修 士 論 文

オープンソースソフトウェア開発に適した CVS リポジトリの階層型分散構成法の研究

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

嶋田 大輔

平成十四年三月

修 士 論 文

オープンソースソフトウェア開発に適した CVS リポジトリの階層型分散構成法の研究

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　権藤 克彦 助教授

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

学生番号: 010052
嶋田 大輔

2002 年 2 月

要 旨

オープンソースソフトウェアの開発プロジェクトは、目的の異なるサブプロジェクトを持つ場合がある。これらの開発で広く用いられている CVS のリポジトリは、単一構成のため、サブプロジェクトごとに異なる運用方針や管理方針を持つことが難しい。本研究では、この問題を解決する CVS リポジトリの階層型分散構成法を提案する。この構成は、リポジトリ間の関連を維持しながら、リポジトリごとに運用方針と管理方針を設定可能にする。

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 背景

近年、インターネットの普及と計算機資源の発達によって、広域分散環境においてソフトウェアの共同開発が行われている。その１つに、オープンソースソフトウェアの開発がある。その開発では、プロジェクトの規模が大きくなると複数の開発拠点や目的の異なるサブプロジェクトを持つことが多い。これらは、ネットワーク的に遠く離れ、プロジェクトの運営方針が大きく異なっている場合がある。したがって、これらのプロジェクトの成果物を管理するリポジトリは、分散構成を取れることが必要である。

既存のソフトウェア構成管理システムには、リポジトリの分散機能を持つものがある。しかし、これらの分散モデルでは、プロジェクトに応じて柔軟な構成をとることや、プロジェクト固有の運用方針をリポジトリごとに適用できない場合があるため、オープンソースソフトウェア開発の持つ特性を支援することが難しい。

本研究では、オープンソースソフトウェアの開発プロセスに適した特徴を持つリポジトリの階層型分散構成を実現する。この構成では、リポジトリ間の関連を維持しながら、リポジトリごとに運用方針を設定可能にする。本研究の手法を用いることで、サブプロジェクトごとに独立した運用方針を設定可能にし、オープンソースソフトウェアの開発プロセスに適した版管理システムの構成を実現する。

1.2 目的

本研究では、オープンソースソフトウェアの開発プロセスに適した特徴を持つリポジトリの階層型分散構成を実現する。この構成では、リポジトリ間の関連を維持しながら、リポジトリごとに運用方針と管理方針を設定可能にする。

まず、階層型とは、複製元 (親リポジトリ) と複製先 (子リポジトリ) の関係から成り立ち、親リポジトリは、自身の全て、もしくは一部の複製を持つ１階層下の子リポジトリを作成できる。親リポジトリと子リポジトリは、互いに開発の成果を反映可能にするため、一貫性を維持し、さらに必要に応じて関連を設定可能にする。ここで一貫性とは、親–子リポジトリ間の版内容の矛盾を防ぐための２つの作業モデルと、リビジョン番号の矛盾を防ぐためのリビジョン番号の操作により成り立つ。また関連とは、リポジトリ間の一貫性を保持した上で、さらに版の同期や委譲など、リポジトリ間の関連付けを行える。本構成では、これらのプリミティブを組み合せることで、数種類の分散構成を実現できる。

たとえば、リポジトリ間の開発スタイルとして、並行開発や開発の委譲を実現する。並行開発では、子リポジトリごとに異なる運用方針や管理方針を設定可能にすることにより、サブプロジェクトごとに異なる開発プロセスを支援できる。また開発の委譲では、親リポジトリの開発の一部を子リポジトリに委譲し、異なる役割を持ったチームに委譲を行える。

開発プロジェクトによっては、複数のサブプロジェクトを持つ場合や、複数の関連プロジェクトを持つ場合、もしくはサブプロジェクトがさらにサブプロジェクトを持つ場合が考えられる。そこで本構成では、親/子リポジトリの１対１関係を基本とし、それを複数組み合せることにより、複雑な分散構成も取れるようにする。

本研究の手法を用いることで、サブプロジェクトごとに独立した運用方針と管理方針を設定可能にし、オープンソースソフトウェアの開発プロセスに適した版管理システムの構成を実現する。

1.3 論文の構成

本論文の構成は、まず 2 章でオープンソースソフトウェア開発の特徴と、その開発で広く利用されている版管理システム CVS についてまとめる。また、CVS や既存のリポジトリ分散モデルの問題点を指摘し、その開発に適したリポジトリの分散構成を提案する。

3 章では、本研究で提案するリポジトリの階層型分散モデルの概要と、その基本シナリ

オを述べる。さらに4章では、リポジトリ間の一貫性と関連性を維持する機構を詳細に述べる。そして5章では、本モデルの適用例を挙げる。

6章では、本モデルに基づくプロトタイプシステムの設計方針と、各コンポーネントの設計を示し、7章でその実装環境と動作手順を示す。

8章では、本モデルとプロトタイプシステムについて考察を行い、最後に9章でまとめと今後の課題を述べる。

第 2 章

オープンソースソフトウェア開発と版管理システム

本章では、まず研究の背景として、オープンソースソフトウェアの開発と、それらの開発で多く用いられている CVS の特徴をまとめる。次に、オープンソースソフトウェアの開発プロジェクトでは、規模が大きくなると複数のサブプロジェクトを持つ場合があり、CVS のリポジトリ構成では充分に対応できない問題点を挙げる。これを踏まえ、これらの開発に適した特徴を持つリポジトリ構成を提案する。また、既存のソフトウェア構成管理システムのリポジトリ分散モデルに関して、不足している特徴を指摘する。

2.1 オープンソースソフトウェア開発の特徴

オープンソースソフトウェア開発は、数多くのインターネットの基盤ソフトウェアや、Linux や FreeBSD のような広く普及したオペレーティングシステムを生み出してきた。その開発プロセスは、一般的な定義が難しく、プロジェクトごとにさまざまな開発スタイルがある。

以下に、一般的な特徴を挙げる。

- 制約のゆるやかなライセンスでソースコードやドキュメントが配布される
- プロジェクトに関心を持つユーザが開発に参加・貢献する
- ユーザは広域分散し、ソフトウェアの開発能力は不均一である
- 階層的な権限構造を持つ

オープンソースライセンスには、配布は自由、プログラムはソースコードを含まなければならない等、11 つの定義がある。基本的には、ソースコードの独占を禁止し、共有を促す内容である。代表的なオープンソースライセンスには、*GNU General Public License (GNU GPL)*、*BSD* **コピーライト**、*Netscape Public License (NPL)* がある。

開発に参加するユーザは、まずプロジェクトの公開 CVS から、ソースコードやドキュメントを入手する。彼らは、入手したソフトウェアに対して貢献を行い、プロジェクトにそれをフィードバックする。その貢献は、ソフトウェアの使用中に発見したバグやテストレポート、そのコードを修正するバグフィックスや、新しい機能の追加等、さまざまである。プロジェクトのメンバは、これらの中から有益なものを選択し、ソースコードやドキュメントに反映させる。

著名なプロジェクトには、*Apache*[1]、*Mozilla*[2]、*Gnome*[3] 等が挙げられ、*Linux*[4]、*FreeBSD*[5]、*NetBSD*[6] 等の高い安定性や安全性を求められるオペレーティングシステムの開発プロジェクトも同様の形態を取っている。これらのプロジェクトには、一般的に階層的な権限構造があり、その強さによって**コアメンバ** *(core member)*、**メンテナ** *(maintener)*、**コミッタ** *(committer)*、**コントリビュータ** *(contributer)* の役割がある。コアメンバは、プロジェクトの管理と運営を行い、その指針を決定する。メンテナは、リポジトリに管理されている成果物 (ソースコード、ドキュメント等) の管理を行う。コミッタは、リポジトリに格納されている成果物に対して、変更権限を持つ。コントリビュータは、広域分散した一般のユーザであり、オープンソースソフトウェアに対し、新しいアイデア、ソースコード、バグフィックスやテストレポート等の貢献を行う。

2.2 CVS の概要

オープンソースソフトウェア開発で広く利用されている版管理システムとして、*CVS (Concurrent Versions System)*[7, 8] がある。CVS は、並行作業モデル、クライアント/サーバ方式や複数のクライアント認証方式のサポート等、インターネットを介した共同開発に適した特徴を持つ。

以下に、主な特徴を挙げる。

- 自由度の高い並行作業モデル

  CVS は、**コピー–修正–マージ** *(copy–modify–merge)* **方式**を採用している。この方式では、ユーザは CVS リポジトリから各自のワークスペースに必要な版をチェックア

ウトし、独自に開発を進める。ユーザがその成果をリポジトリに反映させるためには、チェックインを行う。チェックインを行う際、他のユーザが同一の版に対して作業を行い、リポジトリに反映させていた場合、または同時に反映させようとした場合、競合が生じる可能性がある。競合が発生した場合は、ユーザ間で連絡を取り合い、それを手動で解消する必要がある。

この方式では、ロックを使用せず並行作業を実現しているため、比較的、自由度の高い作業スタイルであり、オープンソースソフトウェアの開発プロセスに適応している。

- モジュール単位の版管理

  版管理は、モジュール単位で行われるため、ユーザは各自のワークスペースにモジュールをチェックアウトして作業を行う。各モジュールは、管理情報と管理対象 (中間成果物) から構成される。

- ファイル単位のバージョン付け

  CVS で管理される中間成果物 (ソースコード、ドキュメント等) は、*RCS* **ファイル形**[9] で保存される。RCS 形式は、最新のリビジョンに対し、各リビジョンとの差分を保持することで、バージョン管理を行う。

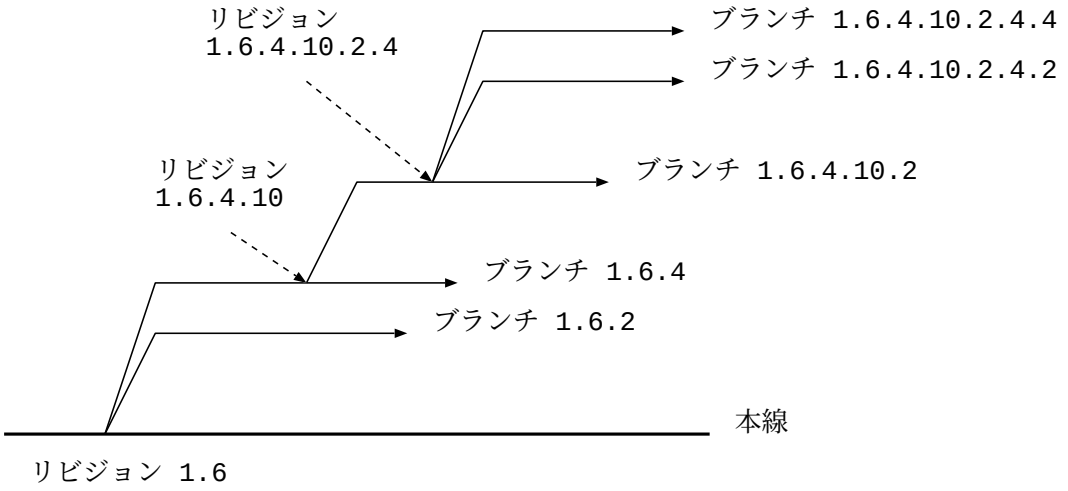

図 2.1: ブランチを用いた開発

また CVS は、ブランチの概念を持ち、開発方針の異なる成果物を分けて管理でき

る。図 2.1 を用いて、CVS におけるリビジョン番号とブランチ番号の関係を説明する。まず、本線のリビジョン番号は 1.6 とする。ここで、ブランチを作成すると、ブランチ番号は 1.6.2 となる。ブランチ番号の最後に追加される番号 (この例では”2”) には、未使用の最初の正の偶数が選択される。また、ブランチの初期のリビジョンは、ブランチポイント (ブランチを派生した本線の位置) の本線のリビジョン番号と同一であるため、リビジョンは 1.6 になる。さらにブランチが作成されると、同様の規則でブランチ番号が付加される。

- クライアント/サーバ方式と複数のクライアント認証方式

  CVS は、クライアント/サーバ方式 [10] を採用し、インターネットを介してリポジトリにアクセスできる。例えば、プロジェクトが地理的に分散したチーム構成であっても、インターネット環境があれば、オープンソースソフトウェアの共同開発に参加できる。

  また、複数のクライアント認証方式もサポートされている。認証方式には、パスワードの検証をする pserver(パスワード認証サーバ) 方式、Kerberos を用いた kserver 方式、Generic Security API(GSSAPI) を用いた gserver 方式や、rsh や ssh を用いた ext 方式がある。特に、匿名アクセスを認める**公開** *CVS* では、そのプロジェクトに関心を持つユーザが自由にモジュールをチェックアウトできるため、一般のユーザを開発に取り込み易い環境を提供する。

2.3 オープンソースソフトウェア開発に適したリポジトリ構成

前述したオープンソースソフトウェアの開発プロジェクトは、**集中型版管理モデル**を採用している。それは、プロジェクトに関するすべての開発や版管理を、単一のマスタリポジトリで行うモデルである。

近年、プロジェクト規模が大きくなるにつれ、ソフトウェアの国際化や多国語化、新しいプラットフォームへの移植や、新しいデバイスドライバへの対応等、目的の異なる複数のサブプロジェクトを持つ場合がある。サブプロジェクトは、基本的にマスタリポジトリにブランチを作成して開発を行い、メインプロジェクトと成果物の管理を分離している。しかし、サブプロジェクトは、メインプロジェクトとは**異なる運用方針**や**成果物の管理方針**を持つ場合が考えられる。単一のリポジトリでは、これらを共存させることが難しい。

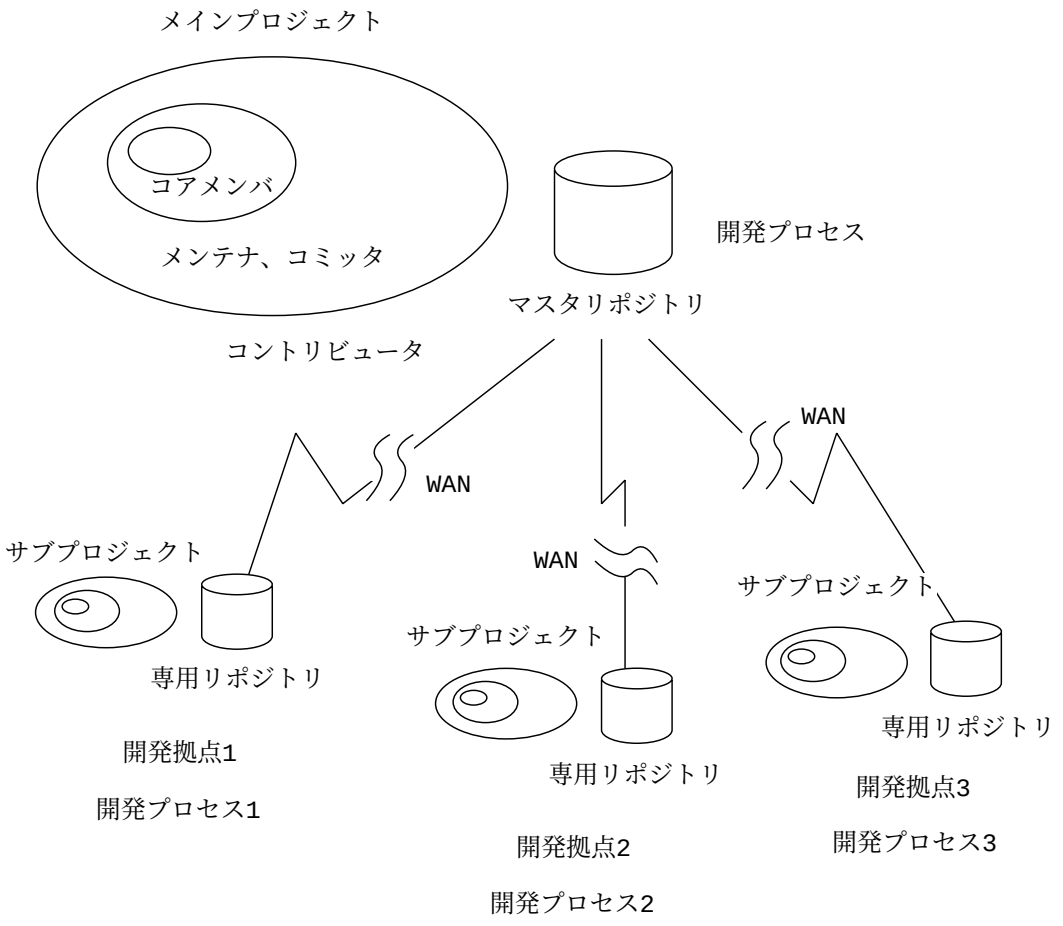

図 2.2: オープンソースソフトウェア開発と分散型版管理

そこで、これらのプロジェクトには、**分散型版管理モデル**が有効であると考えられる (図 2.2)。それは、複数の開発拠点ごとにマスタリポジトリと関連した専用リポジトリを持ち、それぞれが独立した開発と版管理を行うモデルである。ここで、マスタリポジトリのミラーリングは、版の広域配布が目的であり、このモデルには含まれない。

以下に、分散型版管理モデルが有効な場合をまとめる。

- **多くの開発拠点で並行に開発が進行する**

  集中型版管理モデルでは、マスタリポジトリにメインプロジェクトと複数のサブプロジェクトの成果物を一元に管理していた。分散型版管理モデルでは、サブプロジェクトごとに異なる開発拠点で専用リポジトリを持ち、それぞれ並行に開発が進行したり、独立した版管理を行うことが考えられる。

- **各拠点は１人〜複数人の開発者からなり、開発プロセスは拠点ごとに異なる**

サブプロジェクトは、個人またはチームから構成され、プロジェクトの運用方針や成果物の管理方針がそれぞれ異なることが考えられる。分散型版管理モデルでは、サブプロジェクトごとの専用リポジトリに対して、独立した運用方針と管理方針を設定できる。

- **利用可能なネットワーク帯域や計算機資源は拠点ごとに異なる**

  集中型版管理モデルでは、マスタリポジトリに対して、プロジェクトのメンバからの開発目的のアクセスだけではなく、一般のユーザからの参照目的のアクセスも混在する。そこで、リポジトリに対するアクセスは、重要度の違いから目的に応じて分けるべきであると考えられる。分散型版管理モデルでは、目的に応じたアクセスの分離や、マスタリポジトリに集中する負荷の分散ができる。

- **拠点間の分散/協調方針はプロジェクトごとに異なる**

  開発プロジェクトごとに、メインプロジェクト–サブプロジェクト間、またはサブプロジェクト間の開発の並行性や関連性は異なると考えられる。分散版管理モデルでは、プロジェクトに応じた柔軟な分散構成や、必要に応じてリポジトリ間に関連を設定可能にすることで、これに適用できる。

前述したCVSは、インターネットを介した共同開発に適しているため、オープンソースソフトウェア開発で広く利用されている。しかし、そのリポジトリは、集中型版管理モデルであり、メインプロジェクトと異なる運用方針や管理方針を持つサブプロジェクトの要求に、充分に対応できていない。

以下に、その問題点をまとめる。

- **開発拠点ごとに異なる管理方針を適用できない**

  オープンソースソフトウェア開発では、複数の開発拠点を持つ場合や、開発拠点ごとにソースコードの変更方針などの異なる成果物管理方針を持つ場合がある。CVSのリポジトリ構成では、複数の異なる管理方針を共存させることが難しい。

- **開発拠点ごとに異なる運用方針を適用できない**

  各拠点は、中間成果物の管理だけではなく、リソース管理、アクセス権管理、コミット権限管理といった独自のプロジェクトの管理や運用方針を持つ。これらを独自に適用し、運用できるリポジトリが必要である。CVSのリポジトリ構成では、複数の異なる運用方針を適用できない。

- **リポジトリの負荷分散やアクセスの分離ができない**

  世界中から単一のリポジトリにすべてのアクセスが集中するため、ネットワーク帯域と計算機資源に高い負荷がかかり、共同開発に支障をきたす恐れがある。

  また、リポジトリに対して、共同開発を行う目的のアクセスと、開発以外の目的 (参照、版の取得など) のアクセスを分離できない。

これらの問題点から、オープンソースソフトウェア開発を支援する版管理システムは、**リポジトリの分散構成**を持つことが望ましい。

オープンソースソフトウェアは、広域分散したユーザから、アイデア、ソースコード、バグフィックスやテストレポートなどの貢献を受け付け、それらを取り込むことで開発プロジェクトの推進力と発展が得られてきた。そこでリポジトリの分散モデルは、サブプロジェクトの貢献をメインプロジェクトのリポジトリ (マスタリポジトリ) に反映可能にするため、リポジトリ間において開発の成果を反映させる、または取り込む関連が必要である。この関連は、リポジトリ間である程度の**版の矛盾**を許すが、最終的に開発の成果を反映するために**一貫性**を保つべきである。

そこで本研究では、以下に挙げた特徴を持つリポジトリの階層型分散モデルを提案する。

feat.1 **リポジトリの分散は階層型 *(*親–子関係*)* である**

feat.2 **プロジェクト構造ごとにリポジトリの分散構成を柔軟に構成できる**

feat.3 **リポジトリごとに異なる成果物管理方針やプロジェクト運用方針を適用できる**

feat.4 **リポジトリ間の関連を維持できる**

2.4 既存のリポジトリ分散モデルの特徴と問題点

既存のソフトウェア構成管理システムには、リポジトリの分散構成をサポートするものがいくつか存在している。オープンソースソフトウェア開発の要求を満たすリポジトリの分散モデルは、前述の feat.1 から feat.4 の特徴を持つことが望ましい。それらを踏まえ、既存の分散モデルの特徴をまとめ、その問題点について指摘する。

2.4.1 Rational ClearCase MultiSite

Rational ClearCase[11, 12] は、大規模な開発プロセスに適したソフトウェア構成管理システムの製品である。版管理や分散開発支援だけではなく、作業空間管理、プロセス管理、ビルド管理など数多くの機能を備えている。

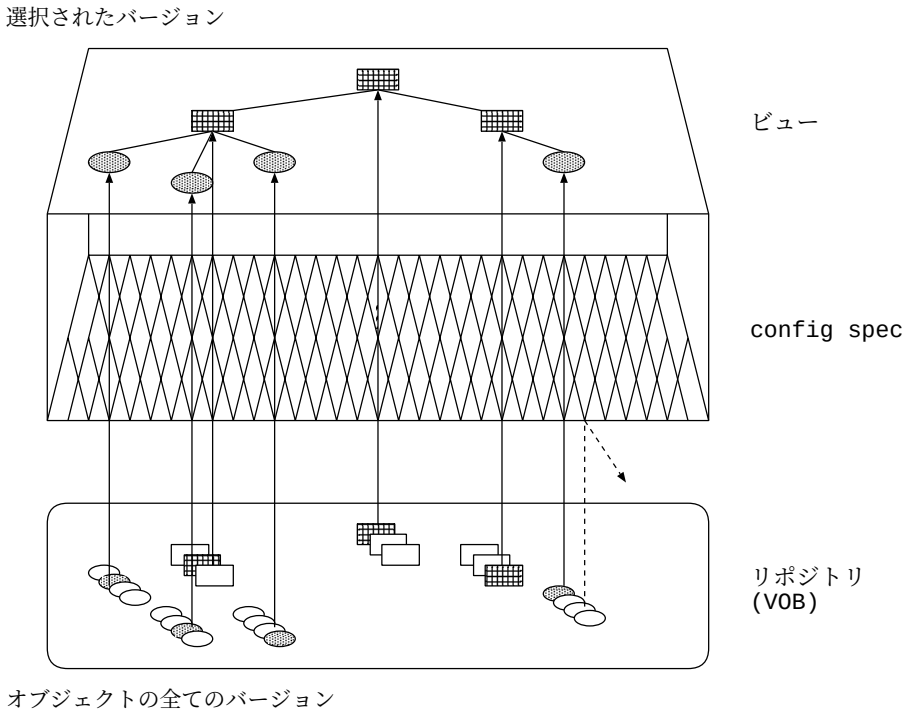

図 2.3: VOB の特徴

ClearCase のリポジトリである *VOB (Versioned Object Base:***バージョン付きオブジェクトベース***)*(図 2.3) は、すべての成果物をオブジェクトとして扱う。ソースコードやドキュメントといったテキストベースのオブジェクトだけではなく、バイナリ、実行可能ファイル、テストスイート、ユーザ定義オブジェクト等、あらゆるオブジェクトのバージョン管理を行える。バージョン管理は、ファイル単位だけではなく、ディレクトリ単位でも行うことが可能であり、管理対象は**エレメント** *(element)* と呼ばれる。

この製品は、各ユーザごとに必要なオブジェクトの構成を提供する**ビュー** *(view)* を持つ。ビューは、**構成仕様** *(config spec)* と呼ばれるユーザ固有のルールに基づいて、バージョンの選択や、特定のバージョンのオブジェクトへのアクセスを提供する。ユーザは、開発、移植やバグフィックスといった、用途別に複数のビューを持つことができる。また、

複数の VOB から１つのビューを構成することもできる。

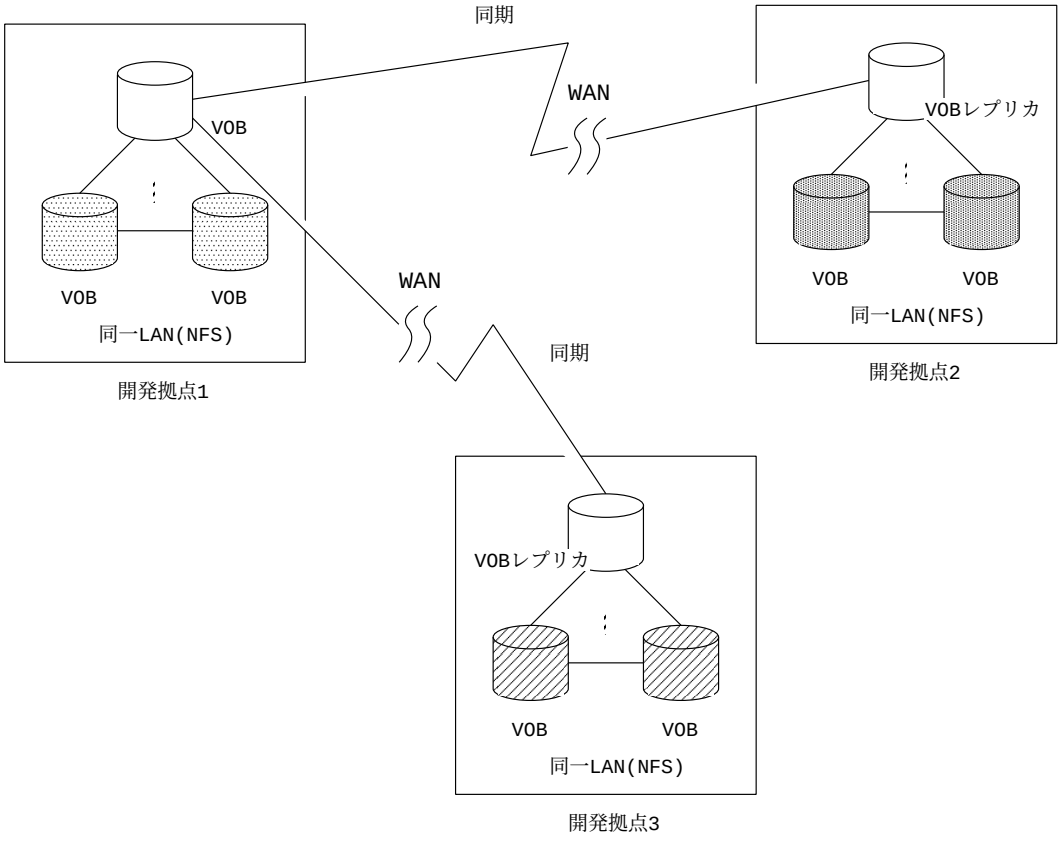

図 2.4: VOB のリポジトリ分散モデル

VOB は分散機能を持っている。この機能は、NFS (Network File System) を利用しているため、同一の LAN 上でリポジトリの分散を行う制約がある。分散した各 VOB は、**非中央型**であり、それぞれが互いにオブジェクトの変更をインポート、またはエクスポートできる。各 VOB は、それぞれ異なる運用方針や管理方針の設定もある程度可能であるが、レプリケーション、ロールバック、ビュー等、高機能なデータベースやファイルシステムのように信頼性や性能の向上に重点が置かれている。

実際のオープンソースソフトウェア開発では、WAN レベルのリポジトリの広域分散を行うことが想定される。VOB の WAN レベルの広域分散 (図 2.4) を実現するためには、オプション製品である Rational ClearCase MultiSite[13] を利用する。MultiSite は、WAN 越しのサイトに VOB のレプリカを作成し、常に VOB のレプリカ間で同期をとる必要がある。常に同期を取っているレプリカは、オブジェクトの固有の構成や管理方針を持つこと

ができない。

これにより、前述の feat.1、2、3 が備わっていない。

2.4.2 BitKeeper

BitKeeper[14] は、オープンソースソフトウェア開発の支援を目的とした版管理システムの製品である。実際に、Linux/PPC の開発 [15]、Linux2.5 カーネルの版管理 [16] や、Intel 社の次期 CPU である Merced へ Linux カーネルを移植するプロジェクト等で利用されている。

この特徴は、リポジトリの分散や圧縮、ディスコネクテッドオペレーションやチェンジセット等、分散開発を考慮した機能が用意されている。リポジトリの分散 (図 2.5) は、clone コマンドを用いて parent(複製元) から child(複製先) を作成することで実現している。

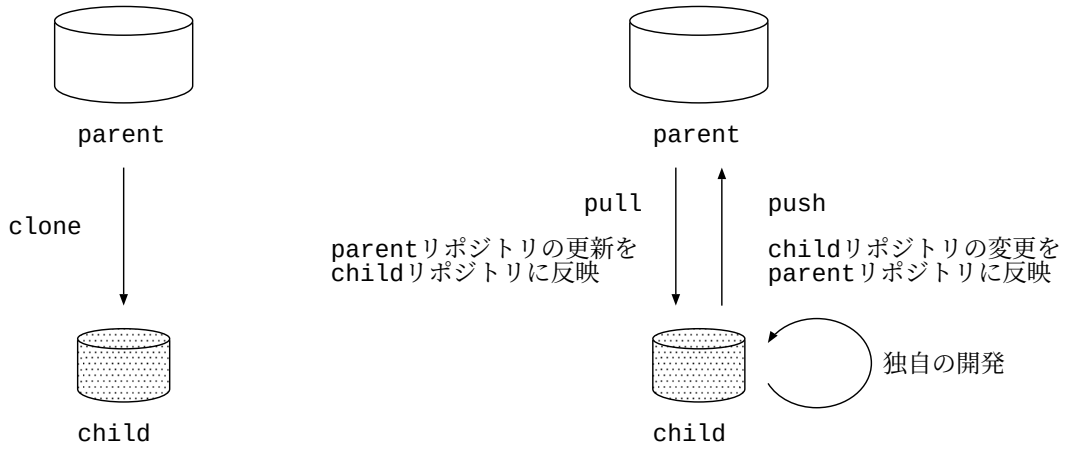

図 2.5: BitKeeper のリポジトリ分散モデル

各開発者は、ローカルリポジトリ child を持ち、作業成果をマスタリポジトリ parent に反映 (push) させる作業スタイルをとっている。しかし、child リポジトリは、基本的に parent リポジトリの全複製であり、リポジトリごとに固有の版構成を取れない。また、版を複製する際、チェンジセットを指定できるが、特定のリビジョンまでの全複製のみの指定である。このため、ある特定の範囲のリビジョンや、特定の版のみといった木目細やかな複製ができない。

これにより、前述の feat.3 が備わっていない。

2.4.3 CVS リポジトリと CVSup

CVSup[17] は、CVS リポジトリの高機能なミラーリングツールである。CVS と組み合せて使用することで、リポジトリの分散構成を実現できる。実際に、FreeBSD や NetBSD プロジェクトのミラーサイトの実現に利用されている。

これは、マスタリポジトリ (複製元) とローカルリポジトリ (複製先) の差分検出を行い、RCS 形式のファイルの高速な複製が行える。ローカルリポジトリでは、マスタリポジトリと衝突の発生しないブランチ番号を付けることによって、マスタリジトリと同期を取りながら、そのブランチで独自の開発 (図 2.6) が行える。

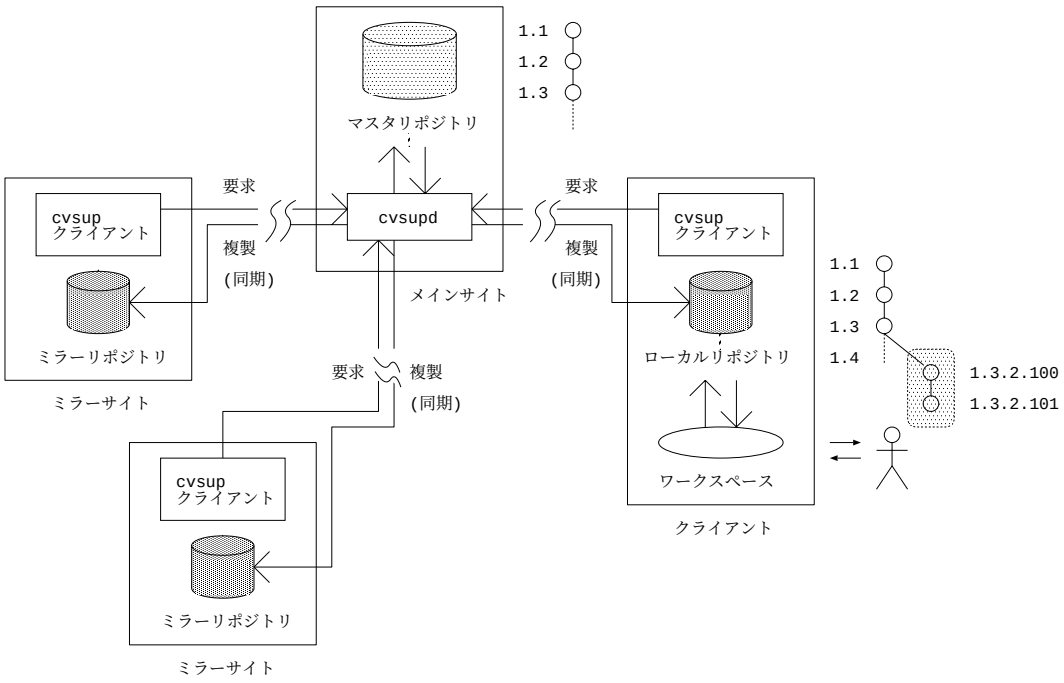

図 2.6: CVS リポジトリと CVSup のリポジトリ分散モデル

マスタ–ローカルリポジトリ間の関連は単方向 (マスタからローカル) であり、ローカルリポジトリの開発の成果をマスタリポジトリに自動的に反映させる機構を持たない。このため、各ローカルリポジトリからのフィードバックは、マスタリポジトリの版との差分を抽出し、パッチの作成を手動で行う必要がある。

これより、前述の feat4 が備わっていない。

2.4.4 arch

arch[18] は、2002 年 1 月 16 日にプレリリースされた、分散型の版管理システムである。このシステムは、ユーザごとに作業ブランチを持ち、それぞれのローカルな開発拠点で作業を行うモデルである。各ブランチは、自由に分散可能であり、それらの自動マージ機能も備えている。

以下に、主な特徴を紹介する。

- パッチセット (patch set)

  パッチセットは、チェンジセットと類似したバージョン管理機構であり、パッチ単位でブランチ間のマージを行う。また、どのブランチのどのパッチとマージしたか情報管理を行い、これを利用したマージアルゴリズムを持つ。

- スター型の分散モデル

  分散形態は、ソース (版の複製元) を中心にブランチがスター型に分散する。このモデルでは、分散したブランチ間のマージも可能にするため、名前空間とパッチセットを利用し、複雑なマージを行う。

- 大域名前空間 (Global Namespace)

  Global Namespace は、プロジェクトに名前空間を与え、公開プロジェクト、プロジェクトの派生、ブランチごとのバージョン情報を扱える。以下に、名前空間を用いたブランチの指定方法の例を示す。

  プロジェクト project の最初のリビジョンの指定：

  hoge@foo.bar–project/project–0.1–base-0

  これは、ユーザ名、計算機名、プロジェクト名を組み合せて指定するため、名前が衝突することなく、ブランチを一意に識別できる。

- ポリシーの設定

この機能は、成果物のバージョンごとに異なるポリシーを設定できる。例えば、バージョンにより、安定版と開発版に区別されている場合、それぞれ異なるリリース管理を行える。

このシステムは、AWK、sed、shell スクリプトで記述され、FTP を利用して版の複製を行うため、shell 環境と FTP サーバが必要である。

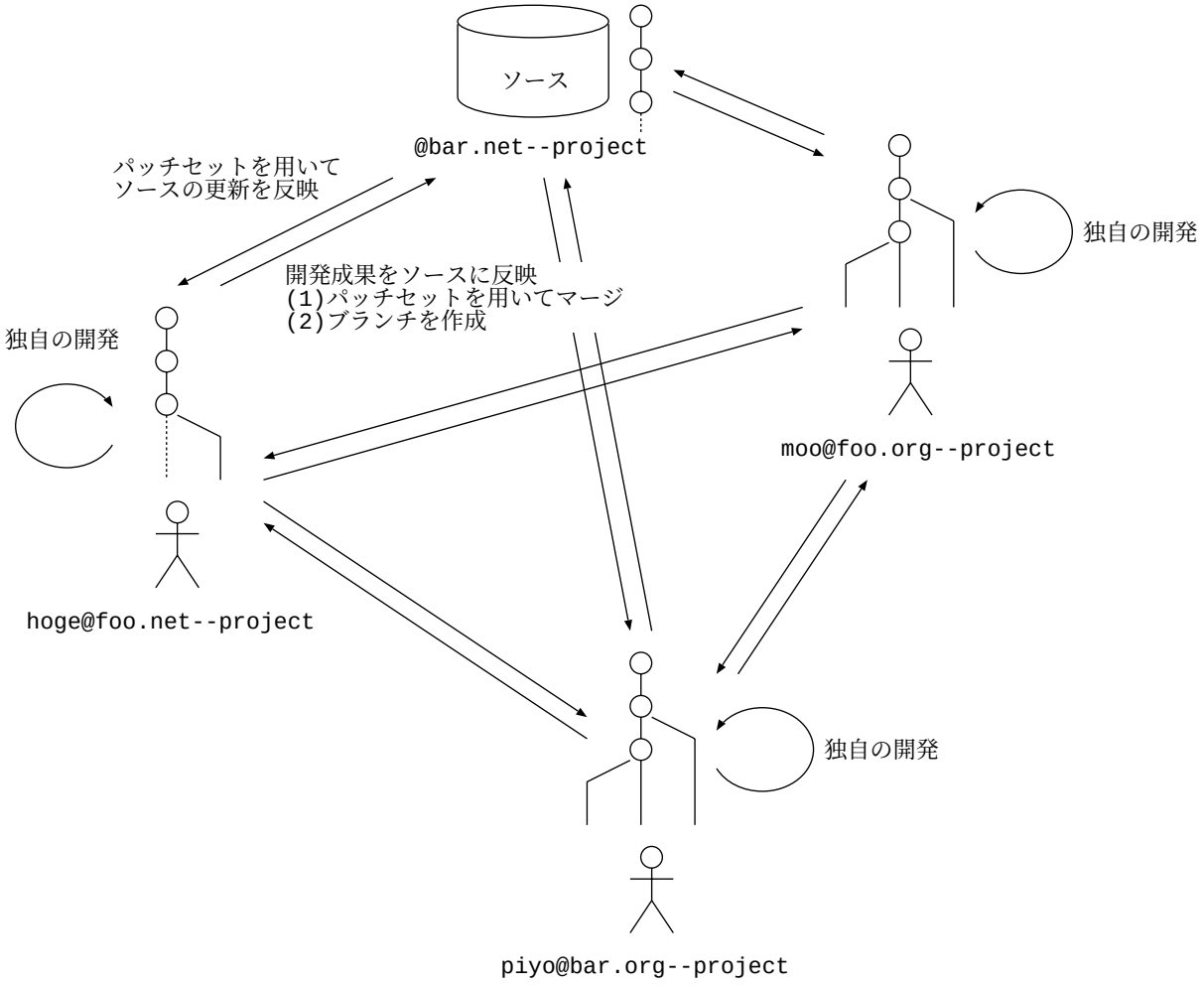

図 2.7: arch のリポジトリ分散モデル

第 3 章

リポジトリの階層型分散モデル

本章では、前章で提案した特徴を持つリポジトリの階層型分散モデルの概要を述べる。まず、基本モデルとして、２階層の分散構成を実現する機構と、このモデルを利用した開発のシナリオを述べる。次に、２階層の構成を組み合せた複雑な分散構成の例を挙げ、その特徴をまとめる。

3.1 モデルの概要

本研究では、オープンソースソフトウェア開発の視点からみた、既存のリポジトリ分散モデルの問題点を解決するため、前述した４つの特徴を持つリポジトリの階層型分散モデルを提案する。図 3.1 に基本モデルを示す。

まず、バージョンモデルは、オープンソースソフトウェア開発で広く用いられている利点があり、かつ、リポジトリの分散構成を持たない CVS を採用する。本研究で提案する分散モデルを適用することにより、前述した CVS の問題点も解決を図る。次に、リポジトリ分散モデルは、版の複製元と複製先のリポジトリ間の一貫性と関連性を維持できる。ここで、親/子リポジトリとは、版の複製元/複製先の関係である。子リポジトリの版は、１つ、もしくは、複数の親リポジトリの版の一部、もしくは、全部から構成される。本モデルでは、リポジトリごとに固有の版構成を実現するため、必要な版のみを選択し複製可能にする。

版の複製を行う際、以下の２つを指定可能にする。

- **モジュール名**

  ある親リポジトリの全モジュール、特定のモジュール、特定のファイルと指定可能

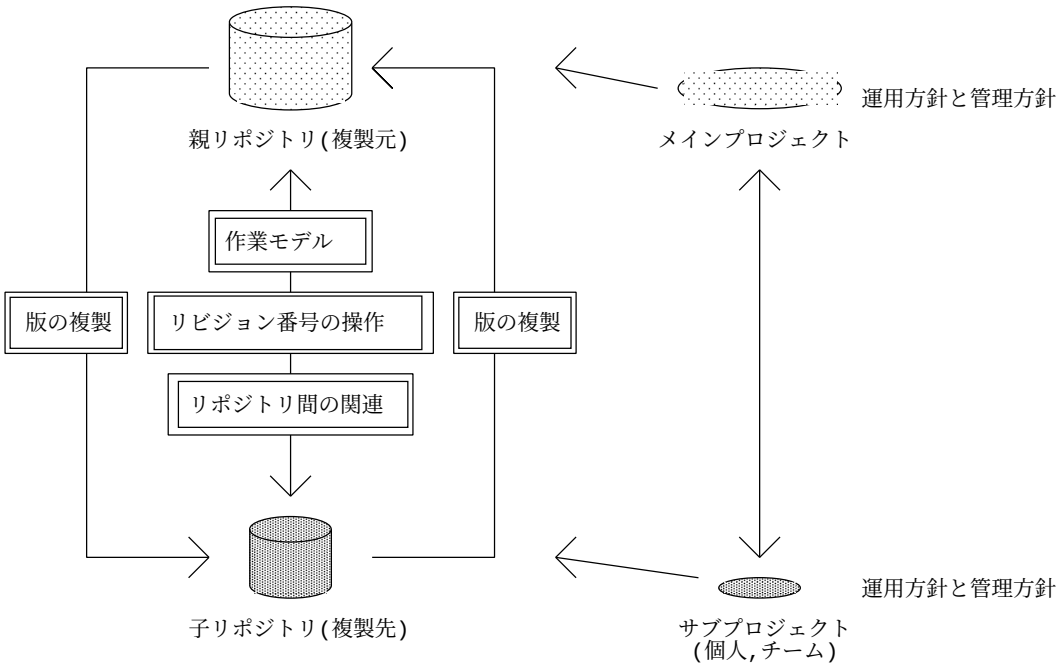

図 3.1: 基本モデル –2階層の分散構成–

- **リビジョン番号**

  RCS形式のあるファイルの全リビジョン、特定の範囲のリビジョン、特定のリビジョンと指定可能

親リポジトリから子に複製した版は、基本的にRead許可されている。親リポジトリの管理者は、子リポジトリの版に対してWrite許可を与えることによって、子リポジトリで独自の開発を許可する。

親–子リポジトリ間では、並行して独立に開発が進行するため、版の内容やリビジョン番号に矛盾が生じる。最終的に、子リポジトリの開発の成果を親リポジトリに反映可能にするため、版内容の矛盾を防ぐ**作業モデル**と、リビジョン番号の矛盾を防ぐ**リビジョン番号の操作**を定義する。また、必要に応じてリポジトリ間の関連を設定できるようにする。

3.2 基本モデルのシナリオ

基本モデルのシナリオを述べる。

(1) 親リポジトリから子リポジトリに必要な版を選択し、複製する

(2) 親リポジトリと子リポジトリでは、それぞれ異なるプロジェクトの運用方針と成果物の管理方針に基づき、並行に開発が進められる

(3) 子リポジトリは、自身の開発の成果を親リポジトリに反映させる

または、親リポジトリは子リポジトリの成果を主導的に取り込む

3.3 さまざまなリポジトリの分散構成

実際のオープンソースソフトウェアの開発プロジェクトでは、メインプロジェクトとサブプロジェクトが1対1関係の場合は少なく、複数のサブプロジェクトが同時に存在する場合や、複数のプロジェクトに関連するサブプロジェクトを持つ場合がある。これらのプロジェクトの形態を支援するためには、親/子リポジトリが1対1の分散構成だけではなく、複数の親リポジトリを持つ場合や、複数の子リポジトリを持つ場合が必要である。また、親/子リポジトリの2階層構成だけではなく、子リポジトリがさらに子リポジトリ(孫リポジトリ)を持つといった多階層の構成も必要である。

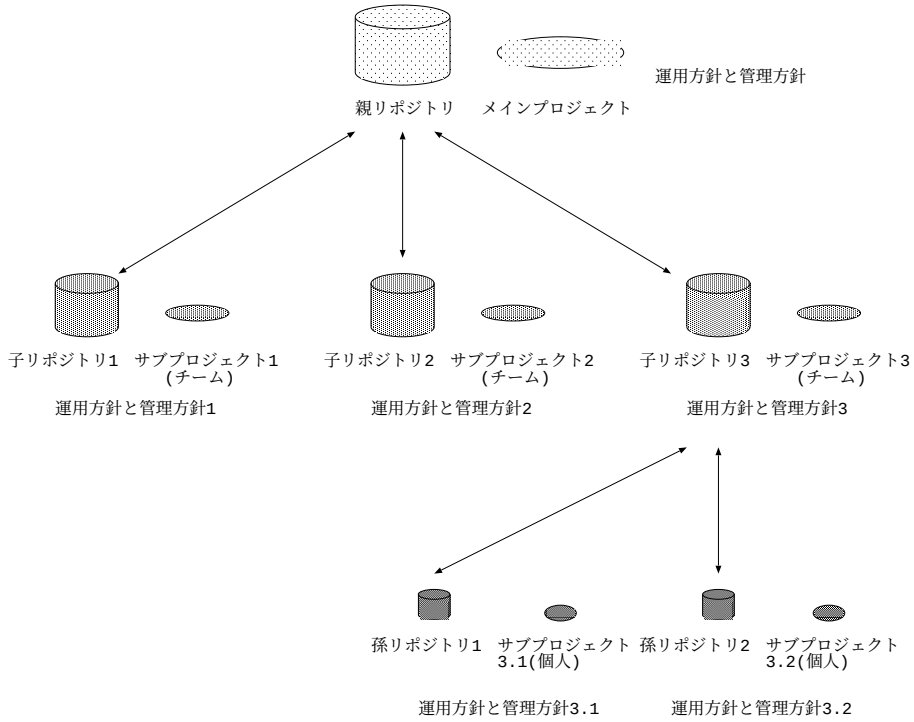

図 3.2: 複数のサブプロジェクトを持つ場合の分散構成

図 3.2 では、メインプロジェクトから、複数のサブプロジェクトが派生した場合の分散構成の例を示している。また、サブプロジェクト 3 の開発者が、ローカルな専用リポジトリを持つ要求に対し、さらに孫リポジトリを持つことでそれに対応している。

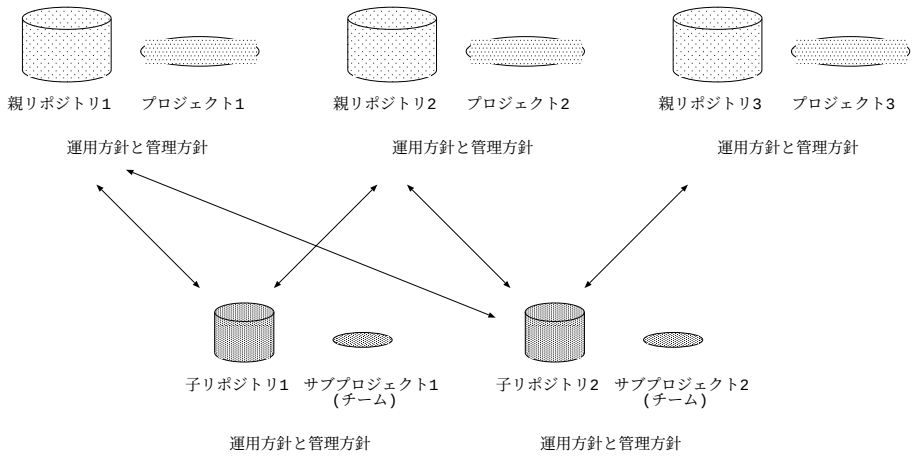

図 3.3: 複数の関連プロジェクトを持つ場合の分散構成

図 3.3 では、サブプロジェクト 1 と 2 が複数の関連したプロジェクトを持つ場合の分散構成の例を示している。サブプロジェクトの開発に必要なライブラリは、異なるプロジェクトで開発が行われている。これらのリポジトリから、必要な版を複製し、子リポジトリで開発を行っている。

以下に、複雑なリポジトリの分散構成をまとめる。

- ***2* つ以上の階層 *(*多階層*)* を持つ場合がある**
- **複数の親リポジトリを持つ場合がある**
- **複数の子リポジトリを持つ場合がある**

これより、本モデルは前述した 4 つの特徴を持つ。まず feat.1 は、**版の複製元**(親リポジトリ) と**複製先**(子リポジトリ) の関係から成り立つ。feat.2 は、本モデルを組み合せることで、**多階層**や**複数の親/子リポジトリ**の構成を取れる。feat.3 は、feat.4 を守った上で、独自に設定できるようにする。feat.4 は、版の矛盾を防ぐ**作業モデル**と、リビジョン番号の矛盾を防ぐ**リビジョン番号の操作**を導入する。また、必要に応じて**リポジトリ間の関連**を設定することで実現する。

次章では、基本モデルの構成要素である、作業モデル、リビジョン番号の操作、リポジトリ間の関連をそれぞれ詳細に説明する。

第 4 章

リポジトリ間の一貫性維持機構

開発が進行するにつれ、子リポジトリに独自のファイルやモジュールが作成され、また親リポジトリから複製した版に対して独自の進化が行われ、親–子リポジトリ間の版に矛盾が生じる。親–子リポジトリ間にある程度の版の矛盾を許可し、子リポジトリの開発の成果を親リポジトリに比較的容易にマージ可能にするため、作業モデルとリビジョン番号の操作を定義する。さらに、これらを保持した上で、必要に応じてリポジトリ間の関連付けを行えるように、基本的な関連を設ける。

4.1 作業モデル

親–子リポジトリ間で並行して独立に開発を行う場合、版の内容の矛盾が生じる。これを防ぐためには、ブランチを利用して親/子リポジトリで開発対象の版を分ける、もしくは、親/子リポジトリの開発をどちらか一方のみ許可する方法が考えられる。それぞれの方法として、ブランチ制御方式とロック制御方式を定義する。

4.1.1 ブランチ制御方式

ブランチ制御方式では、子リポジトリの独自の開発はブランチを作成して行う。子リポジトリが親リポジトリの版を複製し同期を取る場合、その後の親リポジトリの版の進化は、すべて子リポジトリに反映される。子リポジトリで試験的なマージを行いコンフリクトを解消した後、親リポジトリに子リポジトリの成果を反映させる。

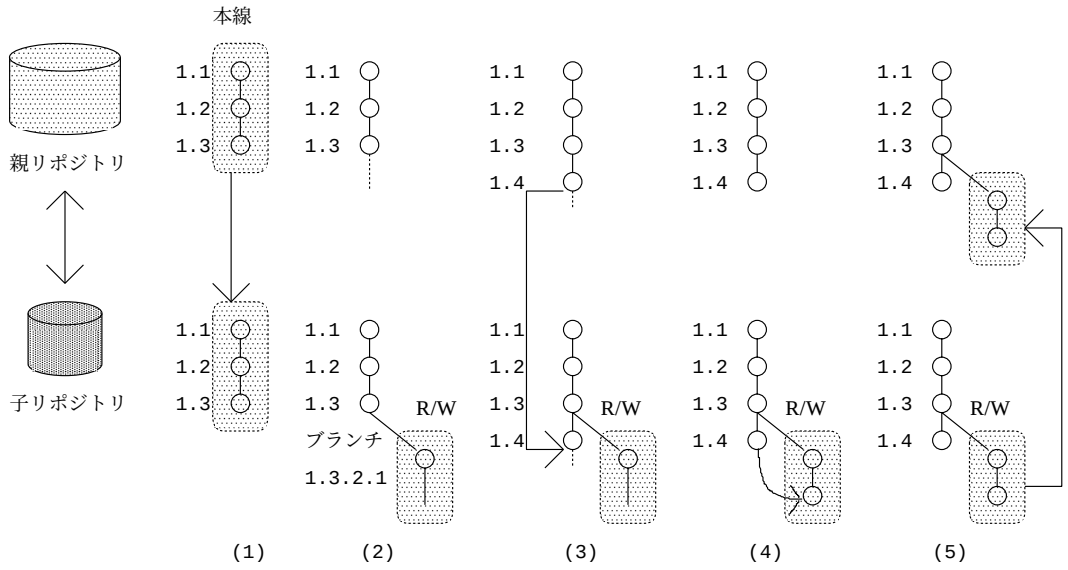

図 4.1: ブランチ制御方式

図 4.1 は、ブランチ制御方式を用いた場合の版の進化の例を示している。(1) まず、親リポジトリから子リポジトリに必要な版を選択し、複製する。(2) 子リポジトリの開発は、ブランチを作成して進められる。本線は Read 専用である。(3) 親–子リポジトリ間の関連として版の同期を設定した場合、親リポジトリの版の進化が子リポジトリに自動的に反映される。(4) 子リポジトリでは、親リポジトリに成果を反映させる前に本線と試験的なマージを行い、コンフリクトを解消する。(5) 子リポジトリは、親リポジトリに反映要求を発行し親リポジトリに承認されたなら、親リポジトリにブランチを作成し子リポジトリの開発の成果を反映させる。または、親リポジトリは、子リポジトリの成果を主導的に取り込む。

子リポジトリが親リポジトリの一部を複製している場合も考えられる。また、プロジェクト間の方針によって、親–子リポジトリ間に版の同期を設定しない場合もある。もしマージを行う際、子リポジトリに複製していない版が必要な場合、不足しているファイルを親リポジトリから子リポジトリにキャッシュする。

4.1.2 ロック制御方式

ロック制御方式では、親リポジトリの該当ファイルにロックを掛け、子リポジトリで独自の開発が行われる。これは、それぞれのリポジトリで並行に開発を行わず、親リポジト

リの開発の一部を子リポジトリに委譲する場合に用いられる。子リポジトリの成果は、子リポジトリから親リポジトリに主導的に反映される。

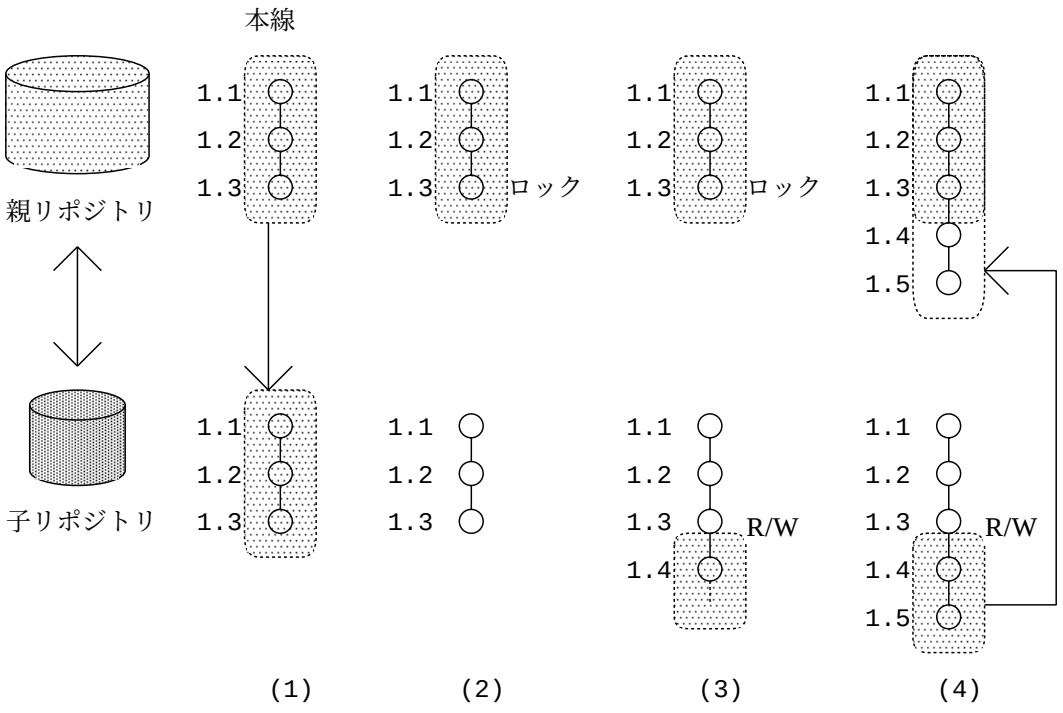

図 4.2: ロック制御方式

図 4.2 は、ロック制御方式を用いた場合の版の進化の例を示している。(1) まず、必要ならば、親リポジトリから子リポジトリに版を複製する。(2) 親リポジトリは、子リポジトリに開発を委譲する該当ファイルにロックを掛ける。(3) 子リポジトリの作業は、本線に対して、またはブランチを作成して自由に行える。(4) 子リポジトリは、開発の成果を親リポジトリに主導的に反映させる。

4.2 リビジョン番号の操作

親–子リポジトリ間の矛盾は、版の内容だけではなく、リビジョン (ブランチ) 番号にも生じる。そこで、リビジョン番号の矛盾を解消するため、リビジョン番号の操作を定義する。基本的には、子リポジトリの番号のみを付け換えるが、場合によっては、親リポジトリの番号の付け換えも必要である。

子リポジトリに対するリビジョン番号の付け換え

子リポジトリにブランチを作成して作業を行う場合、親リポジトリに子の開発の成果を容易に反映可能にするために、そのブランチ番号は親リポジトリの番号と衝突しない番号にする必要がある。開発が進行すると、親リポジトリにも新規にブランチが作成され、子リポジトリの既存のブランチ番号と衝突する可能性がある。そこで、ブランチ番号の衝突を防ぐため、リビジョン番号の付け換えを定義する。

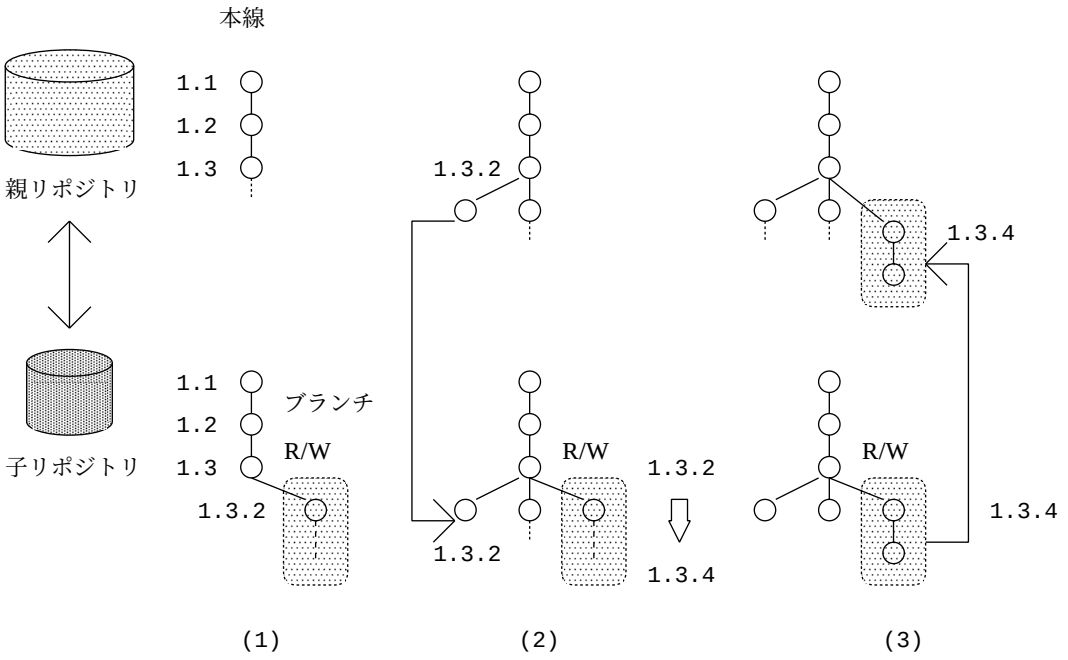

図 4.3: 子リポジトリに対するリビジョン番号の付け換えの例

図 4.3 では、ブランチ制御方式を用いて子リポジトリで開発を行っている。(1) まず、子リポジトリは、親リポジトリから版を複製し、ブランチを作成して作業を進める。(2) 次に、親リポジトリの同一のブランチポイントにブランチが作成され、親リポジトリと子リポジトリとブランチ番号が衝突する。それを自動検出し、該当する子リポジトリのブランチ番号を１つ進める。(3) 子リポジトリの成果を親リポジトリに反映させる際、ブランチ番号の衝突を防ぐことができる。

4.3 リポジトリ間の関連

本モデルでは、親–子リポジトリ間の版の内容とリビジョン番号の矛盾を解消する機構だけではなく、さらに必要に応じて関連も設定可能にする。基本的な関連としては、版の同期、版のフィードバックの期間、版の矛盾の通知、版の委譲を設ける。

4.3.1 版の同期

版の同期は、親リポジトリの版の進化を一定時間ごとに子リポジトリに反映可能にする。この関連では、子リポジトリで独自の開発が行われている場合、親–子間のリビジョン番号が衝突する可能性があるため、自動的にリビジョン番号の付け換えが行われる。

例えば、マスタリポジトリのミラーサイトを構成する場合、リポジトリ間にこの関連を設定することにより、マスタリポジトリと同一の版構成を持つリポジトリを実現できる。

4.3.2 フィードバック期間

フィードバック期間は、親リポジトリに子の開発の成果を自動的に反映させる期間を設定可能にする。親リポジトリは、手動で子の成果を取り込むのではなく、決められた期限、もしくは、一定時間ごとに自動的に成果が反映される。この関連では、親–子間のリビジョン番号が衝突する可能性があるため、子リポジトリの開発成果を親リポジトリにフィードバックする前に、リビジョン番号の付け換えが行われる。

4.3.3 コンフリクト通知

コンフリクト通知は、親リポジトリの版に対して、子リポジトリの版に設定値以上の競合が生じた場合、親リポジトリの管理者に通知を行う。この関連は、親リポジトリの管理者 (メインプロジェクト) が子リポジトリ (サブプロジェクト) の開発状況を把握する指標として利用できる。

4.3.4 版の委譲

版の委譲は、親リポジトリで行われている開発やメンテナンスの全部、もしくは、一部を子リポジトリに委譲できる。これは、前述した作業モデルのロック制御方式を用いて実

現する。まず、子リポジトリに委譲を行う親リポジトリの対象の版にロックを掛ける。この間、親リポジトリでは、委譲した版に対して作業ができない。子リポジトリでは、本線やブランチに対して自由に作業が行える。そして、子リポジトリから親リポジトリに開発やメンテナンスの成果を反映する場合、版やリビジョン番号の矛盾が生じないため、容易に複製できる。

第 5 章

モデルの適用例

これまで述べた本モデルは、プリミティブの組み合せにより、数種類の分散構成を実現できる。たとえば、プロジェクト間の並行開発や、開発の委譲がある。本章では、NetBSD プロジェクトとその国際化を行う Citrus プロジェクト間で並行開発を行う場合と、Linux プロジェクトとデバイスドライバを開発するサブプロジェクト間で開発の委譲を行う場合の適用例を挙げる。

5.1 NetBSD プロジェクトと Citrus プロジェクト –分散開発拠点間において並行/協調開発–

オープンソースソフトウェアは、広域配布されているため、さまざまな国や地域のユーザが利用している。それらのソフトウェアの重要な開発の 1 つに、**国際化**と**多国語化**がある。それぞれ、*i18n(internationalization)* と *m17n(Multilingualization)* と呼ばれている。これらは、直行する概念であり、同一のソフトウェアを世界中の国や地域の人々に使用可能にすることと、同一のソフトウェア上で多国語を混在して処理可能にすることを目的としている。その実現は、フレームワークを用いて、従来のソフトウェアと言語処理の実装の依存を切り離している。これにより、使用したい言語のロケールやフォントモジュールを追加することで、さまざまな言語が扱えるようになる。

このようなプロジェクトとして、*Citrus* **プロジェクト**[19] がある。このプロジェクトは、BSD 系オペレーティングシステムに i18n フレームワークの実装を目的としている。これは、NetBSD プロジェクト等のサブプロジェクトではないが、本家のリポジトリのソースコードに基づき、専用リポジトリで開発が行われていた。現在では、NetBSD 本家のリポ

ジトリにブランチとして取り込まれている。

ソフトウェアの GUI やドキュメント等は、ほとんどが英語で記述されている。このため、その多国語化や国際化は、英語圏以外の開発者やユーザが提唱する場合がある。また、大規模なフレームワークや言語処理機能を追加するため、開発の方針がメインプロジェクトと大きく異なり、リポジトリを独立させたい場合がある。Citrus プロジェクトは、日本が開発拠点になっており、独自の開発方針で開発を行っている (図 5.1)。以下に、これらのサブプロジェクトの特徴をまとめる。

- メインプロジェクトと地理的に広域分散している
- メインプロジェクトと異なる開発や版管理の方針を持つ

このような特徴を持つ場合、分散した複数の開発拠点を持ち、それぞれ並行に開発が行われる。そして、最終的なサブプロジェクトの開発成果は、メインプロジェクトのリポジトリに反映する。または、メインプロジェクトからサブプロジェクトの開発成果を取り込む。

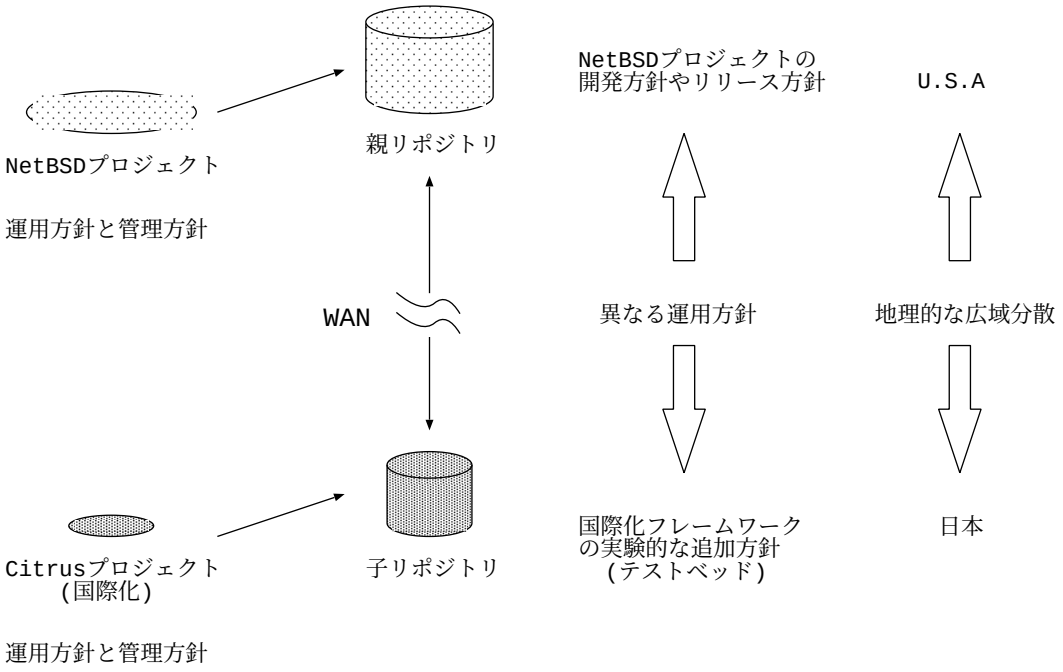

図 5.1: NetBSD プロジェクトと Citrus プロジェクト

本モデルを適用させた場合、以下のプリミティブを組み合せたリポジトリの分散構成が考えられる。

- **ブランチ制御方式**(作業モデル)
- **リビジョン番号の付け換え**(リビジョン番号の操作)
- **版の同期**(リポジトリ間の関連)

リポジトリ間の関連を維持するため、子リポジトリでは、ブランチを用いて開発を進める。リポジトリ間では、版の同期を取り、常に親リポジトリの開発成果を子リポジトリに反映させる。また、版の同期やフィードバックの際、ブランチ番号が衝突する恐れがあるため、自動的に子リポジトリの番号の付け換えが行われる。

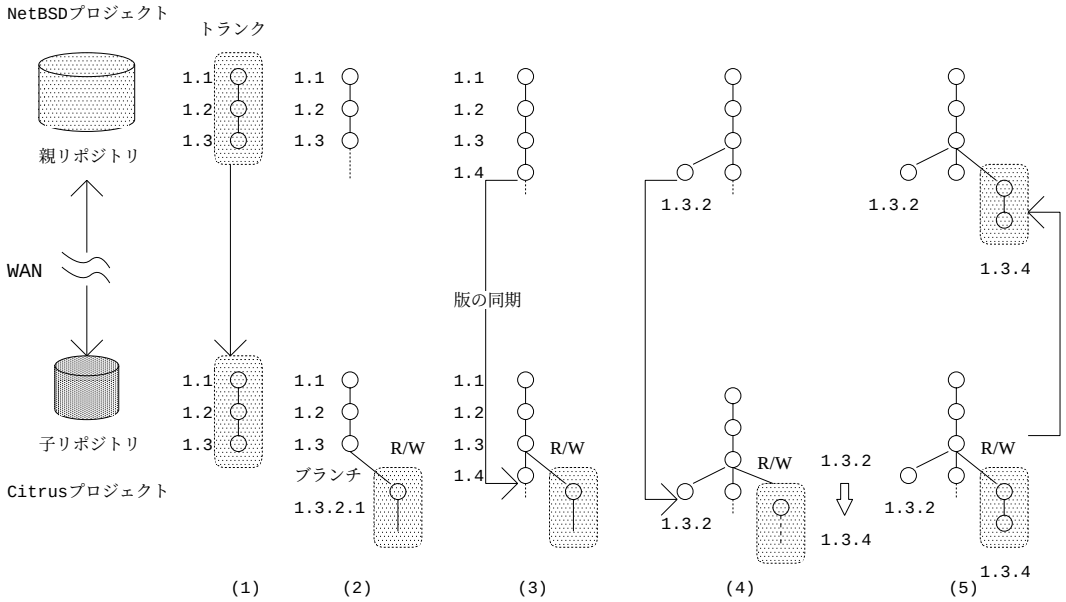

図 5.2: 並行開発の場合のシステム動作例

図 5.2 の例を説明する。(1) まず、サブプロジェクトは、開発に必要な版を親リポジトリから子リポジトリに複製する。(2) 次に、子リポジトリでは、**ブランチ制御方式**に基づいて開発成果の管理が行われる。(3) リポジトリ間の関連は、**版の同期**を設定し、親リポジトリの成果が子リポジトリに自動的に反映される。(4) 開発が進行し、リポジトリ間でブランチ番号の衝突が生じた場合、自動的に**番号の付け換え**が行われる。(5) 最終的に、子リポジトリの成果を親リポジトリに反映する。

5.2 Linux カーネルとデバイスドライバ開発 –プロジェクト間の開発の委譲–

オープンソースソフトウェアは、インターネットの基盤ソフトウェアやオペレーティングシステム等、複数のコンポーネントから成り立つ大規模なソフトウェアが多くある。その１つに Linux がある。その開発プロジェクトは、カーネルの開発を中心に行い、カーネルが標準対応するデバイスドライバは、信頼するサブサブプロジェクトや他のプロジェクトからの貢献を取り込むことで実現している。そこで、カーネル開発チームとデバイスドライバ開発チームの関係に着目すると、Linux プロジェクトは信頼のあるサブプロジェクトに、デバイスドライバの開発を委譲しているとみなせる。そこで、開発スタイルの１つにプロジェクト間の開発の委譲が考えられる (図 5.3)。

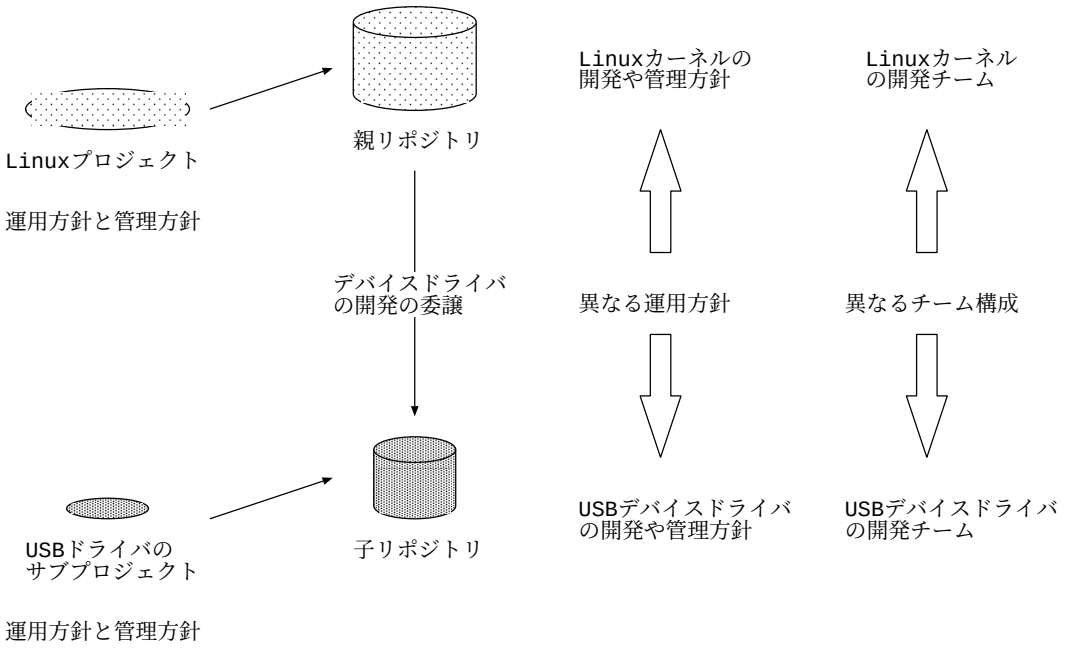

図 5.3: Linux プロジェクトとデバイスドライバプロジェクト

本モデルを適用させた場合、以下のプリミティブを組み合せたリポジトリの分散構成が考えられる。

- **ロック制御方式**(作業モデル)
- **フィードバック期間、版の委譲**(リポジトリ間の関連)

親リポジトリの特定の版は、子リポジトリに委譲するため、開発の責任は子リポジトリが持ち、自由に作業を進められる。子リポジトリの成果は、親リポジトリ側が設定した一定の期間ごとに、親リポジトリに自動的に反映される。

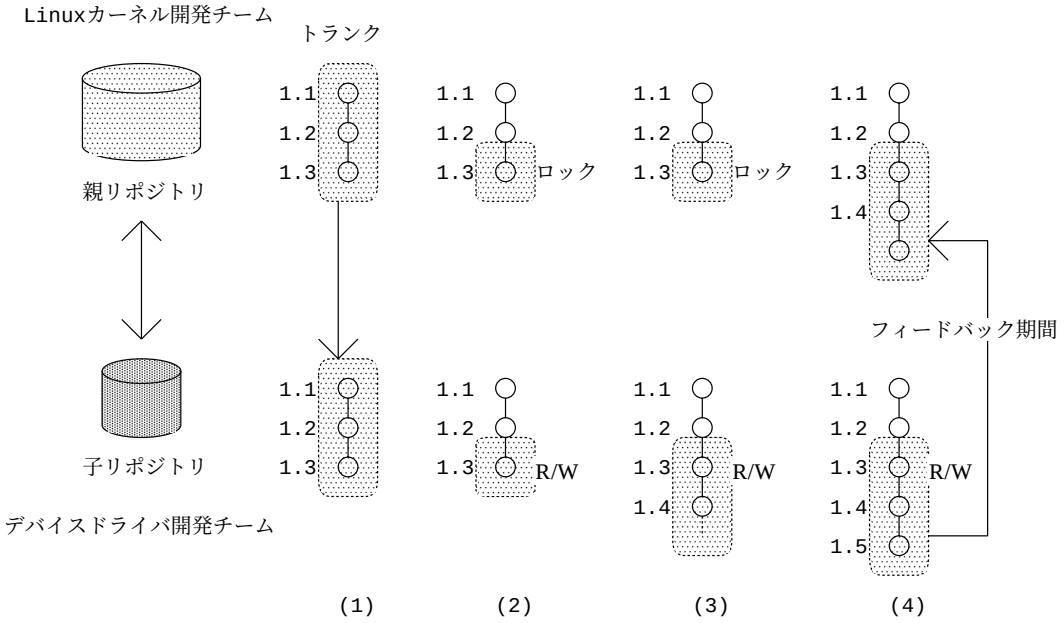

図 5.4: 開発の委譲の場合のシステム動作例

図5.4の例を説明する。(1) まず、親リポジトリから子リポジトリに版の複製を行う。(2) 次に、メインプロジェクトは、**ロック制御方式**に基づき、サブプロジェクトに開発を委譲する版にロックを掛ける。(3) 子リポジトリでは、自由に作業が行われる。(4) リポジトリ間の関連として**フィードバック期間**を設定し、一定期間ごとに子リポジトリの成果が親リポジトリに自動的に反映される。

第 6 章

distCVSの設計

プロトタイプシステムの設計では、CVS と CVSup を再利用し、リポジトリ間の関連を維持する親/子リポジトリサーバと、これらのサーバにコマンドを発行する distCVS クライアントの設計を行う。まず、システム全体の構成図を示し、各コンポーネントのユースケース図とクラス図を示す。

6.1 システムの設計方針

プロトタイプシステム distCVS の設計では、*CVS* と *CVSup* プログラムを再利用し、リポジトリ間の関連を維持する**親/子リポジトリサーバ**と、*distCVS* **クライアント**の設計を行う。親/子リポジトリサーバは、互いに協調動作し、親–子リポジトリ間の版の複製、一貫性制御、リビジョン番号の操作、リポジトリ間の関連の処理を担当する。

distCVS クライアントは、CVS クライアントのラッパーとして機能し、CVS コマンドであれば CVS サーバに、親/子リポジトリに関するコマンドであれば親/子リポジトリサーバに命令を転送する。

図 6.1 に本システムの構成図を示す。

親–子リポジトリ間の関連を操作する命令は、CVS コマンドに親/子リポジトリコマンドを追加する。CVS のユーザは、CVS コマンドと同じ書式で distCVS クライアントからそれらの命令を発行することで、リポジトリ間の関連を操作できるようにする。

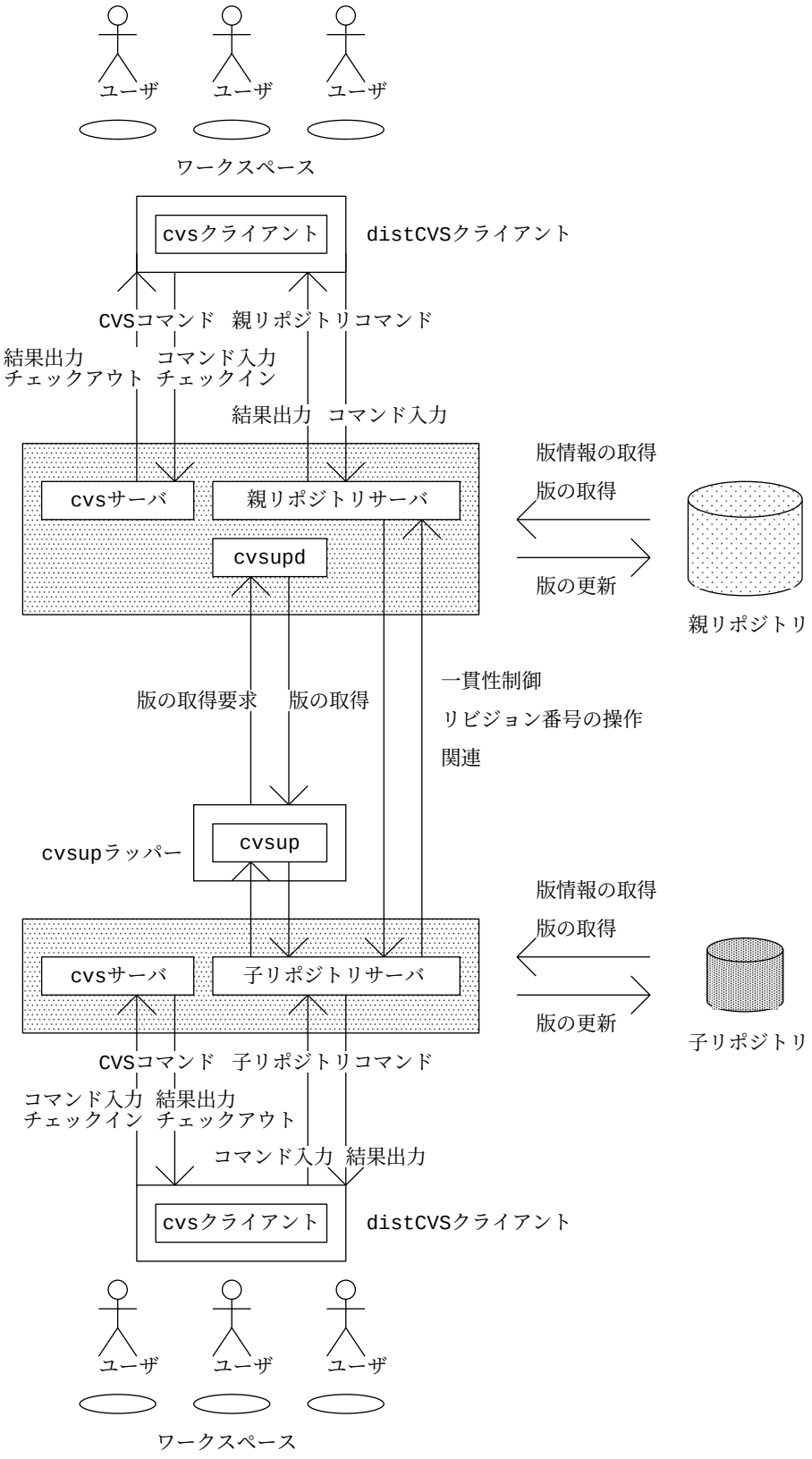

図 6.1: プロトタイプシステム distCVS の構成図

6.2 再利用するソフトウェア

プロトタイプシステムの実装は、CVS と CVSup を再利用する。以下に、それぞれの利用方法を説明する。

6.2.1 CVS サーバ/クライアント

CVS クライアントは、CVS サーバに CVS コマンドを発行する。各ユーザは、CVS クライアントを使用してそれぞれのリポジトリにアクセスを行える。

CVS サーバは、CVS クライアントから転送された CVS コマンドを受け、リポジトリに対してチェックイン、チェックアウト等の処理を行う。CVS サーバには、認証方式の違いからいくつかの利用方法があるが、本システムでは pserver を用いる。

6.2.2 CVSup サーバ/クライアント

親リポジトリから子リポジトリに版の複製を効率良く行うため、CVSup を利用する。cvsup クライアントは、設定ファイルに基づき、複製元のリポジトリの cvsupd に版の複製要求を出し、クライアント側のリポジトリに複製が行われる。

CVSup のクライアント cvsup は、子リポジトリサーバから内部的に起動されるようにする。子リポジトリサーバは、ユーザから発行されたコマンドに応じて設定ファイル supfile を生成し、CVSup の動作を決定する。親リポジトリサイドの CVSup のサーバ cvsupd は、cvsup からの要求を受け、指定された版を親リポジトリから子リポジトリに複製する。

6.3 親リポジトリサーバの設計

親リポジトリサーバは、親–子リポジトリ間の一貫性と関連性を維持する親リポジトリサイドのデーモンプログラムである。CVS プロキシから転送された親リポジトリコマンドに基づき、版の取得、一貫性制御、リビジョン番号の操作、リポジトリ間の関連の処理を担当する。

図 6.2 に、親リポジトリサーバのユースケース図を示す。まず、アクタには CVS ユーザと distCVS ユーザがいる。それぞれ、CVS の基本機能と、それに加え distCVS の機能を利用する。親リポジトリサーバは、版の複製、リポジトリ間の関連に関するユースケー

スを持ち、アクタと相互作用することで親–子リポジトリ間の操作が行われる。

表 6.1 に親リポジトリコマンドの一覧を示す。各コマンドは、CVS の管理者グループ (cvsadmin) に属するユーザが実行権限を持つ。各コマンドは、従来の cvs コマンドと同様に、dcvs コマンドの引数として指定可能にする。

命令の実行例 1(子リポジトリの情報取得)：

dcvs show *REPOSITORY*

命令の実行例 2(子リポジトリの特定のモジュールの取り込み)：

dcvs pull *MODULES*

表 6.1: 親リポジトリコマンド

命令	引数	説明
show	*REPOSITORY*	版情報の表示
push	*REPOSITORY* *MODULES* *FILES* [*REVS*]	版の複製
pull	*MODULES* *FILES* [*REVS*]	版の取得
accept		反映要求の承認
sync	[-t *sec*] *REPOSITORY*	版の同期の設定
term	[-t *sec*] *MODULES* [-t *sec*] *FILES*	フィードバック期間の設定
confict	[-c *percent*] *MODULES* [-c *percent*] *FILES*	コンフリクト通知の設定
delegation	*MODULES* *FILES*	版の委譲の設定

※REPOSITORY(子リポジトリ)、FILES(１つ以上のファイル)、MODULES(１つ以上のモジュール)、REVS(全部、特定の部分、特定のリビジョン番号)、-t(時間設定オプション)、-c(コンフリクト値設定オプション)

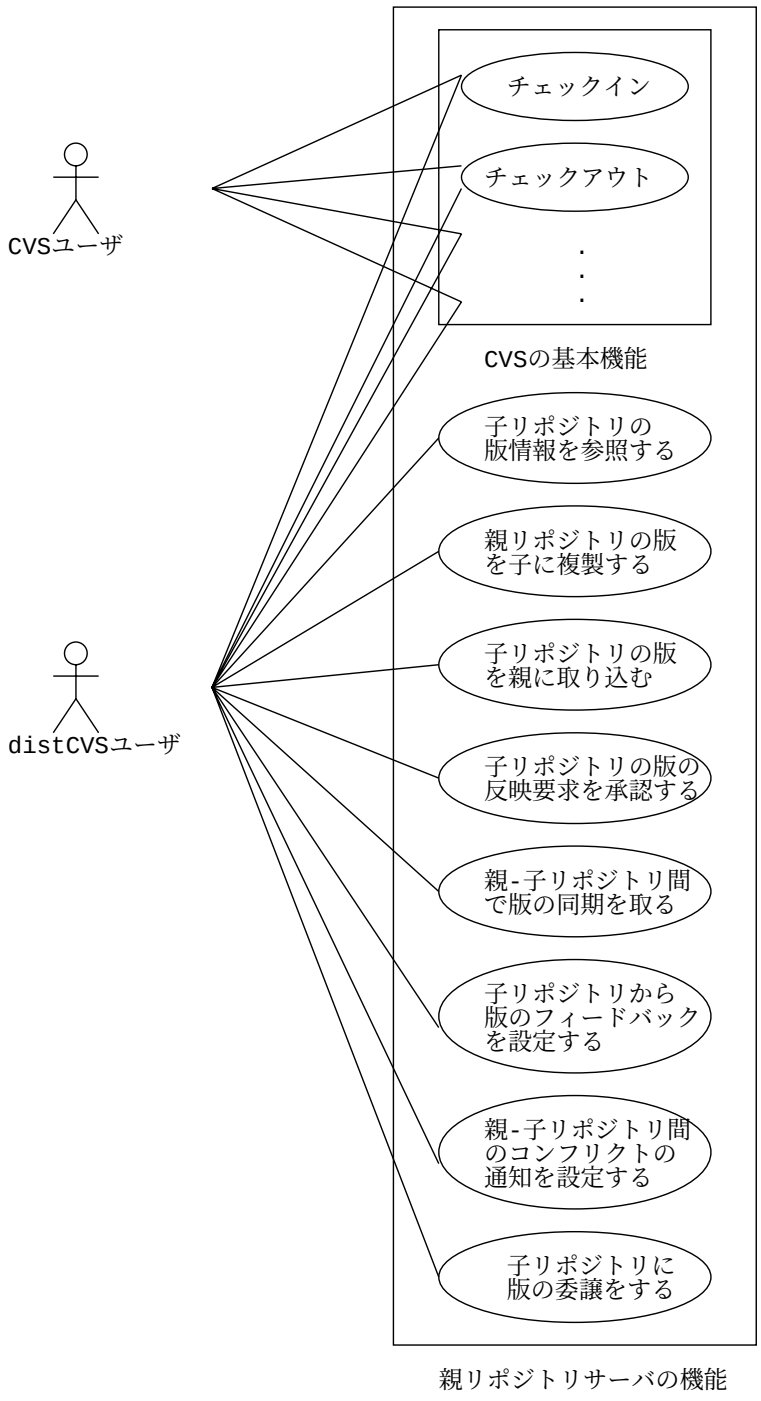

図 6.2: 親リポジトリサーバのユースケース図

6.4　子リポジトリサーバの設計

子リポジトリサーバは、親–子リポジトリ間の一貫性と関連性を維持する子リポジトリサイドのデーモンプログラムである。CVS プロキシから転送された子リポジトリコマンドに基づき、版の複製、一貫性制御、リビジョン番号の操作、リポジトリ間の関連の処理を担当する。

図 6.3 に、子リポジトリサーバのユースケース図を示す。まず、アクタには、CVS ユーザと distCVS ユーザがいる。それぞれは、CVS の基本機能と、それに加え distCVS の機能を利用する。子リポジトリサーバは、版の複製、リポジトリ間の関連に関するユースケースを持ち、アクタと相互作用することで親–子リポジトリ間の操作を行う。

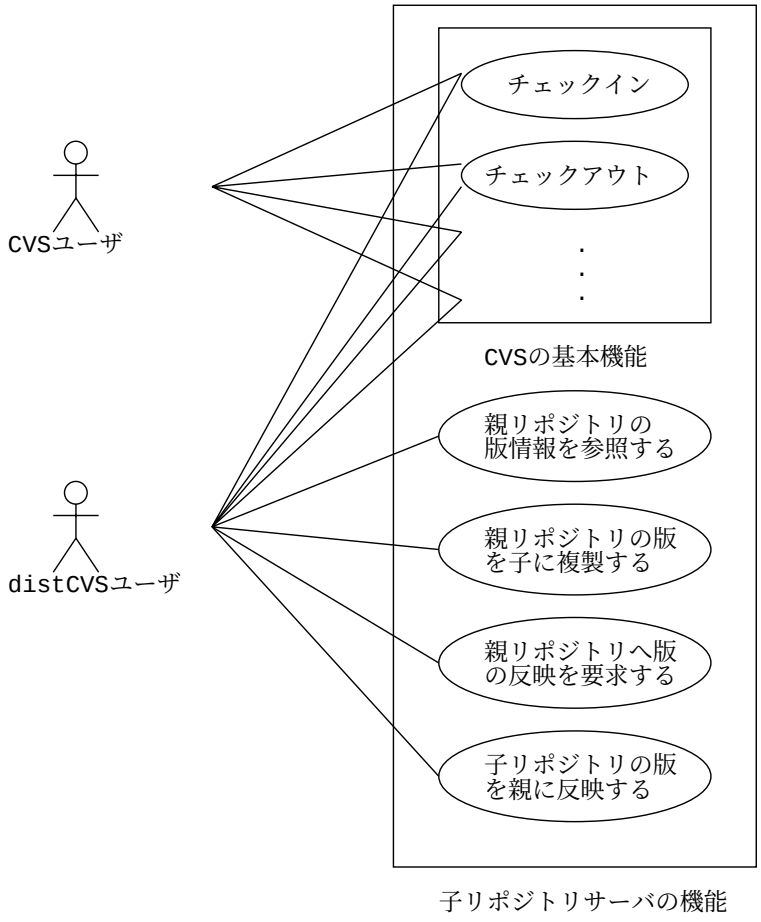

図 6.3: 子リポジトリサーバのユースケース図

表 6.2 に子リポジトリコマンドの一覧を示す。親リポジトリコマンドと同様に dcvs コ

マンドの引数として指定可能にする。

表 6.2: 子リポジトリコマンド

命令	引数	説明
show	*REPOSITORY*	版情報の表示
pull	*REPOSITORY* *MODULES* *FILES* [*REVS*]	版の取得
request	*MODULES* *FILES*	反映要求
push	*MODULES* *FILES* [*REVS*]	版の複製

※REPOSITORY(親リポジトリ)、FILES(１つ以上のファイル)、MODULES(１つ以上のモジュール)、REVS(全部、特定の部分、特定のリビジョン番号)

6.5 distCVS クライアントの設計

distCVS クライアントは、CVS クライアントのラッパーとして機能し、ユーザからの命令を受け、命令によって各デーモンプログラムに処理を依頼する。CVS コマンドであれば、CVS サーバに命令と引数を転送する。また、親/子リポジトリコマンドであれば、親/子リポジトリサーバに命令と引数を転送する。

第 7 章

distCVSの実装

7.1 実装環境

プロトタイプシステムの設計に基づき、現在、その実装を行っている。各コンポーネントは、Java のクラスとして実装し、それぞれはメソッド呼び出しで協調動作させる予定である。分散オブジェクトの通信には、比較的設定や開発コストのかからない HORB[20] を利用する。版管理システムとミラーリングツールは、それぞれ CVS と CVSup を利用し、システムから内部的に起動させるようにする。

表 7.1 に、実装環境を示す。

表 7.1: 実装環境

	ソフトウェア	バージョン
OS	FreeBSD	4.3-RELEASE
	NetBSD	1.5.2
ORB	HORB	ORB Version 2.0.2
プログラミング言語	Java	java version "1.3.1_01"
		java version "1.3.0beta_refresh"
版管理システム	CVS	1.11 (client/server)
ミラーリングツール	CVSup	CVSup client GUI version
		Software version: REL_16_1
		Protocol version: 16.1

7.2 distCVS の実行手順

プロトタイプシステム distCVS を実行するために、まず、その実行環境を整える必要がある。親リポジトリ側の計算機には、CVS サーバ/クライアント、CVSup サーバ、HORB サーバ、そして親リポジトリサーバに関連する Java クラスが必要である。また、CVS リポジトリを作成し、cvsupd を起動させておく必要がある。

子リポジトリ側の計算機には、CVS サーバ/クライアント、CVSup クライアント、HORB サーバ、そして子リポジトリサーバに関連する Java クラスが必要である。また、CVS リポジトリを作成しておく必要がある。

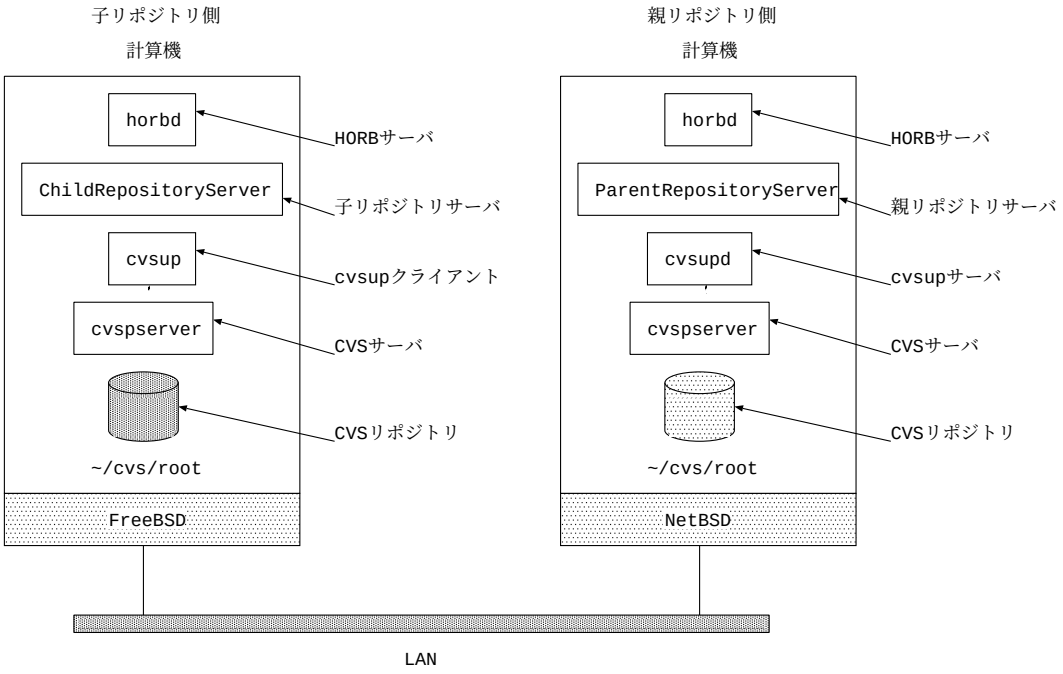

図 7.1: distCVS の実行環境

次に、実行の手順を示す。

1 親リポジトリサーバの起動

まず、以下のコマンドを発行し、親リポジトリサーバを起動する。horbd の起動の際、設定ファイル distCVS.conf を指定し、ParentRepositoryServer のインスタンスの生成と活性化を行う。

>horb -conf disCVS.conf

以下のメッセージが出力されれば、親リポジトリサーバのインスタンスの生成と活性化が成功している。

ParentRepositoryServer(prs) started

2 子リポジトリサーバの起動

同様に、以下のコマンドを発行し、子リポジトリサーバを起動する。horbd の起動の際、設定ファイル distCVS.conf を指定し、ChildRepositoryServer のインスタンスの生成と活性化を行う。

>horb -conf disCVS.conf

以下のメッセージが出力されれば、子リポジトリサーバのインスタンスの生成と活性化が成功している。

ChildRepositoryServer(crs) started

3 dcvs クライアントの起動とコマンドの実行

各サーバのインスタンスが生成され活性化されたなら、distCVS のユーザは、親/子リポジトリコマンドを発行して、それらを動作させることができるようにする。コマンドの発行は、distCVS クライアントから各サーバのインスタンスに接続し、引数にそれを指定して実行可能にする。

>java dcvs [command] [arguments]

第 8 章

考察

本章では、提案したリポジトリの階層型分散モデルと、開発を進めているプロトタイプシステムについて考察する。

8.1 CVS リポジトリの階層型分散モデルの考察

作業モデルと一貫性維持機構のトレードオフについて

親/子リポジトリで並行開発を行う場合、子リポジトリでは、ブランチを作成して作業を行うスタイルを採用している。この際、子リポジトリのユーザにとっては、ある程度の自由な作業は許されるものの、通常の CVS を利用する場合より利便性に欠けるであろう。理想的な分散リポジトリの並行開発は、各リポジトリを利用するユーザが、何の制約を受けることなく通常の作業が行え、なおかつ、リポジトリ間の関連を維持できることである。しかし、CVS は本来分散構成を考慮した設計ではなく、理想的な分散を行うためには CVS のバージョンモデルから分散リポジトリに適したバージョンモデルを導入する必要性が考えられる。今回は、CVS のバージョンモデルに基づく代わりに、ユーザの利便性とリポジトリ間の関連維持のトレードオフを行い、分散モデルを構築した。

リポジトリ間の関連では、版に関する３つの関連と、作業モデルに関する１つの関連をまとめた。この他にも、開発の委譲において親/子リポジトリの立場を逆転させる関連や、プロジェクト間の運用方針を反映させた関連が考えられる。また、必要な関連は用意した基本的なものだけとは限らない。したがって、関連は必要に応じて拡張可能にすることで、プロジェクトの要求により適切に対応できると考えられる。

開発スタイルについて

本モデルは、プリミティブの組み合せによって、数種類の分散構成がある程度柔軟に実現できる。しかし、実際に利用される構成は、親/子リポジトリで並行開発を行う場合や、親リポジトリの開発の一部を子リポジトリに委譲する場合など、開発スタイルが限られている。それらを本モデルに反映することで、さらに容易に分散構成を取れる上位のモデルが明らかになると期待できる。

プロジェクト管理モデルについて

現在、多くのオープンソースソフトウェアの開発プロジェクトは、版管理モデルだけではなく、プロジェクト管理も集中型の傾向がある。本モデルの有効性を高めるため、従来の集中型から分散型の管理モデルを作成することが挙げられる。

その１つとして、プロジェクト間の方針によって決められた関連を維持すれば、サブプロジェクトごとの権限構造や管理モデルの違いを意識しせずに、プロジェクト全体の管理が行えるモデルを考えている。

分散型版管理に適したバージョンモデルについて

本モデルは、オープンソースソフトウェアの開発に広く利用されている点から、CVS のバージョンモデルに基づき構築した。しかし、CVS は RCS ファイル形式でリビジョン情報を管理しているため、リポジトリを分散させた場合、リビジョン番号の衝突が起こる問題がある。その番号の操作は、複雑な機構になり、ユーザの作業に制約をかしてしまう可能性がある。子リポジトリにおいて自由度の高い作業を実現するために、番号の操作はなるべく行わないほうが良い。また、同一ファイルに複数のリビジョンが格納されているため、版の複製や同期を行う際、木目細やかな選択の実現が難しい。

そこで、将来的には CVS のバージョンモデルから、リポジトリの分散構成を実現し易いバージョンモデルを採用、もしくは提案することが望ましい。現段階では、チェンジセットを候補の１つに考えている。チェンジセットとは、リビジョンを差分ファイルとして持ち、特定のバージョンを構成するために必要な差分ファイル情報を保持することで、バージョン管理を行っている。これは、変更がファイル単に分れているため、必要な版のみ複製や同期の粗/細粒度の選択に比較的適しているといえる。

8.2 プロトタイプシステム distCVS の考察

プロトタイプシステム distCVS の実現について

distCVS の実装にあたり、本モデルで用意したプリミティブな操作同士がどのように作用し合うのか明確にし、機能の衝突が起こらないようにシステムを実現させるべきである。特に、複数の関連を同時に設定した場合、リポジトリ間の一貫性が保持されるのか検証する必要がある。

さまざまなリポジトリの分散構成について

実際の開発プロジェクトでは、版の複製元 (親リポジトリ) が複数ある場合や、開発拠点 (子リポジトリ) が複数ある場合も考えられる。また、開発者が個人専用のローカルリポジトリを持つ場合は、親/子リポジトリだけではなく、孫リポジトリを持つ構成も考えられる。そこで、基本モデルを組み合せ、複雑な構成を取る場合の技術的課題を洗い出し、それらを解決する必要がある。

オープンソースソフトウェア開発のシュミレーション実験について

本システムの評価は、実際にオープンソースの開発プロジェクトで用い、検証する必要がある。しかし、プロトタイプシステムレベルでは、実際にソフトウェアの共同開発に使用することが難しい。また、これらの開発に用いられる WAN レベルのネットワーク環境の帯域を把握することも難しい。

そこで、まずトラヒックシェーパを用いて、仮想的に不均一なネットワーク帯域や、計算機資源への負荷を作り出し、シミュレーション環境の構築を行う。次に、実際の開発プロジェクトの構造やチーム構成を想定し、それに応じてプロトタイプシステムのリポジトリを構成し、実験を行うことを考えている。

ワークスペース管理について

本研究室の研究成果に、未来版機構 [21] がある。この特徴の１つに、プライベートなバージョン管理機能がある。各自のワークスペースにある成果物をバージョン管理することで、開発者ごとに専用の作業空間 (未来版空間) を提供し、独自の開発を進めることが

できる。また、未来版空間からリポジトリ (永続版空間) に成果を反映させる前に、プライベートバージョンを作成しその有効性を確認することで、先を見越した開発プロセス支援を可能にしている。

オープンソースソフトウェアの開発スタイルでは、リポジトリの分散構成だけではなく、個人レベルの開発支援を実現する未来版空間も有効であると考えられる。そこで、ワークスペースの拡張として、未来版空間を取り入れることを検討している。

プロジェクト管理ルールやプロセス支援機能について

本モデルに、オープンソースソフトウェアの管理モデルを反映させた場合、システムにもそれらを反映させる機構が必要である。そこで、プロジェクトの方針やルールといった、従来手動で行われてきたプロジェクト管理を CVS に反映可能にすることが考えられる。さらに、プロジェクトごとに必要な機能が異なるため、他の機能も容易に追加・プログラミング可能な拡張インタフェースを備えることも考えられる。また、リポジトリ間の一貫性維持機構を CVS サーバ間のプロトコルとして実装することで、CVS を集中型版管理モデルから分散型版管理モデルに移行させる枠組みを提供できる。

これらを実現するため、CVS をオブジェクト指向のアプリケーション・フレームワーク [22] として再構成し、上記のものを拡張機能として実現することを検討している。

第 9 章

おわりに

9.1 まとめ

本研究では、オープンソースソフトウェアの開発プロジェクトが目的の異なるサブプロジェクトを持つ場合に着目し、CVS のリポジトリでは複数の異なる運用方針や管理方針の共存が難しい問題点を取り上げた。これを解決するため、本研究では、以下の特徴を持つ CVS リポジトリの階層型分散構成法を開発した。

- 開発プロジェクトの要求によってリポジトリを柔軟に構成できる

  基本モデル (2 階層) を組み合せることで、複数の親/子リポジトリを持つ構成や、子リポジトリがさらに子を持つ多階層の構成を取ることができる。これにより、サブプロジェクトを複数持つ場合や、関連するプロジェクトを複数持つ場合、もしくは個人専用のリポジトリを持つ場合等、開発プロジェクトの要求に応じて、リポジトリを構成できる。

- リポジトリ間の関連を維持できる

  作業モデルとリビジョン番号の操作を用いて、リポジトリ間の版の矛盾を防ぎ、一貫性を保持できる。また、基本的な 4 つの関連を用いることで、必要に応じてリポジトリ間に関連付けを行える。これにより、サブプロジェクトの開発成果をメインプロジェクトに容易に反映させることや、メインプロジェクトの開発の一部をサブプロジェクトに委譲することが可能である。

- リポジトリごとに異なる管理方針や運用方針を設定できる

リポジトリ間の関連を維持した上で、リポジトリごとに独立したプロジェクトの管理方針や成果物の運用方針を設定できる。これより、メインプロジェクトと目的の異なるサブプロジェクトの開発プロセスを支援できる。

以上の特徴から、本研究により、オープンソースソフトウェアの開発プロセスを適切に支援可能な版管理システムを構成できる。

9.2 今後の課題

8 章で述べた通り、今後の課題として以下のものが残されている。

CVS リポジトリの階層型分散モデルに関する課題：

- 現状の開発スタイルの体系化と上位モデルの構築
- 分散型版管理に適したプロジェクト管理モデルの導入
- 分散型版管理に適したバージョンモデルの提案と導入

プロトタイプシステム distCVS に関する課題：

- プロトタイプシステム distCVS の実装
- 複数の親/子リポジトリや多階層のリポジトリ構成の実装
- CVS アーキテクチャの再構成と拡張機能 (プロジェクト管理、プロセス支援) の実現
- オープンソースソフトウェア開発のシミュレーション実験とその評価

謝辞

本研究を進めるにあたり、落水浩一郎教授には日頃よりご指導、ご助言いただき心より深く感謝致します。

また、藤枝和宏助手には、本学情報科学センターの業務で多忙の中、終始ご指導いただき厚く感謝致します。

本論文審査にあたり、篠田陽一教授、権藤克彦助教授にはご意見、ご指摘をいただき深く感謝致します。

そして、村越広享助手、服部哲助手、猪俣敦夫氏、特に藤田充典氏と早坂良氏には、本研究に関する貴重なご意見と関連情報や、励ましをいただき心より感謝致します。落水研究室の諸氏には、日頃より快適な研究環境の提供や、励ましをいただき感謝致します。

最後に、北陸での生活を支援していただいた両親や、常に励ましの言葉を掛けていただいた恩師の方々、友人達、遠く海の向こうのホストファミリに感謝致します。

参考文献

[1] The Apache Software Fundation http://www.apache.org/

[2] The Mozilla Organization http://www.mozilla.org/

[3] The GNOME project http://www.gnome.org/

[4] Linux Online! http://www.linux.org/

[5] The FreeBSD Project http://www.freebsd.org/

[6] The NetBSD Project http://www.netbsd.org/

[7] Brian Berliner: CVS II: *Parallelizing Software Development*. White paper, Prisma, Inc., Colorado Springs, CO, 1990.

[8] Karl Fogel: *Open Source Development with CVS: Learn How to Work With Open Source Software*. Coriolis Group Books, October 1999.(K. ホーゲル 著 (でびあんぐる 監訳, 竹内里佳 訳): *CVS* –**バージョン管理システム**–, Ohmsha, 2000 年 5 月)

[9] Walter F. Tichy: *RCS–A System for Version Control*. Software–Practice and Experience, Vol.15, No.7, 1985, pp 637-654.

[10] *CVS Client/Server*. Included in CVS-1.10.6 Distribution, 1999.

[11] David B.Leblang.: *The CM Challenge: Configuration Management that Works*. Configuration Management, 1994, pp1-37.

[12] Larry Allen, Gary Fernandez, Kenneth Kane, David Leblang, Debra Minard, John Posner: *ClearCase MultiSite: Supporting Geographically-Distributed Software Development*. SCM lecture notes LNCS 1005, pp 194-214.

[13] Rational ClearCase http://www.rational.com/products/clearcase/index.jsp

[14] BitKeeper http://www.bitkeeper.com/

[15] FSMLabs http://fsmlabs.com/

[16] Linux kernel 2.5 is hosted at BitMover http://linux25.bkbits.net/

[17] CVSup http://www.polstra.com/projects/freeware/CVSup/

[18] arch http://www.regexps.com/#arch

[19] The CITRUS Project http://citrus.bsdclub.org/

[20] HORB http://www.horb.org/

[21] 堀 雅和:**ソフトウェア分散開発に適した中間成果物の管理法に関する研究**, 北陸先端科学技術大学院大学 博士論文, 1998 年 12 月

[22] 早坂 良，藤枝 和宏，落水 浩一郎：**オープンソース開発プロジェクトのための** *CVS* **機能拡張法**, 平成 13 年度電気関係学会北陸支部連合大会，pp315, 2001

本研究に関する発表論文

[1] 嶋田 大輔, 藤枝 和宏, 落水 浩一郎: 階層的分散機能を持つ版管理システムに関する研究, 平成 13 年度 電気関係学会北陸支部連合大会, pp.316, October 2001.

[2] 嶋田 大輔, 藤枝 和宏, 落水 浩一郎: オープンソースソフトウェア開発プロセスに適した CVS リポジトリの階層的分散構成のモデル化と設計, 情報処理学会ソフトウェア工学研究会, 2002–SE–136, pp.49–56, March 2002.