The University of Osaka
Institutional Knowledge Archive

Title	MonoPrcoessを用いた開発作業連絡支援システムの提案
Author(s)	山本, 哲男; 松下, 誠; 井上, 克郎
Citation	情報処理学会研究報告. ソフトウェア工学研究会報告. 1998, 1998-SE-120(64), p. 11-18
Version Type	VoR
URL	https://hdl.handle.net/11094/50147
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 120－3
（1998. 7. 23）

MonoPrcoess を用いた開発作業連絡支援システムの提案

山本 哲男　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

近年，分散型で複数人によるソフトウェア開発作業がごく一般的に行われるようになってきている．このような形態によるソフトウェア開発では，各開発者によって作業を分担して行い，各作業間が相互に連携することによって全体の開発作業を進めていく．しかし，作業者間での認識の違いや，合意した作業内容の確認が行えないなど，作業者間の連絡の欠如に起因するさまざまな問題があることが指摘されている．本稿では，まず作業者の役割やその責任範囲，作業分担などを明確に表現するための枠組について考察を行う．これらを表現するために，我々が既に提案しているオブジェクト指向モデル MonoProcess を用いた．また，これらに基いた開発作業連絡支援システムの提案を行う．

Communication Support Mechanism with MonoProcess Process Modeling

Tetsuo Yamamoto　Makoto Matsushita　Katsuro Inoue

Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka 560-8531, Japan

Recently software is being developed in more distributed way, with lots of developers. In this circumstance, software development activities are divided into sub-activities; cooperation of each activities is very important issue to progress software development. However, there are some problem caused by miss-coordination between activities; communication fault of the developers, lost the confirmation of the activities, and so on. In this paper, we propose a framework to clarify the roles, responsibilities, activities of each developers. We employ MonoProcess software process modeling to describe our mechanism. We also propose a communication support system based on our mechanism.

1 はじめに

近年，ソフトウェア開発が大規模化しており，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 [2]，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 [7, 8]．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」に着目した物であり，[1, 3, 4, 10] それらに基づいた環境もまた開発作業に着目した物となっている．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．

これらの問題に対して，我々は既にオブジェクト指向モデル MonoProcess の提案を行なっている [5, 6]．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している．

一方，ソフトウェア開発作業は分散する傾向にあり，共同作業による開発は、各開発者が作業を分担し、分担した各作業を連携させてソフトウェアの開発を行なう．しかし，共同作業においては作業者間で合意事項やプロダクトの内容の認識などに誤解が生じることがある．このような状況は開発状態の不安定さをもたらすため，プロダクトの欠陥や開発の遅れにつながる．したがって，作業者間の相互連絡は重要な問題である [9]．これらの問題を解決するためには，開発者の役割と責任範囲，開発者間の作業分担の明確化が必要と考えられる．

本稿では MonoProcess を用いて開発者間における開発作業の連絡を支援する枠組について考察を行なう．また，その枠組を実現する支援システムの提案を行う．

2 MonoProcess

本節では，我々が提案しているプロセス記述言語である MonoProcess について説明する．MonoProcess では，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」を単位としたプロセスの記述を行う．以下では MonoProcess におけるオブジェクト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．

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，`.Product.Source` と名付けられたあるソースコードを記述している．

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 で示されるソースコードを示すオブジェクト中で定義される属性 `@Owner` の値 `.Person.Yamamoto` は「このオブジェクトの管理者は `.Person.Yamamoto` というオブジェクトで表される人物である」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

```
Object .Product.Source def
    Attribute @Owner .Person.Yamamoto;
    Attribute @Type "Source Code";
    Attribute @Input (.Product.Doc.Desgin);
    Attribute @Localtion "source1.c";
    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

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．

3 枠組

前節で述べたモデル MonoPrcoess では実際のプロセスを記述するには難しい．そこで，本節では開発者の作業連絡を支援するための枠組について説明する．そして，それを実現するためのプロセス記述方法を述べる．

3.1 基礎オブジェクト

MonoProcess を用いてオブジェクトを定義しソフトウェア開発を行なうことはできるが，何もないところから単にオブジェクトを作成するだけでは作業連絡を行なうことは難しい．そこで，オブジェクトを定義する際に作業連絡を支援するために必要な属性，メソドを持つ基礎オブジェクトを考える．実際にオブジェクトを定義する時には，これらの基礎オブジェクトから派生し，作成を行なう．

メソドの実行開始，終了を開発者に自動的に伝えることで作業連絡を行なう．そのために，作業を行なう開発者，チームを表すオブジェクトが必要である．また，作業を行なう対象となるプロダクトも必要とであると考えられる．

基礎オブジェクトを用いてオブジェクトを定義することで，オブジェクトのメソドを実行すると，そのオブジェクトの責任者，また，そのオブジェクトに関係のあるオブジェクトの責任者に，誰がいつそのメソドを実行したかという情報が伝えられる枠組を提供する．これらの情報はすべて保存され，現在までにどのような作業が行われたかが分かる．また，メソドの実行開始，終了通知以外にも，作業に関する事柄を通知するために人，チームに対してメッセージを送るためのメソドも必要である．

基礎オブジェクトの種類として次の 4 種類を定義する．

- 人
  ソフトウェアを開発する作業者．
- チーム
  ソフトウェアを開発するチーム．人の集合である．
- プロダクト
  開発する生成物，中間生成物 (ソースなど)．開発期間中に変化するもの．
- その他
  その他のオブジェクト．

3.1.1 人 (Person)

基礎オブジェクト Person は，ソフトウェア開発に携わる人を定義する．管理者，プログラマー，テスターなど開発現場におけるすべての人のを表すオブジェクトは，この基礎オブジェクトから派生して定義する．

属性としては以下のものがある．

- チーム (Team)
  所属するチームのオブジェクトラベルのリストを持つ．
- 作業 (Task)
  作業のメソドのリスト．人が行なう作業を表す．開発者はこの属性を見て自分が行なうべき作業を決定する．また，他の開発者の作業を見ることでその開発者が何を行なうかを理解出来る．

メソドとしては以下のものがある．

- メッセージ (Message)
  定義した人に対してメッセージを送るメソドである．開発者間での伝言などに用いる．

3.1.2 チーム (Team)

基礎オブジェクト Team は，ソフトウェアを開発におけるチームを定義する．ここで，チームとは人の集合のことである．人を役割などでチーム分けするときに定義する．

チームの属性としては以下のものがある．

- 構成員 (Memeber)
  チームに所属する人のオブジェクトのリストを持つ．

メソドとしては以下のものがある．

- メッセージ (Message)
  定義したチーム全員に対してメッセージを送るメソドである．チーム内での告知，伝言などに用いる．

3.1.3 プロダクト (Product)

基礎オブジェクト Product は，ソフトウェアを開発における生成物，中間生成物 (仕様書，ソース，オブジェクトコードなど) を表すオブジェクトである．これらのオブジェクトとして定義されたものは，開発期間中に変化するものである．

プロダクトの属性としては以下のものがある．

- 責任者 (Owner)
  オブジェクトに対して責任をもつ人またはチームのオブジェクトのリストを持つ．責任者はプロダクトに関して責任を持つ．メソドを実行するとこの属性を参照し通知を行なう．
- 関係のあるオブジェクト (Relation)
  オブジェクトと関係をもつオブジェクトのリストを持つ．関係とは，オブジェクトのメソドの中で参照しているオブジェクトを指す．
- スケジュール (Schedule)
  プロダクトの開発期間を定義する．

3.1.4 その他 (Other)

基礎オブジェクト Other は，人，チーム，プロダクト以外のためのオブジェクトである．ソフトウェア開発に用いられるツールや計算機などを表す．

3.2 オブジェクトの作成

オブジェクトの作成方法を述べる．まず，新たに定義するオブジェクトが，基礎オブジェクトである人，チーム，プロダクト，それ以外のどれにするかを考察する．人，チーム，プロダクトなら，それぞれの基礎オブジェクトをコピーし得られたオブジェクトに属性，メソドの追加を行なう．このとき，新たなオブジェクトはオブジェクト名として，次のような名前で始まる名前を付ける．

人	.Person.
チーム	.Team.
プロダクト	.Product.
その他	.Other.

図 2 はある開発環境の例を示している．開発者として A，B がいる．また，A はコーディング担当者であり，B はテスト担当者とする．コーディング作業はエディタを用いソースコードを作成する作業で，テスト作業はソースコードから生成された実行ファイルに対してテストファイルを用いテストを行なう．

これらの作業を MonoProcess でプロセス記述を行なうと，まず，人である A，B を定義する．そして，ソースコード，テストをプロダクトとして，エディタをその他として定義する．また，A，B を開発チーム A として定義する．ソースコードにたいしてはコーディングをするメソドを定義し，A にそのメソドを割り当てる．また，実行ファイルにたいしてはテストをするメソドを定義し，B にそのメソドを割り当てる．ここで定義されるオブジェクトは以下の 7 個である．

```
.Person.A
.Person.B
.Team.A
.Product.Source
.Product.Objeccode
.Product.Test
.Other.Tool.Editor
```

これらのオブジェクトとその関係を表したのが図 3 である．

図 4 は図 3 の開発者 B とオブジェクトコードを表すオブジェクトの例である．

ここで，`@Team`, `@Task`, `@Owner`, `@Relation` は先に述べた基礎オブジェクトで定義されている属性であり，`@Input`, `@Input`, `@Location` はオブジェクトに新たに加わった属性である．

メソドに関しては，`&Test` が新たに加わったメソドである．

開発者 B がテスト作業を行なうときの作業連絡の流れは以下のようになる．

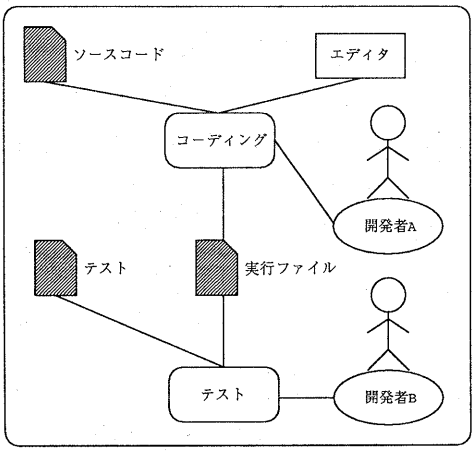

図 2: 開発環境の例

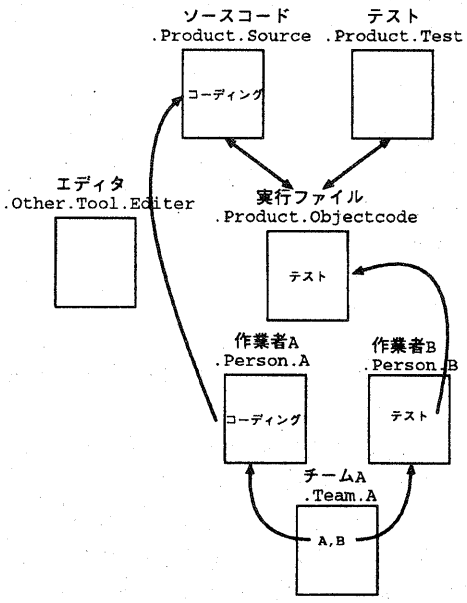

図 3: オブジェクトとその関係

```
Object .Person.B def
    Attribute @Name "Tetsuo Yamamoto";
    Attribute @Team (.Team.A);
    Attribute @Task (.Product.Objectcode&Test);
endObject

Object .Product.Objectcode def
    Attribute @Owner (.Team.A);
    Attribute @Relation (.Product.Source
                         .Product.Test);
    Attribute @Input (.Product.Doc.Desgin);
    Attribute @Localtion "object";
    Method &Test def
        ...
    endMethod
endObject
```

図 4: Object の記述例

1. 開発者 B は自分自身を表すオブジェクト (.Person.B) の作業属性から実行するメソド .Product.Objectcode&Test を決める.
2. そのオブジェクト .Product.Objectcode のメソド Test を実行する.
3. オブジェクト .Product.Objectcode の責任者である属性 @Owner の値である .Team.A にメソド Test の実行開始通知がいく.
4. オブジェクト .Product.Objectcode の属性 @Relation で表されているオブジェク (.Product.Source .Product.Test) の属性 @Owner の値が表す人にもメソド Test の実行開始通知がいく.

メソドが終了した時点でも同様にメソド実行終了通知がいく.

4 開発作業連絡支援システム

本節では，前節で述べた枠組を実現するためのソフトウェア開発作業連絡支援システムについて述べる.

4.1 システム構成

開発者は本システムを用いて開発を行う．本システムを用いることで作業連絡を行なうことができ，開発者がどのような作業をしてきたか，して

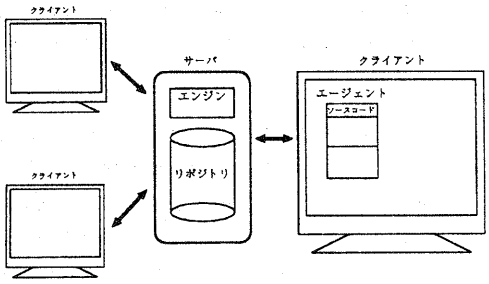

図 5: システム構成

いるか，プロダクトがどこまで出来てるか，といった情報が把握出来る．

図 5 に本システムの構成を示す．

各開発者は連絡するためのエージェントプログラムをもつ．このエージェントを使用してソフトウエアの開発，作業連絡を行う．また，オブジェクト，メソド実行履歴を保存しているリポジトリがあり，リポジトリに対してすべての操作を行なうエンジンが存在する．各開発者は各自のエージェントを動かし，エージェントはエンジンを通してリポジトリ内のオブジェクトの情報の取得，作業連絡を行なう．

エージェントからメソドを実行すると，エンジンを経由し他のエージェントへ連絡を行なう．それらの情報はエンジンによりリポジトリに格納される．

4.2 リポジトリ

オブジェクト，メソド実行履歴のすべての情報はリポジトリ内に保存する．リポジトリに対しての操作はすべてエンジンによって行なわれる．リポジトリを操作したければ，このエンジンと通信を行ない実現する．

4.3 エージェント

エージェント自体は状態をもたず，状態は常にリポジトリ内に存在する．エージェントは以下の機能をもつ．

```
Start
B host1 .Product.Objectcode&Test
20/Feb/1998 19:18:10
```

図 6: エージェントの表示例 1

```
B host1
20/Feb/1998 19:18:10
Messeage is
        ...
        ...
```

図 7: エージェントの表示例 2

- オブジェクトの参照．
- メソドの実行．
- メッセージの受取．
- 作業状況の確認．

オブジェクトの参照とは，定義されたオブジェクトの一覧，あるオブジェクトに関する属性やメソドの一覧や各属性の値を見る機能である．

メソドの実行とは，メソドを指定することでそのメソドを実行させる機能である．実際に実行するのはエージェントではなく，エージェント自身は指示するだけでメソドを実行するための別のプログラムが実行する．

メッセージの受取とは，誰かがメソドを実行し，その通知を知らせる時にエージェントがその通知を受け取るための機能である．

作業状況の確認とは，エージェントから，だれがどのような作業を行なっているか，また行なってきたかを知る機能である．

メッセージメソドを実行するとメッセージの宛先である作業者のエージェントに送られ，そのメッセージが表示される．

例として，開発者 B が host1 という計算機からオブジェクト .Product.Objectcode のメソド Test の実行を開始すると関係する人のエージェントに図 6 の情報が表示される．

また，開発者 B が host1 という計算機から開発者 A の Message メソドを実行すると 開発者 A のエージェントに図 7 の情報が表示される．

5 まとめ

本稿では，我々がすでに提案を行なっているオブジェクト指向プロセスモデル MonoProcess を用いた開発作業連絡支援システムについて，そのシステムを実現するための枠組とシステムの概要を述べた．本システムを用いることにより開発者の役割と責任範囲，開発者間の作業分担の明確化が行なえる．

現在，本システムの実装を進めている．今後の課題としては，本システムを完成させ，実際の開発環境に用いて評価を行なうことが挙げられる．

参考文献

[1] Bandinelli, S. and 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] Curtis, B. and Kellner, M. and Over, J.: "Process Modeling", Communication of the ACM, 1995, Vol.35, No.9, pp.75–90 (1995).

[3] Kaiser, G.E. and Feiler, P.H. and Popovich, S.S.: "Intelligent Assistance ofr Software Development and Maintenance", IEEE Software, Vol.5, No.3, pp.40–49, (1988).

[4] Katayama, T.: "A Hierarchical and Functional Software Process Description and Its Enaction", Proceedings of 11th International Conference on Software Engineering, pp.343-352 (1989).

[5] 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).

[6] 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).

[7] 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, 1995, Vol.95, No.SP-2-1, pp.1–10 (1995).

[8] 落水浩一郎: "ソフトウェアプロセスに関する研究の概要", 情報処理, Vol.36, No.5, pp.379–391 (1995).

[9] 落水浩一郎: "分散開発に適したソフトウェアプロセスモデルの構築に向けて", 日本ソフトウェア科学会 FOSE'97, pp.43–50 (1997).

[10] Sutton Jr., S.M. and Heimbigner, D. and Osterweil, L.J.: "APPL/A: A Language for Software Process Programming", ACM Transactions on Software Engineering and Methodology, Vol.4, No.3, pp.221–286 (1995).