オブジェクトモデルを用いた ソフトウェアプロセス記述用言語 MonoProcess

松下 誠[†]　大下 誠[†]　飯田 元[‡]　井上 克郎[†]

[†]大阪大学基礎工学研究科
〒 560 大阪府豊中市待兼山町 1-3

[‡]奈良先端科学技術大学院大学
情報科学センター
〒 630-01 奈良県生駒市高山町 8916-5

これまでのソフトウェアプロセス環境やプロセス記述言語は，プロダクトを生成する手順に着目していた．しかし，ソフトウェアの再利用，コンポーネントベースの開発等，最近のソフトウェア開発においては，開発の際に作るべき対象に着目しているため，既存のプロセス環境や言語が充分に開発を支援できているとは言えない．本稿では，オブジェクトモデルを用いたソフトウェアプロセス MonoProcess の提案を行なう．MonoProcess は，開発環境における要素や資源を表現するオブジェクトの集合で表される．オブジェクトには属性とメソドがあり，これによりオブジェクトの特性やオブジェクトに対する操作を表現する．MonoProcess によって，ソフトウェア開発環境をありのままに表現することができ，ソフトウェアプロセスの記述，管理，進化のためのフレームワークを提供する．

MonoProcess: Software Process Description Language with Object Model

Makoto Matsushita[†]　Makoto Oshita[†]　Hajimu Iida[‡]　Katsuro Inoue[†]

[†]Graduate School of Engineering Science
Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka 560, Japan

[‡]Information Technology Center
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma,
Nara 630-01, Japan

Most of process-centered software engineering environments and those languages focus on *how to produce the product*. However, recent software development tend to require to focus on *what should be made*, because of emergence of various types of software developments; e.g., software reuse, component-based composition, and so on. To achieve this, we propose in this paper a new process model named *MonoProcess*, an object-centered software process description model. MonoProcess consists of a set of objects which represent artifacts and resources in the software development. An object has attributes and methods, which represent characteristics and operations of the object, respectively. MonoProcess illustrates software development environment as it is, and provides a framework for software process description, management, and improvement.

1 Introduction

ソフトウェア開発作業を効率よく行ない，かつ，高品質のソフトウェアを作成するために，ソフトウェアの開発プロセスをあらかじめ記述し，それに基づいた開発作業を行なうようになってきている [10, 11]．プロセスの記述を行なう際にはプロセス記述言語が用いられるが，既存の多くの言語は「ソフトウェアを作る際の作業手順」に着目した物ととなっている [7, 8, 9]．しかし，オブジェクト指向プログラミングやソフトウェアの再利用，コンポーネントに基づいたプログラム開発等に代表される，近年さかんに行なわれている開発形態は「ソフトウェア開発の際に作られるべき生成物」に着目して作業を整理している．このような生成物に着目している開発形態を考慮したプロセス記述言語を用いてプロセスを記述することによって，よりわかりやすいプロセスの記述を行なうことができると考えられる．

本敲では，オブジェクトモデルに基づくソフトウェアプロセス記述言語である MonoProcess の提案を行なう．MonoProcess は開発環境を直接表現することによって，ソフトウェアプロセスを表現，管理，進化させるための枠組を提供することを目的とする．

MonoProcess は，開発環境中に存在する生成物や資源を表現する「オブジェクト」を単位とした記述を行なう．オブジェクトは属性とメソドから構成され，属性によってオブジェクトの持つ特性を，メソドによってオブジェクトに適用する操作を定義する．MonoProcess は任意のオブジェクトを集合化する機能を提供しており，これによって複数のオブジェクトを 1 つのオブジェクトとして扱える．オブジェクトの継承機構を用いることにより，複数のオブジェクト間で情報の共有を行なえる．また，属性の参照等のオブジェクトへのメッセージはオブジェクトの操作履歴として自動的に保存される．

MonoProcess を用いることによって，プロセスを記述する際に，オブジェクト指向分析のプロセス記述への応用，異なる粒度のオブジェクトに対する統一的な操作，オブジェクトのうち興味のある部分だけの抽出，開発プロジェクトの変更への柔軟な対応を行なうことができる．

2 MonoProcess

本節では，我々の提案するプロセス記述言語である MonoProcess について説明する．最初に，MonoProcess の定義について述べ，MonoProcess によってソフトウェアプロセスがどう表現されるか説明する．また，MonoProcess の持つ機能についても紹介する．

2.1 オブジェクトの定義

オブジェクトはソフトウェア開発環境におけるさまざまな生成物や資源を表現する．図 1 は MonoProcess オブジェクトの記述例である．この記述は，`.Doc.Design` と名付けられたあるデザインドキュメントを記述している．

```
Object .Doc.Design def
    Attribute @Owner .Person.Matsushita;
    Attribute @Type "Design Document";
    Attribute @Input (.Doc.Specification
                          .Doc.Schedule);
    Attribute @Localtion .ShareDisk.Document
    Method &Edit def
        $editor = .caller&GetEditor(@Type);
        if ($editor) {
            &View;
            invoke($editor,
                   @Localtion . "design.doc");
        }
    endMethod
    Method &View def
        $viewer = .caller&GetViewer();
        if ($viewer) {
            invoke($viewer, @Input);
        }
    endMethod
endObject
```

図 1: Object の例

オブジェクトの名前 (ラベル) は “.” (ドット) に続く 1 つの単語で始まり，必要に応じて複数の「ドット+単語」を繋げることによって構成される．オブジェクトは属性とメソドから構成される．属性はオブジェクトの特性を表現し，メソドはオブジェクトに適用される関数である．

属性にはどのような種類かを示すためにラベルが一意に付けられている．図 1 では 4 つの属性が定義されており，`@Owner`, `@Type`, `@Input` そして `@Location` がそれぞれの属性に付けられている属性ラベルである．属性値としては次のような種類

がある．

- 整数/実数 (例: 1, -3.5)
- 文字列 (例: “ABC”, “あいうえお”)
- オブジェクトラベル (例 .A, .X.Y.Z)
- 上記 3 つの型を要素として持つリスト

例えば，図 1 における属性 `@Owner` の値 `.Person.Matsushita` は「このオブジェクトの管理者は `.Person.Matsushita` として表現されるオブジェクトである」ことを意味する．

属性と同様，メソドに対しても，その操作内容を示すためにラベルが一意に付けられている．図 1 では 2 つのメソドが定義されており，`&Edit` や `&View` がメソドに付けられたメソドラベルであり，それぞれ，編集と閲覧の作業を表わしている．

メソドによって表現される作業の内容は，オブジェクトの集合内で閉じた演算であるメソド関数として記述される．メソド関数によって行なわれる操作には，次のものがある．

- オブジェクトの保持する情報への操作: 属性値の参照，属性値の変更，属性/メソド一覧の入手，メソドの実行．
- その他のオブジェクトに関する操作: 新規オブジェクトの生成，外部ツールの起動．
- 通常のプログラミング言語に見られる操作: 整数/実数の演算，文字列演算，集合演算．

また，MonoProcess ではメソド関数の機能のみを定義しており，その文法等は定義しないため，複数のメソド記述言語を用いることができる．図 1 では，Perl 風の簡単な記述言語を用いてメソド内の記述を行なっている．

2.2 ソフトウェア開発プロセス

部分オブジェクト Op を，あるオブジェクト O に対して定義する．Op は O の持つラベルを接頭語として持ち，O の持つ属性やメソドの部分集合を持つものとする．Op は対象となるオブジェクト O の部分情報を持っており，ある観点において興味のある特徴を抽出したものである．

このようにして定義された部分オブジェクトを用いて，状態オブジェクト Os を定義する．Os は，ソフトウェア開発環境におけるある状態を表現し，部分オブジェクトの集合とする．我々は，開発環境におけるある状態は，開発環境中に存在する生成物や資源によって表現できると考えている．MonoProcess では，ソフトウェア開発プロセスをこれら状態オブジェクトの遷移系列として定義する．

簡単な例として `.SPEC`, `.CODE`, `.TEST` という 3 つのオブジェクト (それぞれ，仕様書，ソースコード，テスト結果，とする) が存在する状況を仮定する．各オブジェクトには同一の属性ラベル `@FINISHED` があり，これによって，それぞれの生成物に対する作業が完了しているかどうかを表現しているとする．

この時，開発の進行状況を示す `.SAMPLEOBJ` と名付けられた状態オブジェクトを，`.SPEC`, `.CODE`, `.TEST` それぞれのオブジェクトから `@FINISHED` だけを抜きだした部分オブジェクト 3 つの集合として定義する．作業が進行するにつれて，`.SAMPLEOBJ` は次のように状態を変化させる．

1. まず，開発開始時には`.SAMPLEOBJ` の持つ 3 つの状態オブジェクトの属性が全て “false” になっている状態になっている．
2. 仕様書の作成が終了した時点で，`.SPEC` の持つ属性 `@FINISHED` が “true” へ変化する．
3. さらにコーディングの作業が終了すると，`.CODE` の持つ属性が`@FINISHED` “true” へと変化する．つまり，`.TEST` だけが “false” の状態になる．
4. 作業が終了して，全てが “true” に変化した状態になる．

このような `.SAMPLEOBJ` の遷移系列を，MonoProcess におけるソフトウェア開発プロセスと考える．

2.3 MonoProcess の機能

MonoProcess は，プロジェクト管理，ソフトウェア開発，開発者の協調作業を支援するためにさまざまな機能を提供する．以下，MonoProcess の提供するさまざまな機能について説明する．

オブジェクトの参照範囲

一般的には，全てのオブジェクトは他のオブジェクトから参照され，また他のオブジェクトを参照することができる．しかし，それぞれのオブジェクトは自分自身の属性によって，自分自身が参照される範囲を定義することができる．

参照範囲は以下の 2 つを用いて定義する．

- グループによる範囲の指定
  グループはグループの名前によって特定されるオブジェクトの集合であり，各オブジェクトは自由にグループに所属することができる．所属したグループの名前は，@GROUP 属性にて表わされる．@GROUP オブジェクトを持っていないか，持っていても値が定義されていないオブジェクトはどのグループにも属していないものとする．

- 範囲/範囲外から行なえる操作の指定
  @ACCESS 属性によってこのオブジェクトに対して行なえる操作を定義する．操作には 4 つの種類があり，それぞれ「属性値の参照」「属性値の変更」「メソドの実行」「継承の許可」である．この 4 種類の操作について,「同一のグループに所属しているオブジェクトから」および「同じグループに所属していないオブジェクトから」行なわれた際に許可するかどうかを@ACCESS に保存する．

オブジェクトへの操作履歴

属性値の参照等，オブジェクトに対する操作は，対象となるオブジェクトに対してメッセージを送り必要な処理を要求することによって行なわれる．MonoProcess では，全てのオブジェクトに対する操作は操作履歴として次のような形で記録される[1]．

- 属性に対する操作
  操作したオブジェクト，操作した時間，操作の内容を@ATTRIBUTE.HISTORY に保存する．

- メソドに対する操作
  操作を行なったオブジェクト，操作の開始時間，操作の終了時間，実行結果の戻り値を &METHOD.HISTORY に保存する．

これらの履歴は自動的に記録される．すなわち，ある属性 @X の値の参照は，実際には @X.HISTORY に対する履歴の保存の後に実行され，あるメソド &Y の実行の後には &Y.HISTORY に対する履歴の保存が行なわれる．

オブジェクトの生成と継承

オブジェクトの生成は .OBJECT というあらかじめ定義されたオブジェクトの雛型か，既に存在しているオブジェクトからの属性/メソドの継承によって行なわれる．全てのオブジェクト生成はオブジェクトからの継承と等価であるため，以下の説明ではオブジェクトの継承に着目して説明する．

既に存在しているオブジェクト .A から，新たにオブジェクト.B を継承によって作成する場合には，以下の順序で行なう．

1. .B を .A の複製として作成する．
2. .B に対する初期化を行なって，このオブジェクトに新規に加える属性/メソドの追加や，属性値の変更等を行なう．

このように MonoProcess では，オブジェクトの継承は既存のオブジェクトの複製によって行なう．なお，オブジェクトの継承は，オブジェクトの雛型で定義されている &FORK メソドの実行によって行なうことができる．このメソドは引数として新しく生成されるオブジェクトで用いられる追加定義の情報を取る．また，&FORK を再定義することによって，あるオブジェクトに固有の継承方法を定義することができる．

3 考察

3.1 プロセス記述言語としての特徴

ソフトウェアプロセスモデルや，それを元にするソフトウェアプロセス環境にはさまざまな目的がある [5]．Curtis らは，プロセス記述，プロセスモデル，プロセス中心型開発支援環境には，プロ

[1] 我々は MonoProcess による記述の粒度に関してある仮定を置いている．それは，MonoProcess によって行なわれるプロセスの記述は非常に細かな粒度にはならず，MonoProcess では極端に頻繁なオブジェクトの生成および削除等は行なわないということである．この仮定を置くことにより，MonoProcess における操作履歴の収集は非現実的ではないと考える．

セスの理解から自動実行といった幅広い分野において，5 つの基本的な使われ方があるとしている [3]．それぞれは次のように纏めることができる．

- プロセスの理解や伝達を容易にする
- プロセスの進化を支援する
- プロセスの管理を支援する
- 自動的なプロセスの誘導を行なう
- 自動的な作業の実行を行なう

プロセス記述言語である MonoProcess は，これらの点を次のような形で実現している．

- 開発環境はオブジェクトの集合として表現されており，開発者も管理者も共通の表現を利用している．各オブジェクトに関する情報はそのオブジェクト内部で閉じている．
- MonoProcess で用いるオブジェクトはオブジェクト指向分析等を用いて分析を行なうことができるため，プロセスの進化を助けることができる．

また，これ以外の観点については，MonoProcess を用いたプロセス中心型開発管理支援環境である MonoProcess/SME によって実現される．

3.2 オブジェクトを用いた記述

MonoProcess はソフトウェア開発環境における制御/データ統合を行なうために，オブジェクトという単一の構造に情報と作業の両方を記述する．記述されたオブジェクトを用いて「開発作業中で現在何が行なわれているのか」を表現し，開発管理の際に用いる．これによって，作業を自由に記述することができるようになる．これは，他のプロセス中心型ソフトウェア開発環境 [4, 6] で用いられている言語に見られる「何をしなければならないのか/どうしなければいけないのか」に着目して「予想されない例外に対してどう対処を行なうかを記述する」こととは異なる．

我々は開発プロセスを開発時の結果として捉えており，それはさまざまな観点から充分に記述することは非常に困難であるという立場を取っている．そこで，MonoProcess は現在の開発環境それ自身を直接描写できる機能を提供している．これにより，MonoProcess はプロセス記述を行なう際に現実的な基盤となり得ると考える．

あるオブジェクトの持つ内容を継承したオブジェクトを定義することは，MonoProcess 以外のプロセス記述言語でも行なうことができる．しかし，それらは全体構造が木構造になることを仮定している物が多い [9]．このため，多重継承を表現することや，全体におけるあるまとまった一部分を変更することが困難である．Marvel は strategy によって作業をグループ化することができるが，strategy 自体は Marvel の持つデータ構造の定義に依存しているため，自由なグループ化が行なえるとは言えない．

MonoProcess では，作業の分類は MonoProcess の持つグループ機能によって任意の作業に対して行なえる．また，継承機能を用いることによって，複数のオブジェクトで利用する属性やメソドを共有できる．さらに，さまざまな粒度における資源や作業等を，同一の属性/メソドラベルを用いることによって，一貫した記述が行なえる．

3.3 開発作業の記述

APPL/A[9] 等の手続き型のソフトウェアプロセス記述言語では，ソフトウェア開発作業は構造化に基づいたプロセスの定義が行なわれる．すなわち，プロセスを記述する際には，作業の構造を静的に決定する必要がある．Marvel[1, 2, 7] などの，ルールベースのソフトウェアプロセス記述では，作業の記述を行なう場合に何が前条件や後条件となるのかを決定する必要がある．MonoProcess においては，オブジェクト指向分析の結果として得られた結果をプロセス記述へ導入することが容易に行なえると期待される．また，開発環境中に存在するさまざまな要素に関する情報をオブジェクトを単位として集中的に記述することが可能である．

MonoProcess の記述では，開発手順はオブジェクトにおけるメソドとなるため，ある一連の作業の流れが複数のオブジェクトの複数のメソドとして記述されてしまう可能性がある．これによって一連の記述が分散してしまい可読性を損ねる危険がある．しかし，MonoProcess の持つグループ化の機能を用いることにより，複数のオブジェクトに分散した情報を単一のオブジェクトへ纏めるこ

とが自由に行なえる．また，また参照される範囲を適切に限定することができる．

MonoProcess の枠組では，メソドの記述に関して特定の言語仕様を仮定しておらず，その機能のみを定義している．既存のプロセス記述言語は，言語の機能に強く依存した構造であり，作業の表現方法に何らかの制約を課してしまうことになっている [8]．MonoProcess をメソド記述言語から独立して定義することによって，メソド記述の際，目的に応じて言語を選択することが可能となる．

3.4 作業履歴

MonoProcess の持つ作業履歴の管理は，他には見られない特徴と言える．既存のオブジェクト指向プログラミング言語やデータベース言語においても，メソドの呼び出し回数や実行を中断して内部状態を観察するための同様の機能が提供されている．しかし，それらは外部ツール等によって実現されているものが多く，言語仕様に含まれているわけではない．MonoProcess では，記録されている履歴は属性として扱われているために，他の属性を扱う時と同様にして作業履歴を扱うことができる．

また，作業履歴の収集は，キーボードやマウスに対する割り込みを利用したり，オペレーティングシステムやアプリケーションに対する追加操作として収集することも考えられる．しかし，このようにして収集される履歴は非常に細かな粒度のデータになってしまうため，これらを用いて「作業者がどの生成物に対してどのような操作を行なったのか」を分析するのは非常に困難であろう．MonoProcess はオブジェクト中の属性やメソドを単位とした履歴の収集を行なうため，オブジェクトを収集したいデータに応じて定義することによって，履歴の解析を行ないやすいデータを効率よく収集することができる．

4 まとめ

本稿では，オブジェクト中心型プロセス記述言語 MonoProcess の提案を行なった．MonoProcess はオブジェクトを用いて開発作業環境中にある生成物や資源を表現する．MonoProcess ではソフトウェアプロセスはオブジェクトの状態遷移として表現される．MonoProcess を用いることにより，ソフトウェアプロセスをよりわかりやすく表現することが可能となり，プロセスの管理をより効率良く行なうことができる．

現在，MonoProcess によって記述されたプロセスを実行できるソフトウェア開発環境 MonoProcess/SME の設計を進めており，今後その実装を行なう予定である．また，実装された環境を用いた言語やシステムの評価を行ないたい．

参考文献

[1] N. Barghouti. Supporting Cooperation in the MARVEL Process-Centered SDE. In *Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments*, pages 21–31, 1992.

[2] I. Ben-Shaul, G. Kaiser, and G. Heineman. An Architecture for Multi-User Software Development Environments. In *Proceeding of 5th ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, pages 149–158, 1992.

[3] B. Curtis, M. Kellner, and J. Over. Process Modeling. *Communication of the ACM*, 35(9):75–90, 1995.

[4] C. Fernstrom and C. G. Innvation. PROCESS WEAVER: Adding Process Support to UNIX. In *Proceedings of 2nd International Conference on Software Process*, pages 12–26, 1993.

[5] A. Fugetta. Functionality and architecture of PSEEs. *Information and Software Technology*, 38(4):289–293, 1996.

[6] H. Iida, K. Mimura, K. Inoue, and K. Torii. HAKONIWA: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model. In *Proceedings of 2nd International Conference on Software Process*, pages 64–74, 1993.

[7] G. Kaiser, P. Feiler, and S. Popovich. Intelligent Assistance ofr Software Development and Maintenance. *IEEE Software*, pages 40–49, 1988.

[8] T. Katayama. A Hierarchical and Functional Software Process Description and Its Enaction. In *Proceedings of 11th International Conference on Software Engineering*, pages 343–352, 1989.

[9] S. Sutton Jr., D. Heimbigner, and L. Osterweil. APPL/A: A Language for Software Process Programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221–286, 1995.

[10] 井上克郎. ソフトウェアプロセスの研究動向. ソフトウェア科学会研究報告, 95(SP-2-1):1–10, 1995.

[11] 落水浩一郎. ソフトウエアプロセスに関する研究の概要. 情報処理, 36(5):379–391, 1995.