ソフトウェア工学 118－6
（1998．3．10）

未来デルタ機構による版管理法

堀 雅和[†,‡] 篠田 陽一[†] 落水 浩一郎[†]
北陸先端科学技術大学院大学[†]
(株) インテック・システム研究所[‡]

ソフトウェア成果物の迅速な配付を実現するためには、将来の変更に備えてあらかじめ可能な部分については開発作業を進めておくといった、先を見越した開発プロセスを採る必要がある。本稿では、このような開発プロセスをシステマティックに支援するため、以下のような特徴を持つ二つの基本機構を提案する。

1. 未承認の変更要求に基づき開発した中間成果物の版を管理することで、先を見越した作業プロセスを支援することを目的とした未来デルタ機構
2. 依存関係のある中間成果物の間に発生する矛盾に基づき他の開発者の作業の影響を通知することを目的とした汚染マーク機構

以上の2つの機構の概要を示し、ソフトウェア共同開発への適用について例を用いて説明する。

Version Management by Future Delta Mechanism

Masakazu HORI[†,‡] Yoichi SHINODA[†] Koichiro OCHIMIZU[†]
Japan Advanced Institute of Science and Technology, Hokuriku[†]
INTEC Systems Laboratory Inc.[‡]

A quick delivery of a software product requires such a work style that developers can prepare for expected future changes in advance. In this paper, we propose a scheme with the two mechanisms to support such a development process systematically. The proposed mechanisms have the following features:

1. Future Delta Mechanism, which manages an artifact developed for non-approved change requests to support such a work style that developers prepare for expected future changes in advance.
2. Pollution Marking Mechanism, which provides functionalities to detect inconsistent states between related artifacts and notify influences of other developer's work.

We show the overview of the two mechanism and the way how to support a development process using an example.

1 はじめに

ソフトウェア成果物を迅速に配付するためには、予想される変更に備えてあらかじめ可能な部分については開発作業を進めておくといった、先を見越した作業プロセスを採る必要がある。例えば、ネットワークやパソコン関連のソフトウェアは、絶えず変化し進化し続けるため、常に先を見越して作業することは非常に有効である。このようなプロセスを支援する場合、予想される変更要求に基づいて開発した中間成果物の版を管理する必要があるが、現在の版管理システムは中間成果物の変更履歴を管理することが主目的であり、将来起こりうる中間成果物の変更を管理するには適切ではない。

本稿では、予想される変更要求に基づく先行作業を支援する以下のような特徴をもつ二つの基本機構を提案する。

1. 未来デルタ機構 (Future Delta Mechanism)

   未来デルタ機構は、先を見越した作業プロセスを支援することを目的とした版管理機構である。未承認の変更要求を用いて開発した中間成果物を管理する未来版空間と承認済の変更要求を用いて開発した中間成果物を管理する永続版空間という二種類の版空間により構成されている。未来版空間に属する版のうち承認された変更要求に対応する作業結果を選択的に永続版空間に反映することで開発プロセスを支援する。

2. 汚染マーク機構 (Pollution Marking Mechanism)

   汚染マーク機構は、関連する中間成果物の状態間に発生する矛盾を検出することで、他の開発者の作業の影響を通知することを目的とした機構である。汚染マーク (Pollution Marker) のアイデアは Balzer[3] が提唱し、Cugola ら [6] が汚染マークのアイデアを拡張してソフトウェアプロセスに適用した。本研究では、他の開発者と共有している中間成果物を制約条件に違反して変更した場合にその中間成果物に汚染マークを付け、依存関係のある中間成果物に対して汚染マークを伝播することにより共有情報を変更したことで発生する矛盾発生の可能性を通知する。

本稿の構成は、以下のとおりである。第 2 節では、提案する未来デルタ機構の概要について説明する。次に第 3 節において、汚染マーク機構を適用するうえでの基本的方針について述べる。第 4 節では、予想される変更に備えた作業プロセスに基づきソフトウェア開発する場合の提案する機構による支援法について例を用いて説明する。第 5 節では、関連研究を紹介し、本研究の位置付けを明らかにする。最後に第 6 節にて、本稿のまとめと今後の課題について述べる。

2 未来デルタ機構

未来デルタ機構は、予想される変更に備えた作業プロセスを支援するための版管理機構である。開発成果物の版と予想される変更要求に基づき開発した中間成果物の版を別の種類の版として管理し、承認された変更要求に基づき開発した中間成果物を選択的にソフトウェアレポジトリに反映する機能を提供することで、作業プロセスを支援する。これより、図 1を用いて未来デルタ機構の三つの構成要素である「変更要求 (change request)」、「永続版空間 (persistent version space)」、「未来版空間 (future version space)」の特徴および、未来デルタ機構による作業プロセスの支援方法の概要について説明する。

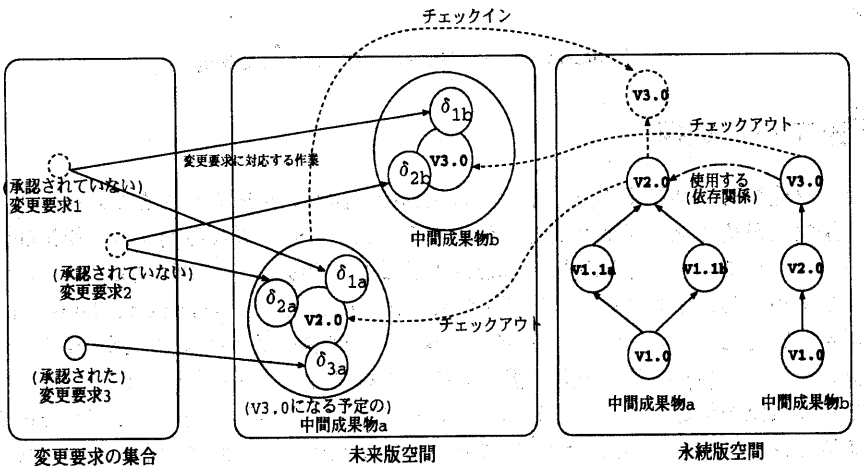

図 1: 未来デルタ機構の概要

変更要求は、変更要求の内容 (例: ファイル読み込みのパフォーマンスの向上)、具体的な変更対象となる中間成果物とその内容 (例: モジュール 1 のうち f_a という関数を新たに開発する関数 f_b に変更)、変更要求の承認に関する状態などの属性によって規定される。変更要求は、承認の状態が変化するだけでなく、変更対象となる中間成果物とその内容も変化する。すなわち、関係者によって承認され実際に対処することが確定した変更要求だけでなく、変更内容が承認されておらずしかも具体的な対処方法も確定していない兆しとしての変更要求も変更要求の集合に含まれる。

永続版空間は、既に永続化され安定した状態の版の版木 (version tree) で構成され、それぞれの版は変更することはできない。版の進化は、状態に基づき定義され (state-based versioning)、進化の方向には、バリアント (variant) とリビジョン (revision) の二種類ある。一方未来版空間は、変更要求に基づき作業している不安定な状態の中間成果物の版で構成されている。未来版は、変更要求に基づき定義されており (change-based versioning)、共通部分と変更要求に依存する部分を分離して管理する二レベル構成になっている。既に存在している永続版を用いて変更作業する場合、用いた永続版を $PersistentVersion$ で表わし、変更要求$_i$ $(i = 1, \cdots, n)$ に対応する変更内容を $PersistentVersion$ との差分情報 δ_i $(i = 1, \cdots, n)$ で表わすと、未来版 $FutureVersion$ は以下の式 (1) のように表わすことができる。

$$FutureVersion = PersistentVersion \cup \delta_1 \cup \cdots \cup \delta_n \quad (1)$$

ここで δ_i を $\cup$ するという操作は、変更要求$_i$ に対する作業が、$FutureVersion$ に含まれることを表わしている。δ_i は、未来デルタと呼ぶことにする。

次に図 1 を用いて未来デルタ機構による作業プロセスの支援方法について説明する。図 1 において変更要求$_1$ および変更要求$_2$ はまだ承認されておらず、この変更要求が将来システムに組込まれるかどうかはこの時点では、未定であるものとする。一方変更要求$_3$ は、システムに組込まれることが承認されているものとする。変更要求$_3$ に関して発生する変更作業は、中間成果物 a に対して行われるものとする。また中間成果物 a は、変更要求$_3$ 以外にも既に変更要求$_1$ と変更要求$_2$ の作業が行われているものとする。承認された変更要求に関して発生した作業で終了した作業の結果だけを反映すると、反映する未来版の中間成果物 a は以下の式 (2) のように表わすこと

ができる。

$$a_{FutureV3.0} = a_{PersistentV2.0} \cup \delta_{\text{変更要求}_3} \tag{2}$$

3 汚染マーク機構

汚染マーク機構は、開発中の異なる中間成果物の状態間の矛盾の発生を検知し、依存関係のある中間成果物に矛盾の発生可能性通知を伝播することにより、異なる開発者の作業間の調整をすることを目的としている。本機構は、Balzer が提案した汚染マーク [3]、Cugola らの汚染マークを拡張して適用することでプロセスモデルからのズレに耐えうる (tolerating deviation) プロセス環境に関する研究 [6] をベースにしている。汚染マーク機構の導入により、例えば以下のような状況の改善を試みる。(a) 将来の変更に備えて作業する場合、ソフトウェアレポジトリに反映する時の版の構成状態を想定して作業するのが一般的である。しかし、前もって作業をしておくと実際の作業時とレポジトリへの反映時とに遅延が生じるため、その間に想定していた版の構成状況と実際の状況にズレが生じる可能性がある。(b) また、使用している共有情報が知らぬ間に別の開発者によって変更されていると、自分の作業がどのような影響を受けるのかわからず混乱が発生するだけでなく、作業の大きなロールバックの発生原因となりうる。

本研究では、未来に備えた共有情報の変更処理に関する作業プロセスを以下のように想定している。

> 共有されている (すなわち共有作業空間に属する) 永続版を私有作業空間に属する未来版空間において作業するためにチェックアウトする。変更要求の承認および変更作業が終了したら、永続版空間にチェックインする。

ここで私有作業空間とは開発者個人の作業空間であり、所有者以外の他の開発者が参照することはできない。一方共有作業空間は共有している複数の開発者によってアクセス可能な作業空間である。

以上に述べた作業プロセスでの矛盾の扱いに関する基本的方針は以下のとおりである。前提条件 (precondition) と不変条件 (invariant assertion) という 2 つの種類の制約条件を定義する。前提条件は、プロセス開始のための条件を定義する。具体的には、「共有情報をチェックアウトする時、その情報を共有している開発者達によって変更要求が承認されている必要がある」とする。不変条件は、永続版空間の版構成の一貫性を保つための条件を定義する。前提条件が守られている限り共有情報の変更を安全に実施できることが保証される。しかし、前提条件に違反しても処理は継続可能であるものとする。その場合共有情報には、矛盾の発生可能性を示す汚染マークが付けられる。式 (3) において Q が前提条件、P が汚染マークが設定される条件を表している。汚染マークの条件を表わした P は、Q が成り立たない時、その時に限り真となる。

$$P(x_1, \ldots, x_n) \Leftrightarrow \neg Q(x_1, \cdots, x_n) \tag{3}$$

この汚染マークにより、設定された制約条件を変更することなく、矛盾の発生可能性を検出することが可能となる。また、汚染マークの付いた中間成果物と依存関係にある中間成果物を特定するために、汚染マークの伝播ルールを定義する。

4 例題

本節では、提案した機構のソフトウェア共同開発への適用例について説明する (図 2参照)。

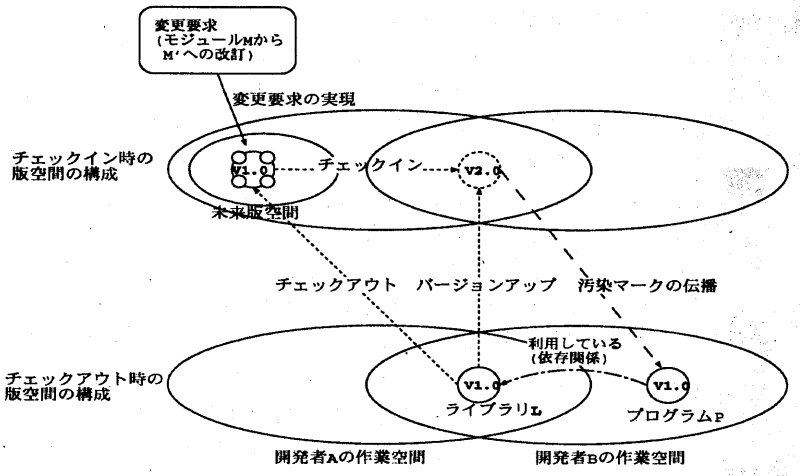

図 2: 例題の状況

開発者 A は ライブラリプログラム L を、開発者 B はライブラリ L を用いてプログラム P を開発しているものとする。すなわちプログラム P はライブラリ L を "使用する" という依存関係にあり、ライブラリ L は、開発者 A と 開発者 B の共有作業空間に属している。開発者 A がライブラリ L をチェックアウトする前は、ライブラリ L の版は V1.0、プログラム P の版は V1.0 であるとする。このような状況において開発者 A が、ライブラリ L が提供するモジュール M のインタフェースが不十分で使いにくいという他の開発者からの不平を聞き、先行してモジュール M の改訂版 M' を開発する場合について考察する。開発者 A は、ライブラリ L の次の版である V2.0 としてリリースするつもりで、私有作業空間内でライブラリ L の V1.0 の版をもとに未来版を開発しているものとする。この時、共有者である開発者 B より承認を得ることなくチェックアウトした場合、未来版 $L_{FutureV2.0}$ に汚染マークが付けられる。その後、モジュール M を M' に改訂する変更要求が承認され、M' の作業が終了したら、未来デルタを組込んだ版 $L_{FutureV2.0}$ が永続版 $L_{PersistentV2.0}$ として永続版空間にチェックインされる。この場合、チェックアウトした版の子供の版としてチェックインしたので、永続版空間の一貫性を破壊することがなく、チェックインは成功する。しかし、$L_{PersistentV2.0}$ には汚染マークが付いているため、ライブラリ L と依存関係のあるプログラム P の最新の版 $P_{PersistentV1.0}$ に汚染マークが伝播する。汚染マークの伝播により開発者 B は、プログラム P に変更が必要である可能性が生じたことが通知される。

5 関連研究

開発途中に発生するさまざまな変化にスムースに対応することは、開発工数を短縮するうえで重要である。現実のソフトウェア開発では、発生する変更を見越してあらかじめ可能な部分について開発作業を進めるというプロセスがしばしば観測される。しかし現状のソフトウェアリポジトリ技術では、このようなプロセスを十分に支援できないという問題がある。初期の版管理システム [21, 22] は、高価な資源を有効活用することに主眼が置かれていた。また構成管理技術の発達により、任意の版のソフトウェア成果物を取り出すことが可能となり、ソフトウェアの共同開

発の支援 [2, 4]、環境の分散化への対処 [8, 14]、プロセス環境との統合 [9] などを考慮したシステムが数多く報告されている。今までの版管理は、ソフトウェアの構成を管理することを主眼としているため、開発されたソフトウェア成果物に関してプロダクト空間と版空間をどのように対応付けるかという視点から版管理を捉えている [5]。しかし開発作業を支援することを考えた場合、開発者の作業内容に適した管理法を実現する必要がある。今回提案した未来版空間と永続版空間は、異なる版管理法にて作業空間を構成することに対応している。

複数の開発者が共同でソフトウェア開発をするために発生する大きな制約の一つは、他の開発者からのレスポンスなしには次の作業に進めない箇所が存在するということである。例えば、他の開発者の承認を得る必要がある場合や他の開発者の作業結果を用いて作業する場合などである。このような制約を常に守らなければならないという方針でソフトウェア開発環境を構築すると、開発プロセスのあらゆるところにボトルネックとなる箇所が表われ、開発者はシステムの支援を無視して作業するようになると考えられる。すなわち制約のなるべく少ない作業環境を構築することが重要である。このようなソフトウェア開発の特性を考慮した研究として例えば、共有情報の同時アクセス制御に関する研究がある。Copy-Modify-Merge パラダイム [1] は、共有情報の楽観的変更を支援する。また、ソフトウェア開発を長時間トランザクションと捉え、トランザクションの構造をマルチレベル [7, 15] で構成したり、動的に再構成 [20] できるようにすることで同一資源を長時間確保することを防ぐアプローチなどがある。共有情報の同時アクセス制御よりも大きな枠組として、ソフトウェア開発を矛盾が存在する中での作業と捉え、矛盾をよりシステマティックに扱うための研究が進められている。文献 [17] では、矛盾が存在する中での行動の取りかたには、(a) 矛盾を全く無視する (b) 矛盾を回避する (c) 間違いを修正するか競合を解消することで矛盾を取り除く (d) 矛盾した状況を改善し、将来の矛盾解消の可能性を増すようにするという 4 つの場合があると述べられている。Balzer が提案した汚染マーク [3] の機構は、(b) の矛盾を回避するという範疇に入る。Cugola らは、定義されたプロセスモデルと観測されたプロセスのズレ (deviation) をを検知するために汚染マークの機構を適用した [6]。Cugola らは、汚染マークの伝播の機構を導入することで Balzer の機構を拡張している。今回適用した汚染マークに関連する研究以外にも、矛盾を扱うためのいくつかのアプローチが提案されている。Gabbay と Hunter は、矛盾を外部あるいは内部アクションを採るためのシグナルととらえ、時間論理を用いてメタルールを定義する手法を提案している [11]。Finkelstein は、ViewPoints と呼ばれる部分的な仕様間の論理的な矛盾を古典論理で検知し、矛盾を処理するルールを時間論理で定義している [10]。Narayanaswamy と Goldman は、提案する変更をアナウンスすることによる "lazy" な一貫性維持に関する研究を報告している [16]。

6 まとめと今後の課題

本稿では、ソフトウェア開発の共同作業の効率向上を目的として以下の二つの機構を提案した。

1. 未来デルタ機構
2. 汚染マーク機構

未来デルタ機構は、未承認の変更要求に基づき作業する場合の中間成果物を管理する未来版空間と確定した変更要求に基づく作業結果が反映されている永続版空間により構成されている版管理機構である。未来デルタ機構により、未来版空間において予想される変更に対して可能な部分に

関する作業を行い、変更要求が承認されると永続版空間に移動することで作業プロセスを支援する。このような作業プロセスを実現することで開発成果物の迅速な配付が可能となる。

汚染マーク機構は、異なる作業間で発生する矛盾を検知し調整を支援するための機構である。我々は既に共有情報の管理機構を提案してきた [12, 13, 18, 19]。しかし提案してきた機構は承認されて初めて次のステップに進むことができるという方針を採用しており、実際に適用する場合制約の大きいものであった。しかし汚染マーク機構を用いることで、共同作業のために発生する制約を緩和できると考えている。

今後の課題は、以下のとおりである。

1. 汚染マーク機構を適用するうえでの制約条件と汚染ルールのフォーマルな定義
2. 未来版管理の一般化
3. 分散環境におけるさまざまな作業スタイルへの適用の検討

参考文献

[1] Adams,E.W., Honda,M., and Miller,T.C.: Object Management in a CASE environment. In Proc. of ICSE-11, 1989, pp.154-163.

[2] Allen,L., Fernandez,G., Kane,K., Leblang,D., Minard,D., and Posner,J.: ClearCase MultiSite: Supporting Geographically-Distributed Software Development. LNCS 1005, (1995), pp.194-214.

[3] Balzer,R.: Tolerating Inconsistency. Proc. of ICSE-13, (1991), pp.158-165.

[4] Berliner,B.: CVS II: Parallelizing Software Development. In Proc. of 1990 Winter USENIX Conference, Washington D.C., 1990.

[5] Conradi,R.: Version Models for Software Configuration Management. to be appear ACM Computing Surveys. (can be obtained from http://www-i3.informatik.rwth-aachen.de/private/bernhard/westfechtek.html)

[6] Cugola,G., Nitto,E. Di, Ghezzi,C., and Mantione,M.: How To Deal With Deviations During Process Model Enactment. Proc. of ICSE-17, (1995), pp. 265-273.

[7] Deacon,A., Schek,Hans-Jorg, Weikum,G.: Semantics-based Multilevel Transaction Management in Federated Systems. Proc. of ICDE-10, (1994), pp.452-461.

[8] B.O'Donovan and J.B.Grimson: A Distributed Version Control System for Wide Area Networks. Software Engineering Journal, September 1990.

[9] Estublier,J., and Casallas,R.: The Adele Configuration Manager. In W. Tichy, editor, Software Configuration Management, No.2 in Trends in Software, pp. 99 - 134, Wiley, London, 1994.

[10] Finkelstein,A.C.W., Gabbay,D., Hunter,A., Kramer,J., and Nuseibeh,B.: Inconsistency Handling in Multiperspective Specifications. IEEE Transactions on Software Engineering, Vol. 20, No. 8, (1994), pp. 569-578.

[11] Gabbay,D. and Hunter,A.: Making Inconsistency Respectable: Part 2 - Meta-level handling of inconsistency. LNCS 747, (1993), pp.129-136.

[12] 堀雅和、落水浩一郎: ソフトウェア開発における自己反映オブジェクト指向モデルに基づく共有情報の管理法. コンピュータソフトウェア 13-1, (1996), pp.37-54.

[13] Hori,M., Shinoda,Y., and Ochimizu,K.: Shared Data Management Mechanism for Distributed Software Development Based on a Reflective Object-Oriented Model. LNCS 1080, (1996), pp.362-382.

[14] Hung,T., and Kunz,P.F.: UNIX Code Management and Distribution. Techinical Report SLAC-PUB-5923, Stanford Linear Accelerator Center, Stanford, California, September 1992.

[15] Moss, J.E.B.: Nested Transaction: An Approach to Reliable Distributed Computing. The MIT Press, Cambridge, Mass., (1985).

[16] Narayanaswamy,K., and Goldman,N.: "Lazy" Consistency: A Basis for Cooperative Software Development. Proc. of CSCW '92, (1992), pp.257-264.

[17] Nuseibeh,B.: To Be and Not to Be: On Managing Inconsistency in Software Development. Proc. of 8th IEEE International Workshop on Software Specification & Design (IWSSD-8), March 1996, pp. 164-169.

[18] Ochimizu,K., Kadowaki,C., and Hori,M.: Design of an Information Repository to Support Cooperative Works over Computer Network. Proc. of the IPSJ International Symposium on Next-Generation of Information Technologies, (1997), pp.79-86.

[19] Ochimizu,K.: Toward a Software Process Model for Distributed Cooperative Software Development. Proc. of the 2nd International Symposium on Future Software Technology, (1997), pp.55-62.

[20] Pu,C., Kaiser,G.E., and Hutchinson,N.: Split Transactions for open-ended activities. In Proc. of the 14th International Conference on Very Large Data Bases, (1988), pp.26-37.

[21] Rochkind,M.J.: The Source Code Control System. IEEE Trans. on Software Engineering, Vol. SE-1, No.4, Dec. 1975, pp. 364-370.

[22] Tichy,W.F.: RCS - A System for Version Control. Software-Practice and Experience, Vol. 15(7), (1985), pp.637-654.