ソフトウェア工学 126－7
（2000．3．9）

開発環境中におけるオブジェクトの状態を用いた進捗状況表示モデル

湯本貴雪† 松下 誠† 井上克郎†‡

† 大阪大学大学院基礎工学研究科
〒560-8531 大阪府豊中市待兼山町 1-3
06-6850-6571

‡ 奈良先端科学技術大学院大学情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5
0743-72-5314

あらまし

既存のプロセスモデルを用いたソフトウェア開発管理システムでは，動的に変化する開発の状況を正確に表現することは困難である．本研究では，既に提案しているソフトウェアプロセスモデルを拡張し，進捗状況を表すことを目的としたモデルである，MonoProcessEX の提案を行う．また，既に試作している開発管理システムを本モデルを用いて拡張し，進捗管理の機能を構築する．MonoProcessEX では，開発環境中の生成物や資源を表すオブジェクトの持つ状態を組み合わせることにより，開発環境を抽象化する．そのため，開発環境の状態の変化を容易に把握することができる．これにより，開発の状況を正確に表現し，把握することができる．また，拡張したシステムを用いて効率の良い開発管理が行える．

キーワード： ソフトウェアプロセス，プロセスモデル，ソフトウェア開発管理

Project Management Model with Object States of Software Development Environment

Takayuki Yumoto† Makoto Matusita† Katsuro Inoue†‡

† Graduate School of Engineering Science,
Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka, 560-8531, Japan
+81 6 6850 6571

‡ Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma,
Nara, 630-0101, Japan
+81 743 72 5314

Abstract

Existing software project management system based on a software process model does not illustrate an actual ongoing software development environment. In this paper, we propose an software project management model MonoProcessEX which is based on our software process model. We also implements software project management features by extending the our software development environment. MonoProcessEX employs the states of software objects which represents products and/or resources in actual software development environment to illustrate an abstracted state of software development environment. Using MonoProcessEX model, software development environment can be easily grasped and pricisely illustrated.

key words： Software Process, Process Model, Software Project Management

1 はじめに

ソフトウェアの開発過程 (ソフトウェアプロセス) を表現する，ソフトウェアプロセスモデルの研究が広く行われている．また，それらのプロセスモデルで表現されたソフトウェアプロセスを実行する開発管理環境の研究も盛んに行われている．これらの環境では，開発の状況や行う作業をプロセスモデルを用いて表し，開発を管理することで，ソフトウェアの生産性を向上させることを目的としている．

一般的に，ソフトウェアの開発管理には，開発の進捗状況を正確に把握し，その状況に応じて実行すべき作業を決定し，開発者に掲示することが重要である．しかし，従来のプロセスモデルやその実行環境 [1, 2, 3, 4, 5, 6] では，プロセスを実行した時の振る舞いをあらかじめ予測して記述し，記述通りに開発を行うことが前提とされている．ところが，実際の開発プロセスは単純ではなく，作業の順序が変更され，状況に応じて作業が決定される．そのため，あらかじめ予測して作業の流れを記述することは難しく，開発の状況を正確に表現，把握することは困難である．また，状況に応じた作業についても，正確な作業を掲示することは難しい．

開発の状況を把握する際には，プロダクトやプロセスなど様々な要因を合わせて状況を把握することが一般的である．また，行う作業は，その把握した状況に応じて決定される．そのため，開発の状況を正確に表すためには，プロダクトや資源などを一貫して表現できるモデルが必要となる．我々は，既にプロセスモデル MonoProcess を提案している [7, 8]．また，そのモデルに基づく開発管理システムを既に試作している．本プロセスモデルでは開発環境中の生成物や資源をオブジェクトとして定義する．また，生成物や資源の性質をオブジェクトの属性として表し，オブジェクトのメソドを通して生成物や資源を操作する．これにより開発環境の変化はオブジェクトの属性の変化で表現される．これを用いて，様々な要因となるオブジェクトの属性の情報を合わせて状況を把握することができる．

本研究では，MonoProcess で表現される開発環境中のオブジェクトの状態を組み合わせることで，進捗状況の表示，状況に応じた作業の掲示を行うためのモデルを提案する．本モデルでは，オブジェクト群で表される開発環境を抽象化して，その開発環境の状態の変化を表示するための枠組を提供する．

また，既に試作を行った開発管理システムに進捗状況の把握を行うための状態表示ビューアと作業のガイダンス機能の構築を行う．本システムでは，状態表示ビューアによって進捗管理が行え，開発作業のガイダンス機能によって効率の良い開発を行うことができる．

以降，2 節では我々の提案しているプロセスモデル MonoProcess について述べる．そして，3 節ではこれから提案する進捗状況表示モデルについて述べる．また，4 節で共通例題 [9] に提案するモデルを適用した結果について述べる．5 節では提案したモデルを用いた開発管理システムについて述べる．さらに，6 節で関連する研究に関して考察を述べる．最後に，7 節で本研究のまとめと今後の課題を述べる．

2 プロセスモデル MonoProcess

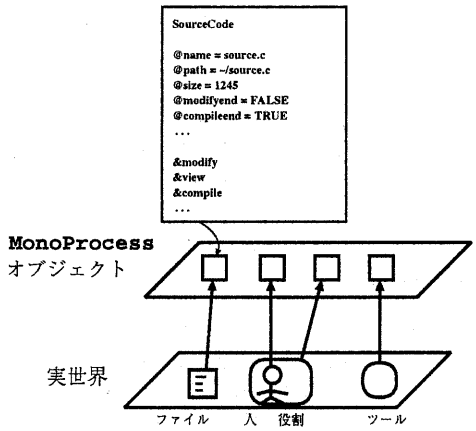


図 1: プロセスモデル MonoProcess の概要図

我々は開発環境中の生成物や資源に着目したプロセスモデル MonoProcess を提案している [7, 8]．プロセスモデル MonoProcess は，開発環境中の生成物や資源をオブジェクトとして定義する．生成物や資源の性質はオブジェクトの属性として定義し，生成物や資源への操作はオブジェクトのメソドとして定義する．オブジェクトのメソドを実行することで開発を進め，開発環境の変化はオブジェクトの属性の変化として表される．

図 1 でプロセスモデル MonoProcess の概要とオブジェクトの定義の例を示す．定義の例としてはソースコードオブジェクトの例を載せている．@ で示されるのは属性であり名前，サイズ，修正済などの情報を表している．&で示されるのはメソドであり，修正する，参照するなどの操作を定義している．

3 MonoProcessEX

進捗状況表示モデル MonoProcessEX は，プロセスモデル MonoProcess で表現されるオブジェクトの状態を組み合わせることで進捗管理，状況に応じた作業の掲示，および分析，分析結果の適用を行える枠組を提供する．

一般的に，開発状況の把握は様々な要因を合わせて行う．また，行う作業は，その把握した状況に応じて決定する．開発の任意の時点で状況を把握するための要因は異なり，各時点で決定された作業を並行して行っている．

本モデルで，開発環境中のオブジェクト群の属性は，開発の状況を表す要因となる．そのため，それらの状況を表す要素を属性を用いて定義し，要素を組み合わせることで開発環境を抽象化する．そ

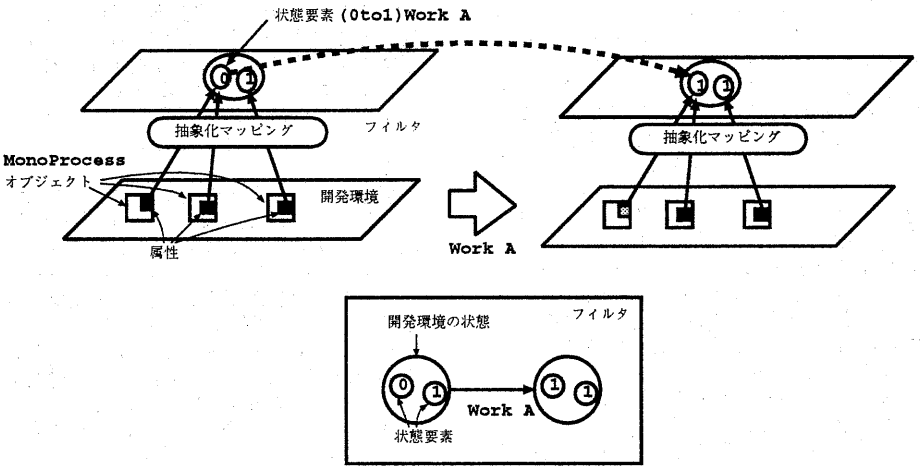


図 2: MonoProcessEX の概要図

して，各要素の変化により開発環境の状態変化を表す．これにより，開発の状況を素直に表現することができる．

また，状況を表す要素に対して作業を定義することで，開発の任意の時点で行う作業は，それらの組み合わせにより表すことができる．

3.1 MonoProcessEx の概要

本モデルでは，属性の変化の情報を用いて，開発環境の状態変化を状態遷移図で表現する．属性の変化は MonoProcess で定義されるため，開発の任意の時点で，開発環境の変化を抽出する観点を追加することができる．状態や状態の変化は，あらかじめ記述しなくても過去の情報で過去の変化の遷移を表すことができる．これを見ることで開発の状況を正確に把握でき，様々な観点から分析を行うことができる．

開発環境の各状態において行う作業は書く状態で必要とされる作業より導出できるため，状況に応じて作業を掲示できる．

さらに，分析の結果判明した問題を引き起こす原因を状態を定義するための要素として追加し，回復するため作業を新しく定義することにより，将来，同様の状況が生じた際に回復する作業を動的に追加，変更できる．

また，状態要素の値を変化させるための作業量を整数値で定義することにより，過去に行った作業量や任意の状態間での作業量を計算できる．これにより進捗状況を把握することができる．

3.2 MonoProcessEX の構成

本モデルは，フィルタ，状態要素，シーケンスから構成される．

フィルタは開発環境の状態を抽出するもので，開発環境の状態を状態要素の組合せで表す．状態要素はオブジェクトの属性の値の演算により論理値を取り，開発環境の状態の変化は状態要素の値の変化で表される．

また，状態要素の値を変化させるための作業を定義し，状態要素を組み合わせることで，開発の任意の時点での状態において行わなければならない作業を表す．シーケンスは状態要素の変化の順序を制御し，それにより作業の順序を制御する．

図 2 で，進捗状況表示モデルの概要を示す．フィルタ上でオブジェクトの属性の演算により状態要素を定義し，抽象化された開発環境の状態を状態要素の組み合わせで表している．開発環境の状態の変化は状態要素の値の変化で表現されている．

3.2.1 フィルタ

フィルタはオブジェクト群で表される開発環境を抽象化して状態を表し，開発環境の状態変化を抽出し表示するフレームである．フィルタは状態要素を定義し状態要素の値を変化させるために必要な作業，作業量を定義することにより開発管理および分析を行う目的で作成できる．

フィルタは開発の任意の時点で複数作成できる．そのため様々な観点で分析できる．また状態要素も任意の時点でフィルタに付け加えることができるため，状況に応じた作業を追加することができ，その状況で作業が掲示できる．

フィルタは以下の内容で構成される

- 状態要素の集合：観点を特徴づける状態要素群
- シーケンスの集合：状態要素の値の変化を順序づける
- 状態:ある観点で見た開発環境の状態 (丸で表現)
- 状態要素の値の遷移：開発環境の変化を表す一つの状態要素の変化 (矢印で表現)

3.2.2 状態要素

状態要素は開発環境の抽象化された状態の極小の要素である．状態要素の組合せにより開発環境の状態を表し．状態要素の値の変化で開発環境の状態の変化を表す．

状態要素は，それを特長づける状態式を持つ．状態式はオブジェクトの属性を用いた式で論理値を

とる．その論理値が状態要素の値となる．また，状態要素の値は論理値であるため状態要素の値の遷移は0から1と1から0の二つある．その二つの値の遷移 ($0to1$,$1to0$ と表現) どちらかに対して値を変化させるための作業とその作業量を定義できる．さらに，両方の遷移に対して遷移終了時に自動的に実行する作業 (例えば作業終了メッセージを送る) を定義できる．

値を遷移させるための作業は，基本的に二つの観点で定義する．それは，開発を進めるために必要な作業と問題が発生した時に回復するために必要な作業である．そのため理想的に開発を進めるうえで余分になる作業は，開発を進めている時に生じる作業の後戻りと問題が発生した時に回復するために必要な作業である．

状態要素は以下の内容で構成される．

- 名前：状態要素の名前
- 状態式：オブジェクトの属性を用いた式 (論理値)
  開発環境の状態の変化を特徴づけ，抽象化する．
- 値の遷移：状態式の値の変化 ($0to1$,$1to0$ の二つがある)
  - 開始プロシージャ：どちらか一方の値の遷移に対して必要な作業開発環境を変化させるために必要な作業
  - 終了プロシージャ：各遷移終了時に自動的に実行される作業開発環境の変化によって自動的に実行される作業

名前：Module1_Modify
状態式：Module1@ModifyEnd=TRUE
開始プロシージャ(0to1)：Module1&Modify
終了プロシージャ(0to1)：Module1&ModifyEndMessage
終了プロシージャ(1to0)：Module1&ModifyAgainMessage

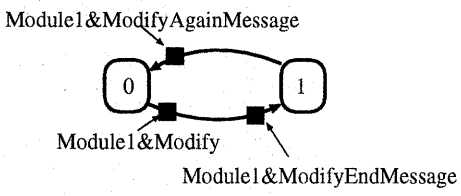


図 3: 状態要素の例

図 3 に状態要素の例を示す．モジュール 1 の修正に関する状態要素で値が 0 であれば修正前で 1 であれば修正済であることを示す．$Module1$ の $Modify$ メソドを実行することで $Module1$ の $ModifyEnd$ 属性が $TRUE$ に変わるため修正済の値へと遷移する．終了時に終了通知，再修正の際の再修正通知が自動で行われる．

3.2.3 シーケンス

シーケンスは状態要素の値を遷移させるための条件を表す．つまり，状態要素で定義されている作業を行うための条件を示している．これにより，開発環境の状態変化を順序づける．

シーケンスは以下の内容で構成される．

- 条件要素式：状態要素を用いた簡単な論理式
- 実行要素：ある状態要素 (実行要素)

条件要素式は状態要素の値を指定しその論理和，論理積で定義する．条件要素式が真の時，実行要素の開始プロシージャが定義されているものとする．つまり，その時のみ実行することができる．

シーケンスは条件要素式 => 実行要素式と表現する．

図 4 でシーケンスの例を示す．X,Y は状態要素で，$X = 1$ は条件要素式，Y は実行要素である．$Procedurey_1$ は開始プロシージャで $Procedurey_2$, $Procedurey_3$ は終了プロシージャである．終了プロシージャは条件要素式の値に関係なく Y の値の遷移によって自動的に実行される．しかし，開始プロシージャである $Procedure1$ は $(X,Y) = (1,0)$ の時のみ実行できる．つまり，状態要素 X を遷移させるための作業 $Procedurex$ を実行した後でなければ，状態要素 Y を遷移させるための作業 $Procedurey_1$ は実行できない．

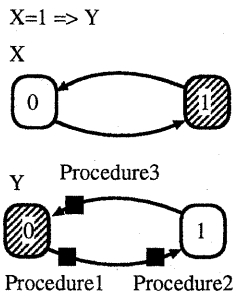
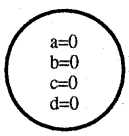


図 4: シーケンスの例

3.3 開発環境の状態に応じた作業の割出し

フィルタ上の状態要素とシーケンスによって開発環境の状態において実行しなければならない作業を計算することができる．



a:start(0to1):work A
b:start(0to1):work B
c:start(0to1):work C
d:start(1to0):work D

A=1 => B

図 5: 状態に応じた作業の割出し例

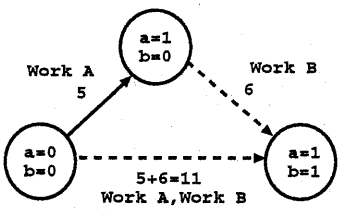
図??を用いて計算の例を示す．フィルタ上に 4 つの状態要素と 1 つのシーケンスが定義されているとする．状態要素は a, b, c, d で，a, b, c に関しては状態要素の値が 0 の時 1 にするための作業がそれぞれ $WorkA$, $WorkB$, $WorkC$ として定義され，d に関しては状態要素の値が 1 の時 0 にするための作業が $WorkD$ として定義されているものとする．シーケンスは $A = 1$ => B が定義されていて，$A = 1$ の時に B に付随する作業が行

えることを意味する．

今，$(a,b,c,d)=(0,0,0,0)$ の状態であったとする．実行できる作業は a，b，c，d に関して 0 から 1 にする作業である．つまり状態要素の定義から $WorkA$，$WorkB$，$WorkC$ が実行できる．シーケンスでは B の作業は，$A=1$ の時にのみ実行できるため残りの $WorkA$，$WorkC$ だけを実行することができる．

このようにして，各状態において掲示される作業が決定する．このフィルタに関しては，$(a,b,c,d)=(1,1,1,0)$ の状態の時に実行できる作業が無くなる．フィルタは複数用意することができるため，ある時点での開発環境において実行しなければならない作業は，各フィルタで実行できる作業の和になる．そして，それらの作業が掲示される．それにより，開発プロセスを望む観点でフィルタで分割し進捗管理できる．

3.4 フィルタを用いた進捗計算



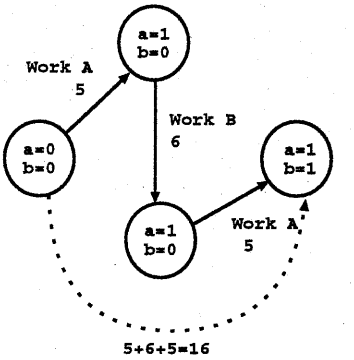

```
a:WorkAEnd==TRUE
  Start(0to1):Work A (5)
b:WorkBEnd==TRUE
  Start(0to1):Work B (6)

a=1 => b
```

図 6: 進捗計算 1

図 6 は簡単な進捗計算のための例である．a，b という状態要素が定義されていて，0 から 1 に値を遷移するための作業としてそれぞれ $WorkA$，$WorkB$ が定義されているとし，$WorkA$，$WorkB$ の作業量をそれぞれ 5，6 とする．$(a,b)=(0,0)$ の状態から作業を開始して $(a,b)=(1,1)$ の状態が作業の終了状態とする．この時，a，b 両方の値の遷移が必要であり遷移に必要な作業 11 を全体の作業量として概算できる．実際の開発によって $(a,b)=(1,0)$ の状態に遷移したとすると，作業 A が終了して残り b の状態遷移に必要な作業 $WorkB$ とその作業量 6 が必要となることが分かる．つまり残りの作業量 11 から 6 へと進捗している．残りの作業の割合は全体が 11 であるため 11/11 から 6/11 に変化している．

また，$WorkB$ が終了して $(a,b)=(0,1)$ の状態に達したとする．この場合，$(a,b)=(1,0)$ と $(a,b)=(0,1)$ の残りの作業量は 6 と 5 であるため，終了までの進捗は 1 だけになる．行った作業量は 5 から 11 へと変化するため，総作業量の見積もりは 11 から 16 へと変わる．総作業量に対する残りの作業量の割合は，6/11 から 5/16 に変化し，総作業量に対する行った作業量の割合は 5/11 から 11/16 へと変化する．これらの進捗計算は，起こりうる状態の変化を予想してあらかじめ記述することにより正確に計算できる．

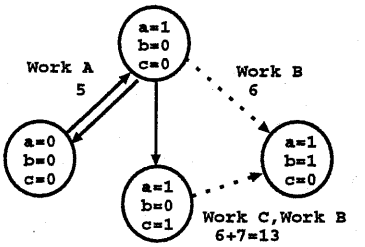



```
a:WorkAEnd==TRUE
  Start(0to1):Work A (5)
b:WorkBEnd==TRUE
  Start(0to1):Work B (6)

a=1 => b
```

図 7: 進捗計算 2

図 7 は理想とする遷移を書いた場合である．$WorkA$ の後 $WorkB$ を実行し，その後再び $WorkA$ を実行しなければならないことが分かっていてその遷移を書いている．この場合正確に $(a,b)=(1,1)$ の状態に達するまでに必要な作業と作業量が計算できる．すなわち作業は $WorkA$，$WorkB$，$WorkA$ で全体の作業量は 16 となる．




```
a:WorkAEnd==TRUE
  Start(0to1):Work A (5)
b:WorkBEnd==TRUE
  Start(0to1):Work B (6)
c:WorkBFailure==TRUE
  Start(1to0):Work C (7)

a=1 => b
```

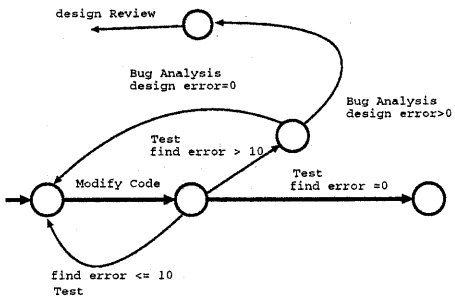
図 8: 進捗計算 3

図 8 は，さらに $WorkB$ を行った結果，失敗すれば回復するために $WorkC$ を行わなければならないという条件が加わえる．$WorkA$ を実行し $(a,b,c)=(1,0,0)$ に達した時点で行った作業量は 5 で，残りの作業量は 6 である．$WorkB$ を実行し失敗して $(a,b,c)=(1,0,1)$ という状態に遷移したとする．この時，残りの作業量は 6 から 13 に変

化し作業が後退したことを示す．行った作業量は11であるため，総作業量は，24と見積もることができる．そして，この時点で，総作業量に対する残りの作業量の割合は，6/11から13/24に変化し，総作業量に対する行った作業量の割合は5/11から11/24へと変化する．作業量24で作業が終了した場合．理想的な遷移に対して余分な作業は$24-11=13$を行ったことになる．つまり余分な作業の割合は13/24となる．

これにより，進捗状況を把握でき，余分な作業をどれくらい行ったかが分かる．

3.5 フィルタの例




```
C:CodeModifyEnd==TRUE
Start(0to1):CodeModify
CodeModify{CodeModifyEnd=TRUE,find_error=0,code_error=0}

T:TestEnd==TRUE
Start(0to1):Test
Test{CodeModifyEnd=FALSE,find_error=n||TestEnd=TRUE}

FE:TestCase@find_error>10
Start(1to0):BugAnalysis
BugAnalysis{design_error=m,code_error=k
           ,find_error=0,last_error=m+k}
```

図 9: フィルタの例 (コード修正，テスト)

図9は実際のフィルタの例である．状態要素はC, T, FEが定義されていて，Cはコードの修正が終了した時，Tはテストが成功した時，FEはテストの結果エラーが見積もった値を越えた場合1に遷移する．理想とする作業の遷移が太い矢印，作業が後退し総作業量が増える遷移が細い矢印で示される．

4 共通例題の適用

ここでは例としてKellnerらによって提案されているプロセスモデリングのための例題[9]を対象に，実際にモデルを用いた記述を行った．

これにより様々な観点でフィルタを定義でき，開発の状況を正確に表現することにより，作業の掲示，終了までの残りの作業の計算が行えることを検証する．

4.1 共通例題

この例題は，あるソフトウェアシステムの一つのモジュールに対して変更を加える作業を規定しており，8つのサブステップに分けられている．各ステップにはそれぞれ様々な開始，終了条件，入力，出力などが与えられており，プロジェクトマネージャがスケジューリングし，仕事を割り当てることで開始し，変更点がテストに合格することで全体の作業終了する．

4.2 MonoProcessEX を用いた記述

この例題を解くに当たってフィルタを3つ用いて例題のプロセスを表した．図10で例題で作成したオブジェクト，状態要素，シーケンスの定義の例を示す．また，図11は，抽出したオブジェクト，状態要素，シーケンスの個数を示す．

```
Object
----------------
DesignGroup
 @WorkingDesign (Type:Pointer of Design Object )
 @EndDesign     (Type:Pointer of Design Object)
 &Select( Selected-$->$PreviewOK=FALSE
          Selected-$->$ModifyEnd=FALSE
          Selected from EndDesign to WorkingDesign )

Design
 @CorrespondSourceCode (Type:Pointer of SourceCode )
 @ReviewOK   (Type:Boolean)
 @ModifyEnd (Type:Boolean)
 &View
 &Modify{Me@ModifyEnd=TRUE}
 &Review{DesignReviewResultGroup@WorkingDesignReviewResult
        -$->$New DesignReviewResult
              Me@ModifyEnd=FALSE or Me@ReviewOK=TRUE)
 &ModifyEndMessage
 &PreviewEndMessage
 &ModifyAgainMessage


StateElement
----------------
Name:ScheduleAndTaskAssign
Expression:TaskPlan@WorkingTaskPlan!=NULL
           AND Schedule@WorkingSchedule!=NULL

Name:Design
Expression:Design@WorkingDesign->ModifyEnd==TRUE
Start(0to1):DesignGroup@WorkingDesign->&Modify
End{0to1}:DesignGroup@WorkingDesign->&ModifyEndMessage
End{1to0}:DesignGroup@WorkingDesign->&ModifyAgainMessage

Name:ReviewDesign
Expression:Design@WorkingDesign->ReviewOK==TPUE
Start(0to1):DesignGroup@WorkingDesign->&Review
End(0to1):DesignGroup&ReviewEndMessage

Name:ModifyCode
Expression:DesignGroup@WorkingDesign->@ReviewEnd==TRUE
AND DesignGroup@WorkingDesign->@CorrespondSourceCode->@CompileEnd==TRUE
Start(0to1):SourceCodeGroup->WorkingSourceCode@->&Modify
            SourceCodeGroup->WorkingSourceCode@->&Compile
End(0to1):SourceCodeGroup->WorkingSourceCode@->&ModifyEndMessage
End(1to0):SourceCodeGroup->WorkingSourceCode@->&ModifyAgainMessage

Name:ModifyTestPlan
Expression:TestPlanGroup@WorkingTestPlan->@ModifyEnd==TRUE
Start(0to1):TestPlanGroup@WorkingTestPlan->&Modify
End(0to1):TestPlanGroup@WorkingTestPlan->&ModifyEndMessage

Sequence
----------------
ScheduleAndTaskAssign=1 => Design
ScheduleAndTaskAssign=1 => ModifyCode
ScheduleAndTaskAssign=1 => ModifyTestPlan
Design=1 => ReviewDesign
ModifyTestPlan=1 => ModifyUnitTestPackage
(ModifyUnitTestPackage=1 And ModifyCode=1)=>TestUnit
```

図 10: オブジェクト，状態要素定義，シーケンス例

図12は，開発者の視点でのフィルタの例を示している．図には各状態で行える作業，遷移に必要な作業遷移の結果自動的に実行される作業が明記されている．これにより作業の掲示が行える．各作業の終了報告は，状態要素の値の遷移時に自動的に実行することが可能となっている．終了までの作業は状態からの遷移に必要な作業で計算できる．

“デザイン変更”，“コード変更”，“試験計画の変更”の作業は並行に行われるように表現されている．“デザイン変更”が終るまで“デザインレビュー”が行えないなどの作業順序の条件は，シーケンスで表現している．また，“デザインレビュー”が終了するまでに，“コード変更”は終了することがないといった，作業完了の条件は状態要素の状態式で表される．これらを用いて各ステップ間の実行順序やタイミングが定義される．

図で示される各状態は，右下の四角で囲まれた2つの状態を持っているが簡潔のために省略している．この二つの状態は，プロジェクトマネージャの視点から作業中断の指示があった時に自動的に遷移し，作業が行えなくなる．このようにフィルタ

オブジェクト	38		
フィルタ	CCB の視点	プロジェクトマネジャの視点	その他の開発者の視点
状態要素	3	6	9
シーケンス	2	5	12

図 11: 抽出したオブジェクト，フィルタ，状態要素，シーケンスの数

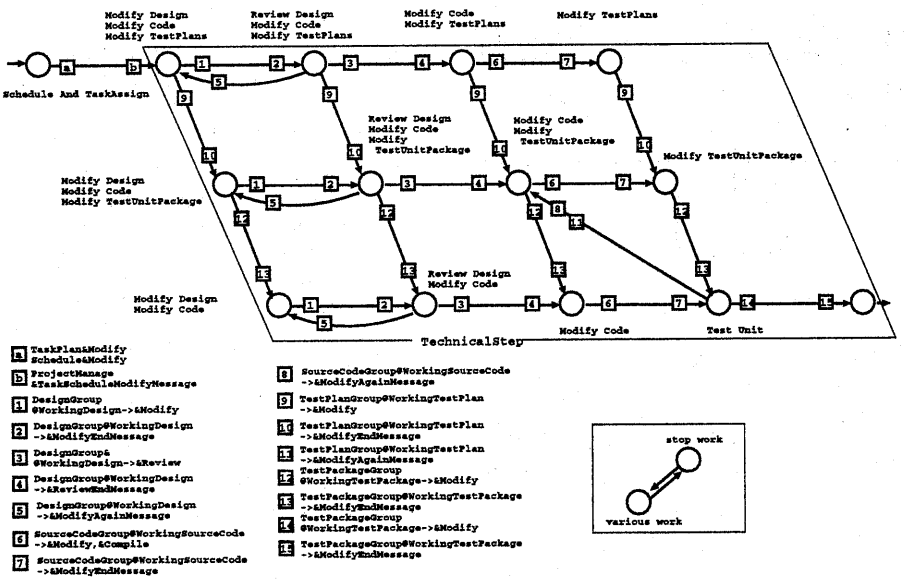


図 12: 開発者の視点でのフィルタ例

を分けて記述することにより役割別や作業種別など様々な観点で進捗状況を把握することができる．

このモデルを利用することで，様々な観点でフィルタを定義でき，開発状況を正確に把握することが可能で，各状態において行わなければならない作業，開発を終了するまでに必要な作業を知ることができる．

5 MonoProcessEX に基づく開発管理システム

本節では，2 節で述べた MonoProcess 及び 3 節で挙げた進捗管理モデル MonoProcessEX に基づくソフトウェア開発管理システムについて述べる．本研究では，以前試作したサーバアプリケーションの変更，クライアントアプリケーションの変更，及び進捗管理機能の構築を行った．

本システムは，ソフトウェア開発の環境を提供するもので，システムを用いソフトウェア開発を行なうことで開発作業の進捗管理を容易にすることを目的とする．管理者は開発作業に必要なクラス，オブジェクト，進捗管理に必要なフィルタを作成し，開発者は掲示されたメソドを実行することで開発を進める．

本システムでは，クラスオブジェクトを用いることでオブジェクトを簡単に作成することができ，プロジェクトオブジェクトを用いることで，プロジェクトごとにオブジェクトを管理することができる．

また，フィルタを用いることで，開発の進捗管理が行える．さらに，開発の任意の時点でフィルタを作成し分析を行うことができるため，問題となる事象が発生した時に分析し，そのような事象を表す状態要素を追加することで，再びその事象が起こった時に回復する作業を定義することができる．そして，管理者は開発環境の状態遷移を見ることで開発の現状を把握することができ，達したい状態までの残りの作業と作業量，過去の作業と作業量，それらの総作業量に対する割合を見ることで進捗状況を把握することができる．

図 13 はシステムの概要図である．

6 関連研究

プロセス中心型ソフトウェア開発環境として，ペトリネットを用いた SPADE[1, 2] ，Process Weaver[3]，正規文法を用いた Hakoniwa[4]，ルールベースを用いた Marvel[5, 6] などがあるが，いずれも基本的に作業手順を中心に記述し開発を行う．あらかじめ基本的な作業の流れを記述することは作業を進める点で必要であるが，作業は状況に応じて決定されることが多いため，記述通りに作業が進むとは限らない．開発管理には，まず開発の状況を正確に表現することが重要である．提案したプロセスモデル MonoProcessEX では，開

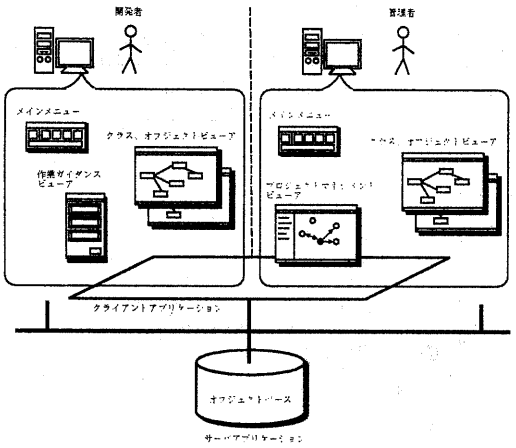


図 13: システム概要図

発環境中のオブジェクトの属性を用いて状態要素を定義し，状態要素の組み合わせによって開発環境の状態を抽象化して表現する．また，状態要素を変化させる作業を定義することにより，状況に応じた作業を割出す．これにより，開発の状況を正確に表現，把握することができ．状況に応じて正確な作業を掲示することができる．

また，開発プロセスを状態遷移図などで表現することで，開発をシミュレーションし，スケジュールや総工数を見積もる研究がある [10, 11]．このようにプロセスを状態遷移図などで表現し前もってシミュレーションすることは，大規模なソフトウェア開発において生産性を向上させるために必要なことである．我々のモデルでは，動的なプロセスを状態遷移図で正確に表現できる．そのため，開発が進行することによって正確に計算することができる．

7 おわりに

本研究では，開発環境中のオブジェクトの状態を組み合わせることで，開発の進捗状況表示，および作業のガイダンス行うためのモデル MonoProcessEX を提案した．また，既に試作を行ったプロセスモデル MonoProcess に基づく開発管理システムに進捗状況の把握や分析を行うための状態表示ビューアと作業のガイダンス機能の構築を行った．

このシステムの状態表示ビューアを用いることで正確な状況把握，詳細な進捗管理が行える．また作業掲示ビューアの作業のガイダンス機能を用いることで効率のいい作業を行うことができる．

今後，管理者側と開発者側のシステムのユーザビリティの評価や進捗計算の正確性の評価の研究課題を進めていきたい．

参考文献

[1] Bandinelli, S., Fuggetta, A. and Grigolli, S.: "Process Modeling in-the-large with SLANG", Proceedings of the Second International Conference on the Software Process, pp. 75–83(1993)

[2] Bandinelli,S.C.,Fuggetta,A. and Ghezzi,C.: "Software Process Model Evolution in the SPADE Environment", IEEE Transaction of Software Engineering, Vol.19, No.12, pp.1128–114(1993).

[3] Christer, F.: "PROCESS WEAVER: Adding Process Support to UNIX", Proceedings of 2nd International Conference on the Software Process, pp. 12–26(1993).

[4] Iida, H., Minura, K.,Inoue, K., Torii, K.: "Hakoniwa: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model.", Proceedings of 2nd International Conference on the Software Process, pp. 64–74(1993).

[5] Kaiser, G. E., Feiler, P. H.: "An Architecture for Intelligent Assistancd In Software Development", IEEE Software 5(3),pp. 40–49(1988).

[6] Heineman, G. T., Kaiser, G. E., Barghouti, N. S., Israel, Z. B.: "Rule Chaining in Marvel: Dynamic Binding of Parameters", IEEE Expert 7(6),pp. 26-32(1992).

[7] Matsushita, M., Oshita, M., Iida, H., and Inoue, K.: "Conceptual Issues of Object-Centered Process Model", 1997 Asia-Pacific Software Engineering Conference and International Computer Science Conference (1997)

[8] Matsushita, M., Oshita, M., Iida, H., and Inoue, K.: "Distributed Process Management System Based on Object-Centered Process Modeling", In Proceedings of 2nd International Conference on Worldwide Computing & Its Applications '98 (1998, to appear).

[9] Kellner, M. I. , Feiler, P. I. , Finkelstein, A. , Katayama, T. , Osterweil, L. J. , Penedo, M. H. and Rombach, H. D. : "Software Process Modeling Example Problem", Proceedings of the 6th International Software Process Workshop, pp.19–29 (1990).

[10] 永島淳, 飯田元, 松本健一, 鳥居宏次.: "作業の並行化による影響を考慮した開発プロセスシミュレーションモデル", 情処研報,96-SE-109-5,pp.33–40(1996).

[11] Kellner, M. I.: "Software process modeling support for management planning and control", Proceedings of the 1st International Conference on Software Process, pp8–28 (1991).