修 士 論 文

共同ソフトウエア開発における成果物に関する情報の管理方式

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

押目 晴義

2002 年 3 月

修 士 論 文

共同ソフトウエア開発における成果物に関する情報の管理方式

指導教官　落水浩一郎 教授

審査委員主査　落水浩一郎 教授
審査委員　篠田陽一 教授
審査委員　片山卓也 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

010027 押目 晴義

提出年月: 2002 年 2 月

概 要

本論文では、共同ソフトウェア開発における開発過程において生成される様々な成果物と、それらの構成に関する情報の管理方式について検討する。反復作業が繰り返し行われる共同ソフトウェア開発においての、成果物の生成・変更作業を支援する有効な方法として、成果物を統括的に管理する管理モデルと管理方式を提案する。具体的には、「チーム構造モデル」と呼ばれる「プロセスモデル」に成果物間の依存関係を取り入れた管理モデルと、それから導出される「変更スレッド候補」という概念を用いることで、変更管理を実現する。また、それらを基に管理システムのプロトタイプを作成し、シミュレーションによる評価実験を行った結果を報告する。

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

多くの場合、ソフトウェアは共同で開発される。共同ソフトウェア開発では、様々な役割を持つ開発者により、多くの成果物 (例えばUML 図 [1][2] やソースコード) が生成、または更新される。このような状況下では、成果物の構成に関する情報を管理することが重要であり、成果物に記述・描画されている内容だけでなく、成果物の生成理由や変更履歴、また成果物間の依存関係などの成果物の構成に関する情報が管理されている場合、成果物の再利用や一貫性の保持、また開発プロセスの理解などを支援可能である。これにより、成果物の変更や新たな成果物の生成を容易に行えると考えられる。

しかし、従来の紙による文書の管理方式や、プロダクトベースのデータベーススキーマ、あるいはHTML[5] によるリンク構造では、柔軟な構成管理が困難である。近年登場した文書記述言語で、今後デファクトスタンダードとなりうるXML(eXtensible Markup Language)[6] を用いれば、タグによる柔軟な文書記述が可能である [10][12]。また、XML はワークフロードメイン連携での、企業間における異種アプリケーション間のデータ交換を容易に実現可能にする。このメリットを生かした情報の管理方式の研究は盛んに行われているが、変更の波及効果が検知可能であるような成果物の生成・変更作業を支援する管理方式は、筆者の知る限り未開発である。

1.2　研究の目的

本論文では、成果物の生成・変更を行う際に必要となる情報を柔軟な視点から検索できる成果物の管理モデルを提案する。具体的には、XML のメリットをソフトウェア開発に対応させ、成果物を格納した「XML の封筒」に依存関係などの情報をタグとして格納し、それらのタグの部分集合を軸とする検索空間を与えることや、タグを規則的にトレースすることにより、波及効果などの様々な可能性を提示する管理モデルを提案する。

また、複数の変更要求に伴って発生する変更作業に関する複数のスレッドを保持し、スレッド間の相互作用を解析できる管理手法を開発することにより、成果物の生成・変更作業の支援を図る。

1.3　本論文の構成

本論文の構成は以下のとおりである。第 2 章では、一般的な文書管理における様々な技術についての経緯と現状を分析する。第 3 章では、プロセスモデル (チーム構造モデル) を取り入れた、提案する管理モデルとそのモデルに付随する概念について述べる。第 4 章では、その管理モデルを使用した管理手法と、その手法により得られる成果物に関する情報について述べる。第 5 章では、管理手法の基本的な要素を実際に構築・実現する。第 6 章では、シミュレーションにより管理システム (手法) を総合的に評価する。最後に第 7 章で本論文をまとめる。

第2章　文書管理技術における現状と問題

文書管理とは、一般的に文書をどのように保存し、また保存されている文書を再利用する目的のために、その効率を向上させる作業の事をいう。具体的には、文書を検索して表示/変更したり、そのための方法を検討する作業のことである。

文書管理には様々なものがあるが、本研究における文書管理は、共同ソフトウェア開発における成果物を対象にすることから、構成管理や知識管理の意味も含まれる。

共同ソフトウェア開発では、様々な役割を持つ開発者により、多くの成果物が生成・更新されることから、ソフトウェア開発プロジェクトの管理項目の一つである版管理や変更管理を含むソフトウェア成果物の (構成) 管理は、工程管理や外注管理と並ぶ非常に重要な項目の一つである。

共同ソフトウェア開発における文書管理に望まれる点を以下に述べる。

成果物の再利用 : 目的とする成果物の情報の検索

成果物の一貫性 : ある成果物の変更が、他の成果物に与える影響の検索

プロセスの理解 : 成果物の情報の背景にある開発プロセスの検索

次節では、文書管理における様々な技術についての経緯と現状を分析し、本研究における文書管理の特徴を述べる。

2.1　文書管理の経緯

文書管理の歴史は、紙ベースの文書管理に始まる。紙ベースの文書はフォームとキングファイルにより管理され、一度ファイルにまとめられたの文書が膨大な数である場合、再び検索することは非常に困難である。また、高度な検索や整理が不可能であることから、検索容易性においては非常に乏しい。さらに、物理的な制約が大きいことからも、再利用には限界がある。

その後、紙文書の電子化が進み、物理的な制約はほぼ解消され、電子化された利点を生かしたコンピュータによる支援が始まる。ここ数年、様々な技術が開発され、その特性を利用した支援が盛んに行われているが、代表的なものを以下に取り上げる。

2.1.1 RDB

RDB(Relational Database) は、1970 年代、E.F.Codd によって提案され、1980 年代以降、実用システムにおけるデータベースの標準言語となった SQL(Structured Query Language) と共に、圧倒的な成功を収めている。

和 (union), 差 (difference), 直積 (Cartesian product), 射影 (projection), 選択 (seleceion) の 5 つの基本演算子から成る関係演算の組み合わせにより、様々な条件下での検索を可能とし、これによって、検索容易性はほぼ達成された。

しかし、RDB での情報の構成は、「テーブル (表)」として、管理される必要があるため、成果物と成果物情報を別々に管理しなければならず、成果物の変更に伴う成果物情報の更新作業には問題が残る。また、RDB は単純で規則的なデータに向いているため、共同ソフトウェア開発のような不規則なデータを扱う場合には、半構造データと呼ばれるようなデータモデルが必要とされる。

2.1.2 HTML

HTML(Hyper Text Markup Language)[5] とは、WWW 上で文書を記述するための言語であり、W3C[1]による勧告である。HTML は文書の構造や表示方法などの要素を定義し、image や URL を取り込むことにより、ハイパーテキストを実現する。HTML は単純な構成であり、かつテキスト文書であることことから、端末への依存性が低い。さらにタグとハイパーリンク構造により、ある程度の不規則なデータを構成することも可能である。ハイパーリンク構造を図 2.1 に示す。

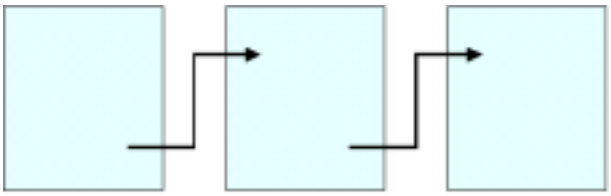


図 2.1: ハイパーリンク構造

しかし、HTML 文書の構造は、一般的な文書管理に考えられる要素 (章、節、項といった階層構造や、作者、制作日時など) を、明確に規定することができないという問題点がある。したがって、HTML 文書をデータベースのように扱うには限界がある。

[1]The World Wide Web Consortium

2.1.3 XML

XML(eXtensible Markup Language)[6] とは、1998 年に W3C から勧告され、SGML や HTML を基に、その欠点を補完するように設計されたマークアップ言語である。

一般的に、インターネット上でのデータ交換フォーマットとして、また、異種アプリケーション間のデータ交換フォーマットとして、データ構造を定義するための DTD とともに用いられる。XML はタグにより意味内容を表現可能であり、タグの名前を自由に決定可能なことから、情報内容や用途などを明確に表すことが可能である。図 2.2 に XML の記述例を示す。

図 2.2: XML 記述の例

XML 文書は、XML 宣言、DTD(文書型定義)、XML データ (XML 文書本体) の 3 つの部分から構成される。XML の基本単位である要素を表 2.1 に示す。

表 2.1: XML の基本単位 (要素)

< 要素名 > 要素内容 < /要素名 >
開始タグ + 文字データ + 終了タグ

XML の文書構造は、階層構造や任意の要素を明確に規定可能である。

DOM

DOM(Document Object Model) とは、XML とアプリケーションの中間に位置し両者をつなぐ役割を果たすもので、ひとつながりのデータである XML 文書をツリー構造に階層化し、プログラムから操作可能な対象物 (オブジェクト) に変換したものである。XML 文書の要素や属性は、パーサを通すことで DOM オブジェクトになり、個々の要素や属性に対して DOM API を通じてアクセス可能となる。

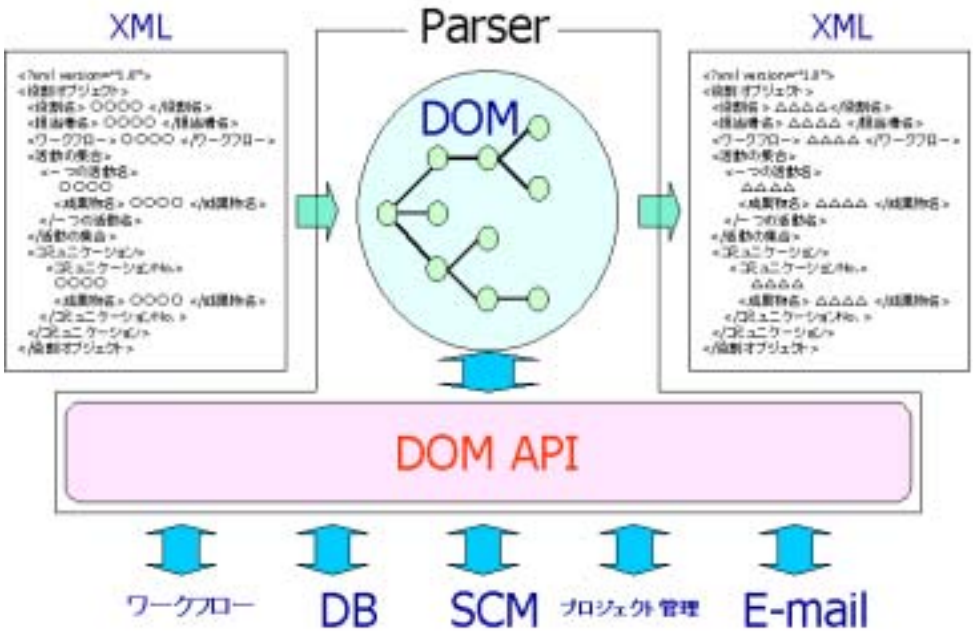


図 2.3: XML と DOM とアプリケーションの関係

DOM は、基本的にツリー構造で (各要素間の包含関係として) 表現されている XML データから、柔軟なアクセスが可能になるツリー構造をメモリ上に生成する。DOM には多様な API が用意されていて、アプリケーションデータから、DOM のツリー構造データを作成し、変更することが可能である。DOM API を利用することにより、XML データへのアクセスを統一的に行える。また、要素などの小さな情報の単位を扱えることから、マッピングなどが容易に実現可能である。

XML 関連技術

XML には様々な関連技術が用意されている。図 2.4 にその概要を示し、調査した結果の主なもの以下にまとめる。

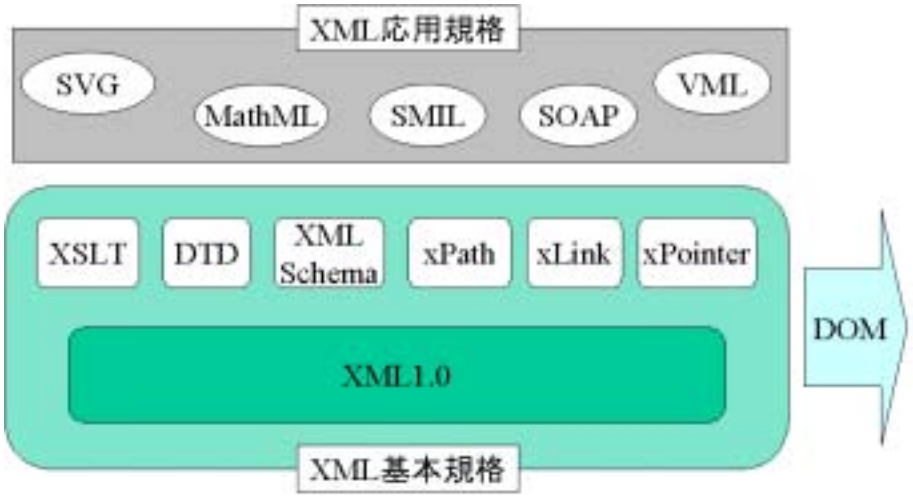


図 2.4: XML の関連技術

DTD

DTD(Document Type Definition) とは、XML 文書内に現れる要素や属性、階層構造などのデータ構造を定義するためのものであり、文書の型となるタグ (要素) と階層構造、要素の属性を宣言し、XML インスタンスのタグ付けを規定する。

XML には、XML の文法に適合している整形式 (well-formed) の XML 文書と、XML の文法と DTD の定義に適合している妥当 (Valid) な XML 文書の 2 形式がある。文法上、正しい XML 文書であっても、要素名や要素構造が明確でなければ、アプリケーション間での交換、共有は不可能である。よって、アプリケーション間でどのようなタグ (要素) を使うか、タグの階層構造がどのようになっているか明確にしておく必要があり、そのために DTD が使用される。DTD がある場合は、整形形式の XML 文書であっても、要素情報に従いインスタンスを書く必要がある。

現在、DTD の他に XML 文書の構造を定義するスキーマ言語として、RELAX (REgular Language description forXML) や、XML Schema がある。

XSL

XSL(eXtensibleStylesheet Language)[7] とは、2001 年に W3C から勧告された、XML のスタイルシート言語であり、XML と共に用いられる。

XML は、紙や画面に表示するためのスタイル情報を含まないため、XML を表示する場合は、XML の構造を出力用に変換し、それにフォーマット情報を付加することにより表示を行う。この作業を行うのが XSL である。CSS(Cascading Style Sheet) との違いは、CSS がタグごとに表示情報を設定するだけに対し、XSL は表示だけでなく、全体の構成を変えたり、その情報を表形式で表示するといったアプリケーション的な処理も可能である。

SVG

SVG(Scalable Vector Graphics)[14] とは、2001 年に W3C から勧告された、高度な 2 次元グラフィックの標準規格である。

SVG は、JPEG や PNG 等と異なり、点 (ラスタ、ビットマップ) で画像を表現するのではなく、直線や曲線 (ベクトル) で画像を表現している。ベクタルで表現することにより、ラスタで起る拡大・縮小などの損失もなくなる。SVG は XML に準拠していることから、フォーマットが統一されており、また、実体がテキスト形式であることから、様々な応用 (検索、リンク、表示方法など) が可能となる。

SOAP

SOAP(Simple Object Access Protocol)[9] とは、ネットワーク上のアプリケーション間 (オブジェクト間) の情報を交し合うためのシンプルなプロトコルの仕様である。現在、W3C により技術ノート (Note) として仕様が公開されている (2001 年 2 月現在)。このプロトコルを表現するための手段として XML が利用されている。

SOAP は、XML ベースで通信を行うため、既存の XML ツールや環境を応用することが可能である。また、様々なプロトコル (HTTP や SMTP など) にバインディングして使用可能であるため、環境に依存しない「開かれた」アプリケーションの作成可能である。

SOAP のメッセージ構成は、SOAP メッセージは大きくわけて「ヘッダ (Header)」と「SOAP エンベロープ (SOAP Envelope)」から構成される。さらに「SOAP エンベロープ」は「SOAP ヘッダ (SOAP Header)」と「SOAP 本体 (SOAP Body)」から構成される。

2.2 文書管理の現状

代表的な文書管理技術の現状を以下に述べる。

2.2.1 PCTE

PCTE(Portable Common Tool Environment)[11] とは、開放型リホジトリのための共通ツールインターフェース (PTI:Public Tool Interface) 標準であり、CASE ツールが高度な統合性と可搬性とを持つことを可能にするための、共通で標準的なサービス群を規定するものである。

PCTE は、ツール、ソフトウェアプロダクト、仕様書などといった、ソフトウェア生産環境において必要な情報の全てを蓄えておくための場所であるリポリトリに蓄えられた共通の情報を、異なる種類のツールがアクセスし、操作することを可能とするための共通スキーマ機構を提供する。各々のツールは、これらの情報を適当なサブスキーマを通じて、そのツールに適した形で取り扱うことが可能である。

PCTE の目的には、以下の三種類の共有があげられる。

- データ共有：ツールと環境の間での情報共有
- 表示共有：環境内のツールによるユーザインタフェースの共有
- 制御共有：環境内の個々のツール間の強調と通信制御の共有

PCTE は 1983 年から ESPRIT [2] のプロジェクトの一環としてスタートし、NATO 拡張、ECMA 標準、そして現在では、分散したソフトウェア開発環境 (CASE) の統合のためのリポジトリとして、ISO 標準になっている。

2.2.2 Rational ClearCase

現在、もっとも多く使用されているソフトウェア構成管理製品として、Rational 社が開発した ClearCase がある。ClearCase は、バージョン管理、作業領域管理、ビルド管理、プロセス管理を包括的に行うことによって、チームによるソフトウェア開発を支援する、ソフトウェア構成管理 (SCM) システムである。VOB(Versioned Object Base) と呼ばれる独自のリポジトリを使用して、ソースコード、ディレクトリ、テストスクリプト、ドキュメントなど、すべての開発資産をバージョン化し、ソフトウェアの完全なバージョン履歴を管理する。ClearCase は、開発プロセスを効率化することで、短期間で高品質のソフトウェアを作成することを目的としている。

[2] European Strategic Program for Research and Development の略称.

2.3 文書管理技術における本研究の立場

本研究では、管理モデルに「チーム構造モデル (プロセスモデル)」を用い、変更管理に「変更スレッド候補 (成果物の変更に関する一連のスレッド)」を用いることで、共同ソフトウェア開発におけるの成果物の修正変更作業を支援する管理モデルを構築する。具体的には、ソフトウェア開発において生成される成果物を「XML という封筒」に格納 (エンベロープ技術) して管理し、依存関係などの情報をタグとして格納しておくことにより、利用者の要求を満たす検索を可能にする。

2.3.1 管理対象

本研究で管理する対象は、ソフトウェア開発のと一連のワークフロー (要求→分析→設計→実現→テスト) において生成される、図 2.5 に示す成果物の各版とする。

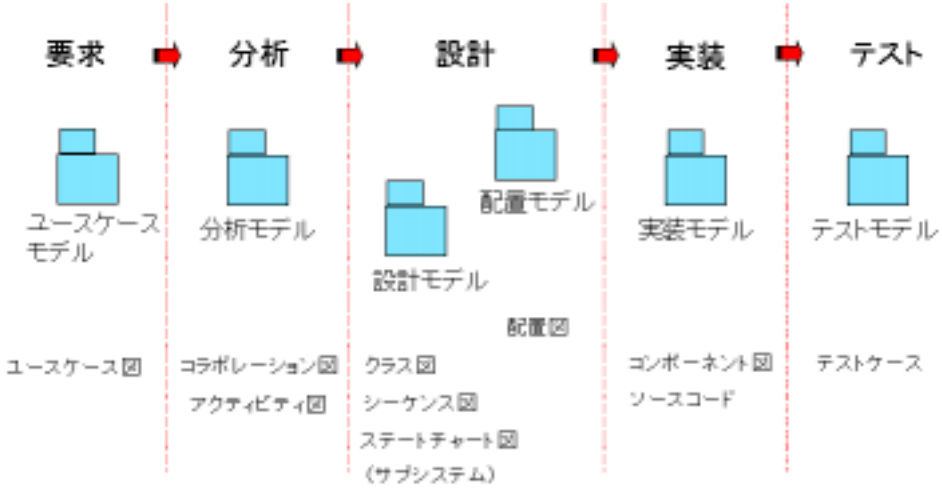


図 2.5: 管理対象

具体的には、Unified Process[3] の開発過程に生じる、UML で記述されたユースケース図、コラボレーション図、アクティビティ図、クラス図、シーケンス図、スレートチャート図、配置図、コンポーネント図等の成果物の各版である。

2.3.2 管理方針

成果物を管理する管理方針を以下に述べる。

- 開発者の登録時の負担は、最小限にする
  (あいまいな記述や、表現しにくい情報も存在するため、可能な限り自動化し、手動による入力は単純なものにする)

- 登録した付加情報以上の情報を提供する
  (管理モデルを基に、タグの組み合わせやトレースアルゴリズムを考える)

- 開放的なシステムとする
  (入力情報と出力情報を絶対的なものと考えず、あくまで可能性として扱い、システム自体も修正変更が行える開放的なものとする)

※ 波及効果の及ぶ可能性のある成果物、また成果物の変更順序の可能性を提示することは、修正変更作業の支援に有意義であると考える。

2.3.3 XML の適用

本研究では、成果物の生成・変更を行う際に必要となる情報を柔軟な視点から検索できる成果物の管理モデルを提案するために、XML 技術 (XML タグによる由な視点からの柔軟な構成・ワークフロードメイン連携での、企業間における異種アプリケーション間のデータ交換) をソフトウェア開発に対応させる。

- 開発ワークフロー間における異種アプリケーション間のデータ交換に適応
- 任意のタグの決定を、管理モデルの構築に適応
- 変更スレッド候補の管理に DOM を適応

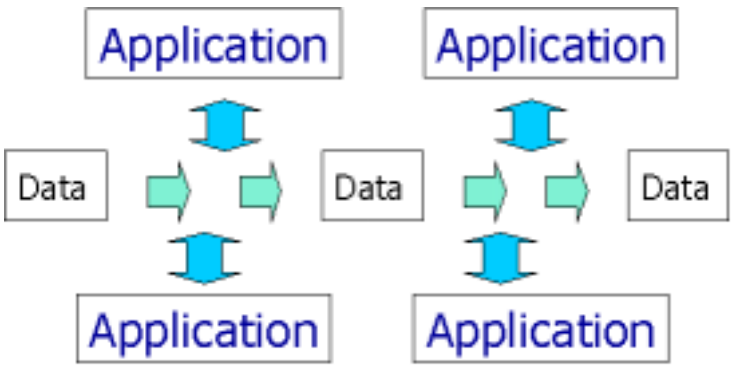


図 2.6: アプリケーション・ワークフロー

成果物を「XML の封筒」に格納することは、ワークフロー間・アプリケーション間の連携を可能にする。さらに、封筒に依存関係などの付加情報をタグとして格納することは、付加情報の利用はもちろんであるが、成果物と付加情報が一緒に管理されていることによるタグと成果物間の一貫性を高める利点もある。

以上の利点をもと、次章では管理モデルを検討していく。

第3章　管理モデル

3.1　プロセスモデルの適応

本研究では、成果物の管理モデルにソフトウェアプロセスモデルを取り入れ、全ての成果物をプロセスの基に管理することにより、開発プロセスの変更にともなう成果物 (付加情報) の変更や、開発プロセスからの各成果物の背景にある情報の参照を可能にすることを目的とする。

3.1.1　チーム構造モデル

本研究室 (自在プロジェクト) におけるソフトウェア開発支援において定義された、**チーム構造モデル** [15] と呼ばれるプロセスモデルがある。チーム構造モデルは、Coplien による組織パターン [17] の知見に基づいており、プロセスの静的側面を定義するモデルである。以下、チーム構造モデルについて述べる。

チーム構造モデルは、役割オブジェクト、コミュニケーション・パス・オブジェクト、分散作業空間オブジェクトから構成される。

役割オブジェクト

Unified Process におけるアクターと Coplien による役割 (密接に関係する活動の集合) を対応させる。また、アクターが利用するユースケースに含まれる活動を Coplien による責任 (役割間に結合された活動) に対応させる。責任は、オブジェクト関連責任とコミュニケーション関連責任に分類する。さらに、役割 (ロール・アクター) 毎にワークフローを定義し、進歩状況表現の基礎とする。ワークフローはアクティビティ図で表現する。

コミュニケーション・パス・オブジェクト

分析モデルまたはコラボレーション図におけるイベントの流れに従って、(通知する・通知される)、(質問する・答える)、(共に検討する) などの、アクター (役割) 間の結合を定義し、設計の基礎とする。これにより、役割オブジェクト間にコミュニケーション・パスが張られる。コミュニケーション・パスもオブジェクトとして表現する。

分散作業空間オブジェクト

役割毎の作業の独立性を可能な限り保証する立場をとり、役割毎の作業の責任範囲

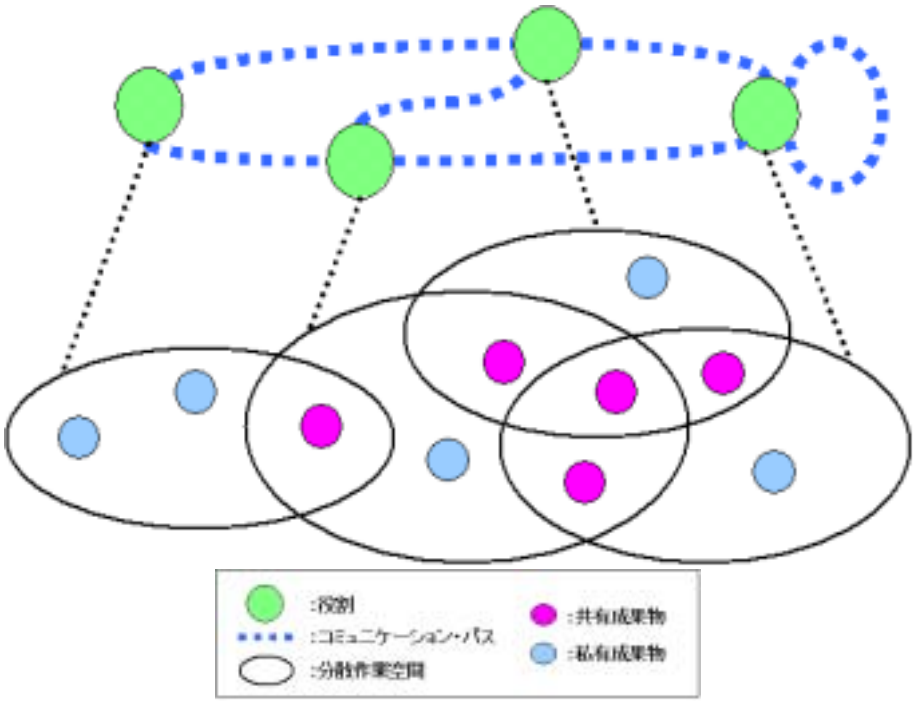


図 3.1: チーム構造モデル

と他の役割との関係を記述することにモデル定義の力点を置く。具体的には、役割毎の仕事の責任範囲を成果物オブジェクトの集合として捉える (以後タスクと呼ぶ)。分散作業空間は、一部の成果物が共有されているタスクである。分散作業空間の積集合の要素が共有成果物オブジェクトであり、他の役割との作業インターフェースである。複数の分散ワークスペースの重なりの時間軸上での動的変化をプロセスの実行として定義する。

成果物オブジェクト

UML を利用した各種方法論における各種成果物 (ユースケース図、コラボレーション図、クラス図、シーケンス図、ステートチャート図、配置図、コンポーネント図、テストケース) と Java プログラムを対象としている。成果物間には UML における trace 依存関係が設定されていること、版管理がなされていることを前提とする。

まず、ソフトウェア開発における密接に関係する活動をの集合を役割と考える。それぞれの役割は、役割毎の作業の責任範囲 (作業空間) をもっており、そこには私有成果物が存在する。役割間には、役割に結合された活動間の意味的結合としてコミュニケーションパスが張られ、これを通して「通知する-通知される」、「質問する-答える」といったコミュニケーションを行うことにより、責任範囲を変化させる。これにより、ある役割の私有成果物が他の役割に受け渡され、責任範囲を共有するという意味で共有成果物となる。

3.2　修正変更作業を支援する管理モデル

共同ソフトウェア開発において生成される成果物を実際に管理するには、頻繁に発生する成果物の変更要求に対して、成果物の修正変更作業を支援する管理モデルが必要である。

3.1.1 節で述べたチーム構造モデル (プロセスモデル) を基に、修正変更作業に関する情報を取り入れたモデルを、「成果物の管理モデル」とする。そのモデルを図 3.2 に示す。

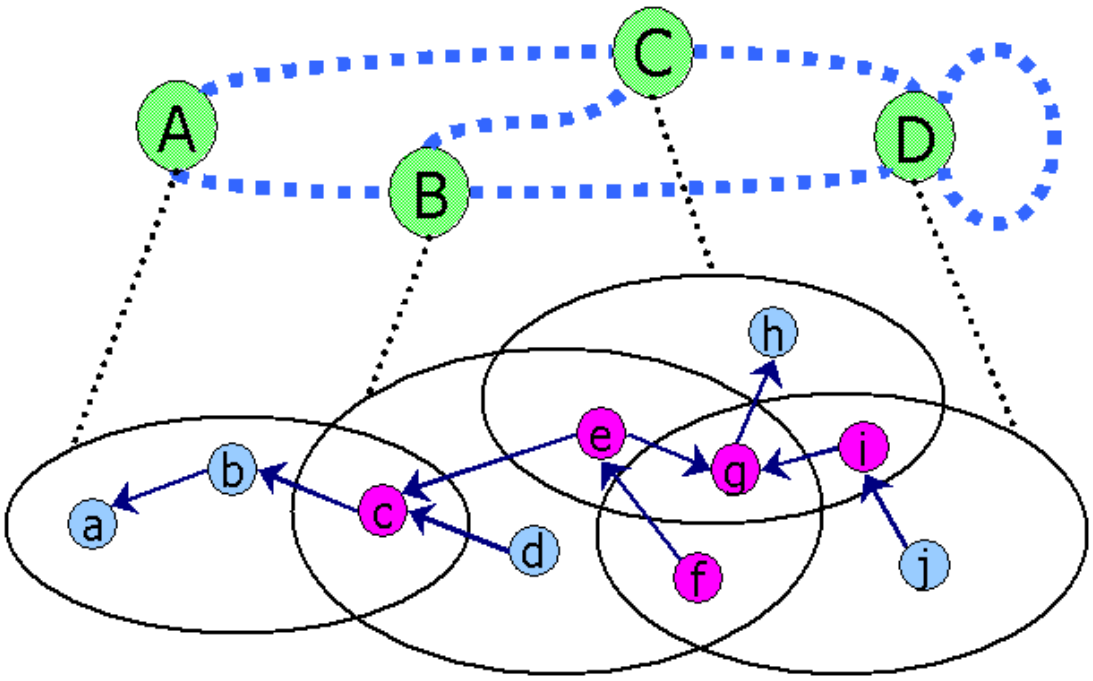


図 3.2: 成果物の管理モデル

- “a ← b” は、「成果物 b」は「成果物 a」に依存していることを表す。
  (b が a に依存しているとは、「a を変更すれば、b を変更する必要がある」ことを意味する。)
- “c ← e → g” は、「成果物 e」は「成果物 c」と「成果物 g」に依存していることを表す。
  (e が c と g に依存しているとは、「c または b を変更すれば、e を変更する必要がある」ことを意味する。)

この管理モデルは、チーム構造モデルに成果物間の依存関係情報を追加することにより得られる、以下 3.2.1 節に述べる「変更スレッド候補」の概念を用いて、修正変更作業を支援することを目的とする。

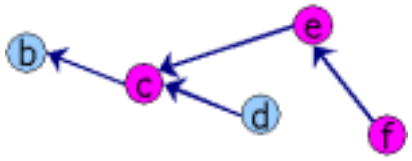
3.2.1 変更スレッド候補

依存関係グラフ中 (図 3.2 参照) のひとつの部分グラフとして、「変更スレッド候補」を定義する。「変更スレッド候補」とは、変更要求のある成果物を起点に、依存関係を順番にトレース (矢印を逆向きにトレース) することにより導出する一連のスレッドである。

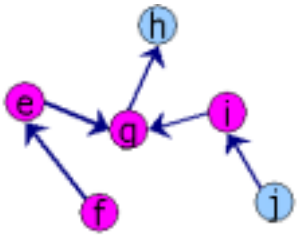
ある成果物の変更によって、(確実に) 影響を受ける成果物を、変更要求が発生した時点において検知することは不可能である。そのため、変更の影響を受ける可能性のある成果物を依存関係をもとに導出し、そのスレッドを、変更の影響を受ける “可能性” のある成果物のスレッドという意味で、特に「変更スレッド “候補”」として定義する。

定義：変更スレッド候補
依存関係をもとに導出した変更の影響を受ける可能性のある成果物を、変更順序に従って並べた成果物の変更に関する一連のスレッドである

具体的な変更スレッド候補の例を以下に示す。



成果物 b を変更する場合



成果物 h を変更する場合

※ 特殊な事例
成果物 b を変更する場合を例に補足する。成果物 b に問題があり、成果物 b を変更するには、成果物 b の参照先にある成果物 a に対しても、その変更を反映しなければならない場合が考えられる。本来、これは変更スレッド候補の導出の際に考えなければならない問題であるが、本研究では、そのような場合を特殊な場合と位置づけ、また、変更スレッド候補は完全な変更波及の検知を目的としないため、反映しないものとする。

依存関係から導出した変更スレッド候補は、一般的に、同時に複数個存在する可能性が考えられる。

3.2.2 コミュニケーション頻度

ソフトウェアエンジニアリング諸活動には、オブジェクト関連活動とコミュニケーション関連活動の 2 種類がある。両者の関係を図 3.3 に示す。

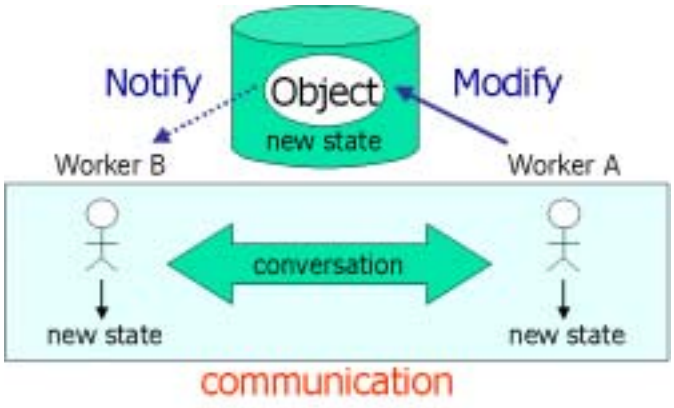


図 3.3: オブジェクト関連活動とコミュニケーション関連活動の関係

一般的に共同作業の参加者は、コミュニケーションのゴール、成果物やその要素の機能・役割・実装方法に関する共通認識に達する。共同作業のバックグラウンドとなる共通認識は、まず、状況の把握から始まり、認識のズレの解消へと進む。コミュニケーションの結果、各自は新しい認識状態に達し、その状態を各自の責任範囲下にある成果物オブジェクトに反映させる。このような成果物の状態は、再び関係者間のコミュニケーションを引き起こす。

つまり、成果物の変更に関して考えた場合、変更スレッド候補中に存在する成果物のうち、他の役割と責任を共有している成果物 (共有成果物) の変更においては、コミュニケーション・パスを使用してネゴシエーションをとる必要がある。

このことから、共同ソフトウェア開発では、変更の及ぶ成果物だけでなく、その成果物に責任を持つ役割、特に共有成果物の場合においては、役割間のコミュニケーションの頻度 (パスの必要とする回数) もまた、変更にかかるコストとして考慮する必要があると考える。

コミュニケーション頻度は、以下の要素により影響されるものと考えられる。

- 共有成果物数
- 責任を共有する役割数

さらに、コミュニケーション・パスの欠如などによる役割間のルート (道順) の違いについても、要素となる可能性が考えられる。また、成果物の変更は、役割間のコミュニケーションのルート (道順) が確立されていることを前提に行うべきであると考える。

具体的なコミュニケーション頻度の例を以下に取り上げる。

- **成果物 b を変更した場合** (変更スレッド候補中に成果物 b,c,d,e,f が存在)
  - **共有成果物 c** (役割 A,B で共有) : A-B で 1 回
  - **共有成果物 e** (役割 B,C で共有) : B-C で 1 回
  - **共有成果物 f** (役割 B,D で共有) : B-D で 1 回

  コミュニケーション頻度 : 計 3 回

- **成果物 h を変更した場合** (変更スレッド候補中に成果物 h,g,e,f,i,j が存在)
  (役割 D 自身のコミュニケーションを 1 と仮定)
  - **共有成果物 g** (役割 C,B,D で共有) : B-D-C で 4 回
  - **共有成果物 e** (役割 B,C で共有) : B-C で 2 回
  - **共有成果物 f** (役割 B,D で共有) : B-D で 2 回
  - **共有成果物 i** (役割 C,D で共有) : C-D で 2 回

  コミュニケーション頻度 : 計 10 回

「変更スレッド候補」と共に、「コミュニケーション頻度」を用いることで、変更成果物の変更に費やすおおよそのコスト (変更の影響を受ける成果物の、変更コストやコミュニケーション頻度等の総和) の計算と、修正変更作業を円滑に進めるための大まかな対策 (コミュニケーション・パスの確立、各役割のワークフローの計画等) を立てることが可能になると考えられる。

3.3　管理モデルに対応するタグ

管理モデルを構築するにあたり、一般的な文書管理に必要なタグと管理モデルに必要なタグを表 3.1、表 3.2 に示す。また管理モデルとタグの対応を図 3.4 に示す。

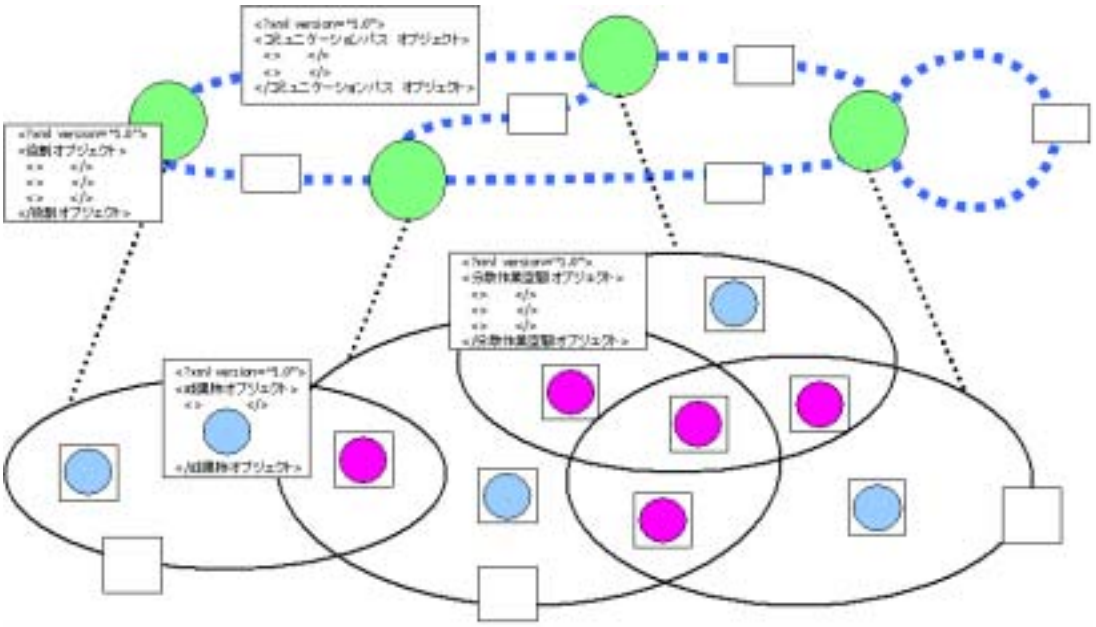


図 3.4: 管理モデルとタグの対応

表 3.1: 一般的な文書管理に必要なタグ

・文書情報 (成果物名、ID、版 No)
・作成情報 (作成開始日時、作成終了日時、作成者)
・ショートカット情報 (管理者、URI、ファイルタイプ)
・コメント情報 (自由にタグ付け)

追跡依存情報、内容情報については、成果物情報と同様に成果物と共に格納する。役割情報、コミュニケーション・パス情報、分散作業空間情報、成果物情報については、チーム構造モデルにおいて、既に定義済みである。また、**追跡依存情報**に位置する < 参照した他の図 > と < 参照した自分の版 No> については、3.4 節で詳細に述べる。

表 3.2: 管理モデルに必要なタグ

役割情報
- 役割名
- 担当者名*
- 使用する活動のワークフロー
- オブジェクト関連責任
  - 活動名 (成果物名、アクセス権)*
- コミュニケーション関連責任
  - コミュニケーション・パス名 (グループ名、成果物名)*

コミュニケーション・パス情報
- コミュニケーション・パス名
- 関連する役割名*
- 関連する討議スレッド
  - 討議スレッド名*
- 参加グループ名
- 状態

分散作業空間情報
- 作業空間名
- 関連する役割名*
- 私有成果物
  - 成果物名 (未来版空間 No)*
- 共有成果物
  - 成果物名 (トランク No)*

成果物情報
- 成果物名
- 型
- 版 No
- 依存*
- 状態

追跡依存情報
(参照した自分の版 No、参照した他の図、作成・変更理由)

内容情報
(内容説明、内容履歴、キーワード、アブストラクト)

3.4　タグの定義とトレース方法

管理モデルを構築するにあたり、はじめに、変更スレッド候補を導出するためのタグの定義とその使用方法を検討する。

コラボレーション図、シーケンス図、クラス図等に各版が存在すると仮定する。これらの成果物を、依存関係などの付加情報ともに XML 文書の封筒に入れて管理し、＜参照した他の図＞と＜参照した自分の版 No＞という 2 つの依存関係を使用して構造化した。図 3.5 に、成果物間の構造と変更スレッド候補を示す。

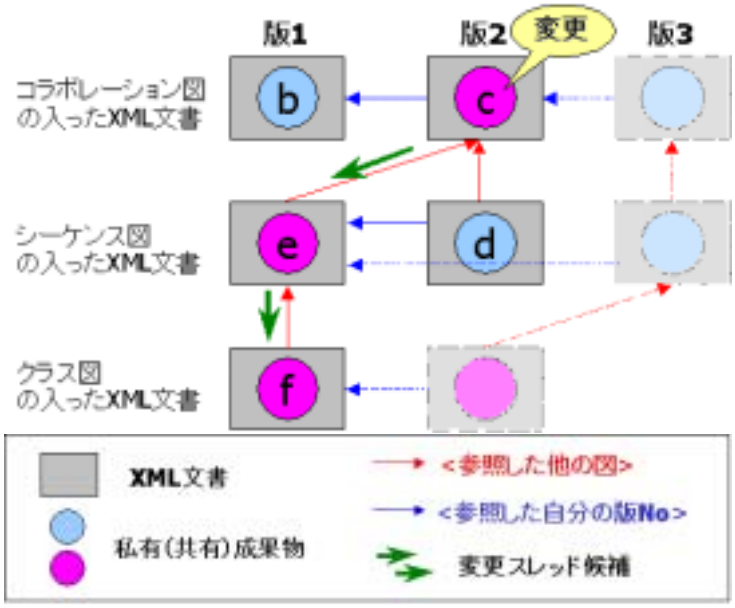


図 3.5: 変更スレッド候補の導出

全ての依存関係をトレースすると、様々な種類の依存関係により、ほとんどの成果物が変更スレッドの候補に入ることになる。しかし、実際に変更が必要な成果物は、その全てではないことから、変更スレッド候補をある程度しぼる必要がある。よって、変更スレッド候補を絞るために、＜参照した他の図＞と＜参照した自分の版 No＞の依存関係を区別してトレースする。

成果物 c を変更する場合を例として、変更スレッド候補を導出する手順を以下に示す。

1. ＜参照した他の図＞で依存関係にある最も新しい版 (成果物 d) を選択する。
2. 成果物 d には＜参照した他の図＞の依存がないので、＜参照した自分の版 No＞をたどり、以前の版 (成果物 e) を選択し直す。
3. 成果物 e には＜参照した他の図＞の依存が存在するので、それを選択する。

3.5 タグ (追跡依存) の有効性の確認

定義したタグの有効性の確認を目的として、追跡依存に関するタグ < 参照した自分の版 No>,< 参照した他の図 > を例に、以下に示す条件の下で評価を行った。

- 対象：本研究内の「自在プロジェクト」における、Web アプリケーションを開発する過程で生成された成果物
- 成果物数：22 個
- 成果物の種類：11 個
- 版 No：1〜4
- 変更パターン
  - パターン 1： コラボレーション図のあるコントロールオブジェクトを変更
  - パターン 2： クラス図のあるクラスを変更

評価の尺度には、正解文書を漏れなく検索できる能力の指標である「再現率」と、正解でない文書を検索しない能力の指標である「適合率」を用いた。各定義を表 3.3 に示す。

表 3.3: 評価の尺度

再現率 $=$ $\frac{\text{検索成果物中の該当成果物数}}{\text{該当成果物数}}$
適合率 $=$ $\frac{\text{検索成果物中の該当成果物数}}{\text{検索成果物数}}$

実験結果を、以下に示す。

- 再現率 $7/7 = 100\%$(パターン 1) $6/6 = 83\%$(パターン 2)
- 適合率 $7/12 = 58\%$(パターン 1) $6/11 = 55\%$(パターン 2)

変更内容や依存関係のトレース方法により、結果は大きく変化する可能性が予想されるが、この実験から、単純なタグと簡単なトレース方法であっても、ある程度の結果は得られるという知見が得られた。

本研究における成果物の管理方式は、必要最低限の情報をもとに、変更に及ぶ可能性を提示することにより、共同ソフトウェア開発を支援する目的のため、複雑で厳密なことより、シンプルで利便性があることに重点をおく。よって、この知見を基に、次章で、管理手法について検討する。

第4章　管理手法

第 3 章で提案した管理モデルを基に、本章では、成果物の管理手法について述べる。

4.1　管理手法の概要

本研究で提案する管理手法は、基本的に管理モデルと変更スレッド候補を中心にした変更管理手法であるが、それを補助するための手法として、検索空間、視覚化表示についても検討する。本管理手法により、成果物の変更の波及効果、変更に費やすコスト、変更作業のスケジューリングの可能性を提示することにより、共同ソフトウェア開発を支援する。

本管理手法におけるソフトウェア開発サイクルとともに、管理手法の全体像を図 4.1 に示す。

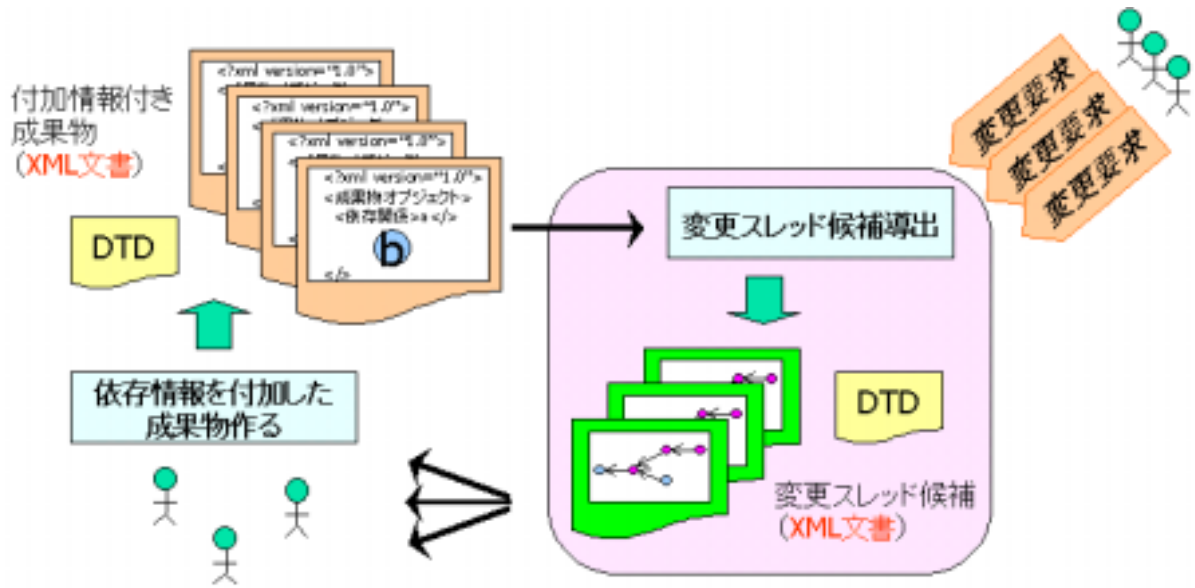


図 4.1: 管理手法の全体像

開発過程は、以下の 4 つの過程に分けて考えられる。

1. 開発者よって作成 (変更) された成果物は、手動 (一部自動) により「成果物に関する情報」を付加した後、リポジトリに格納する。

2. (複数の) 変更要求の発生に伴い、(複数の) 変更スレッド候補を導出する。

3. 変更スレッド候補を統括的に管理し、変更に必要な情報 (成果物の変更の波及効果、変更に費やすおおよそのコスト、変更順序) を提示する。

4. 開発者は提示された情報をもとに、成果物の変更を行う。(1 へ戻る)
(なお、変更スレッド候補は変更作業が進む度に、常に更新される。)

本管理手法は、チーム構造モデルと変更スレッド候補を用いることに加え、XML による利点を用いることにより、共同ソフトウェア開発における成果物の変更作業の支援を実現する。繰り返し修正変更作業が行われる共同ソフトウェア開発において、本管理手法を用いた利点を以下に述べる。

- ソフトウェアの成果物は XML 文書の封筒に格納して管理する。また、封筒には依存関係などの情報も同時に格納しておく。一般的に、エンベローブと呼ばれる方法により、成果物と付加情報を一つの XML 文書で管理できることは、頻繁に更新作業が行われるソフトウェア開発において、付加情報との一貫性を保つのに非常に有効であり、また付加情報と成果物間のリンクをたどる必要がないことから、利便性に優れている。

- 成果物 (リンクでも可) を XML の封筒に格納することにより、異なった事業所間、異種アプリケーション間でデータ交換を可能にする。これにより、異なる企業で異なるアプリケーションを使用している場合でも、共同開発をスムーズに行うことが可能である。つまり、分析を A 社、設計を B 社、実現を C 社で行う場合などのワークフローにおける連携も比較的スムーズに行える。

- 管理モデルにプロセスモデル (チーム構造モデル) を取り入れ、全ての成果物をプロセスの基に管理することから、開発プロセスの変更にともなう成果物 (付加情報) の変更や、開発プロセスからの各成果物の背景にある情報が参照可能である。

- 変更スレッド候補を用いることにより、変更に必要な情報 (成果物の変更の波及効果、変更に費やすコスト、変更順序) の可能性を測定することが可能である。

4.2　変更管理

ソフトウェア開発では、様々な役割を持つ開発者により、多くの成果物が、頻繁に生成・更新されることから、成果物の管理において、変更管理は非常に重要な項目の一つである。

4.2.1　管理モデルにおける変更スレッド候補

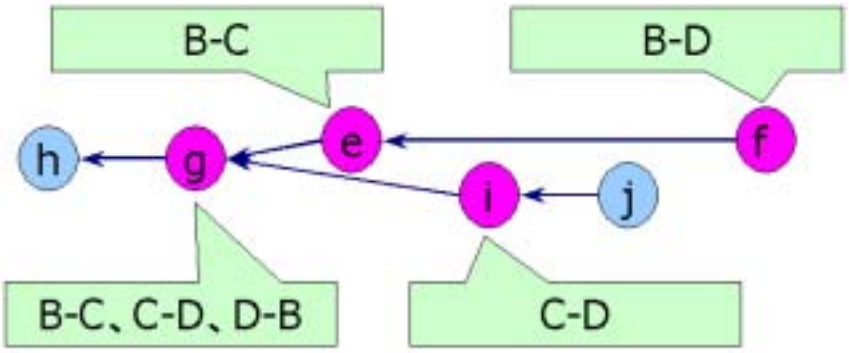


図 4.2: 変更スレッド候補とコミュニケーション・パス

ある成果物 A を変更したい場合には、まず変更スレッド候補を使用して変更の及ぶ可能性のある成果物を検知する。次に波及の及ぶ成果物はどの責任範囲に属していて、どの役割に管理されているかが分かれば、それらの役割間のコミュニケーションパスの経路やコミュニケーション頻度も分かるはずである。最終的に成果物 A を変更することの影響やコストの可能性を利用者に提示することにより、ソフトウェア開発を支援を図る。

4.2.2　複数の変更スレッド候補間の関係

共同ソフトウェア開発では、複数の変更要求の発生に伴い、複数の変更スレッド候補が導出される。よって、変更スレッド候補が、同時に複数存在する場合を考慮する必要がある。

図 4.3 に複数の変更スレッド候補の例を示す。

複数のスレッド候補間に共通して現れる成果物間には、変更の競合の生じる可能性がある。しかし、実際には、変更スレッド候補上のすべての成果物が変更されるわけではなく、競合可能性のある成果物間にも競合が起らない場合がある。「変更スレッド候補間の競合が解消された時点で競合可能性があるとしないこと」と同時に「最小限の付加情報で競合可能性の精度を上げること」を目的とし、この問題について検討する。

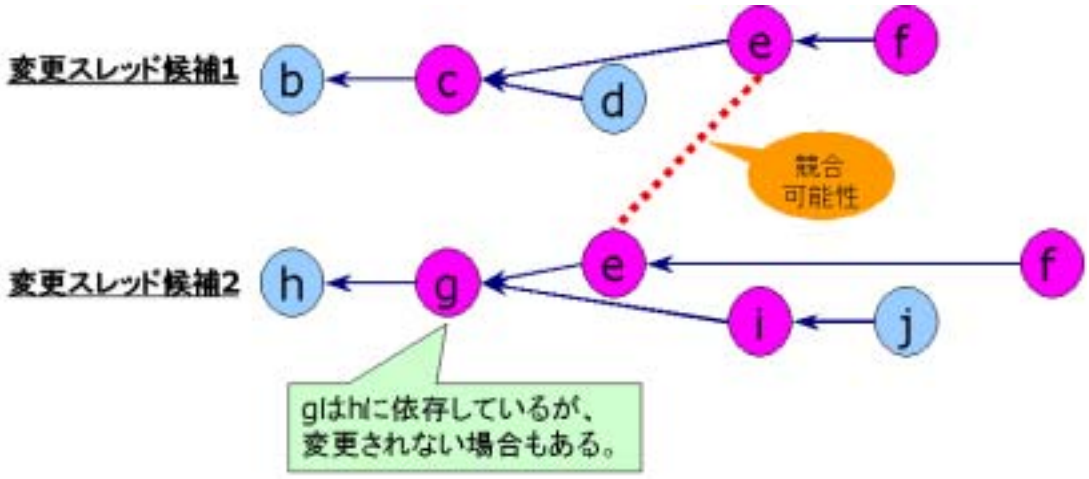


図 4.3: 複数の変更スレッド候補間の関係

4.2.3 変更管理のプロセス

変更管理は、変更要求が発生した時点で、変更スレッド候補を導出することから始まる。利用者は、導出された変更スレッド候補から得られる情報を基に、変更するか否かを決定する (波及効果の検知のみに使用することも可)。

変更のステップを以下に述べる。

1. 変更スレッド候補を導出する。(一般的に変更スレッド候補は、同時に複数存在する。)
2. 各スレッド間における競合可能性を検出し、2 通りのロック制御 (「通常ロック」「条件付きロック」) のうち一方を選択して、ロックをかける。
3. 全スレッドにおける停滞率 (全スレッドにおけるロックされるスレッドの割合) を求め、それがある閾値を超えた場合、停滞/デッドロックとみなし、全スレッドを解除する。逆に閾値を超えていない場合は、選択した一方のロック制御を用いて、スレッドの進行/停止を行う。
4. 変更スレッド候補は、成果物が変更された時点で再導出される。(2 に戻る)

変更管理手法のプロセスを、図 4.4 に示す。

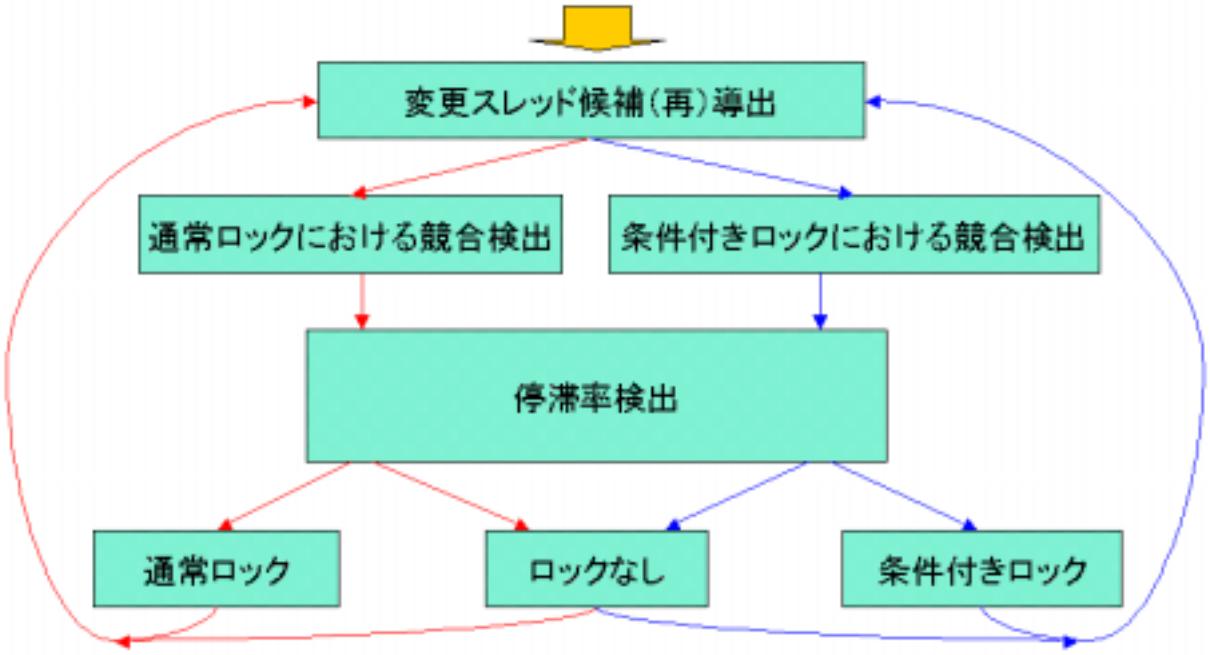


図 4.4: 変更管理手法のプロセス

デッドロック

2 つ以上スレッドがデッドロック状態になること

停滞状態

「全スレッドにおけるロックされているスレッドの割合」が、ある閾値を超えた場合、停滞状態にあるとする

通常ロック

スレッド間に競合状態が検出された場合、常にロックする

条件付きロック

スレッド間に競合状態が検出された場合、スレッドの状態を調査し、ロックするかしないかを判断する。

各詳細について以降の節で述べる。

4.2.4 変更スレッド候補の導出規則

変更スレッド候補は、あらかじめ決められた規則に従って導出する。基本的には、変更要求のある成果物からスタートし、依存関係に関するをタグを順にトレースしていくことにより、変更スレッド候補を導出する。変更スレッド候補を導出するには、依存関係を一定の方向にトレースする必要があるが、一般的に依存関係をトレースする場合、ループや循環を含む複雑な依存関係が存在し、一定方向にトレースするには、ある一定の規則が必要である。そのため、「循環する依存」「ループする依存」「複数を参照する依存」が存在する場合においては、それぞれ図 4.5 に示す規則を用いる。
(依存関係を有効グラフとして表現)

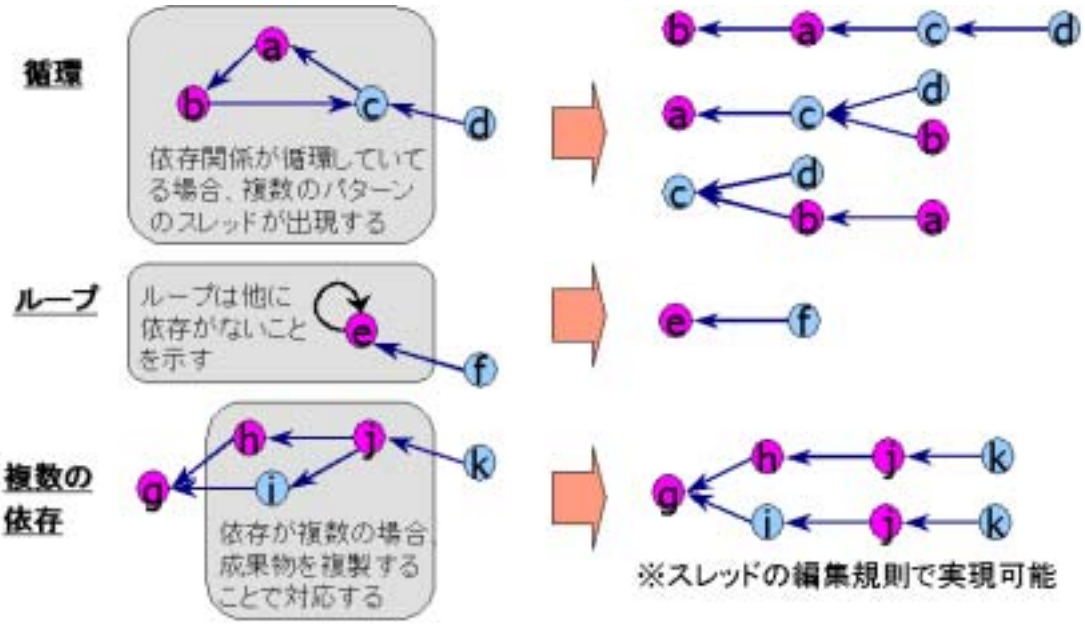


図 4.5: 変更スレッド候補の導出規則

1. **循環する依存関係**
   依存関係が循環している場合、トレース過程で循環と認識された時点で次の依存 (一度トレースされた依存) はトレースしない。
   ※ 変更要求が異なる場合、変更スレッド候補もそれぞれ違う構造を持つ。

2. **ループする依存関係**
   依存関係がループしている場合、自分自身を参照しているものとみなし、その依存はトレースしない。

3. **複数の成果物に依存する依存関係**
   ある成果物が２つ以上の成果物に依存している場合、それぞれの依存を独立なものと考え、個別にトレースする。
   ※ なお、この場合の変更スレッド候補は、スレッドの編集作業 (スレッドの合成) により、もとの依存関係グラフを再現可能である。

以上の規則を用いることで、変更スレッド候補は木構造を持つグラフとして、導出可能である。

4.2.5 変更スレッド候補の編集規則

変更スレッド候補は、その構造をもとに、編集規則に従って最適な形に編集される。編集作業は、変更作業の効率 (変更成果物の更新回数と変更作業にかかる行程数) を高めるために行うものである。以下、編集規則を述べる。

1. **変更スレッド候補の分解**
   変更スレッド候補に分岐が存在する場合、その分岐点の成果物が変更された時点で、変更スレッド候補を分解することが可能である。分解された (新たな) スレッドは、別々のスレッドとして進行し、また別々のスレッドとして管理される (図 4.6 参照)。

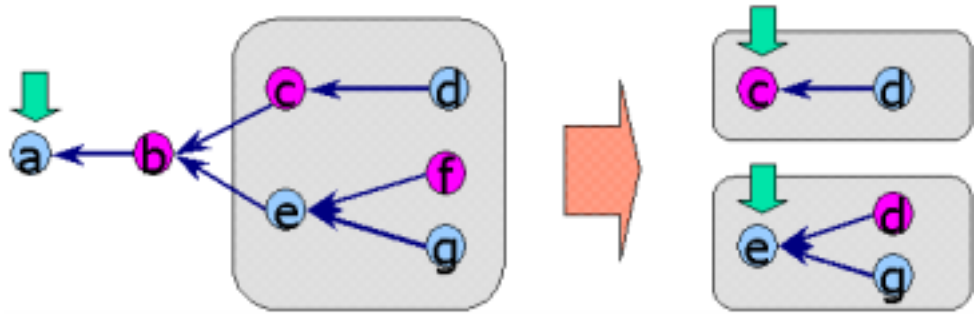


図 4.6: 変更スレッド候補の分解

2. **変更スレッド候補の合成**
   複数の変更スレッド候補に共通する構造を持つ部分 (グラフ) が存在する場合、変更スレッド候補の構造が等しくなった時点で、一つの変更スレッド候補に合成することが可能である。合成された (新たな) スレッドは一つのスレッドとして進行し、また一つのスレッドとして管理される (図 4.7 参照)。
   ※ なお、この場編集作業により、前述した変更スレッド候補の導出規則 3 が実現可

能となる。

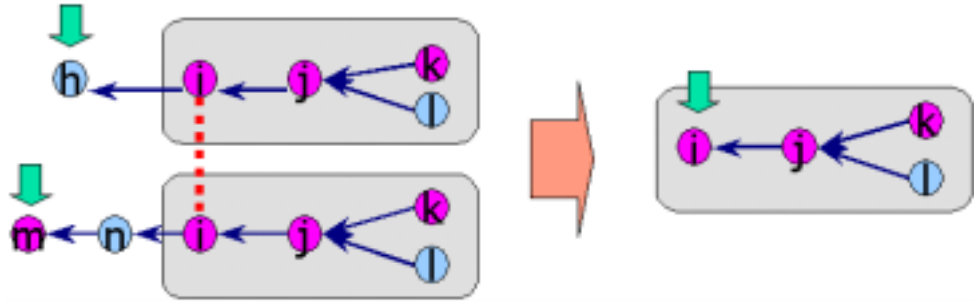


図 4.7: 変更スレッド候補の合成

3. **変更スレッド候補の吸収**
ある変更スレッド候補 (スレッド A) が、ある変更スレッド候補 (スレッド B) を包含している場合、それらの変更スレッドの先頭の成果物が一致した時点で、一方 (スレッド A) がもう一方 (スレッド B) を吸収することが可能である。吸収が行われた後の (新たな) スレッドは一つのスレッドとして進行し、また一つのスレッドとして管理される (図 4.8 参照)。

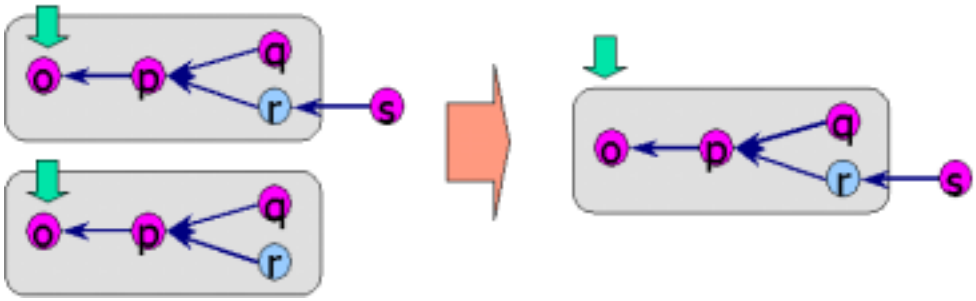


図 4.8: 変更スレッド候補の吸収

4.2.6　通常ロックと条件付きロックによる競合解消方法

スレッド間に成果物の競合が認められる場合、ロックによる競合解消を行う。ロックを用いることで、スレッド間の同期をとり、無駄な変更作業の削減による、変更コストの削減を目指す。ここでのロックは、変更スレッド候補が進行する過程において、スレッドの先頭の成果物にかけるものとする。本研究では 2 種類のロックを用いる。

通常ロック

スレッド間に競合状態が検出された場合、常にロックする

条件付きロック

スレッド間に競合状態が検出された場合、スレッドの状態を調査し、ロックするかしないかを判断する。具体的には、「自分のスレッドにおける競合可能性のある成果物 A に依存する成果物の変更コストの総和」と「他スレッドにおける競合可能性のある成果物 A に至るまでの成果物の変更コストの総和」を比較して、ロックを行うか否かを決定する。その概要を図 4.9 に示す。

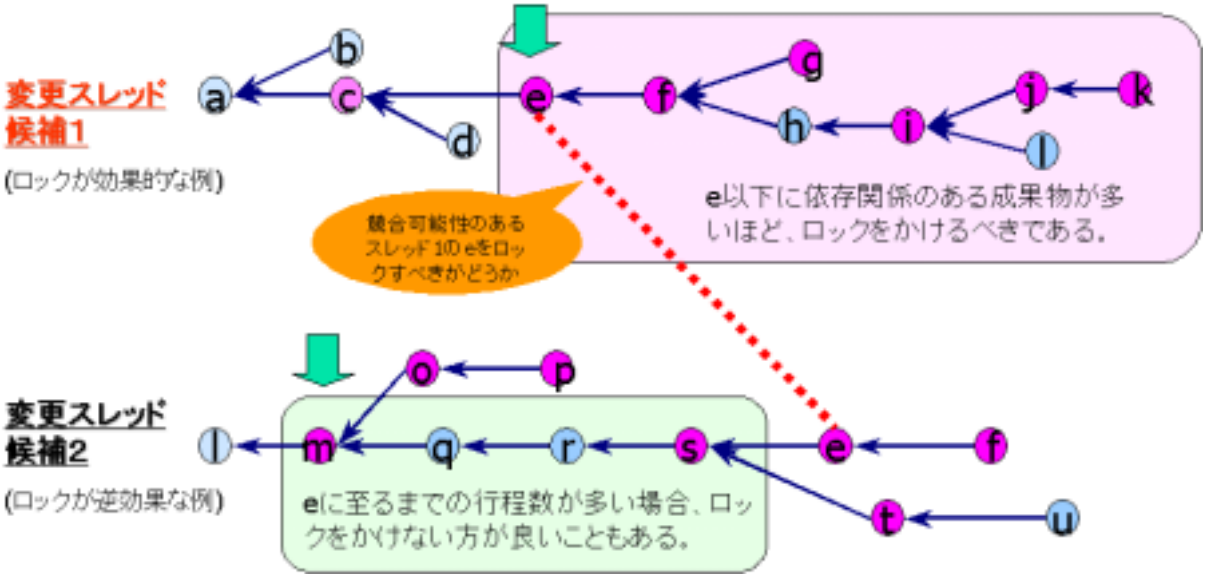


図 4.9: 条件付きロックの概要

4.2.7 デッドロック解消/回避方法

4.2.4 節の導出規則やロック制御などが要因となり、スレッド間にデッドロックが発生する場合がある。この場合、何らかの手段を使用して、デッドロック状態を解消する必要がある。デッドロックの起こる例を図 4.10 に示す。

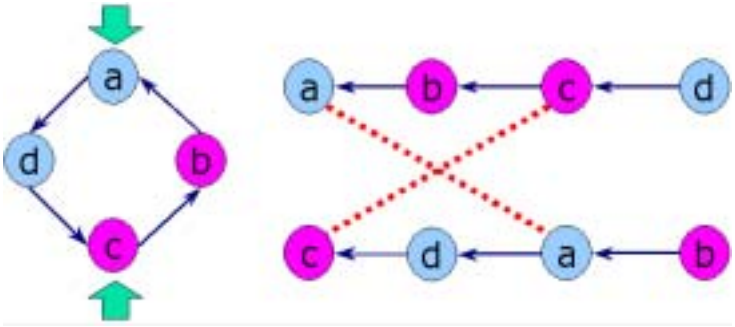


図 4.10: デッドロックの例

全スレッドがデッドロック状態 (全変更作業が停止) になった場合の解消方法と全スレッドがデッドロック状態にならないようにするための回避方法について述べる。

デッドロック解消方法

デッドロックの解消方法としては、成果物の優先順位を用いる方法や、スレッドの優先順位を用いる方法など、様々な手法が考えられる。本論文では、全てのスレッドのロックを解除するというシンプルな方法を用いる。

デッドロック回避方法

デッドロックには管理している全スレッドがデッドロック状態になる場合と、ある部分的なスレッド間にだけがデッドロック状態になる場合が考えられる。本来ならば、部分的なデッドロックにだけ解消方法を用いるべきであろうが、それを発見するのは容易ではないため、全スレッド中のロックするスレッドの割合を閾値とし、それを越えた場合、停滞状態にあるものとし全スレッドのロックを解除することで、全スレッドがデッドロック状態になることを回避するものとする。

4.3 検索空間

開発者全員が成果物に関する知識をすべて持つことは困難である。開発者は成果物の変更に必要な情報を得るために、多大な時間と労力を費やす。本研究では変更管理の補助的な要素として、利用者の要求を満たす自由な視点 (XML タグの部分集合を軸とする検索空間) を提供する。検索空間から得られる、成果物に関する種々の情報により、成果物の修正変更作業を支援する。

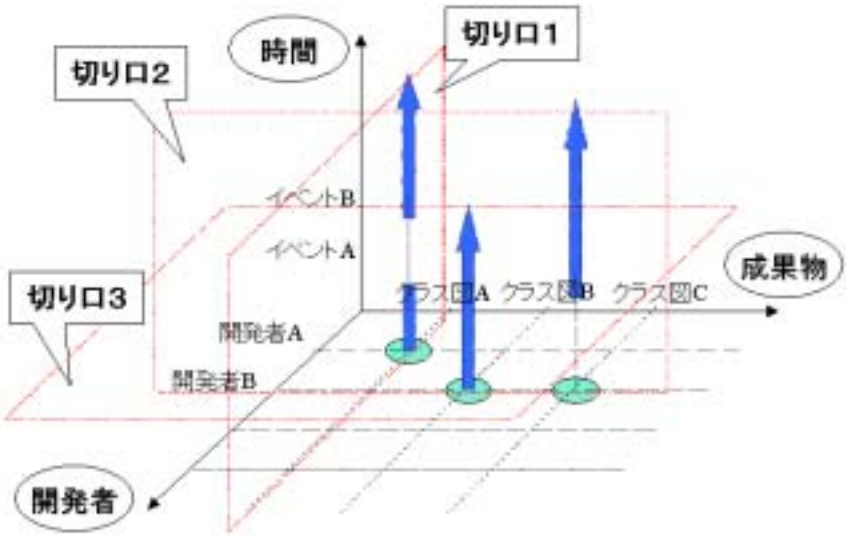


図 4.11: 多次元のタグ軸とする検索空間

図 4.11 に検索空間の一例を示す。太い矢印は成果物が役割によって時間軸方向に開発されていることを表している。管理モデルは複数の軸を組み合わせることにより、様々な視点 (切り出し) を提供する。

- 切り出し 1 (成果物軸)：クラス図 A に関わった開発者やクラス図 A の作成過程を読み取ることが可能。
- 切り出し 2 (開発者軸)：開発者 B がどのクラス図に関わり、いつ活動したのかを読み取ることが可能。
- 切り出し 3 (時間軸)：イベント B の時、どの開発者がどのクラス図に関わっていたかを読み取ることが可能。

本研究で提案する管理モデルは、プロセスモデルを取り入れたことで、多次元の検索空間から検索可能な、検索軸の要素またはそれらの関係以外に、成果物からモデルタグをトレースすることにより、その成果物に関するプロセスを含む背景情報を参照することが可能である。

4.4 視覚化表示

多次元のタグの組み合わせと共に、検索結果を視覚化表示する方法も重要である。視覚化することにより、微妙な状態の変化を読み取ることも可能である。また、検索結果を成果物として管理することも可能である。特定のタグのパターンを視覚化した例を図 4.12 に示す。

図 4.12: 視覚化表示の一例

図 4.12 は、タグの組み合わせ (< 作成終了時刻 >+< ワークフロー >+< 成果物名 >+< 版 No>) から導出した開発プロセスの流れを視覚化したものである。この図から、ある成果物はどの成果物を経由して作成されたのか等の絶対時間における成果物の推移を読み取ることが可能である。また、繰り返し変更が行われている個所などから、問題点が存在する可能性を読み取ることも可能である。

第5章　管理システムのプロトタイプ

第4章で述べた成果物の管理手法の基礎的な部分を、管理システムのプロトタイプとして実現する。実際に変更スレッド候補を制御し、成果物の変更を支援するシステムとして機能する。

5.1　システムの構成

管理システムは、変更スレッド候補をXML文書として表現したものを、XMLパーサが提供する機能のサブセットであるDOMを使用して、動的に制御することにより実現する。プログラム言語にはJavaを用い、XMLパーサにはJAXP(Java API for XML Processing)、XML4J(XML Parser for Java)を用いた。プログラムコードは、2500行程度である。システムは図5.1に示すクラス群によって構成される。

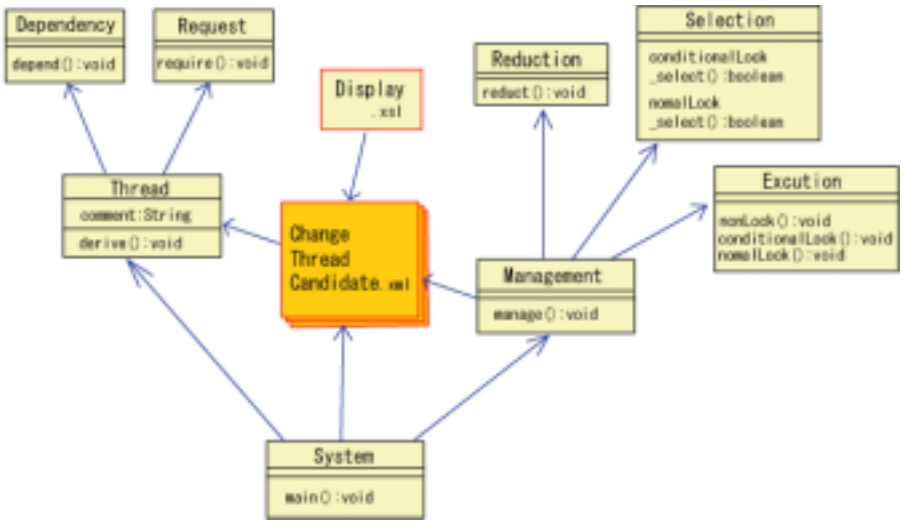


図 5.1: 管理システムのクラス図

4.2.4節で述べたように、変更スレッド候補は構造が木構造をしていることから、XML文書の階層構造を使用することで、表現可能である。よって、(複数の)変更スレッド候補は、(複数)XML文書として扱う。

システムを構成する主な機能を以下に概説する。

Thread クラス

変更スレッド候補の導出 (4.2.4 節の導出規則を使用) を行うクラス
変更成果物を起点に、成果物の依存情報をトレースすることにより、変更スレッド候補を導出する。

Reduction クラス

変更スレッド候補の合成/吸収 (4.2.5 節の編集規則を使用)・妥当性検証を行うクラス
変更スレッド候補 (xml ファイル) を読み込み、空白文字の除去や DTD を使用した妥当性の検証、またスレッドの合成/吸収といった編集作業を行う。

Selection クラス

変更スレッドのデッドロック/停滞検知を行うクラス
各スレッドの全体における状態 (競合成果物の位置やその前後関係などを要素とする各スレッドの状態と、競合成果物に対する他スレッドの状態) を調べる。

Excution クラス

変更スレッド候補の進行/分解 (4.2.5 節の編集規則を使用) を行うクラス
変更スレッド候補自体の状態更新する。

Management クラス

全変更スレッド候補を管理するクラス
Reduction クラス、Selection クラス、Excution クラスを使用することにより、全変更スレッド候補を統括的に制御する。

System クラス

全てのクラスを管理するクラス
クラス全体を統括的に制御するクラスであり、システム本体である。今後の管理機能拡張のため、拡張性の高いものとする。

※変更スレッド候補 (XML)

実体は XML の文書であるが、DOM に Java の属性にマッピング可能なことからクラスとして扱える。また XML 文書であることから、XML の利点や他の関連技術を使用可能。

※視覚化表示 (XSL)

XSL を使用して XML 文書である変更スレッド候補をブラウザに表示する。

5.2 管理するデータ

変更スレッド候補の一例 (XML 文書) を図 5.2 に示す。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE ctc SYSTEM "./../ctc.dtd">
<ctc>
  <cca>art3
    <cca>art16
      <cca>art11
    <cca>art30
      <cca>art6
        <cca>art17</cca>
      </cca>
      <cca>art29
        <cca>art15
          <cca>art27</cca>
        </cca>
      </cca>
    </cca>
      </cca>
      <cca>art13</cca>
      <cca>art20</cca>
    </cca>
  </cca>
</ctc>
<!--art16-->
<!--art13-->
<!--art19-->
```

図 5.2: 変更スレッド候補の一例 (ctc.xml)

変更スレッド候補は DTD により、データ構造が定義されている。図 5.3 に変更スレッド候補の DTD を示す。

```
<!-- CTC Markup Language Document Type Definition -->
<!ELEMENT ctc (cca)>
<!ELEMENT cca (#PCDATA | cca)*>
```

図 5.3: 変更スレッド候補の DTD(ctc.dtd)

図 5.2 の XML 文書に、XSL を使用してブラウザ表示させた、変更スレッド候補を図 5.4 に示す。

図 5.4: 変更スレッド候補のブラウザ表示

成果物にはリンクが張られ、そこから成果物情報、分散作業空間情報、役割情報、コミュニケーション・パス情報が得られる。さらに、共有成果物に関しては、関連する役割間のコミュニケーション・パスが得られる。関連する役割とコミュニケーション・パスの提示の一例を図 5.5 に示す。

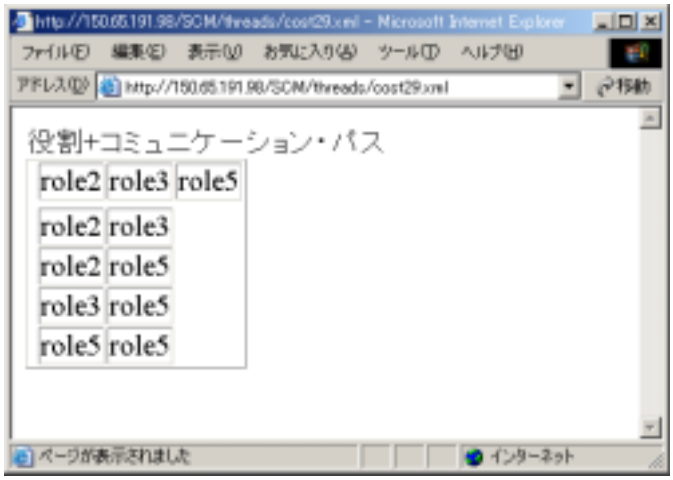


図 5.5: 関連する役割とコミュニケーション・パスの提示

変更コストを算出する手段として関連する役割とコミュニケーション・パスは有効である。

第6章　シミュレーションによる管理システムの評価

第5章でプロトタイプとして実現した管理システムの有効性の評価を行う。評価方法として、一般性を持たせるため、実際の運用に必要と思われる要素の値を、考えられるいくつかのパターンによって、変化させながら、行程数、更新回数を求めるシュミレーションによる評価を用いた。

6.1　実験概要

第5章で作成した管理システムのプロトタイプを、様々な場合において(シミュレーションによるパラメータ値を変化させて)稼働させ、管理システム(手法)の長所短所、問題点を洗い出し、最適な管理システムの使用法を考えるとともに、管理システム全体を評価することを目的とする。

仮想的に成果物を生成させ、成果物間に依存関係を張り、構造化された成果物の集合において、変更要求を発生させる。成果物の変更要求に対し、変更スレッド候補を導出し、システムがそれらのスレッドを統括的に管理する。変更作業はシュミレーション時間で進行するものとし、システムがスレッドの進行/停止を制御する。システムの制御方法と、行程数、変更回数の関係を検討しつつ、以下に示す関係を中心に管理システム(手法)の評価実験を行った。

- 停滞/デッドロック判定基準
- 条件付きロックの閾値とその最適解
- 通常ロックと条件付きロックの比較
- 管理システムの有効性

また、本実験に用いたパラメータを以下に列挙する。

- 成果物数 (10個 〜 50個)

- 変更要求数 (2 個 〜 20 個)
- 依存関係の強さ (弱、 中、強)
- ロック制御方法 (通常ロック、条件付きロック)
  - 条件付きロックの閾値 ($\times 5$ 〜 1 〜 $\div 5$)
- 停滞/デッドロックの閾値 (0% 〜 100%)

6.2 実験手順

成果物数、変更要求数、依存関係の強さ等のパラメータを変更しながら、(最短) 行程数重視、(最低) 更新回数重視、(行程数と更新回数の) バランス重視、各ロック制御方式の場合についてデータを収集する。

本実験に使用するシュミレータについて、以下に述べる。

1. 仮想的に成果物を生成し、それらの成果物間に依存関係を設定する。(依存関係はランダム値を用いる)
2. 管理システムに対して、変更要求を一定の数だけ発生させる。(変更要求はランダム値を用いる)
3. (管理システムはそれぞれの変更要求に対し、4.2.4 節の導出規則に従い変更スレッド候補を導出する。)
4. (管スレッド間のロック制御や編集作業を行うことで、スレッドの進行を管理する。)
5. 変更作業はシュミレーション時間で進め、作業完了と同時にシステムに通知するものとする。(全成果物ともに、変更コストは 1 とする。)
6. (システムにより、変更スレッド候補は更新される。)(4 に戻る...)

依存は各成果物に対して 1 個または 2 個用意し、依存関係の強さに関しては、各変更スレッド候補導出の際に、変更の行われない成果物をランダムに設定する。ロック制御方法としては、4.2.6 節で述べた「通常ロック」と「条件付きロック」を使用する。「条件付きロック」の条件の要素となる変更コストについては、単に成果物数として考える。

以上の手順で評価実験を行う。

6.3　実験結果

シュミレーションにより得られた実験データについては、付録に掲載した。以下、得られたデータを基に評価を行った結果を報告する。

6.3.1 停滞/デッドロック判定基準

図 6.1 に、停滞率の閾値 (%) を変化させた場合の更新回数と行程数の関係を示す (付録 A.1,A.3 参照)。なお、停滞/デッドロックと判定された場合は、全スレッドのロックは解除されるものとする。

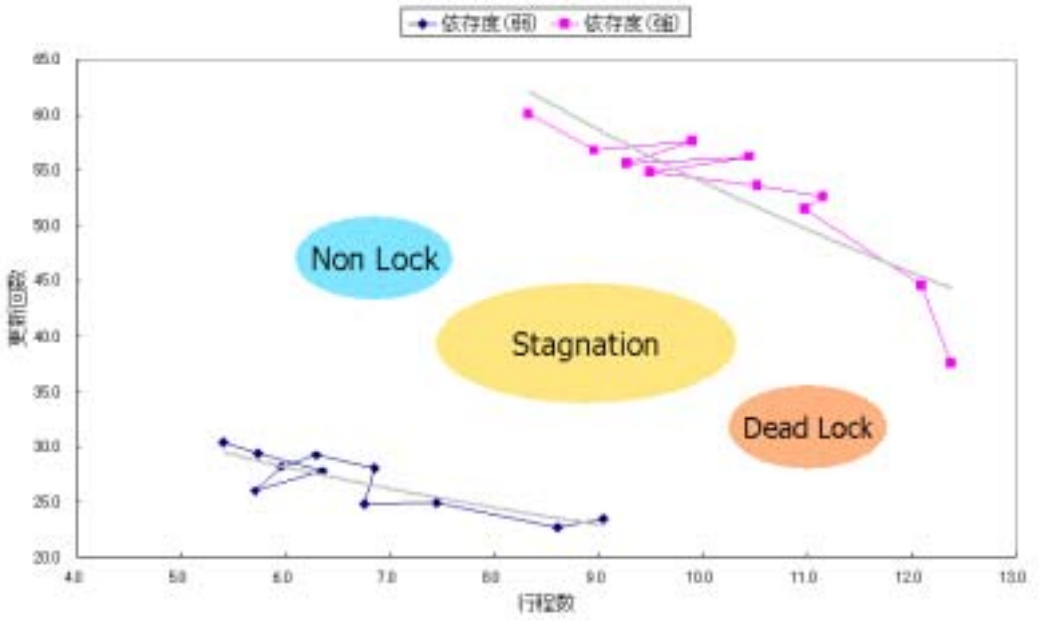


図 6.1: 全ロック解除の判定基準

- 依存関係の依存度が強い場合 (ロックを全解除する閾値を変化)
  - 成果物数/変更要求数：固定
  - ロック制御方式：通常ロック
- 依存関係の依存度が弱い場合 (ロックを全解除する閾値を変化)
  - 成果物数/変更要求数：固定
  - ロック制御方式：通常ロック

依存関係の依存度が弱い場合

更新回数があまり減少しないのに対し、行程数が増加することから、ロック制御を行わない方が良いという結果が得られた。

依存関係の依存度が強い場合

更新回数の減少に反比例して、行程数が増加することから、ロック制御を行うこと

により、「更新回数を優先するか or 行程数を優先するか or それらの中間をとるか」、状況に応じての選択が可能である。

以上のことから、極端に依存関係の依存度が低い場合に対しては、システムは意味をなさないが、ある程度の依存度がある場合に対しては、状況に応じてのシステムの使用が可能であるといえる。

6.3.2 条件付きロックの閾値とその最適解

図6.1に、条件付きロックの閾値を変化させた場合の更新回数と行程数の関係を示す(付録A.4参照)。

図 6.2: 条件付きロックの最適解

- 条件付きロックの閾値を変化(成果物数を変更コストとして計算)。
  - 依存関係の依存度：中
  - 成果物数/変更要求数：固定
  - ロック制御方式：条件付きロック

行程数

8行程〜9行程の間にあり、あまり変化がみられない。

更新回数

33回〜37回の間にあり、多少の変化がみられる。

条件付きロックの閾値を変化させた場合、行程数にあまり変化がないのに対し、更新回数は閾値1を最低更新回数とする変化がみられた。行程数を一定とみなすと、変更作業にかかるコストは更新回数に比例する。よって、閾値1の最低更新回数を中心に増加の傾

向が見られる。この結果から、閾値の最適解は、閾値 1 周辺であることが分かる。このことから、「自分自身のスレッドにおける競合可能性のある成果物に依存する成果物の変更コスト」と「他スレッドにおける競合可能性のある成果物を変更するまでの変更コスト」(図 4.9 参照) のバランスは 1:1 の時が最適であることが分かる。

6.3.3 通常ロックと条件付きロックの比較

通常ロックと条件付きロックにおける、停滞/デッドロックの閾値を変化させた場合の、更新回数と行程数の関係を図 6.3 に示す。(付録 A.1,A.3 参照)。

通常ロック・条件付きロックに対して、依存関係の依存度が、弱い場合・強い場合で実験を行った。

- 通常ロック (ロックを全解除する閾値を変化 0%〜100%)
  - 成果物数/変更要求数：固定
  - 依存度：弱、強
- 条件付きロック (ロックを全解除する閾値を変化 0%〜100%)
  - 成果物数/変更要求数：固定
  - 依存度：弱、強
  - 条件付きロックの閾値：1

通常ロックの場合

依存度が弱、強、両方の場合において、閾値が 0%〜40%、60%〜100%の間で、条件付きロックより有効である。さらに依存度が弱い場合より、依存度が強い場合の方がより有効である。

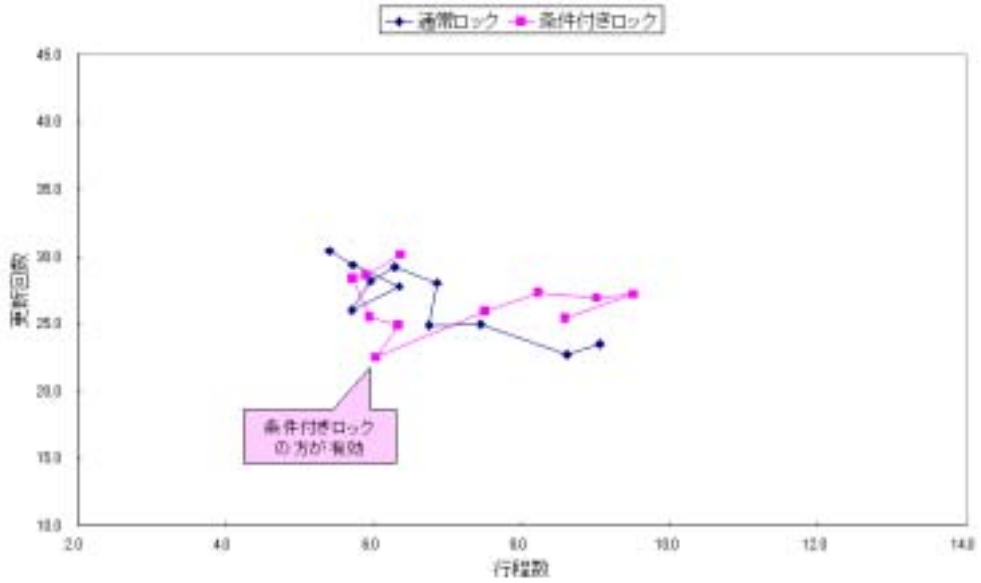
条件付きロックの場合

依存度が弱、強、両方の場合において、閾値が 40%〜60%の範囲で、通常ロックより有効である。

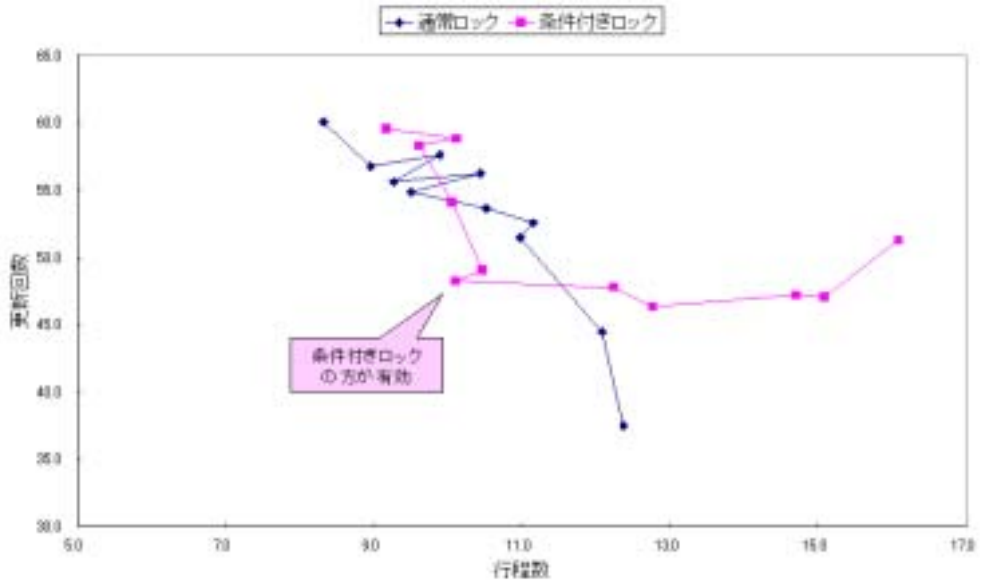
※ 注意：全ロック解除の閾値が 100%に近い場合 (行程数を重視する場合) においては、部分的なデッドロックが通常ロックに完全に劣っているどころか、ロック制御を用いない方が良い可能性がある。

依存度が弱い、強い、どちらの場合においても同様の結果が得られたことから、「変更回数と行程数をバランス (中間) を重視する状況」においては、通常ロックよりも依存条件付きロックの方が有効であるといえる。逆に「更新回数を重視する状況」または 「行程数を重視する状況」においては、条件付きロックよりも通常ロックの方が有効であるといえる。

なお、依存度が弱いよりも、強い方がロック制御 (ロック制御方式を適時選択) による効果は現れることが分かる。



(依存度が弱い場合)



(依存度が強い場合)

図 6.3: 通常ロックと条件付きロックの比較

6.3.4 管理システムの評価

成果物数と、変更要求数を変化させた場合の更新回数と行程数の関係を図 6.4, 図 6.5 に示す (付録 A.5,A.6 参照)。

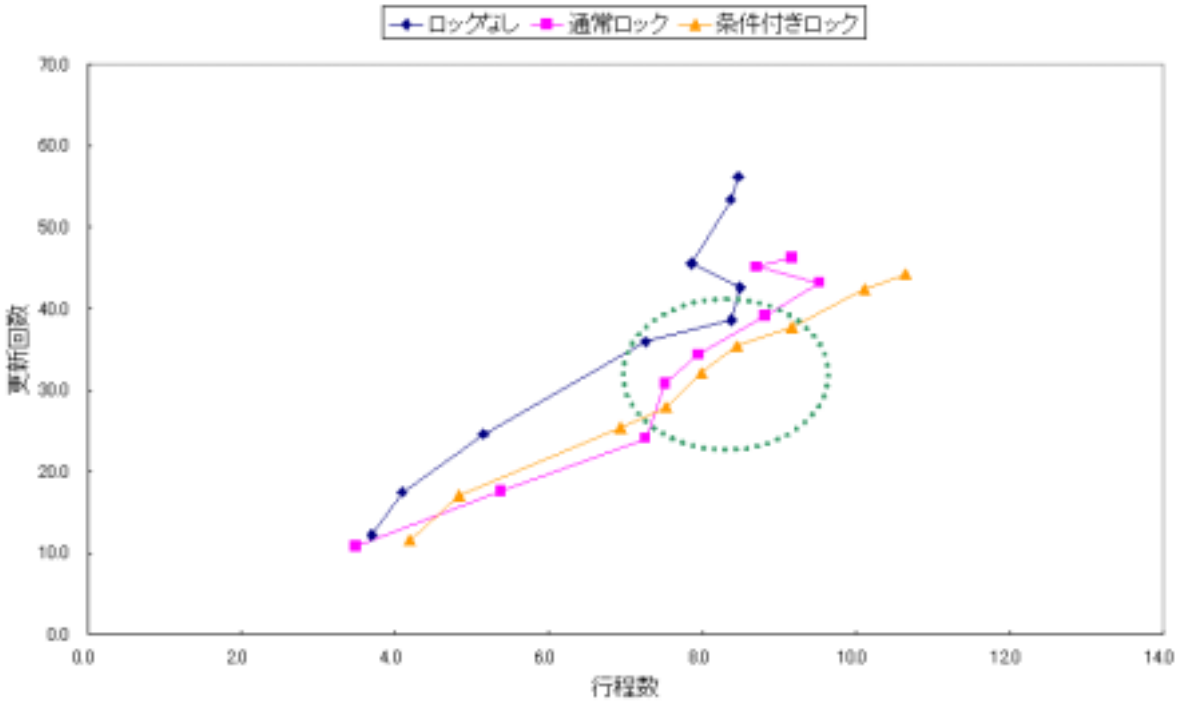


図 6.4: (成果物数に対する) システムの評価

- **成果物数を変化させた場合**
  - 依存関係の依存度：中
  - 変更要求数：固定
  - ロック制御方式：ロックなし、通常ロック、条件付きロック
- **変更要求数を変化させた場合**
  - 依存関係の依存度：中
  - 成果物数：固定
  - ロック制御方式：ロックなし、通常ロック、条件付きロック

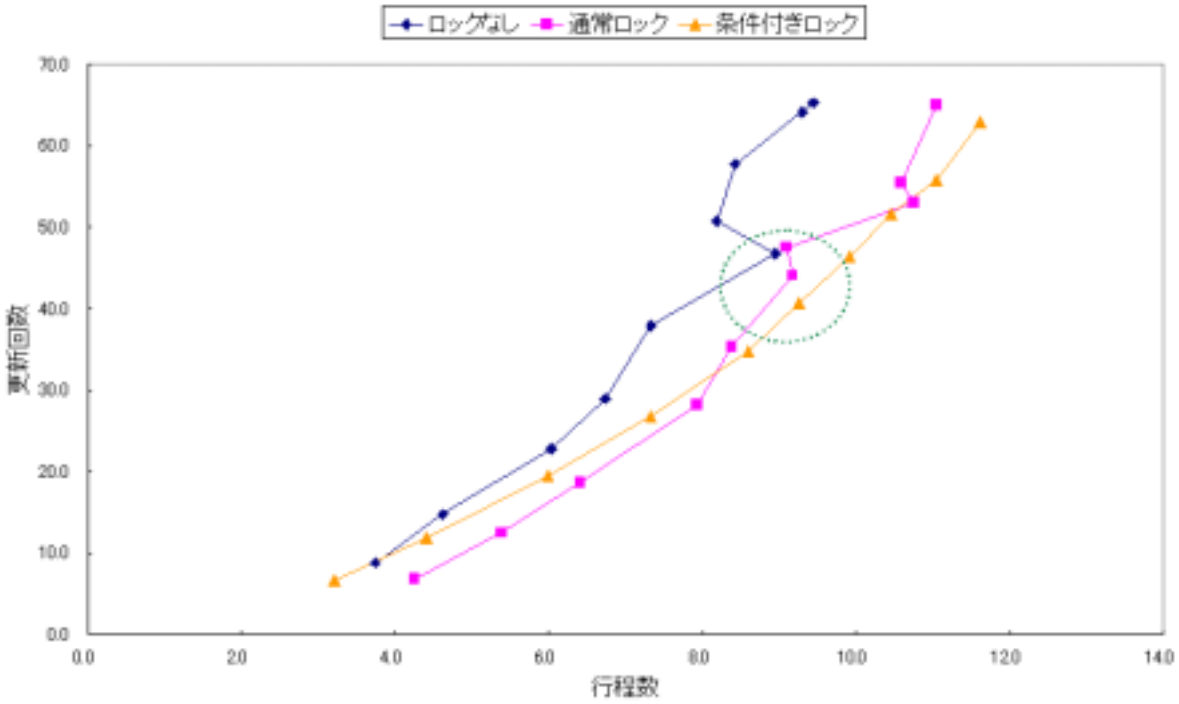


図 6.5: (変更要求数に対する) システムの評価

成果物数を変化させた場合

成果物数と変更要求数のバランスが重要であるため、例えば成果物数が極端に少ない場合に変更要求数が 10 も発生すると、ほぼ停滞/デッドロック状態に陥ることから、成果物数が極端に少なすぎる場合においては、システムを使用しない方がよいと考えられる。

変更要求数を変化させた場合

同様に、成果物数と変更要求数のバランスは重要であり、成果物数に対して変更要求が多すぎると、ほぼ停滞/デッドロック状態に陥ることから、成果物数に対して変更要求が多すぎる場合においては、システムを使用しない方が良いと考えられる。

両方の場合から、一般的な成果物数と変更要求数の場合において、システムは有効であると推測される。6.3.3 節で述べたように、行程数、更新回数のどちらかを重視する状況においては、条件付きロックよりも通常ロックを、行程数と更新回数のバランスを重視する状況においては、通常ロックよりも条件付きロックを選択することにより、シテムはより効果的なものになる。また、依存関係の依存度が強い場合においては、さらにシステムの有効性は高くなると考えられる。

6.4 考察

今回行った評価実験の結果から、成果物数と変更要求数のバランスが極端でない場合において、システムは有効であることが分かった。なお、システムは依存関係の依存度が強いほど、システムの有効的は高くなるといえる。

行程数や更新回数のどちらか一方を重視する状況、両方のバランスを重視する状況など、利用者に要求する状況に応じたシステムの利用が可能である。また、条件付きロックは、行程数と更新回数のバランスを重視する状況において、最も有効であることが分かったが、成果物の変更コストを基にロックの有無を決定するものであるため、変更コストの求め方によっては、結果が変化する可能性がある。

パラメータによって、結果が多少変化する可能性はあるが、状況にあった閾値を選択することができれば、管理システムの使用は効果的であるといえる。実際には状況を把握することや、実際のソフトウェア開発工程を管理モデルに対応した形で、管理システムにパラメータ値を入力することが困難であるかもしれないが、変更コストを見積もることだけに使用したとしても、ある程度の変更に有益な情報を得ることは可能である。

管理システムは、ある成果物の変更により起る可能のある様々な事象や、それに伴う成果物の情報、また変更スレッドの進行に関する制御といった、様々な変更に有益な情報 (可能性) を提示可能である。

管理システムが提示する情報 (可能性) を以下にまとめる。

- 変更の及ぶ成果物
- 変更の及ぶ役割
- 役割間のコミュニケーションの頻度
- 変更順序 (スケジュール)
- 検索空間
- 表示方法

変更スレッド候補を使用した管理システムは、それだけで変更に関する全ての管理を行うシステムではなく、管理システムが提供する成果物に関する様々な情報を基に共同ソフトウェア開発を支援するというものである。管理システムが提供する情報は絶対的なものではないが、ソフトウェア開発における、開発者の要求を満たす、様々なの可能性の提示という観点から考察すると、十分に効果的であると思われる。

第7章　おわりに

7.1　まとめ

本論文では、共同ソフトウェア開発における開発過程において生成される様々な成果物と、それらの構成に関する情報について検討し、反復作業や再利用の面での変更の効率を上げる有効な方法として、チーム構造モデルを基にした、成果物を変更するための情報を管理できる管理モデルと変更スレッド候補を提案した。

また、管理モデルを基に検索空間、表示方法、変更スレッド候補を使用した管理手法を提案した。さらに、管理手法を基に管理システムのプロトタイプを作成し、管理システムのシュミレーションによる評価実験を実施し、その結果を報告した。

本研究の成果を以下にまとめる。

- 変更管理
  - 管理モデルと変更スレッド候補を使用することで、変更に費やすおおよそのコストを推測することが可能
  - 変更スレッド候補を用いることで、おおよその成果物の変更に伴う波及効果を推測することが可能
  - 変更スレッド候補を管理することにより、動的な変更作業の進行状況を把握可能
  - 変更スレッド候補を操作することにより、その場の状況に適応した変更作業を最適な方法で行うことが可能
- 検索空間
  - 管理モデルは複数の軸を組み合わせることにより、様々な視点 (切り出し) を提供することが可能
  - 本研究で提案する管理モデルにより、プロセスモデルを取り入れたことで、多次元の検索空間から検索可能な、検索軸の要素またはそれらの関係以外に、成果物から、モデルのタグをトレースすることにより、その成果物に関するプロセスを含む背景情報を参照することが可能

- 表示方法
  - タグを多次元に組み合わせることによって、より複雑な情報を検索することが可能である。さらに、表示方法を工夫することにより、検索だけでは分かりにくい状況の把握も可能になり、役割に応じた検索パターンを用意しておくことも有効であると考えられる

変更管理を中心に、検索方法、表示方法について以上の知見が得られた。

7.2 今後の課題

今後の課題は以下のとおりである。

- コミュニケーション頻度を変更コストとして考慮に入れた評価や、実際の共同ソフトウェア開発の現場における評価を行う。
- 部分的なデッドロックについてのみ解消方法を適応するために、部分的なデッドロックを検知する方法を検討する。
- 現在のシステムをより実用的なものにするために、版管理を考慮したシステムを検討する。
- 変更スレッド候補上の成果物ごとに、変更に有する時間がある程度予測できれば、その予測を基にスレッドを管理することが可能である。よって、変更に有する時間をある程度予測可能なタグ、またはその組み合わせについて検討する。

筆者は、変更スレッド候補が、変更の波及効果をすべてを含んだものであったり、絶対的な変更順序を示すものであると考えるのは幻想であると思うし、むやみやたらに依存関係などの付加情報を増やせばよいという考え方には反対する。当面、最低限の単純な要素の組み合わせとトレース方法を考えることが、文書の構成管理に関しては、もっとも建設的であると考える。それと共に、管理モデルに関する研究を押し進める必要がある。

謝辞

本研究を行なうにあたり、多大な御助言、御指導を賜わりました北陸先端科学技術大学院大学落水浩一郎教授に深く感謝の意を表します。

日頃から研究に協力していただき、大変有益な御助言、御指導をいただきました落水研究室猪俣敦夫氏、情報科学センター助手藤枝和宏博士に深く感謝致します。

本研究を進めるにあたり、種々の有益な御助言をいただきました、落水研究室助手服部哲博士、村越広享博士をはじめ、研究を行う上で非常に役立つ御意見、御感想を寄せていただいた研究室の皆様に深く感謝致します。

本論文の審査委員として加わっていただきました本学の片山卓也教授、篠田陽一教授には、本論文を完成させるにあたり、論文査読において、種々の有益な御助言をいただきました。深く感謝致します。

最後に、 学業だけでなく、私生活の面からもサポートしてくれた家族、友人に深く感謝します。

参考文献

[1] Object Managemant Group : “Unified Modeling Language (UML) 1.4”. http://www.omg.org/technology/documents/formal/uml.htm

[2] Grady Booch : “UML ユーザーズガイド”, ピアソン, 1999.

[3] Ivar Jacobson, Grady Booch, Jamess Rumbaugh : “Unified Software Development Process”, Addison Wesley Publishing Company, 1999.

[4] Jim Conallen, “Building Web Applications with UML”, Addison-Wesley, 1999.

[5] World Wide Web Consortium : “HTML 4.01 Specification”. http://www.w3.org/TR/html4/

[6] World Wide Web Consortium : “Extensible Markup Language (XML) 1.0)”. http://www.w3.org/TR/REC-xml

[7] World Wide Web Consortium : “Extensible Stylesheet Language (XSL) 1.0)”. http://www.w3.org/TR/xsl

[8] World Wide Web Consortium : “Scalable Vector Graphics (SVG) 1.0 Specification)”. http://www.w3.org/TR/SVG

[9] World Wide Web Consortium : “Simple Object Access Protocol (SOAP) 1.1)”. http://www.w3.org/TR/SOAP/

[10] “XML.ORG”. http://www.xml.org/

[11] Lois Wakeman, Jonathan Jowett : “PCTE: the Standard for Open Repositories”. Prentice Hall 1993.

[12] 丸山 宏, 田村 健人, 浦本 直彦 : “XML と Java による Web アプリケーション開発”, ピアソン, 1999.

[13] 浅海 智晴 : “XML/DOM Programming”, 秀和システム, 2001.

[14] World Wide Web Consortium : “Scalable Vector Graphics (SVG) 1.0 Specification”. http://www.w3.org/TR/SVG/

[15] 落水 浩一郎 : “ソフトウェア分散共同開発のためのチーム構造モデルの定義と調整支援への応用”, 情報処理学会, ソフトウェア工学研究会, SE-131, pp.17-24, 2001.

[16] 昆 雅士 : “研究環境におけるリソースサーバ labdb の設計と実現”, 北陸先端科学技術大学院大学 情報科学研究科 修士論文 1994.

[17] J.Coplian : “A Development Process Generative Patterrn Language, Pattern Languages of Program Design, 1995.

[18] Martin Fawler : “Analysis Patterns: Reusable Object Models”, Addison Wesley Publishing Company, 1996.

[19] Erich Gamma, Richard Helm, Ralph Johnson, John Vlssides : “Design Patterns”, Addison Wesley Publishing Company, 1995.

[20] Volker Gruhn, juri Urbainczyk, Software Process Modeling and Enactment: An Experience Report ralated to Problem Tracking in an Industrial Project, IEEE 1998.

[21] Mikio Aoyama : “Agile Software Process and Its Experience”, IEEE 1998.

付 録A　実験データ

実験データ 1 : 停滞/デッドロックの閾値を変化 (依存度:弱)

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
30	10	15	なし	−	−	30.4	5.4	0.0	6.1
30	10	15	通常	−	1	29.4	5.7	0.0	5.0
30	10	15	通常	−	2	27.8	6.4	0.0	3.4
30	10	15	通常	−	3	26.0	5.7	0.4	3.2
30	10	15	通常	−	4	28.2	6.0	1.1	3.2
30	10	15	通常	−	5	29.2	6.3	1.1	3.5
30	10	15	通常	−	6	28.0	6.9	2.7	2.7
30	10	15	通常	−	7	24.9	6.8	5.3	1.9
30	10	15	通常	−	8	24.9	7.5	10.0	1.7
30	10	15	通常	−	9	22.7	8.6	20.1	1.2
30	10	15	通常	−	DeadLock	23.5	9.1	23.2	1.2
30	10	15	なし	−	−	30.2	6.4	0.0	7.5
30	10	15	条件	1	1	28.6	5.9	0.0	2.9
30	10	15	条件	1	2	28.4	5.7	0.5	2.3
30	10	15	条件	1	3	25.4	6.0	1.6	1.9
30	10	15	条件	1	4	24.9	6.3	3.0	1.7
30	10	15	条件	1	5	22.5	6.0	3.5	1.4
30	10	15	条件	1	6	25.9	7.5	6.7	1.4
30	10	15	条件	1	7	27.3	8.2	9.1	1.5
30	10	15	条件	1	8	26.9	9.0	9.7	1.2
30	10	15	条件	1	9	27.1	9.5	13.2	1.3
30	10	15	条件	1	DeadLock	25.4	8.6	11.0	1.0

表 A.1: 停滞/デッドロックの閾値 (依存度:弱)

- ロック全解除の閾値を変更 (0%〜100%)
  - 依存関係の依存度 : 弱
  - 成果物数 : 固定
  - 変更要求数 : 固定
  - ロック制御方式 :
    - 通常ロック
    - 条件付きロック (条件の閾値:1)

実験データ 2 : 停滞/デッドロックの閾値 (依存度:中)

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
30	10	10	なし	–	–	36.8	6.9	0.0	7.3
30	10	10	通常	–	1	37.8	6.6	0.0	5.6
30	10	10	通常	–	2	40.2	7.5	0.1	5.3
30	10	10	通常	–	3	37.0	8.3	0.4	4.4
30	10	10	通常	–	4	35.9	7.1	1.3	4.0
30	10	10	通常	–	5	38.3	7.8	2.1	4.6
30	10	10	通常	–	6	36.3	7.8	5.3	3.2
30	10	10	通常	–	7	34.5	7.9	8.7	2.6
30	10	10	通常	–	8	30.3	8.3	13.2	1.8
30	10	10	通常	–	9	33.0	10.6	28.3	1.9
30	10	10	通常	–	DeadLock	32.4	11.2	27.1	1.9
30	10	10	なし	–	–	36.0	6.8	0.0	6.8
30	10	10	条件	1	1	37.0	7.5	0.0	3.4
30	10	10	条件	1	2	34.1	7.0	0.7	2.8
30	10	10	条件	1	3	32.5	7.1	2.0	2.0
30	10	10	条件	1	4	33.1	7.5	3.7	2.1
30	10	10	条件	1	5	32.5	8.5	5.2	1.9
30	10	10	条件	1	6	36.2	9.5	9.0	1.8
30	10	10	条件	1	7	31.9	9.5	10.0	1.4
30	10	10	条件	1	8	35.1	11.5	16.8	1.5
30	10	10	条件	1	9	31.8	11.0	15.8	1.5
30	10	10	条件	1	DeadLock	32.6	11.2	15.9	1.4

表 A.2: 停滞/デッドロックの閾値 (依存度:中)

- ロック全解除の閾値を変更 (0%〜100%)
  - 依存関係の依存度 : 中
  - 成果物数 : 固定
  - 変更要求数 : 固定
  - ロック制御方式 :
    - 通常ロック
    - 条件付きロック (条件の閾値:1)

実験データ 3 : 停滞/デッドロックの閾値 (依存度:強)

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
30	10	5	なし	−	−	60.1	8.3	0.0	10.3
30	10	5	通常	−	1	56.8	9.0	0.0	6.4
30	10	5	通常	−	2	57.7	9.9	0.9	8.5
30	10	5	通常	−	3	55.6	9.3	0.7	6.3
30	10	5	通常	−	4	56.2	10.5	1.6	7.4
30	10	5	通常	−	5	54.9	9.5	3.2	5.4
30	10	5	通常	−	6	53.6	10.5	8.0	4.4
30	10	5	通常	−	7	52.6	11.2	11.9	3.6
30	10	5	通常	−	8	51.4	11.0	16.4	3.6
30	10	5	通常	−	9	44.5	12.1	37.2	2.2
30	10	5	通常	−	DeadLock	37.5	12.4	38.8	1.8
30	10	5	なし	−	−	59.5	9.2	0.0	11.2
30	10	5	条件	1	1	58.8	10.1	0.1	5.0
30	10	5	条件	1	2	58.3	9.6	1.3	4.5
30	10	5	条件	1	3	54.1	10.1	3.5	3.2
30	10	5	条件	1	4	49.0	10.5	6.4	2.4
30	10	5	条件	1	5	48.2	10.1	8.4	2.2
30	10	5	条件	1	6	47.8	12.2	11.7	1.9
30	10	5	条件	1	7	46.3	12.8	15.6	1.9
30	10	5	条件	1	8	47.2	14.7	19.8	1.8
30	10	5	条件	1	9	47.0	15.1	27.3	2.0
30	10	5	条件	1	DeadLock	51.3	16.1	29.1	1.9

表 A.3: 停滞/デッドロックの閾値 (依存度:強)

- ロック全解除の閾値を変更 (0%〜100%)
  - 依存関係の依存度 : 強
  - 成果物数 : 固定
  - 変更要求数 : 固定
  - ロック制御方式 :
    - 通常ロック
    - 条件付きロック (条件の閾値:1)

実験データ 4：条件付きロックの閾値

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
30	10	10	条件	×5	6	36.9	8.6	3.6	0.4
30	10	10	条件	×4	6	35.5	8.8	5.3	0.6
30	10	10	条件	×3	6	35.2	9.2	6.5	1.0
30	10	10	条件	×2	6	35.0	9.5	8.2	1.5
30	10	10	条件	1	6	34.2	9.1	8.8	1.6
30	10	10	条件	÷2	6	33.8	9.1	9.7	1.6
30	10	10	条件	÷3	6	35.2	9.1	8.1	2.2
30	10	10	条件	÷4	6	34.5	9.0	9.0	1.9
30	10	10	条件	÷5	6	35.2	9.3	9.8	2.2
30	10	10	条件	×5	5	33.8	8.0	2.9	0.6
30	10	10	条件	×4	5	36.2	8.5	4.5	0.7
30	10	10	条件	×3	5	33.6	8.1	4.9	1.2
30	10	10	条件	×2	5	34.2	8.8	5.2	1.7
30	10	10	条件	1	5	32.9	8.6	5.3	2.0
30	10	10	条件	÷2	5	32.1	8.0	5.3	1.9
30	10	10	条件	÷3	5	38.0	8.1	6.1	2.7
30	10	10	条件	÷4	5	36.7	8.0	5.1	2.5
30	10	10	条件	÷5	5	35.9	7.9	5.4	2.7
30	10	10	条件	×5	4	38.2	9.2	2.5	0.5
30	10	10	条件	×4	4	36.9	8.1	3.7	0.8
30	10	10	条件	×3	4	36.1	8.2	3.6	1.1
30	10	10	条件	×2	4	37.8	8.4	4.6	1.9
30	10	10	条件	1	4	34.1	8.3	3.6	2.2
30	10	10	条件	÷2	4	38.7	8.1	3.7	2.8
30	10	10	条件	÷3	4	36.0	7.9	3.6	2.3
30	10	10	条件	÷4	4	35.6	8.0	2.9	2.5
30	10	10	条件	÷5	4	36.5	8.0	3.7	3.0

表 A.4: 条件付きロックの閾値

- 条件付きロックの閾値を変更 (× 5 〜 1 〜 ÷ 5)
  - 依存関係の依存度：中
  - 成果物数：固定
  - 変更要求数：固定
  - ロック制御方式：条件付きロック (全解除の閾値:4〜6)

実験データ 5 : 成果物数

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
10	10	10	なし	−	−	12.2	3.7	−	−
15	10	10	なし	−	−	17.5	4.1	−	−
20	10	10	なし	−	−	24.6	5.2	−	−
25	10	10	なし	−	−	35.9	7.3	−	−
30	10	10	なし	−	−	38.6	8.4	−	−
35	10	10	なし	−	−	42.6	8.5	−	−
40	10	10	なし	−	−	45.6	7.9	−	−
45	10	10	なし	−	−	53.4	8.4	−	−
50	10	10	なし	−	−	56.2	8.5	−	−
10	10	10	通常	−	7	10.8	3.5	2.3	1.2
15	10	10	通常	−	7	18.6	5.5	4.0	2.0
20	10	10	通常	−	7	24.1	7.3	5.4	2.1
25	10	10	通常	−	7	30.8	7.5	6.5	2.7
30	10	10	通常	−	7	33.3	7.7	8.3	2.8
35	10	10	通常	−	7	39.1	8.9	10.5	3.2
40	10	10	通常	−	7	43.1	9.5	11.8	3.0
45	10	10	通常	−	7	45.1	8.7	12.0	2.5
50	10	10	通常	−	7	46.2	9.2	11.3	2.6
10	10	10	条件	1	5	12.3	4.3	1.5	1.0
15	10	10	条件	1	5	18.7	5.1	3.0	1.5
20	10	10	条件	1	5	25.4	6.9	4.3	1.5
25	10	10	条件	1	5	27.9	7.5	4.5	1.6
30	10	10	条件	1	5	32.2	8.0	5.0	1.7
35	10	10	条件	1	5	36.8	8.8	7.7	2.1
40	10	10	条件	1	5	37.7	9.2	6.5	1.8
45	10	10	条件	1	5	42.3	10.1	6.8	1.4
50	10	10	条件	1	5	44.2	10.7	6.7	1.7

表 A.5: 成果物数

- 成果物数を変更 (10 個〜50 個)
  - 依存関係の依存度 : 中
  - 変更要求数 : 固定
  - ロック制御方式 :
    - ロックなし
    - 通常ロック (全解除の閾値:7)
    - 条件付きロック (条件の閾値:1)(全解除の閾値:5)

実験データ 6 : 変更要求数

成果物数	変更要求数	依存度	ロック制御	条件ロック閾値	停滞度閾値	更新回数	行程数	ロック回数	全解除回数
30	2	10	なし	−	−	8.8	3.7	−	−
30	4	10	なし	−	−	14.7	4.6	−	−
30	6	10	なし	−	−	22.8	6.0	−	−
30	8	10	なし	−	−	29.0	6.4	−	−
30	10	10	なし	−	−	37.9	7.3	−	−
30	12	10	なし	−	−	46.7	9.0	−	−
30	14	10	なし	−	−	48.6	7.9	−	−
30	16	10	なし	−	−	53.3	7.7	−	−
30	18	10	なし	−	−	64.5	9.5	−	−
30	20	10	なし	−	−	64.1	9.4	−	−
30	2	10	通常	−	7	6.9	4.3	1.8	0.2
30	4	10	通常	−	7	12.5	5.4	3.7	0.9
30	6	10	通常	−	7	18.7	6.4	5.6	1.4
30	8	10	通常	−	7	30.1	8.4	7.1	2.5
30	10	10	通常	−	7	35.2	8.4	9.5	2.7
30	12	10	通常	−	7	44.0	9.2	10.4	3.3
30	14	10	通常	−	7	47.6	9.1	9.2	3.6
30	16	10	通常	−	7	53.1	10.8	11.2	3.7
30	18	10	通常	−	7	55.4	10.6	11.8	3.9
30	20	10	通常	−	7	65.0	11.1	12.2	4.4
30	2	10	条件	1	5	6.7	3.2	0.2	0.3
30	4	10	条件	1	5	11.9	4.4	0.8	0.5
30	6	10	条件	1	5	21.2	6.4	2.2	1.3
30	8	10	条件	1	5	26.8	7.3	3.9	1.3
30	10	10	条件	1	5	34.8	8.6	6.2	1.9
30	12	10	条件	1	5	41.3	9.5	8.1	2.4
30	14	10	条件	1	5	47.6	10.2	9.5	2.6
30	16	10	条件	1	5	51.6	10.5	10.3	2.8
30	18	10	条件	1	5	55.8	11.1	12.1	2.8
30	20	10	条件	1	5	62.9	11.6	12.7	2.8

表 A.6: 変更要求数

- 変更要求数を変更 (2 個〜20 個)
  - 依存関係の依存度 : 中
  - 成果物数 : 固定
  - ロック制御方式 :
    - ロックなし
    - 通常ロック (全解除の閾値:7)
    - 条件付きロック (条件の閾値:1)(全解除の閾値:5)