LO-002

アスペクト指向による開発プロセス記述の実現検証

Aspect-Oriented Implementation for Software Process

上野 浩一郎† 茂木 強†
Koichiro Ueno Tsuyoshi Motegi

1．まえがき

近年、ソフトウェアの QCD(Quality, Cost, Delivery)に対する要求がますます厳しくなり、開発のリスクが増している。このため、成功したプロジェクトを再現できるように開発プロセスを記述し標準化しておく重要性が増している。開発プロセスを記述するツールとして、例えば Eclipse では"EPF Composer"[1]を公開している。このようなツールは記述作業を効率化するが、適切な内容を記述することは依然として難しい。なぜなら、開発プロセスには様々なコンサーン(関心事)があり、それらのコンサーンが「もつれている(Tangling)」ことがその記述を難しくしているからである。このため開発プロセスでも、ソフトウェアと同じく SoC(Separation of Concern)すべきである[2]。

この SoC を実現するために、設計や実装などのエンジニアリング作業をコアコンサーン、エンジニアリング作業を横断的に管理/支援する作業をクロスカッティングコンサーンとし、それらのもつれをアスペクトとして分離する方式が報告されている[3][4][5]。更に筆者は、開発プロセスに関するアスペクトをそのメタレベルに応じて2つに分類すべきことを報告している[6]。

本論文では、プロセスエンジニアによる開発プロセス記述を支援するために、その目的に適したメタレベルでアスペクトを実装し、その効果を評価する。

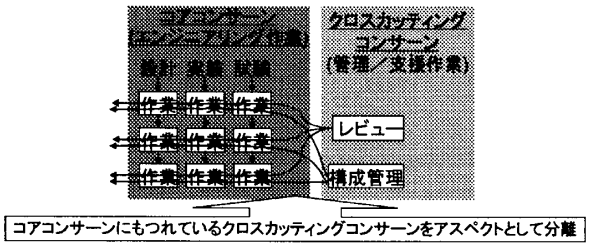

図 1 プロセスのもつれをアスペクトで分離

2．関連する従来研究

2.1 アスペクト指向によるプロセス記述

Reis02[3]は、開発プロセスに対してアスペクト指向の適用を初めて言及した論文である。Reis02 では例として「"開発"作業の後には"検証作業"が続かなければならない」というプロセス内容をポリシーとして宣言的に記述することが示されている。また Mishali06[5]では、XP(eXtreme Programming)のプラクティス"テストファースト"に対して「コード作成前に単体試験コードが無いと承認しない」というアスペクトが AspectJ[7]のコードで示されている。

2.2 メタレベルに応じたアスペクト記述

OMG の SPEM(Software Process Engineering Metamodel)[8]では、開発プロセスに関するメタレベルを表 1に示す4レベルに分類している。

表 1 プロセスに関するメタレベル

メタレベル	内容	説明
メタレベル 0	Performing process	具体的なプロジェクトにおいて計画して実行されるインスタンスが位置付けられる
メタレベル 1	Process Model	具体的な開発プロセスのインスタンスが位置付けられる
メタレベル 2	Process Metamodel	開発プロセスそのものを表現する要素が位置付けられる
メタレベル 3	MetaObject Facility	メタレベル 2 を表現する要素が位置付けられる

筆者の従来研究である Ueno04[4]/Ueno05[6]では、SPEM に基づきメタレベル0とメタレベル1の要素に適用するアスペクトを区別している。メタレベル0に対するアスペクト(M0 アスペクト)は、メタレベル1の要素を用いて定義し、メタレベル0の要素に対して織込まれる。メタレベル1に対するアスペクト(M1 アスペクト)は、メタレベル2の要素を用いて定義し、メタレベル1の要素に対して織込まれる。Ueno05 では、M0 アスペクトとして「レビュー可能な成果物が登録されると、それを担当者 A に入力してレビューを開始させる」、M1 アスペクトとして「新しい種類の成果物が定義されると、それをレビュー作業の入力成果物種類として追加する」という内容を擬似的な AspectJ のコードで示している。

2.3 従来研究に関する考察

従来研究を比較するために、Ueno05 で定義した M0 アスペクトと M1 アスペクトが、誰の何の作業に関わるかを図 2に示す。M1 アスペクトはプロセスエンジニアがメタレベル 1 要素を生成/編集するタイミングで織込まれるので、プロセスエンジニアの作業支援となる。M0 アスペクトは、メタレベル0要素をプロジェクト計画者が生成/編集あるいはプロジェクト員が実施するタイミングで織込まれるので、プロジェクト計画者とプロジェクト員の作業支援となる。

Reis02 と Mishali06 は M0 アスペクトに関する取組みなので、プロジェクト計画者とプロジェクト員の作業支援が範囲となる。本論文が目的とするプロセスエンジニアのプロセス記述作業を支援するには M1 アスペクトが必要である。

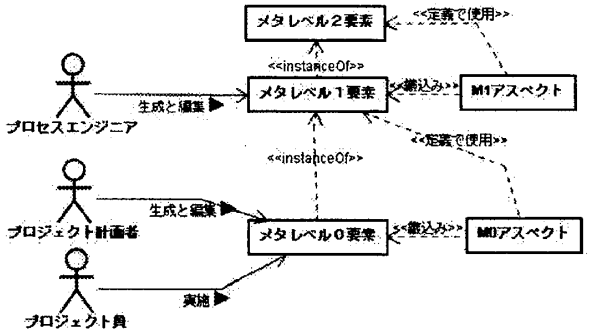

図 2 メタレベル毎のアスペクト

†三菱電機株式会社 情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

3．メタレベル 1 に織込むアスペクトの検証実装

本章では、メタレベル1に織込むアスペクトがプロセス記述を支援できるかを検証するための実装結果を報告する。

3.1 実装に使用したソフトウェア

実装に使用したソフトウェアを表 2に示す。

表 2 実装に使用したソフトウェア

種類	名前	文献
アスペクト指向実装言語	AspectJ	[7]
検証対象の開発プロセス	OpenUP/Basic	[1]
メタモデルの実装	jBPM サンプル実装	[9]

3.2 プロセスメタモデル

検証実装のために Java 実装したプロセスメタモデルを図 3と表 3に示す。図と表で白抜表示している要素は jBPM サンプル実装である。塗潰表示している要素は本論文で拡張実装したものである。なお"TaskDescriptor"と"Step"は OpenUP/Basic のメタモデル UMA[1]の用語に準拠している。図 3のメタモデルの特長は、エンジニアリングに関わる CoreTaskDescriptor と、その管理と支援に関わる CrossTaskDescriptor を明示的に区別している点である。

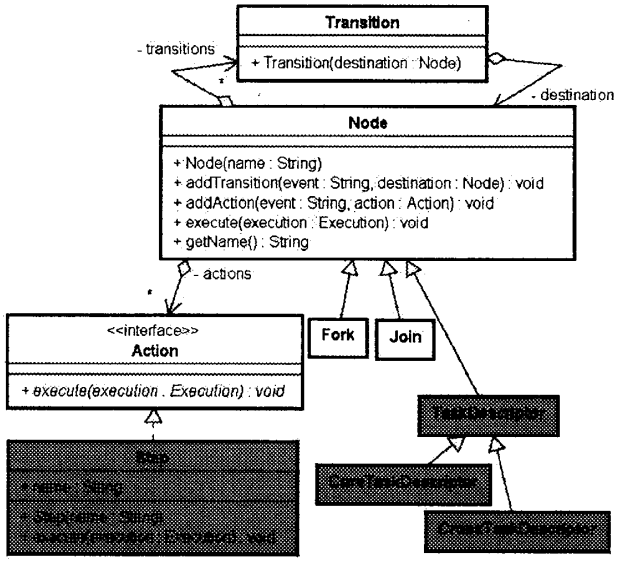

図 3 プロセスメタモデル(クラス図)

表 3 プロセスメタモデルの要素一覧

Java 要素名	説明
Node	グラフ構造のノード
Transition	グラフ構造の遷移
Fork と Join	並行動作開始と同期動作
Action	Node の実行によって起動される動作(node-enter と node-leave の 2 種類有る)
TaskDescriptor	開発プロセスを構成する作業
CoreTaskDescriptor	エンジニアリングに関わる作業
CrossTaskDescriptor	管理や支援に関わる作業
Step	TaskDescriptor のサブ作業

3.3 検証対象の開発プロセス

図 4は、開発プロセス OpenUP/Basic の一部を示すプロセスモデル(オブジェクト図)である。この図は"Develop Solution"下の CoreTaskDescriptor インスタンス群(例えば図 4上部にある developTheArchitecture や designTheSolution)と、"Project Management" 下の CrossTaskDescriptor インスタンス assessIteration(図 4の左下)を示している。CoreTaskDescriptor インスタンス間は Transition インスタンスで連結されている。また各 TaskDescriptor インスタンスには Step インスタンスがリンクされている。

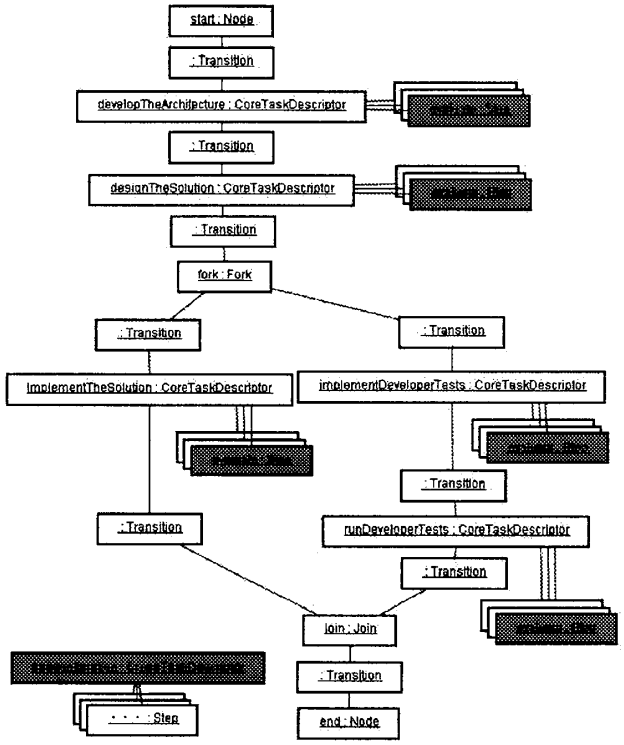

図 4 OpenUP/Basic の内容を示すプロセスモデル

このプロセスモデルに対して、エンジニアリング作業と管理/支援作業がもつれている箇所を2つ示す。

■もつれA

CoreTaskDescriptor のそれぞれのインスタンスに、作業成果を評価する Step "evaluate"(図 4で塗潰されている Step インスタンス)が繰り返し定義されている。Step "evaluate"は管理/支援に関わる作業であり、これが CoreTaskDescriptor インスタンスのそれぞれに埋め込まれており冗長である。

■もつれB

Iteration の最後が Activity "Develop Solution"の場合、"Develop Solution"を完了するためには管理/支援作業である CrossTaskDescriptor"assessIteration"(図 4で塗潰表示)を実行する必要があり、もつれている。しかし OpenUP/Basic では CrossTaskDescriptor の実行タイミングを形式的に表現できていない。

3.4 メタレベル 1 に織込むアスペクト記述

まず図 5で、2つのもつれに対して定義するアスペクトを説明する。もつれAに対しては、CoreTaskDescriptor インスタンスのそれぞれから Step "evaluate"を分離し、その"evaluate"を CoreTaskDescriptor インスタンスに挿入する Aspect

"ConcernEvaluateResults"を定義する。もつれBに対しては、Node "end"に遷移する前に、CrossTaskDescriptor "assessIteration"を挿入する Aspect "ConcernAssessIteration"を定義する。

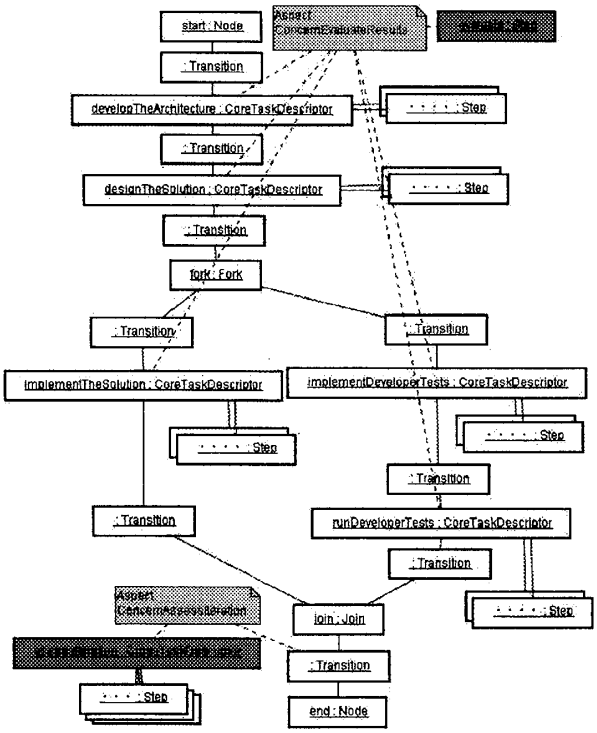

図 5 アスペクト導入後のプロセスモデル

リスト1は Aspect "ConcernEvaluateResults"、リスト2は Aspect "ConcernAssessIteration"を AspectJ で実装したコードである。なおリスト中の太字文字はメタレベル2要素(表 3参照)、下線付き文字は AspectJ 予約語を示す。

まずリスト1を説明する。行番号 2 で、CoreTaskDescriptor が生成されるポイントカット newCoreTaskDescriptor を定義する。行番号 3 で、ポイントカット newCoreTaskDescriptor の after アドバイスを宣言する。行番号 4 がアドバイスの中身であり、Step("Evaluate results")を生成し、それを CoreTaskDescriptor インスタンスに追加する。

次にリスト2を説明する。行番号 2～6 で、Node "end"へ遷移するポイントカット transit2End を定義する。行番号 4～5 は遷移を追加する addTransition()が call され、その第2引数である遷移先ノード destination が"end"である条件を定義している。行番号 7～8 で、ポイントカット transit2End の around アドバイスを宣言する。行番号 9～13 がアドバイスの中身である。行番号 9 で CrossTaskDescriptor("Assess iteration")を生成し assessIteration に代入する。行番号 10～11 で assessIteration に対して Step を追加する。行番号 12 で、around アドバイスが横取りする前に呼び出していた targetNode と assessIteration に遷移関係を持たせる。行番号 13 で、assessIteration と Node "end"に遷移関係を持たせる。

3.5 アスペクトの織込み動作

本節では、前節で記述したアスペクトが織込まれる動作を説明する。リスト3は、図 5のプロセスモデルを構築する Java コードである。Aspect "ConcernEvaluateResults"は行番号 2～3 のそれぞれで織込まれる。行番号2で織込まれる動作を図 6(シーケンス図)に示す。Aspect "ConcernAssessIteration"は行番号 16 で織込まれる。この動作を図 7(シーケンス図)に示す。

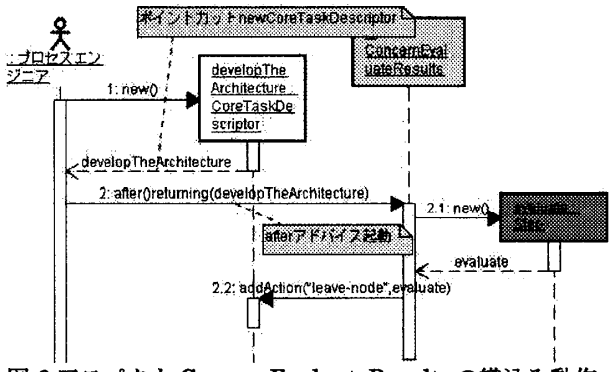

図 6 アスペクト ConcernEvaluateResults の織込み動作

リスト1　メタレベル1に織込まれるアスペクト①

```
1:  public aspect ConcernEvaluateResults {
2:    pointcut newCoreTaskDescriptor() : call( CoreTaskDescriptor.new(..) );
3:    after() returning(CoreTaskDescriptor coreTaskDescriptor): newCoreTaskDescriptor(){
4:      coreTaskDescriptor.addAction("leave-node", new Step("Evaluate results"));
5:    }
6:  }
```

リスト2　メタレベル1に織込まれるアスペクト②

```
1:  public aspect ConcernAssessIteration {
2:    pointcut transit2End(Node targetNode, String event, Node destination):
3:      target(targetNode) && args(event,destination) &&
4:      call(void Node.addTransition(String, Node)) &&
5:      if(destination.getName()=="end") &&
6:      !cflow(adviceexecution());
7:    void around(Node targetNode, String event, Node destination) :
8:      transit2End(targetNode,event,destination){
9:        TaskDescriptor assessIteration = new CrossTaskDescriptor("Assess iteration");
10:       assessIteration.addAction("enter-node",new Step("Gather stakeholder feedback"));
11:       ・・・assessIteration に Step を addAction するコードが続く・・・
12:       proceed(targetNode,event,assessIteration);
13:       proceed(assessIteration,event,destination);
14:     }
15: }
```

リスト 3　図 5の構造を生成するコード

```
1:  //CoreTaskDescriptor の作成
2:  CoreTaskDescriptor
    developTheArchitecture = new
    CoreTaskDescriptor("Develop the
    Architecture");
3:  ･･･CoreTaskDescriptor を生成するコードが続く･･･
4:  //CoreTaskDescritptor へ Step 追加
5:  developTheArchitecture.addAction
    ("enter-node",new Step("Identify
    reuse opportunities"));
6:  ･･･Step を addAction するコードが続く･･･
7:  //Node の作成
8:  Node start = new Node("start");
9:  Node fork = new Fork("fork");
10: Node join = new Join("join");
11: Node end = new Node("end");
12: //Transition の設定
13: start.addTransition("continue",
    developTheArchitecture);
14: developTheArchitecture.addTransi
    tion("continue",
    designTheSolution);
15: ･･･addTransition するコードが続く･･･
16: join.addTransition("continue",
    end);
```

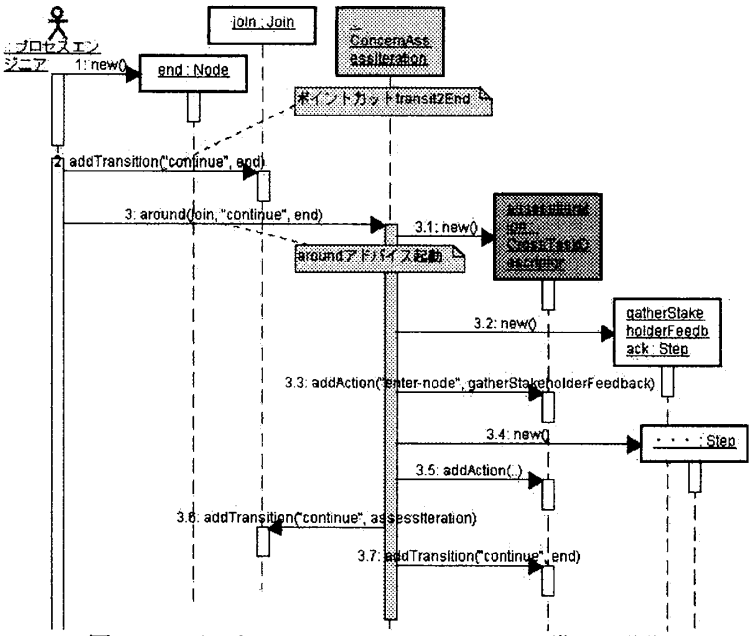

図 7 アスペクト ConcernAssessIteration の織込み動作

4．評価

前章の結果をプロセスエンジニアによるプロセス記述の視点で評価した結果を表 4に示す。結論として、メタレベル1のプロセス記述に対してアスペクト指向を導入することにより、記述性/形式性/テーラリング性が向上する。

表 4 プロセスエンジニアによるプロセス記述の評価

視点	従来方式	アスペクト方式	結果
記述性	エンジニアリング作業毎に同じ管理/支援作業を埋め込む	エンジニアリング作業から管理/支援作業を分離し、その織込みを 1 つのアスペクトで記述	記述量が減少し、変更箇所がアスペクトに局所化
形式性	エンジニアリング作業と管理/支援作業の関係を自然文で記述	エンジニアリング作業への管理/支援作業の織込みを計算機解釈が可能なアスペクトで記述	従来は潜在化していた記述を顕在化
テーラリング性	手作業でエンジニアリング作業に管理/支援作業を埋め込む	アスペクト群を取捨選択して織込む	テーラリング工数が減少

5．おわりに

本論文では、プロセスメタモデル要素を用いてアスペクトを定義し、そのアスペクトをプロセス記述のタイミングで織込む実装を示した。この実装は、プロセスエンジニアによるプロセス記述作業の支援に効果があると評価できた。

今後の課題は3つある。第 1 に、本論文ではシンプルなプロセスメタモデルを自作して、作業の繋がりに関するアスペクトを示したが、今後は UMA[1]のような標準的なプロセスメタモデル実装を用いて Role や WorkProduct に関するアスペクトの検証実装が必要である。第2に、本論文ではプロセスに関するアスペクトを AspectJ で記述したが、プロセスエンジニアがアスペクトを容易に記述できるように開発プロセス用のドメイン特化言語が必要である。第3に、プロセス記述の目的はプロセスを理解/改善/実行(enactment:人間によるものとツールによるものの双方)[10]するためなので、プロセスに関するアスペクトを開発ライフサイクル全体で活用する機能開発が必要である。

参考文献

[1] Eclipse Process Framework: http://www.eclipse.org/epf/

[2] Hruby, P. : Dimensions for the Separation of Concerns in Describing Software Development Processes, OOPSLA99 First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems.

[3] Reis, R.Q., Reis, C.A., Schlebbe, H. and Nunes, D.J. : Towards an Aspect-Oriented Approach to Improve the Reusability of Software Process Models, AOSD2002 Early Aspects Workshop.

[4] 上野浩一郎：アスペクト指向による開発プロセス定義, 情報処理学会第 66 回全国大会 1G-4(2004).

[5] Mishali, O.,et al. : Using aspects to support the software process: XP over Eclipse, Proceedings of the 5th international conference on Aspect-oriented software development (2006).

[6] 上野浩一郎：アスペクト指向によるプロセスモデリング手法, 情報処理学会第 91 回 情報システムと社会環境研究会(2005).

[7] AspectJ : http://www.eclipse.org/aspectj/

[8] OMG : Software Process Engineering Metamodel, http://www.omg.org/technology/documents/formal/spem.htm

[9] JBOSS : jBPM jPDL User Guide, http://docs.jboss.com/jbpm/v3/userguide

[10] Fuggetta, A., et al. : ソフトウェアプロセスのトレンド, 海文堂(1997).