開発プロセスを構成する要素間のインタラクションに関する考察

松下 誠[†]　飯田 元[†]　井上 克郎[†]　鳥居 宏次[†‡]

[†]大阪大学
[‡]奈良先端科学技術大学院大学
[†]〒 560 大阪府豊中市待兼山町 1-3
[‡]〒 630-01 奈良県生駒市高山町 8916-5

ソフトウェア開発における工程やプロダクトの管理, コミュニケーションの支援, ツールの統合などを一貫して支援する手段として, 開発プロセスをその基本構造にとり入れた開発支援環境（プロセス中心型開発環境）がある. プロセス中心型開発環境では, プロセスという形で記述された開発計画を用いることによりそれぞれのプロジェクトに対して, より細かく柔軟な支援を行なうことが期待できる. 本研究では, プロセス中心型開発環境の構成要素を, 作業, ツール, プロダクトなどに分類し, これらの要素間のインタラクションについて考察する. さらに, 各要素についてその振舞いの一部を自動化するような代理プログラムと, それらに対する統一されたアクセス手段を用意することによって, プロジェクト管理やプロダクト管理, コミュニケーションなどを支援する機構について提案する.

An Interaction Support Mechanism in Software Development Processes

Makoto Matsushita[†]　Hajimu Iida[†]　Katsuro Inoue[†]　Koji Torii[†‡]

[†]Osaka University
[‡]Nara Institute of Science and Technology
[†]1-3 Machikaneyama, Toyonaka, Osaka 560, Japan
[‡]8916-5 Takayama, Ikoma, Nara 630-01, Japan

This paper proposes a new modeling method of software development process which focuses on the interactions among the elements of the process. In this method, a software process is modeled as a set of elements and communication channels. With this method, communications performed during the process are clearly represented. The prototype of process-centered environments based on this method is also developed. This environment consists of proxy programs for auto execution of elements, and it provides mechanisms for project management, product management, and communication support.

1 はじめに

ソフトウェアの大規模化や開発にかかる人件費の増大によって, 一箇所で集中してソフトウェア開発を行なうことは非常に困難になってきつつある. そこで, ソフトウェア開発を成功させるために, 開発作業がどのような条件の下で, どういう関係で結びつけられているか, という点に着目, 整理するといった, いわゆるソフトウェアプロセスについての研究が盛んに行なわれるようになってきている [1][2].

ソフトウェアプロセスに対する研究についてはさまざまなものがあるが, その分野の一つとして, 「プロセスを何らかのモデルに基づいてプログラムのように記述し, 記述されたプロセスを用いて開発支援を行なう」ものがある [3][4][5].

しかし, 従来提案されてきたプロセス記述やモデルではコミュニケーションを明示的に扱っていないか, または単純な通信オペレーションとしてしか扱っていない. そのため,

- 連絡する相手先を予め静的に決定しておかなければならない
- 連絡でやりとりされる情報を予め細かく想定しておく必要がある

といった問題点がある.

情報のやりとりは, 離散的なメッセージの送受信の集合としてとらえるのではなく, 特定の「話題」に基づいた構造を持つものとしてとらえるのが重要であろう. したがって, 情報のやりとりを考える際には, 発信者や受信者を主体としてとらえるのではなく, まず「話される内容」を明らかにし, それを基に流れを整理するべきである. そこで, 本研究では, 開発過程におけるさまざまなコミュニケーションに着目したプロセスモデルについて考察を行ない, このモデルに基づく開発支援環境を提案する.

2 対話指向のプロセスモデルに要求される事項

ここでは, コミュニケーションに着目したプロセスモデルとして, 次のような点を満たすものを考える.

コミュニケーションの内容に着目していること

情報のやりとりをとらえる場合に, 「誰と」やりとりを行なうかではなく, 「何を」やりとりするのか, を明確に表現できることを重視する. これによって, やりとりの内容とその流れを整理することができると考えられる. この時, それらの情報が, 必要とされている相手に届くことが保証されなければならない.

コミュニケーションのやりとりの解析ができる

例えば, 通信を記録しておき, それを元にプロセスを解析することを考える際の形式的な枠組として利用できることが望ましい.

連絡支援の枠組としての簡潔さを持つこと

ソフトウェア開発には連絡の他にもさまざまな要素が含まれており, 連絡支援のみを目的に複雑な機構を導入することは望ましくない. したがってモデルはなるべく単純な形で表現されることが望ましい.

3 提案するモデル

前節で考察した要求を満たすためには, やりとりを実際に行なう実体 (例えば, 作業をする人間や作業の対象となるファイル, さらにはプリンタやデータベース, プログラムなど) や, やりとりされる内容 (例えば変更内容やバグ情報など) を直接表現できるべきである, と考えた. そこで, 本モデルでは「エンティティ」と「チャンネル」という二つの要素を用いてこの両者を表現する. モデルを判りやすく説明するため, 本稿では次のようなプロセスを例にとって考える.

> ある小規模なプログラムの開発が行なわれている. 開発は仕様書からコード作成を行ない, テストを行なうとする. テスト実行のためにはテストパッケージを利用する. コンパイル中やテスト実行中に問題が発見されると, それを受けてデバック作業 (コーディングのやり直し) が行なわれる.

上のプロセスを本モデルで表したものが図 1である. 四角がエンティティ, 円がチャンネルを表しており, 各エンティティからはそれが必要とする情報が扱われるチャンネルに点線が引かれている.

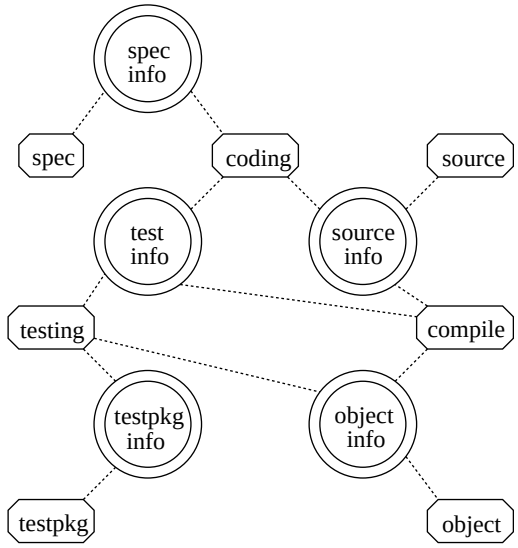

図 1: モデルの概略図

3.1 エンティティ

エンティティは, ソフトウェア開発において「何らかの情報を送信したり受信したりするもの」を表現する. つまり,「人が実際に行なう作業」や「自動化されている作業」,「プロダクト」などは, エンティティとして扱う. 図 1においては, *spec, source, testpkg, object* などはプロダクトに相当し, *coding, compile, testing* は作業に相当する. エンティティは識別子, 属性定義, 動作定義の組として定義される (表 1).

識別子	エンティティ同士の識別に用いる
属性定義	エンティティがどのような目的で存在するのかを示す
動作定義	送られてきた情報に対する条件とそれに対応した操作の組のリストを記述する

表 1: エンティティの構成要素

実際には図 2に示すような記法を用いる. 図 2では, テスト実行を行なうエンティティである *testing* エンティティの記述例を示している. 属性定義の記述には phase(フェーズ), role(役割), sort(種類) という 3 つの属性にそれぞれ develop(開発フェーズ), test (テスト), execute(実行) という値が付けられている. 属性については 3.3節で詳しく述べる. 動作定義の記述では, a) *testpkginfo* チャンネルから testpkg_content(テストパッケージ自身) という種類の情報が送られてきた場合には, それをテストパッケージとして取り込むという動作と, b) *objectinfo* チャンネルから object_content(オブジェクトコード自身) という種類の情報が送られてきた場合には, dotest(テスト実行) という操作によってテストを実行し, その結果を *testinfo* チャンネルに送る, という 2 つの動作が記述されている.

さらに, 動作定義中で使用される変数 testpkg, request_testpkg の宣言, エンティティが実際に行なう動作に新たに dotest を加える宣言, エンティティの初期化動作 initaction の定義が記述される. dotest は *testing* エンティティの持つ固有の操作として扱われ, その具体的な動作の中身については記述しない. また, 動作の記述中で利用することのできる基本的な操作として表 2に示すようなものが予め用意されている.

```
define entity testing
begin
    attribute begin
        phase: develop;
        role: test;
        sort: execute;
    end
    vars begin
        testpkg: testpkg_content;
        request_testpkg: request_testpkg;
    end
    initaction begin
        join(objectinfo);
        join(testpkginfo);
        send(testpkginfo, request_testpkg);
    end
    primitive add dotest;
    action begin
        channel testpkginfo begin
            testpkg_content:
              testpkg = info;
        end
        channel objectinfo begin
            object_content:
              send(testinfo, dotest(info, testpkg));
        end
    end
end
```

図 2: *testing* エンティティの記述

3.2 チャンネル

チャンネルは, ソフトウェア開発において, やりとりされる情報を表現し, 情報の通り道としての役割を

動作名	引数	動作内容
join()	チャンネル名	チャンネルへの所属
leave()	チャンネル名	チャンネルからの離脱
send()	チャンネル名 内容	チャンネルへの情報の書き込み

表 2: エンティティに予め用意されている操作

持つ. 図 1においては *specinfo, sourceinfo, testinfo, objectinfo, testpkginfo* がチャンネルである. チャンネルは識別子, 属性定義, 制約条件, 情報内容の組として定義される (表3). 制約条件には「聞いていて欲しい」エンティティを示す正の制約と「聞いていて欲しくない」エンティティを示す負の制約の二種類について (必要があれば) エンティティの属性の組を列挙する.

識別子	モデル中において一意に決定され, チャンネル同士の区別を行なうために使われる
属性定義	チャンネルがどのような目的で存在するかを表現する
制約条件	チャンネルと接続できるエンティティの制約について記述する
情報内容	チャンネルでやりとりされる内容について記述する

表 3: チャンネルの構成要素

実際には図 3に示すような記法を用いる. 図 3では, オブジェクトコードの情報が流れる *objectinfo* エンティティの記述例を示している. 属性定義としては, phase (フェーズ), sort(種類), deliver(配送手段) という 3 つの属性の種類に, それぞれ develop(開発), object(オブジェクトコード), immediate_large(即時に相手に届けられ, 大量の情報が流れる) という値が付けられている. 属性については 3.3節で詳しく述べる. 制約条件には in(正の制約) として 2 つの組が指定されており, それぞれ「テストの役割を持つすべてのエンティティ」「コンパイル作業を行なうすべてのエンティティ」がこのチャンネルに所属する必要があることを表している. 送られる内容としては object_content(オブジェクトコード自身) があることが記述されている.

```
define channel objectinfo
begin
    attribute begin
        phase: develop;
        sort: object;
        deliver: immediate_large;
    end
    restriction
        in: (any, test, any),
            (any, any, compile);
    infotype
        object_content:
          (filename: name, content: file);
end
```

図 3: *objectinfo* チャンネルの記述

3.3 属性について

エンティティやチャンネルが, 開発過程においてどのような部分に位置し, どのような役割を果たしているか, また, あるチャンネルでの情報はどのような方法でやりとりされるか, といった抽象的な性質は, 各エンティティやチャンネルの持つ属性値によって表される. 本モデルでは, ソフトウェアプロセスに対象を限定するので, エンティティはフェーズ, 種別, 役割の 3 つの属性で分類する. また, チャンネルはフェーズ, 種別, 配送手段の 3 つによって分類する. これらの属性の値域は各プロセスごとに定義する. この例では,

$$
\begin{aligned}
\text{フェーズ} &= \{\text{spec, develop}\} \\
\text{種別} &= \{\text{edit, compile, execute, spec, sourcecode, object, testdata, test}\} \\
\text{役割} &= \{\text{coding, test, product}\} \\
\text{配送手段} &= \{\text{storage\_large, immediate\_large, immediate}\}
\end{aligned}
$$

となっている.

図 1におけるエンティティとチャンネルそれぞれの属性を表 4および表 5に示す.

3.4 モデル上でのプロセスの振舞い

本モデルでは, プロセスの振舞いはエンティティ内の動作とチャンネルを介したエンティティ間のコミュニケーションとして表される. 具体的には以下のような動作を行なう.

名前	フェーズ	種別	役割
coding	develop	edit	coding
compile	develop	compile	coding
testing	develop	execute	test
spec	spec	spec	product
source	develop	sourcecode	product
object	develop	object	product
testpkg	develop	testdata	product

表 4: エンティティの属性

名前	フェーズ	種別	配送手段
specinfo	spec	spec	storage_large
sourceinfo	develop	sourcecode	immediate_large
objectinfo	develop	object	immediate_large
testinfo	develop	test	immediate
testpkginfo	develop	testdata	storage_large
default	any	any	immediate

表 5: チャンネルの属性

チャンネルへの所属

エンティティはチャンネルを通じて情報をやりとりするが, チャンネルを利用する際には, 予め明示的にチャンネルに「所属」する必要がある. エンティティは同時に複数のチャンネルに所属することができ, チャンネルへの所属は, プロセスの初期設定時だけでなく, 任意の時点でも行なわれる.

また, 全てのエンティティは, 必ず「プロセス全体でやりとりされる情報」が流れるチャンネルに所属しているとする (表5の *default* チャンネル). たとえば, 新たにチャンネル C が作成されると, その制約条件がチャンネル *default* に流される. このとき, チャンネル C の持つ正の制約条件を満たすエンティティは join(C) という操作を行ない, C に対する所属の手続きを行なう.

チャンネルからの離脱

エンティティは, 所属しているチャンネルからの情報が必要なくなった時点でチャンネルから「離脱」する. チャンネル C からの離脱は leave(C) という操作によって, 離脱の宣言をそのチャンネルに送ることで行なわれる.

チャンネルの利用

エンティティによる情報の送出, 受取は, 全て対象となるチャンネルへの書き込み, 読み出しとして表現される. チャンネル C への書き込みは send(C, data) という操作で行なわれる. 読み出しは明示的には記述されずに行なわれる.

4 モデルに基づく開発支援環境

これまで述べてきたモデルに基づき, 開発プロセスにおけるコミュニケーション支援環境の設計, 試作を行なっている. この開発支援環境では, 1) エンティティ間のコミュニケーション 2) エンティティが行なう作業の自動実行 3) エンティティを利用する作業者に対してツールの自動起動や作業のナビゲーション, を行なう. システムの概略図を図 4に示す.

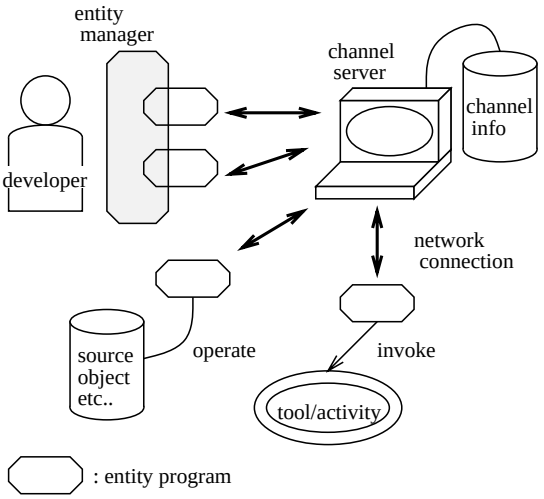

図 4: 開発支援環境の概略図

各エンティティはそれぞれ一つのエンティティプログラムとして実現される. エンティティプログラムは図 2に示したような記述に基づき動作する. チャンネルは本試作ではチャンネルサーバとして集中して管理, 処理する. エンティティプログラムはチャンネルサーバと接続し, チャンネルに対する入出力動作を行なう.

各開発者に対しては, 各自が受け持つ複数のエンティティプログラムをまとめたエンティティマネジャがユーザーインターフェイスとして提供される. エンティティマネジャはエンティティプログラムへのインターフェイスとしての役割や, それぞれのエンティティの現在の実行状況の表示, 実際の作業の際に必要となるツールの起動などを行なう. プロダクトを表

現しているエンティティプログラムは, 該当プロダクトに対する操作の実行を行なう. 自動化された作業を表現するエンティティプログラムは, 実際に作業を行なうプログラムを起動し, その結果をチャンネルを通じて提供する.

チャンネルサーバでは, どのエンティティプログラムがどのチャンネルに所属しているかを管理し, 情報を適切なエンティティプログラムに送り, チャンネル上で交わされた情報を記録する. 記録された情報は, それまでの時点でどのような情報がやりとりされ, どれだけ作業が進捗しているかを把握するために利用される.

5 考察

5.1 本モデルの特長

本モデルによって開発過程を表現することにより, 次のような利点があると考えられる.

- やりとりされる情報はチャンネルとして表現されるため, プロセス中で存在する話題が明示される.
- 連絡先は, チャンネルによって抽象化されるために, 発信者が情報を誰に届けなければならないかを細かく意識する必要がない. また, チャンネルの制約条件によって, 必要なエンティティに確実に情報を届けることができる.

また, 本モデルに基づいた開発支援環境によって,

- 情報の内容はチャンネルサーバが, 作業はエンティティプログラムが表現しているため, それぞれの動作記録を取ることによって進捗状況の把握が可能となる.
- エンティティの動作記述に作業の手順が記述されていることにより, ツールの自動起動や作業の自動化などが表現できる.

といったことが実現されると考える.

5.2 関連研究

次に, ソフトウェア開発におけるコミュニケーションという点に着目して, 既存の研究について考察する. プロセス記述・実行環境の 1 つである APPL/A [3] では, relation や trigger の機構を利用することにより, メッセージ送受信や, 必要な作業の呼出, データのやりとりなどを記述することができる. しかし, 通信動作が記述全体に分散しているため, どのような通信が行なわれるのかわかりにくいといった問題点がある. 作業者間のコミュニケーション支援については, CSCW の分野で盛んに研究が行なわれている [6]. しかし, これらの研究は, オフィスワークにおける比較的固定した対話経路に対する支援を行なうものが多く, ソフトウェアプロセスのように作業の結果や進捗によって頻繁に経路が変化するものには不十分であろう.

6 まとめ

開発環境における作業やプロダクトなどのコミュニケーションに着目したプロセスモデルについて考察を行ない, これに基づく開発支援環境を提案した. 本モデルを利用することによって, 必要とされる情報の流れが明確となり, 作業の自動化や作業の連絡の支援などを行なう際に有用であると考えられる. 現在, 本モデルに基づく開発支援環境の試作を行なっている. 今後の課題としては, エンティティやチャンネルの動的生成, 消滅の機構の実装や, チャンネルで記録された情報の具体的な解析手法の開発などが挙げられる.

参考文献

[1] 井上克郎: “最近のソフトウェアプロセスの研究動向”, ソフトウェア・ツール・シンポジウム '93 論文集, pp.5-14, 1994

[2] ソフトウェアプロセスシンポジウム, 1994

[3] Sutton, S., Heimbigner, D., and Osterweil, L.: “Language Constructs for Managing Change in Proces-Centered Environments”, In *Proceedings of 9th Int. Conf. Software Eng.*, pp.328-338, 1987

[4] Kaiser, G., Feiler, P., and Popovich, S.: “Intelligent Assistance for Software Development Maintenance”, *IEEE Software 5, 3*, pp.40-49, 1988

[5] Katayama, T.: “A Hierarchical and Functional Software Process Description and Its Enaction ”, In *Proceedings of 11th Int. Conf. on Software Eng.*, pp.343-352, 1989

[6] 特集 “グループウェアの実現に向けて”, 情報処理, Vol. 34, No. 8, 1993