性付メニューを用いた選択実行を行う．

従って，正規表現

P = {(A+B+C)*}

にあたる系列は以下のような状態遷移で生成される．

```
if postcondition(S)=true then S
else P( SELECT_EXEC( <CA(S),A(S)>,
                     <CB(S),B(S)>,
                     <CC(S),C(S)>));
```

3.4 文脈自由言語の実行

文脈自由言語の場合には構文則のうち，複数のプロダクションをもつルールに対してメニューを適用し，その他のルールについてはそのまま状態遷移関数にマッピングをとる．

たとえば，モジュールを順次分割して作成を行うようなプロセスはリスト２のようなＣＦＧで定義される．

リスト２　モジュール開発の構文則（例）

```
P → make_a_module

make_a_module → implement | breakdown
breakdown → division interface_design
            make_sub_modules
make_sub_modules →
 make_a_module make_sub_modules | ε
```

この文法に，制約条件を付加することにより，以下の状態遷移で定義される実行プロセスが定義される．

リスト３　モジュール開発のプロセス記述（例）

```
P(S)==make_a_module(S);

make_a_module(S) ==
  SELECT_EXEC(
    <is_a_small_module(S), implement(S)>,
    <is_a_large_module(S), breakdown(S)>
  );

breakdown(S) ==
  make_sub_modules( interface_design(
    division(S) ) );

make_sub_modules(S)==
  SELECT_EXEC(
    <some_modules_not_implemented(S),
     make_sub_modules(make_a_module(S))>,
    <all_modules_implemented(S), S >
  )
```

４．ＰＤＬ処理系による実行

RAISE モデルによるプロセスの定義は，プロセス記述用言語ＰＤＬを用いることによって容易に記述することが出来る．得られた記述は，ＰＤＬのインタプリタ上で動作する開発支援システムのプログラムに相当する．

4.1 ＰＤＬ

我々は，ソフトウェア開発過程の記述に，関数型言語ＰＤＬ（Process Description Language）を用いている．

ＰＤＬは代数的仕様記述言語の部分クラスであり，簡明な意味定義を持つ点や，様々な抽象度における記述能力を持つ点などが特徴である．

ＰＤＬのデータ型の一つにシステム状態型というものがある．システム状態型とはファイルシステムやその他の計算機資源の全て，及び，開発者の状態を抽象的に表わすもので，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．従って，ＰＤＬでは，プロセスをある状態S_1から別の状態S_2へのマッピングを行なう関数として表わす．つまり，ある状態S_0に，ツールの起動やウィンドウの操作などの状態遷移を順次施すことで，目的の状態 S_nに導くことが，ＰＤＬのスクリプトの意味である．

ＰＤＬで記述されたプログラムは，インタプリタを用いて実行することができる．ＰＤＬインタプリタは，ウィンドウ環境への対応やデバグ機能，ツール起動関数を持つなど，開発環境を構築するための様々な機能を備えている．

ＰＤＬシステム上で形式的な開発過程の記述をプログラムとして実行することで，開発支援系を構築することができる．

4.2 ＰＤＬのメニュー関数

ＰＤＬでは制約条件つきメニューによる選択実行を行なう部分は，組み込み関数 menu_branch を用いて簡潔に記述できる．menu_branch は，[文字列,ブール式,状態式]のリストと状態型の引数を取り，システム状態を返す関数で，例えば制約条件 pre_s1, pre_s2, pre_s3 をそれぞれもつ３つの作業：ｓ1, ｓ2, ｓ3 を行なうプロセスは次のように記述できる．ここで記号 {..} はリストを [..] はタプルをそれぞれ表している．

```
P(S) == menu_branch(
        {["s1",pre_s1(S),s1(S)],
         ["s2",pre_s2(S),s2(S)],
         ["s3",pre_s3(S),s3(S)]},S);
```

menu_branch では，まずリスト型の引数の各要素の文字列を取り出してメニュー表示を行ない，選択を促す（但し，引数として与えられたブール値が偽であるものについては選択を許さない）．そして，選択された値に対応するリストの要素（タプル）を取り出し，更に，その３番目の要素である，状態式を評価して返す．この動作は，リスト４に示すように，ＰＤＬで形式的に

定義することができる．

リスト４　関数menu_branchの定義

```
menu_branch(list,S) ==
        element3(get_element(menu(mk_menu(
list),S),list));

mk_menu(list) ==
        if list = nil then ""
        else element1(head(list))+mk_menu(
list);

get_element(item,list) ==
    if item = 1 then head(list)
    else get_element(item-1, tail(list));
where
   menu(string,S)  is a primitive function
   element3(tuple) is a primitive function
```

4.3 記述／実行例

ここでは具体的な例として，Ｃ言語によるプログラム開発の支援系を記述した．開発プロセスの構造は図２に示すように定義した．図からも判るように，このプロセスの文法は正規文法で表される．制約条件には，「コンパイル済みのモジュールにはコンパイルを行なわない」，「リンクを行なうためには，未コンパイルのモジュールが存在してはならない」などが定義されている．ＰＤＬでの記述はライブラリを含めておよそ２００行程度である（リスト５）．メニューにより作業を選択すると，エディタやコンパイラの起動やウィンドゥの開閉を自動的に行なう（図３）．

５．議論

Appl/Aのように手続き的なプロセス記述では，プロセスの構造を比較的明確に記述出来るが，一方で，作業系列の束縛が大きくなる恐れがある．また，Marvelのようなルールベースの記述では記述の容易さ，実行の柔軟さなどが期待できる半面，プロセスの構造や実行系列などを把握しにくいという問題がある．さらに，これら

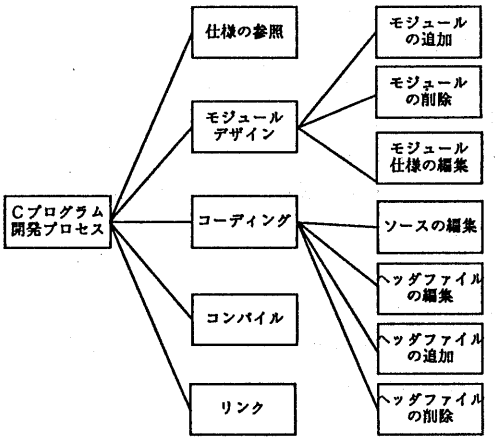


図２　Ｃプログラム開発プロセスの構造

```
//-------------------------------------------------------------
//      LEVEL-0 Process
//-------------------------------------------------------------
C_develop(S) ==
        if post_C_develop(body_C_develop(S):S1) then S1
        else C_develop(S1);
pre_C_develop(S) ==
        exist(PROGRAM_SPEC,SYSTEM);
body_C_develop(S) ==
        menubranch( "C program Support Environment Main Menu",
        {["Spec_Analysis",pre_Spec_Analysis(S),Spec_Analysis(S)],
         ["Ref_Manual", pre_Ref_Manual(S), Ref_Manual(S)],
         ["Module_Design",pre_Module_Design(S), Module_Design(S)],
         ["Coding", pre_Coding(S), Coding(S)],
         ["Compile", pre_Compile(S), Compile(S)],
         ["Link", pre_Link(S), Link(S)],
         ["QUIT", true, exit(SYSTEM)]},
         SYSTEM);
post_C_develop(S) == exist(EXECUTABLE,SYSTEM);

//-------------------------------------------------------------
//      LEVEL-2 Process
//-------------------------------------------------------------
Spec_Analysis(S) ==
        [PROGRAM_NAME,PROGRAM_SPEC,MODULE,EXECUTABLE,
        MORE(PROGRAM_SPEC,SYSTEM)];
pre_Spec_Analysis(S) == true;

Ref_Manual(S) ==
        [PROGRAM_NAME,PROGRAM_SPEC,MODULE,EXECUTABLE,
        XMAN(SYSTEM)];
pre_Ref_Manual(S) == true;

Module_Design(S) ==
        menubranch("Module Design", {
                ["New_Mod", pre_New_Mod(S), New_Mod(S)],
                ["Del_Mod", pre_Del_Mod(S), Del_Mod(S)],
                ["Edit_MSpec", pre_Edit_MSpec(S), Edit_MSpec(S)],
                ["CANCEL", true, S]
        },SYSTEM);
pre_Module_Design(S) == true;
```

リスト５　Ｃプログラム開発支援系の記述

のモデルではプロセスを実行するためのシステム（コンパイラ／インタプリタや推論機構）が複雑になる．RAISE モデルでは単純な制御構造を持ち，実行メカニズムも比較的簡単に実現できる．

現在の RAISEモデルでは，並行した作業を持つ系列には対応していない．これについては，将来への課題である．また，複数の開発者によるプロセスへの対応も検討中であるが，このとき，個々の開発者の作業系列は RAISEモデルがそのまま適用出来，開発者間の通信が問題になると考えている．

6．まとめ

ソフトウェア開発プロセスの性質記述をもとに，インタラクティブな開発支援環境を構築するためのモデル RAISE の提案を行なった．RAISE モデルでは，制約条件つきのメニューによる選択／実行を基本とした動作により，作業の自由度，正しい作業系列の生成，優れた操作性を実現する．現在，RAISE モデルに基づくメニュー記述言語を設計中である．

［参考文献］

[1] 飯田元:"開発支援系における作業系列や対象の実行時選択機能に関する研究",大阪大学基礎工学部修士学位論文(1990).

[2] K.Inoue, T.Ogihara, T.Kikuno, and K.Torii : " A formal Adaptation Method for Process Descriptions " , Proc. of 11th ICSE, pp.145-153(1989).

[3] 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-1), J72-D-I, 10, pp. 742-749(1989).

[4] G.E.Kaiser:"Experience with Marvel", Proc.of the 5th International Software Process Workshop : Experience with Software Process Models, (1989).

[5] L.Osterweil: " Software Processes are Software Too", Proc. of 9th ICSE, pp.2-13(1987).

[6] L.G.Williams:"Software Process Modeling: A Behavioral Approach", Proc. of 10th ICSE, pp.174-186(1988).

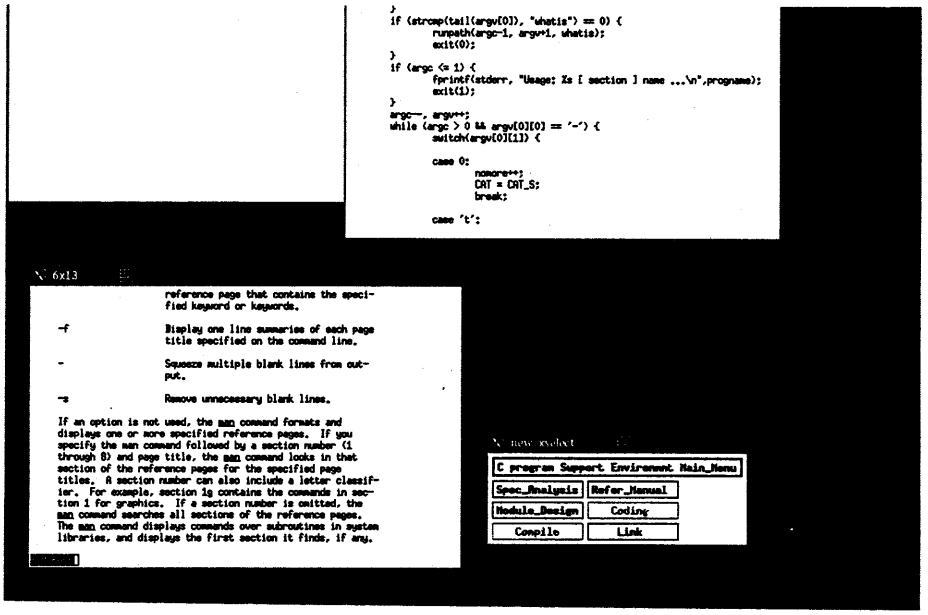


図3　開発支援系の実行画面