Vol. 36 No. 5　　情　報　処　理　　May 1995

解　説

IPS

ソフトウェアプロセス

2. 代表的なプロセス記述言語の特徴†
——共通例題による比較——

鈴　木　正　人††

1. はじめに

ソフトウェアプロセス記述モデルに関してはここ数年の間に多くの研究が行われており，提唱されているモデルの数も数十個に達している．本稿の目的はこれら多数のプロセス記述モデルに関する研究の歴史を説明するという立場から，ISPW 6 で M. Kellner らによって提唱された例題プロセスの記述をとおして代表的なプロセス記述モデルの特徴を比較し，プロセス記述モデルに関する研究動向と今後の方向性を示すものである．

2. プロセス記述言語研究の軌跡と問題点

これまでソフトウェアプロセスに関する研究活動の中心的役割を演じてきたものは 1980 年代半ばに創設された国際ワークショップ（ISPW）といってもよいだろう．この中で第 4 回，および第 5 回のワークショップではソフトウェアプロセス記述モデルに対する要求として以下のようなものが指摘されている[1),2)]．

●表現能力

モデルの基本となる能力であり，アクティビティとプロダクトの関係，アクティビティの実行順序，タイミング，並行，実世界との関係（スケジューリング，予算や作業者の割当て/管理，協調と共動）などが記述できるか．

●分析/検証，シミュレーション能力

プロセスの進行に従ってモニタリングを行い，実行結果の分析や検証が可能か．また新たなプロセスに対して（実世界で実行することなしに）シミュレーションにより結果の予測が可能か．

●適用/進化の容易さ

プロセスは実行中にも実行状況に応じて常に変化する．このような動的な変更に対して記述はどの程度有効であるか．進化のためのガイドラインを示すことが可能か．

●理解のしやすさ

大規模なプロセスにおいても理解が容易であるかどうか．特に階層性，直交性，記述の分解が可能か．

これらの観点からソフトウェアプロセス記述モデルを評価するための尺度として，M. Kellner らによって設定され，第 6 回の国際ワークショップで発表されたものが ISPW 6 共通例題である．

3. ISPW6 における共通例題とその意義

ISPW 6 共通例題はモデルに対する理解と比較を容易にするため，この例題は実世界のソフトウェアプロセスに見られる問題を数多く含むように注意深く設計されており，プロセス記述言語の設計者は自分の言語を使用してこの例題を記述して見ることで，それぞれのモデルの長所と短所を発見することができる．例題は一つの核問題（core problem）といくつかの拡張問題（optional extension）から構成されている．核問題は，ソフトウェア変更プロセスの比較的限定された部分に注目し，ソフトウェアシステムの設計，コーディング，単体テスト，比較的局所的な変更の管理に焦点を当てている．これに対して，拡張問題は与えられたモデリングアプローチの核問題によっては示すことのできない能力を示すさまざまな機会を与えることを目的としている．例題の詳細については本特集の付録を参照されたい．

4. 共通例題の記述例に基づく比較

この例題に対しては全部で 18 個の解答が寄せ

† Features of Typical Software Process Description Models —Comparison with A Common Example— by Masato SUZUKI (School of Information Science, Japan Advanced Institute of Science and Technology).

†† 北陸先端科学技術大学院大学情報科学研究科

られた．本章ではその中から代表的な4つのモデル（APPL/A, MSL/Marvel, Statemate, HFSP）を採りあげ，それぞれのアプローチを簡単に説明する．

4.1 APPL/A

APPL/A[3]はS.Sutton, Jr.らによって提唱された言語であり，手続き型言語Adaに以下のような拡張を行ったものである．

1. relation： オブジェクトのpersistencyを記述するものでデータを表すタプルの定義，操作を表す複数のエントリを持つ．

2. trigger： relationのエントリの呼出しが発生したときおよび完了したときに生成されるシグナルに応答するためのユニットである．

3. predicate： relationの間の関係を論理式により記述したものであり，relationに対する操作の実行後にpredicateが成立している場合は操作を完了する．成立していない場合は操作は取り消され，例外が発生する．

APPL/Aによる例題の記述は以下のようなアプローチをとっている．

●個々のアクティビティは同名のAdaのタスクで表される．これらはその役割（作業者が行うか，マネージャが行うか）によって二つのグループ（Change_Management_TaskおよびChange_Engineering_Tasks）に分けられる．個々のタスクはこれら二つのサブタスクとして階層的に構成されている．

●すべてのプロダクトはSoftware_Productsという名前のパッケージの中でrelationとして記述される．relationにはtaskの割当て，スケジュール，実行状態（実行中，実行終了）なども記述される．またrelation間の関係を表すpredicateもここに記述される．

解答の一部（Review_Design）をスクリプト1に示す．

4.2 MSL/Marvel

MSL/Marvel[4]はG.Kaiserらによって提唱されたE-A-Rモデルに基づくルールベースの環境である．MSL/Marvelの記述はstrategyの集合である．各strategyはデータモデルの定義（objectbase部）とルールの定義（rule部）を持つ．rule部に記述された条件が満たされるとそのルールがactivateされ，実行が行われる．designに関するルールの一部を以下に示す．

```
1  rules
2  design[?c :  CHANGE] :
3  (and (exists MODULE ?m suchthat…)
4  (exists DESIGN ?d suchthat…))
5  ……
6  [DESIGN design ?d]
7  (?d.ready=Yes) ;
```

3，4行目が条件を表す．?が付いているのが変数であり，suchthat以下の条件に当てはまるデータの名前が代入される．6行目はツールDESIGNの呼出しである．呼出しが終了すると7行目で該当するdesignの属性がセットされる．

MSL/Marvelでは以下のような手順によって例題を記述している．

●プロダクトを実体（Entity），プロダクトの構造を属性（Attribute）により記述する．プロダクトはクラスとして階層関係を持つ．また関連のあるEntityをリンクによって結合する．

●プロダクトに対する操作を分析し，build（変更の実行），archive（ソースのアーカイブ），design（設計の変更およびレビュー）などのstrategyに分類する．

●各strategyに対してローカルなデータモデルとルールを記述する．データの状態の変化などはpre/post-conditionとして記述する．

MSL/Marvelによる記述の一部をスクリプト2に示す．

4.3 Statemate™

Statemate™☆5)は本来はリアクティブシステムの仕様記述のために考えられたシステムである．Stetemateではシステムを構造（structural），機能（functional），挙動（behavioral）の三つの観点からとらえ，それぞれをModuleChart, ActivityChart, State-Chartによって表現している．

Statemateでの記述の手順は以下のようになる．

●アクティビティとプロダクトを分析し，プロダクトフローをActivity Chartによって記述する．アクティビティの分解はネストによって表現される．実際のプロダクトのフローを実線で，コ

☆ i-Logix社の登録商標

ントロールのフローを点線で記述する．

● Activity Chart の各アクティビティに対して，内部状態を洗い出す．このとき直交する状態を発見し，分離する．この例ではSCHED ASSIGN_TASK と MONITOR_PROGRESS がマネージャプロセスの状態，他はデザイナプロセスの状態である．初期状態として IDLE を追加し，状態遷移の原因となるイベントと条件分岐(C)，終了状態(T)を記述する．

●構造に関しては作業者（QE，DE 等）および作業者間で交信される情報，イベント，通信チャネルなどを記述する．

Statemate は図的言語であり，いわゆるスクリプトは存在しない．

4.4 HFSP

HFSP[6] は属性文法に基づく関数的，階層的計算モデル HFP[7] に，筆者がソフトウェアプロセスに必要とされる機能を拡張する形で提唱したプロセス記述モデルである．HFSP ではソフトウェアプロセスを以下の三つの側面から記述する．

●機能的側面

アクティビティを数学的関数としてとらえ，入出力プロダクトの関係を記述する．複雑なアクティビティは単純なアクティビティに階層的に分解される．これ以上分解されないアクティビティはツールを使用して直接実行される．

●挙動的側面

実行シーケンスは繰返しなどの制御構造を使用して explicit に記述されるものと，属性の依存関係により implicit に記述されるものがある．また本例題の解答では使用しなかったが，実行順序を動的に変更するための機構をいくつか備えている．

●組織的側面

単一の作業者（個人またはチーム）によって実行されるアクティビティの集まりを単位とし，他の実行単位の実行状態を動的に作成，制御したり，通信を行うための機構としてメタオペレーションを備えている．

HFSP の機能的側面の記述の一部を以下に示す．(タスク名は長すぎるため略号を用いている)

```
1  PM(rc|result)
2  =>SAT(rc|plan, pid)
3  MP(plan, pid|plan.out, log.out)
```

()の前にあるのがタスク（アクティビティ）名，|の変数がタスクの入力，後にある変数が出力である．変数名は変数のタイプも同時に表す．同じタイプの変数が同一コンテキストに複数存在する場合は．の後の名前で区別する．=>は左辺のタスク PM が右辺のタスク SAT と MP に分解されることを意味する．同名の変数は同じ内容を示す．最大の特徴は分解の右辺の実行順序は入出力の間の依存関係のみによって（すなわち記述された順序とは無関係に）決定されることである．たとえば SAT の出力はそのまま入力として MP に渡されるため，SAT は MP より先に実行されなければならない．分解には単純な分解の他に，case による条件つき分解，repeat による反復などがある．詳細は参考文献 6)等を参照されたい．

HFSP による例題の記述は以下のようなアプローチによって行われている．

●例題の 2.1 に記述された各プロセスをその実行者に注目してデザイナのプロセスとマネージャのプロセスに分類する．デザイナプロセスはマネージャプロセスによって実行の過程で作成されるものとする．

●デザイナプロセスのトップレベルを定め(ModifyAndTest)，各タスクをこれの階層的分解として記述し各アクティビティの入出力プロダクトを定義する．同時に persitent なオブジェクトを決定し，オブジェクトベースの定義を行う．

●制御シーケンスが明確なアクティビティ(例：ModifyDesign と ReviewDesign) については，ループ構造などを用いて制御構造を記述する．

●必要ならばモニタリング，再スケジューリングなど動的な操作をメタオペレーションにより実現する．

記述の一部をスクリプト 3 に示す．

4.5 解答の比較

これら 4 つのモデルの特徴を比較したものを**表-1** に示す．なおこれは参考文献 8)で示されたものを元に筆者が若干の項目の追加を行ったものである．

表-1 プロセスモデルの比較

項　　目	APPL/A	MSL/Marvel	Statemate	HFSP
テキスト(T)/グラフィック(G)	T	T	G	T(G)
記述の粒度	小	小	中	大(中)
記述の深さ	深	中	深	中
記述の局所性	△	△	×	○
アクティビティ	○	△	○	○
プロダクト	◎	◎	○	△
アクティビティの実行順序	△	△	○	○
並列性/並行性	○	△	◎	◎
手戻り/フィードバック	△	×	×	○
スケジューリング	△	×	△	△
プロセスの動的作成/制御	×	×	×	◎
分析/検証	◎	△	◎	×
シミュレーション	○	○	◎	△
マネジメント	○	△	○	△
適用/進化	△	×	×	△
理解しやすさ	△	×	○	○

5. 国内外における研究動向

5.1 はこにわ

はこにわ[9]は大阪大学基礎工学部情報工学科の鳥居，井上，飯田らのグループが中心となり進めているプロジェクトであり，複数の作業者の間で協調開発プロジェクトの管理および支援を目標にしたものである．

はこにわは大きく分けて管理者用の**サーバシステム**と，作業者用のクライアントシステム（**タスクオーガナイザ**）より構成される．各タスクの内容は文脈自由言語（CFG）に基づく形式言語で記述，実行される．

はこにわは第6回国際ワークショップではその基本的な考え方が発表されたのみにとどまっており，例題に対する解答は与えられていない．

5.2 Process WEAVER

もう一つヨーロッパでの研究の成果として，フランスCap Gemini Innovation社のC.Fernströmらによる Process WEAVER[10]をあげておこう．これはソフトウェア開発プロジェクトにおけるプロセスおよびリポジトリを管理するための統合ツールであり，すでに市販されている．

Process WEAVERの機能は作業を定義する**プロセスモデリング**と，実際に作業者が使用する環境である**プロセスサポート**に分けられる．前者はプロジェクトの作業を定義する**メソッドエディタ**，作業内容やプロダクトを定義する**アクティビティエディタ**，プロセスの挙動をペトリネットにより記述するための**プロセスエディタ**などのツール群から構成され，後者は各作業者が自分に割り当てられたタスクの内容をアイコンを通じて確認，実行するためのツールの集まりである．

Process WEAVERも1992年の第2回ソフトウェアプロセス国際会議（ICSP 2）で発表されたのが最初であり，例題の解答は与えられていない．

また本稿で示さなかったモデルとしてAP 5, OPIUM, KyotoDB, PDL, OBJ 2などによる解答も寄せられているが，これらに対する詳細については参考文献8)を参照されたい．

6. おわりに―プロセス言語研究の今後の方向性

共通例題によって代表的な4つのプロセス記述モデルおよび言語の特徴を紹介し，比較を行っ

た．ISPW 6例題プロセスに関しては全般的に資源や組織的側面について記述した解答が少なかった．この原因は組織的側面はいわゆるill-structuredなデータが多く，形式的に記述するのが一般に困難であるためであるが，組織的側面の記述に対して例題に詳細な設定がなかったことも原因として考えられる．たとえば以下のような項目が記述の対象から外れている．

●チーム作業

チームで開発作業を行う場合に発生する問題点で，複数のプログラマが並行して作業をする場合のコミュニケーションなど．

●プロセスの動的な変更

プロセス記述自身の変更や進化を扱った問題で，例外処理のための一時的な記述の修正など．

●人間的ファクタの記述

組織の構造，組織内における役割の問題，その動的な変化など．

これらのうちいくつかはISPW 7において拡張という形で例題に組み込まれ，それに対する解答もいくつか寄せられている．このような問題を記述できるようなモデルを構築することが，今後のプロセス記述モデルに関する研究の方向性のひとつであると考える．

参 考 文 献

1) ISPW. Proceedings of the 4th International Software Process Workshop (1988).
2) ISPW. Proceedings of the 5th International Software Process Workshop (1989).
3) Osteweil, L. J., Sutton, Jr. S. M. and Heimbigner, D. : Language Constructs for Managing Change in Process-Centered Environments, In Proceedings of the 4th Symposium of Practical Software Development Environment (1990).
4) Kaiser, G. E. and Feiler, P. H. : An Architecture for Intelligent Assistance in Software Development, In Proceedings of the 9th International Conference on Software Engineering, pp. 180-188 (Mar. 1987).
5) Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sheramn, R., Shtull-Trauring, A. and Trankhtenbrot, M. : Statemate : A Working Environment for the Development of Complex Reactive Systems, IEEE Transaction on Software Engineering, 16 (4) (1990).
6) Katayama, T. : Hierarchical and Functional Software Process Description, In Proceedings of the 4th International Software Process Workshop (1988).
7) Katayama, T. : HFP, Λ Hierarchical and Functional Programming, In Proceedings of the 5th International Conference on Software Engineering, pp. 343-353 (1981).
8) Kellner, M. I. and Rombach, H. D. : Comparisons of Software Process Descriptions, In Proceedings of the 6th International Software Process Workshop, pp. 7-18 (1990).
9) Iida, H., Mimura, K., Inoue, K. and Torii, K. : Hakoniwa : Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model, In Proceedings of the 2nd international conference on the Software Processes, pp. 64-74 (1993).
10) Fernstrom, C. : PROCESS WEAVER : Adding Process Support to UNIX, In Proceedings of the 2nd International Conference on the Software Processes, pp. 12-26 (1993).

(平成6年10月27日受付)

鈴木 正人

1964年生．1987年東京工業大学工学部情報工学科卒業．1989年同大学院理工学研究科情報工学専攻修士課程修了．1992年同大学院博士課程修了．工学博士．同年より北陸先端科学技術大学院大学情報科学研究科助手，現在に至る．ソフトウェアプロセス，ソフトウェア設計/開発環境，高信頼性ソフトウェアに関する研究に従事．日本ソフトウェア科学会，ACM各会員．

付 録

```
package Software_Product is
---単純化されたソフトウェアプロダクトのための関係
type product_type_enum=(reqt, design, src,…)
……
relation Design_Reviews is
  type design_review_tuple is
    proj_id : proj_id_type ;
    ……
  end tuple ;
entries
---入力，更新，検索などの基本的操作
```

```
End Design_Reviews ;
……
End Software_Products ;
task body Review_Design is
---設計のレビューを行うタスク
Begin
  proj_id : proj_id_type ;
  DE_id, QAE_id, SE1_id, SE2_id, mgr_id :
  emp_id_type ;
  ……
  task Eng_Review_Design is
  ---設計チームが実際に設計のレビューを行う.
  ---チーム内でのインタラクションを支援する.
  entry start_up (design_eng_id, qa_eng_id,
  …) ;
  end Eng_Review_Design
  task body Eng_Review_Design is sepa-
  rate ;
Begin
accept start_up(…)do
  mgr_id : =manager_id ;
  DE_id : =design_eng_id ;
  …
end start_up ;
eng_review_design.start_up (DE_id,…) ;
select
  ---正常終了
  accept completed do
    Monior_Progress.review_design
    results (true) ;
  end completed ;
or
  ---マネージャまたはスケジューラによる異常
  終了
  accept deactivate do
  ---チームによる実際のタスクを中断する
  abort eng_review_design ;
  ---メッセージの送信
  send_msg (DE_id, "Review Design は異常
  終了しました.
      設計を更新しますか (y/n) ?") ;
  recv_msg (DE_id, yn) ;
  ---結果の登録処理---
  ……
end select ;
End Review_Design ;
```

スクリプト1: APPL/A による記述(Review Design)

```
strategy data_model
imports none ; exports all ;
#クラス定義
objectbase
### ENTITY はクラス階層のルート
PROJECT : : superclass ENTITY ;
    name : string
    status : (Release, Maintenance, Devel-
          opment)
      =Development ;
    archive-status : (Archived, NotAr-
                chived)
      =NotArchived ;
    build_status : (AllBuilt, NotBuilt)
        =NotBuilt ;
    ……
end
……
DESIGN : : superclass DDCUMENT ;
    module : link MODULE ;
    status : (NotApproved, MajorChanges,
          MinorChanges, Approved) =
          NotApproved ;
    ready : (Yes, No) =No ;
    ……
end
REVIEW : : superclass MEETING ;
  identifier : string ;
  status : (NotDone, Done, Reported) =Not-
  Done ;
  ……
end
end objectbase
……
rules
design[?c : CHANGE] :
    (and (exists MODULE ?m
        suchthat (?m=?c.module))
    (exists DESIGN ?d suchthat
      (member[?d ?m.design])))
```

```
  :
  (and (?c.designer=CurrentUser)
    (or (?d.status=NotApproved)
      (?d.status=MinorChanges)
      (?d.stants=MajorChanges)))
  [DESIGN design ?d]#ツールの起動
  (?d.ready=Yes) ;
review[?r : REVIEW] :
    (exists DESIGN ?d
        suchthat (?d=?r.design))
  :
  (and (?r.status=NotDone)
       (?d.ready=Yes)
      (or (?d.status=NotApproved)
        (?d.status=MajorChanges))
      (?r.scheduled_date=CurrentTime))
  [REVIEWTEAM review ?d ?r]
  (and (?r.status=Done)
    (?d.status=Approved)) ;
.....
```

スクリプト2： MSL/Marvelによる記述 (Review・Design)

```
type
    changeReq, feedback=string
    design, code, plan, resources,
      history, logfile=file
    message=(kind, body)
    kind=enum('major, 'minor, 'approved,
      'finished, 'cancelled,…)
    body=string
    result=message
    process_id=id_type
    .....
object design, code, plan, resources, logfile
//デザイナプロセス
//トップレベル
ModifyAndTest (changeReq|result.review,
result.test)
=>ModifyAndReviewDesign (changeReq
      |design.out, result.review)
    ModifyTestPlan (changeReq|plan.out)
    ModifyCodeAndTest (design.out, plan.
      out|result.test)
//設計の修正とレビュー
ModiayAndReviewDesign (changeReq
  |design.out, result.review)=>
    repeat{
      ModifyDesign (feedback, * design.
      old|* design.new)
      ReviewDesign(design.new|feedback,
      result)
    }until result'kind==’approved
    where
      feedback=makemajorfeedback(chan-
      geReq)
      design.out=design.new
      result.review=result
ModifyDesign (feedback,* design.in
  |* design.out)=>
    [case feedback 'kind==’major=>
      do_major_change(
        feedback, design.in|design.out)
    case feedback'kind==’minor=>
      do_minor_change (
        feedback, design.in|design.out)
    ]
ReviewDesign(design|feedback, result)=>
    doReview(design|result)
    [case result'kind==’approved=>
return
        where feedback=nil
    case result'kind==’minor=>return
      where feedback=MinorFeedback
        (design)
    case result'kind==’major=>return
      where feedback=MajorFeedback
        (design)
    ]
.....
```

スクリプト3： HFSPによる記述 (Review・Design)