修 士 論 文

オープンソースソフトウェア開発に適したリポジトリ分散の支援機構に関する研究

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

中島 健至

2005 年 3 月

修 士 論 文

オープンソースソフトウェア開発に適したリポジトリ分散の支援機構に関する研究

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　片山 卓也 教授
審査委員　鈴木 正人 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

310076 中島 健至

提出年月: 2005 年 2 月

目 次

概 要

既存の構成管理システムや外部ツールのリポジトリ分散機能では, 必要以上に成果物と変更履歴を転送するため, 開発プロジェクトに応じて柔軟なディレクトリ構造やリビジョンツリー構造をとることができない. また, 転送の方向が単方向で開発成果を反映できない場合がある. そのため, 既存のオープンソースソフトウェア開発に見られる多様なリポジトリの分散構成をサポートできない. そこで本研究では、構成の異なるリポジトリ間で成果物と変更履歴を必要な粒度で転送する機構を元に、オープンソースソフトウェア開発を適切に支援可能なリポジトリ分散の支援機構を実現する。本研究が提案する機構により, 開発プロジェクトは, 現在使用している CVS で独自のリポジトリの構築と運用ができる.

図 目 次

第1章　はじめに

1.1　背景

計算機とネットワーク技術の発達により, 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開発がある. この開発プロジェクトでは, Apache [13] や FreeBSD [11] のように規模が大きくなると複数の子プロジェクトを持つことがある. これらは, ネットワーク的に遠く離れ, プロジェクトの管理方針が大きく異なっている場合がある. 目的の異なるプロジェクト同士が１つのリポジトリに混在した場合, 子プロジェクトは親プロジェクトの管理方針を強制される. これを回避するには, 成果物と変更履歴を管理するリポジトリは, 分散構成を取れることが必要となる.

既存の構成管理システムや外部ツールには, リポジトリの分散機能を持つものがある. しかし, これらの構成管理システムや外部ツールでは, 必要以上に成果物と変更履歴を転送するため, 開発プロジェクトに応じて柔軟なディレクトリ構造やリビジョンツリー構造をとることが出来ない. また, 転送の方向が親から子への単方向の場合がある. そのため, オープンソースソフトウェア開発の中で独自のリポジトリの構築と運用を行っている Cygwin [6] や KAME [8] などの開発プロジェクトを支援することが困難である.

1.2　目的

本研究の目的は, オープンソースソフトウェア開発プロジェクトを適切に支援する独自のリポジトリの構築と運用が可能な支援機構を実現することにある. 本研究では, オープンソースソフトウェア開発で広く使用されている CVS [1] を対象としてクライアントシステムの設計と実装を行う. 本研究が提案する機構により, 開発プロジェクトは, CVS の変更や専用のサーバプログラムを必要とせず, 独自のリポジトリの構築と運用ができる.

本研究が実現する機構は次の特徴を持つ.

- **任意の粒度のデータ転送**
  任意の粒度において双方向でデータ転送を可能とする. 粒度とはリビジョン, ファイル, ディレクトリ, リポジトリ全体のことを指す.

- **管理方針の分離**
  転送元のリポジトリの構成, つまりディレクトリ構造, ファイル名, リビジョンツリー

構造に制約を受けず, これらを元に転送先で独自の構成のリポジトリを構築可能とする.

これらの特徴を, 細粒度の操作とデータ転送管理部により実現を行う. 細粒度の操作は転送元のリビジョンツリーから差分またはファイル全体を取得して転送先に 1 つのリビジョンを作成する. この細粒度の操作は, root と exchange という 2 つの操作からなる. この細粒度の操作により転送元のリポジトリと異なるリビジョンツリーを構成できる. データ転送管理部によりリポジトリ間のファイル名の対応やディレクトリ構造の対応を行う. また, このデータ転送管理部は細粒度の操作, 上位操作, ディレクトリの作成を繰り返し実行することでリビジョン単位からリポジトリ全体までのデータ転送を処理する.

本研究では, 細粒度の操作を用いてオープンソースソフトウェア開発において一般的に必要なリビジョンツリーの上位操作の記述を行う.

- **graft** リビジョンツリーの一部を転送する
- **replicate** リビジョンツリー全体を複製する
- **add** トランクまたはあるブランチの開発進展部分を転送する
- **add_tree** リビジョンツリー全体の開発進展部分を転送する

また, FreeBSD などのオープンソース開発プロジェクトで用いられている CVSup [2] の exact と non_exact という 2 つの動作モードの記述を行う.

- **exact** 他拠点のリビジョンツリーと一致する
- **non_exact** 独自のブランチを維持したまま, 他拠点の差分を取り込む

その上で, 本研究の細粒度の操作と上位操作を用いた場合, CVSup を用いる開発者に開発の自由度, 双方向の転送を提供できる例を示す.

本研究が実現する機構をオープンソースソフトウェア開発プロジェクトの Cygwin と KAME に適用するとリポジトリの部分複製と成果物と変更履歴の親プロジェクトへの反映が支援できることを示す.

1.3 論文の構成

本論文の構成は, 以下の章から構成される.

第 2 章では, まず, 分散共同開発にリポジトリの複製を用いる理由を述べる. 次に, オープンソースソフトウェア開発でみられるリポジトリの分散形態の例を示す. そして, 既存の構成管理システムと外部ツールの問題点を説明する.

第 3 章では, 上記の問題を解決する既存の構成管理システムに適応性を持たせる機構について述べ, 第 4 章でシステムの実装環境と動作手順について述べる.

第 5 章では, 本研究の考察として, 細粒度の操作で CVSup のリビジョンツリーの 2 つの動作モードを記述する. これにより, 本研究が提案する細粒度の操作の記述力の検討を行う. また, CVSup の問題点と本研究の機構を用いた場合の解決例を示す.

第 6 章では, 適用例として Cygwin プロジェクトと KAME プロジェクトを元に説明する.

最後に 7 章で, 本研究の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における構成管理システム

本章では, まず, 開発プロジェクトがリポジトリの複製を用いて開発を行う理由を述べる. 次に, オープンソースソフトウェア開発でみられるリポジトリの分散の例を示す. 最後に, 既存の構成管理システムを用いてリポジトリを分散させた場合の問題点を説明する.

2.1　分散共同開発にリポジトリの複製を用いる理由

オープンソースソフトウェア開発に代表される地理的に分散したソフトウェア開発では, 開発の規模が大きくなると複数の子プロジェクトが派生する.

多くの場合, プロジェクトごとに, 開発の規模や方針が大きく異なるため, 親プロジェクトがリポジトリに対して適用している管理方針は, 子プロジェクトを適切に支援していない. そのため, 子プロジェクトは, 親プロジェクトの方針によって制約を受けてしまう.

子プロジェクトは, この制約を回避するために, 親プロジェクトのリポジトリの複製を行い, 複製したリポジトリに独自の管理方針を適用する.

2.1.1　管理方針

リポジトリの管理者は, リポジトリを管理するにあたり, その管理方針を策定する. 以下に一般的な管理方針をあげる.

・メンバー管理方針
ユーザー毎，グループ毎の追加と削除を行う.
・アクセス制御方針
ユーザー毎，グループ毎のファイルやディレクトリのアクセスを制御する.
・コミット前の検査方針
インデントのつけ方といったコーディングスタイル検査，コンパイル検査等を行う.
・ディレクトリ構造
必要なディレクトリ，ディレクトリの階層構造を決定する.
・ファイル名のつけ方
例えば，どのように名前をつけていくかといった命名規則があげられる.

・リビジョンツリー構造
いままでどのように変更され，また，これからどのように変更を加えていくか決定する．

2.1.2 個別にリポジトリを保有することによる、独自の管理方針の適用

目的の異なるプロジェクト同士が１つのリポジトリに混在した場合, 子プロジェクトは親プロジェクトの管理方針を強制される．子プロジェクトは, この制約を回避するために, 個別にリポジトリを保有し, 独自の管理方針を適用する．

2.2 オープンソースソフトウェア開発のリポジトリ分散

オープンソースソフトウェア開発では, 親リポジトリの部分的な複製を行って開発が進められる開発形態がある．ここでは, そのような開発のリポジトリ分散の例を述べる．

2.2.1 FreeBSD Japan

FreeBSD Japan [12] は日本語版の FreeBSD の開発を行う．そのために, 親プロジェクトのリポジトリから部分的複製を行い, 開発を進めている．FreeBSD Japan の開発成果は適宜, FreeBSD に送られる．FreeBSD Japan は FreeBSD と双方向で関係を保ちながらも, 各種ソフトウェアの開発, 日本語版ドキュメントの作成と翻訳を行うために, 独自のリポジトリ構成を持つ．FreeBSD のディレクトリ構造と FreeBSD Japan のディレクトリ構造を以下に掲載する．

・FreeBSD リポジトリのディレクトリ構造

```
CVSROOT/
CVSROOT-doc/
CVSROOT-ports/
CVSROOT-projects/
CVSROOT-src/
distrib/
doc/
ports/
projects/
root/
src/
www/
```

・FreeBSD Japan リポジトリのディレクトリ構造

```
ACPI/
CVSROOT/
ESS/
PAO/
PAO3/
QandA/
distrib/
doc-jp/
man-jp/
newconfig/
pc98/
www/
```

2.2.2 Cygwin

Cygwin [6] は, UNIX 系 OS では一般に広く普及している GNU プロジェクトによる開発ツール群を Windows 環境用に移植するプロジェクトである．図 2.1 に示す Cygwin プロジェクトは MinGW [7] リポジトリの w32api を複製して利用しているため, Cygwin プロジェクトは MinGW の子プロジェクトと考えることができる．Cygwin プロジェクトは MinGW リポジトリの w32api を複製して利用しているだけでなく, w32api のバグなどの修正を MinGW 側へ転送を行う．このように, Cygwin は独自のリポジトリ構造を保ちながら, MinGW と双方向でデータ転送を行っている．

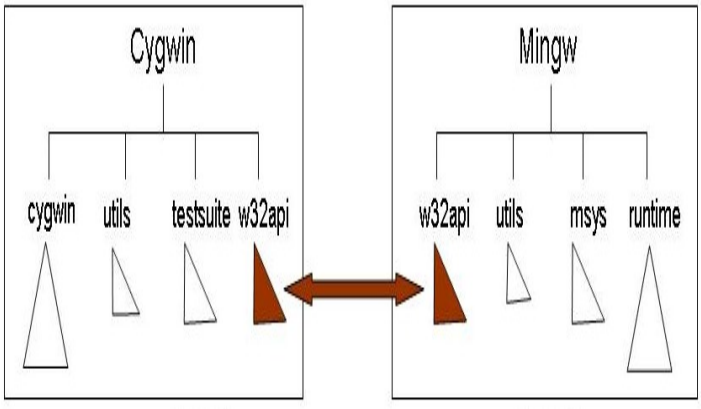

図 2.1: Cygwin

2.2.3 KAME

KAME [8] は FreeBSD [11] リポジトリ, OpenBSD [9] リポジトリ, NetBSD [10] リポジトリの部分的複製をおこない開発を進めている. このため, KAME は FreeBSD, OpenBSD 及び NetBSD を親プロジェクトとして持つ, 子プロジェクトと考えることができる. 図 2.2 に示す KAME は FreeBSD リポジトリの部分的複製を/freebsd に, OpenBSD リポジトリの部分的複製を/openbsd に, NetBSD リポジトリの部分的複製を/netbsd に取り込んでいる. このため, 親プロジェクトとは異なるディレクトリ構造を持つ. KAME は IPv6 対応 TCP/IP ソフトの開発を行い, この成果は適宜, FreeBSD, OpenBSD 及び NetBSD のリポジトリに格納される. KAME は独自のリポジトリ構造を保ちつつ, 複数の親プロジェクトと双方向でデータ転送を行っている.

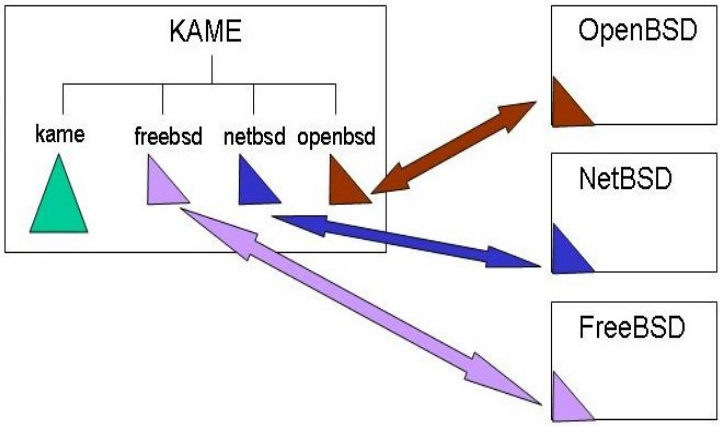

図 2.2: KAME

2.3 既存の構成管理システムと外部ツールの問題点

ここでは既存の構成管理システムと外部ツールの問題点を説明する.

2.3.1 CVS

CVS は [1], オープンソースソフトウェア開発等で広く利用されている構成管理システムである. CVS は, 構成管理のバージョン管理機能だけを提供する.

CVS は, リポジトリの複製機能をサポートしていないため, 多くの開発プロジェクトは, 複製元となる他拠点のリポジトリから手動で複製を行う. また, CVS には, 他拠点と自拠点のリポジトリ間で開発の成果物を相互に反映する機能がない. そのため, 他拠点の開発

の進展部分を手動で自拠点に取り込む必要あり, その結果, リポジトリを管理、維持するコストが高くなる.

リポジトリ分散の外部ツールとして CVSup [2] がある. CVSup は CVS リポジトリの分散を支援する機能を提供するツールである.

CVSup で開発を行う場合には以下の問題がある. CVSup では, 複製元の他拠点のトランクとブランチは, 全て自拠点に反映される. そのため, 自拠点での開発は他拠点に存在しないブランチ上のみとなり, 自拠点は異なるリポジトリ構造をとることが制限される. また, CVSup のミラーリングは, 単方向いわゆる pull モデルのみであり, 自拠点の成果物と変更履歴を他拠点に反映することはできない.

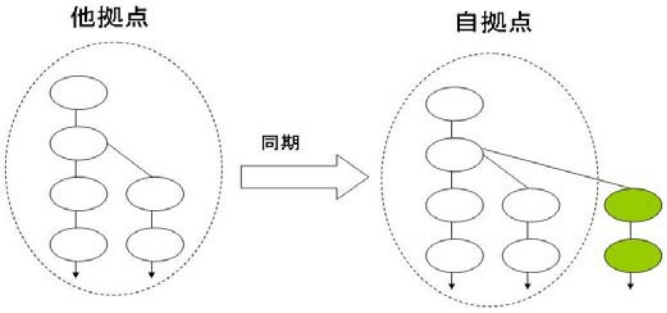

図 2.3: CVSup

2.3.2 ClearCase

ClearCase [4] は, リビジョン管理機能の他に, チームサポート, ビルドサポート, プロセス管理など多くの機能を有する構成管理システムである. この ClearCase はリポジトリの複製を支援する機能を提供する外部ツールの ClearCase Multisite [5] を使うことにより, WAN レベルでリポジトリの分散配置を行う.

ClearCase Multisite の問題は, 異なるリポジトリ構造をとることが制限される点にある. ClearCase Multisite は, 複製元の他拠点の変更は, 常に自拠点に反映される. 図 2.4 にその様子を示す. そのため, 自拠点での開発は他拠点で許可されたブランチ上でのみ行え, 自拠点は異なるリポジトリ構造をとることが制限される.

2.3.3 Bitkeeper

BitKeeper [3] は, 分散型の構成管理システムである. リポジトリの分散は, clone コマンドを用いて複製の対象とするリポジトリから複製を行うことにより実現される. 他拠点と

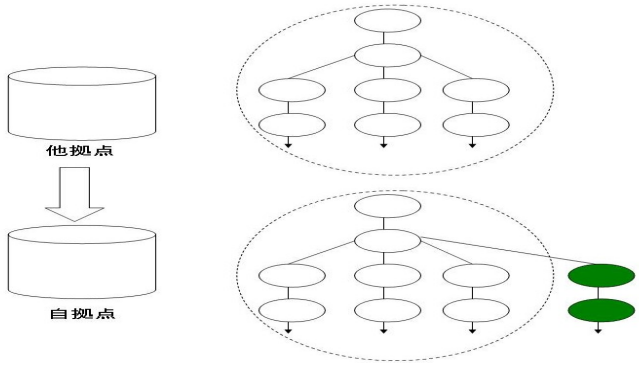

図 2.4: ClearCase Multisite

自拠点のリポジトリ間での開発の成果物と変更履歴の転送は, 他拠点の成果物を自拠点に取り込む pull コマンドと, 自拠点の成果物を他拠点に送る push コマンドによって行われる. その転送方式を図 2.5 に示す.

BitKeeper の複製機能における問題点は, 異なるリポジトリ構造をとることが制限される点である. BitKeeper では, ChangeSet と呼ばれる単位により, 複数のファイルに対する変更をまとめて１つの変更として管理する. pull または push コマンドは, すべての ChangeSet を他方のリポジトリへ送る. そのため, 例えば新しい一部の ChangeSet だけをリポジトリ間で転送できない. その結果, 自拠点のリポジトリは, 他拠点が所有するすべての成果物と変更履歴を共有することになり, 他拠点の完全なコピーになる. このため, 自拠点は異なるリポジトリ構造をとることができない.

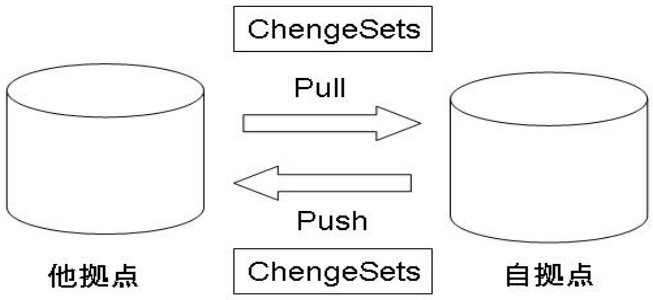

図 2.5: Bitkeeper

第3章　リポジトリ分散の支援機構

本研究は, オープンソースソフトウェア開発を支援することを目的としている. そのために, 独自のリポジトリの構築と運用が可能な支援機構を提案する. 本研究が実現する機構は以下の特徴を持つ.

- **任意の粒度のデータ転送**
  任意の粒度において双方向でデータ転送を可能とする. ここで粒度とはリビジョン, ファイル, ディレクトリ, リポジトリ全体のことを指す.
- **管理方針の分離**
  転送元のリポジトリの構成, つまりディレクトリ構造, ファイル名, リビジョンツリー構造に制約を受けず, これらを元に転送先で独自の構成のリポジトリを構築可能とする.

3.1　概要

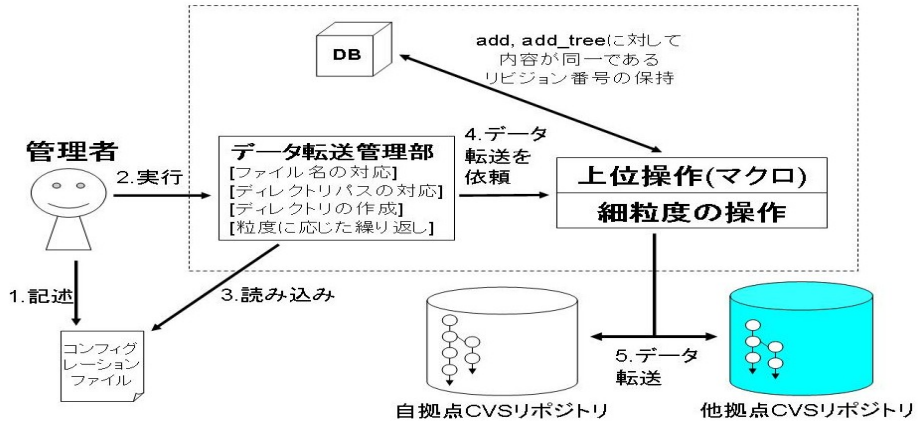

図 3.1: リポジトリ分散の支援機構

リポジトリ分散の支援機構は, 図 3.1 に示す枠内の細粒度の操作, 上位操作, データ転送管理部からなる. ここで, 細粒度とは 1 つのリビジョンを指す. 細粒度の操作は転送元の

リビジョンツリーから差分またはファイル全体を取得して転送先に 1 つのリビジョンを作成する. この細粒度の操作は, root と exchange という 2 つの操作からなる. この細粒度の操作により転送元のリポジトリと異なるリビジョンツリーを構成できる. 本研究が提案する細粒度の操作とそれから構成された上位操作により, リポジトリ間のリビジョンツリーの様々なデータ転送記述が可能となる.

上位操作はリポジトリ間の同一のリビジョンの情報を提供するデータベースや CVS の log 情報を解析し, その情報に基づいて細粒度の操作を呼び出す. 上位操作により, 効率のよいデータ転送が可能となる.

データ転送管理部はリポジトリ間のファイル名の対応, ディレクトリ構造の対応を行う. また, リビジョン単位からリポジトリ全体までのデータ転送を細粒度の操作, 上位操作, ディレクトリの作成を繰り返し実行させることで実現する. 図 3.2 から図 3.10 に本機構が提供するデータ転送のイメージを示す.

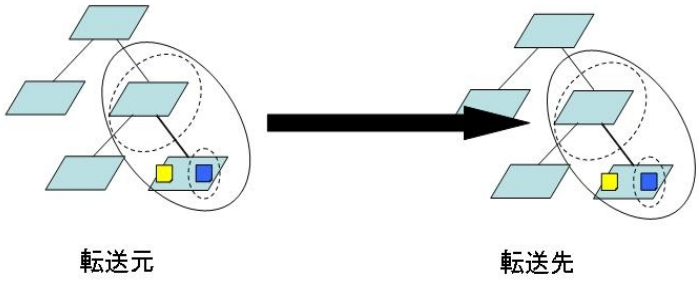

図 3.2: 任意の粒度のデータ転送

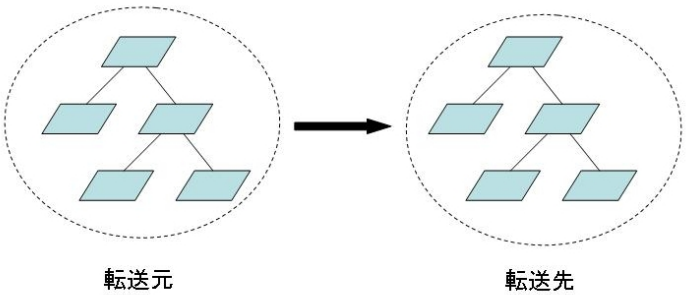

図 3.3: リポジトリ全体のデータ転送

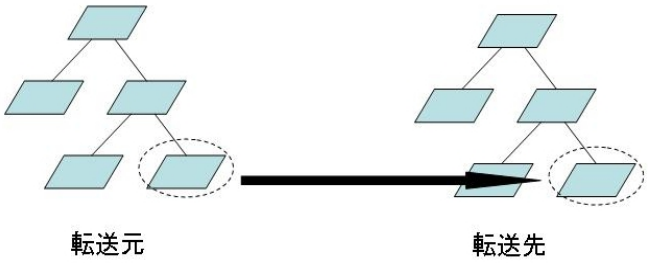

図 3.4: ディレクトリ単位のデータ転送

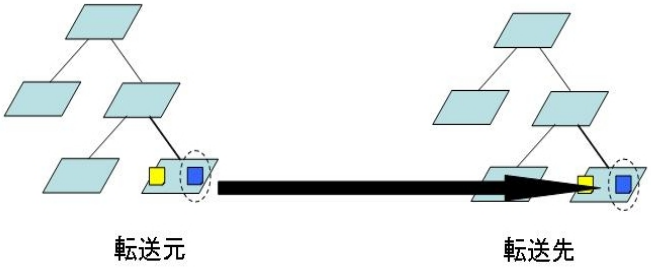

図 3.5: ファイル単位のデータ転送

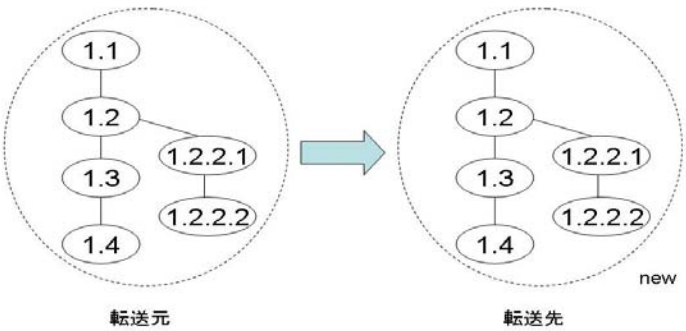

図 3.6: リビジョンツリー全体のデータ転送

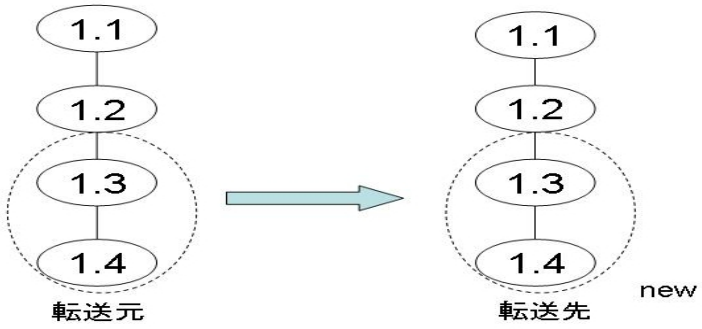

図 3.7: 部分的なリビジョンツリーのデータ転送

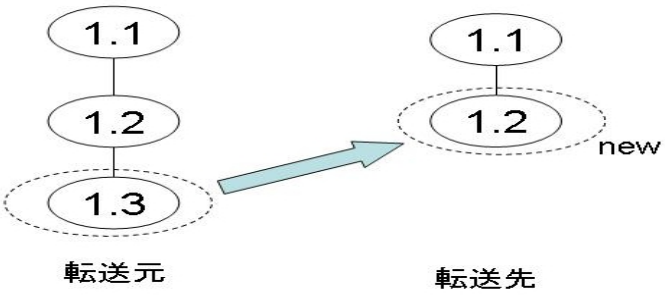

図 3.8: 1 つのリビジョンのデータ転送

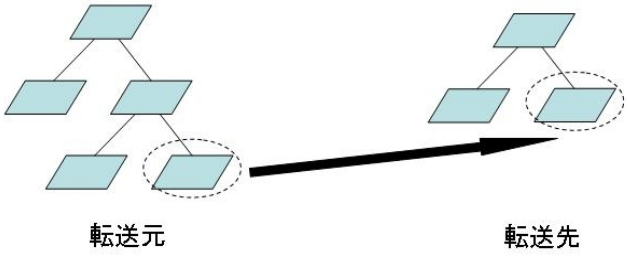

図 3.9: 異なるディレクトリ間のデータ転送

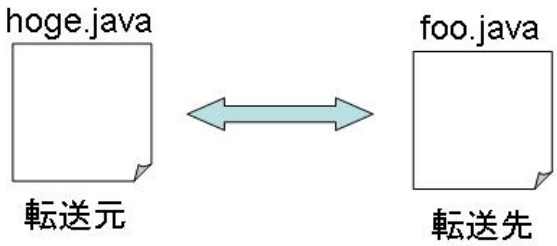

図 3.10: 異なるファイル名間のデータ転送

3.2 データ転送管理部

データ転送管理部は図 3.11 に示すように, リポジトリ間のファイル名の対応, ディレクトリ構造の対応を行う. また, リビジョン単位からリポジトリ全体までのデータ転送を細粒度の操作, 上位操作, ディレクトリの作成を繰り返し実行させることで実現する. データ

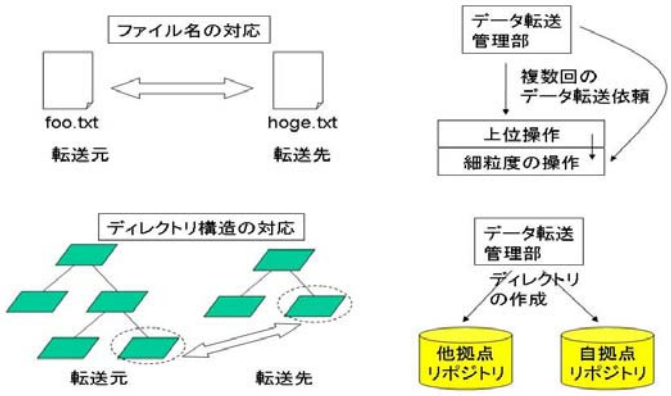

図 3.11: データ転送管理部の概要

転送管理部はリポジトリの管理者が記述したデータ転送記述 (コンフィグレーションファイル) を元に, リポジトリ間の関係を決定する. ここでリポジトリ間の関係とは, ディレクトリ構造の対応, ファイル名の対応である. これらにより, 自拠点のどのディレクトリにあるファイルが他拠点のディレクトリにあるファイルとリビジョンツリーの操作を行うか決定できる. リポジトリ間の関係を決定したら, データ転送記述を元に, 細粒度の操作または上位操作を呼び出す.

このデータ転送管理部はデータ転送記述に基づいて必要に応じたディレクトリの作成,

細粒度の操作, 及び上位操作を繰り返し呼び出す.

なお, 上記で述べたデータ転送記述は拠点数, 粒度, ディレクトリ構造の対応, ファイル名の対応, リビジョンツリーの操作の要素がある.

拠点数は自拠点のリポジトリと関係する全ての他拠点である. 1 つの拠点から複数拠点まで記述できる.

粒度はファイル単位からリポジトリ全体までを記述できる. 粒度の指定により, データ転送管理部はファイル数に応じて繰り返す回数を決定する. 例えば, ディレクトリの指定で, そのディレクトリに 3 つのファイルがあったら 3 回, リビジョンツリーの操作を実行する.

ディレクトリ構造の対応は自拠点と他拠点のディレクトリ構造を記述する. 同一のディレクトリ間も異なるディレクトリ間も記述できる.

ファイル名の対応は自拠点と他拠点のファイル名を記述する. 同一のファイル名間も異なるファイル名間も記述できる.

リビジョンツリーの操作は root, exchange, graft, replicate, add, add_tree を記述する. 複数回の記述も可能である.

3.3 細粒度の操作

細粒度の操作は root と exchange からなる. 本研究が提案するこれらの操作を組み合わせることにより, リポジトリ間の様々なリビジョンツリーのデータ転送操作を記述できる.

3.3.1 root

root は転送元のあるリビジョンの file を元に, 転送先に版管理の対象となる 1.1 を作成する. なお, 転送先にリビジョンツリーがすでに存在する場合は, 全てのリビジョンツリーを削除して, 新たに 1.1 を作成する. root は 2 つの引数を持つ.

・root 方向 リビジョン番号
方向: come/go
転送元のリビジョン番号: rev

1 つ目の引数は方向である. 方向は come と go を選択できる. come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる. 2 つ目の引数は転送元のリビジョン番号の指定である. この指定されたリビジョンを転送して, 転送先に版管理の対象となる 1.1 を追加する.

root を図 3.12 を用いて説明する. ここでは, 方向は特に指定しないが, come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる.

いま, 転送元ではトランク 1.4, ブランチ 1.2.2.2 まで開発が進行している. root を実行すると, 転送元の 1.2.2.2 を複製して転送先に版管理の対象となる 1.1 を作成する. 転送先の 1.1 は 1.1(元)+Δ a1+Δ a4 となり, 転送元の 1.2.2.2 と転送先の 1.1 は同一の内容となる.

root は上書きを行うことができる. 上書きを行った場合, 転送先のリビジョンツリーは全て削除される. その例を図 3.13 を用いて示す. 今, 転送元ではトランク 1.4 まで開発が進行している. 転送先ではトランク 1.3 まで開発が進行している. この状態で root を実行すると, 転送先にあったリビジョンツリーは全て削除され, 転送元の 1.4 を複製して転送先に版管理の対象となる 1.1 を作成する. 転送先の 1.1 は 1.1(元)+Δ a1+Δ a2+Δ a3 となり, 転送先の 1.1 は転送元の 1.4 と同一の内容となる.

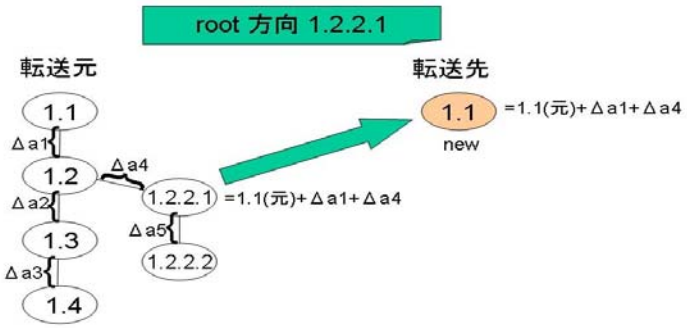

図 3.12: root

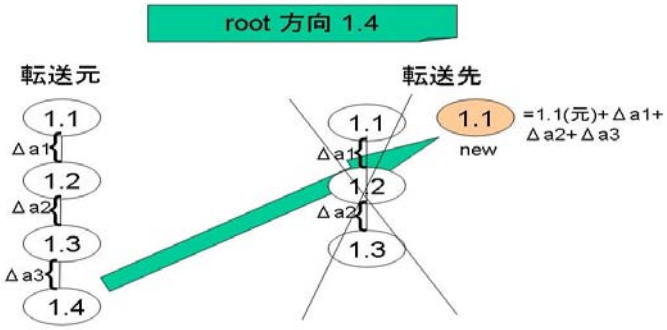

図 3.13: root 上書き

3.3.2 exchange

exchange は転送元の差分または, ファイル全体を元に転送先の指定したリビジョンの先に新たに 1 個のリビジョンを作成する操作である. exchange は 5 つの引数を持つ.

・exchange 方向 転送方式 転送元のデータ指定 転送先のリビジョン番号 ブランチ名
方向:come/go
転送方式:delta/file
転送元のデータ指定:rev-rev/rev
転送先のリビジョン番号:rev
ブランチ名:branch_name

1 つ目の引数は方向である. 方向は come と go を選択できる. come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる.
2 つ目の引数は転送方式である. delta(差分) と file(ファイル全体) が選択できる.
3 つ目の引数は転送元のデータ指定である. 転送方式が delta ならリビジョン番号とリビジョン番号を指定する. 転送方式が file なら, リビジョン番号を指定する.
4 つ目の引数は転送先のリビジョン番号である. この先に新たに 1 個のリビジョンを作成する.
5 つ目の引数はブランチ名である. この引数がある場合, 4 つ目の引数で指定したリビジョンにブランチを作成し, そのブランチ上に新たなリビジョンを作成する.
exchange は 4 つの転送を可能としている. そして 4 つの場合それぞれがリビジョンツリーの上書きを行うことが出来る.
1 つ目の, exchange delta 転送を図 3.14 を用いて説明する. 転送元ではトランク 1.4, 転送先ではトランク 1.3 まで開発が進行している. exchange を実行すると転送元の Δ a2 と Δ a3 を転送して, 転送先にリビジョン 1.4 が作成される. 転送先のリビジョン 1.4 は 1.1(先)+ Δ b1+Δ b2 ＋Δ a2+Δ a3 となる.

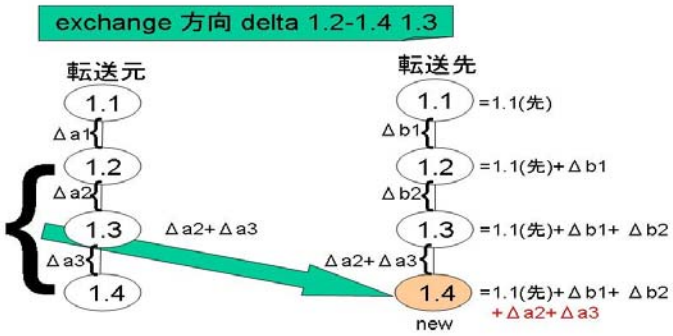

図 3.14: exchange delta 転送

2 つ目の, exchange file 転送を図 3.15 を用いて説明する. 転送元ではトランク 1.4, 転送先ではトランク 1.3 まで開発が進行している. exchange を実行すると転送元のリビジョン 1.4 を転送して, 転送先にリビジョン 1.4 が作成される. 転送先のリビジョン 1.4 は 1.1(元)+ Δ a1+Δ a2 ＋Δ a3 となり, 転送元の 1.4 と転送先の 1.4 は同一の内容となる.

3 つ目の exchange ブランチ作成 delta 転送を図 3.16 を用いて説明する. 転送元ではトランク 1.4, 転送先ではトランク 1.3 まで開発が進行している. exchange を実行すると転送元の Δ a2 と Δ a3 を転送して, 転送先にリビジョン 1.3.2.1 が作成される. 転送先のリビジョン 1.3.2.1 は 1.1(先)+Δ b1+Δ b2 ＋Δ a2+Δ a3 となる.

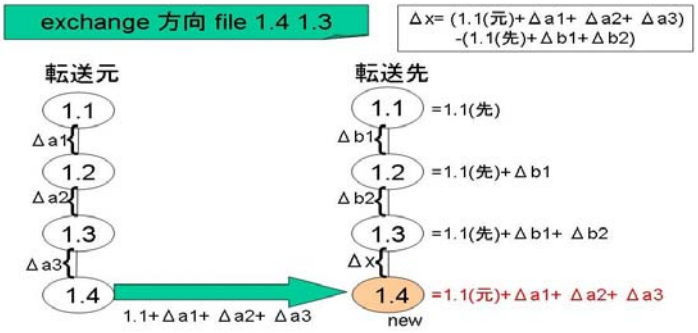

図 3.15: exchange file 転送

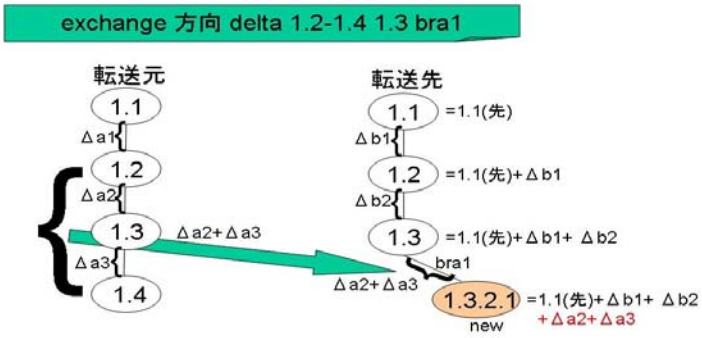

図 3.16: exchange ブランチ作成 delta 転送

4 つ目の exchange ブランチ作成 file 転送を図 3.17 を用いて説明する．転送元ではトランク 1.4, 転送先ではトランク 1.3 まで開発が進行している．exchange を実行すると転送元のリビジョン 1.4 を転送して, 転送先にリビジョン 1.3.2.1 が作成される．転送先のリビジョン 1.3.2.1 は 1.1(元)+Δ a1+Δ a2 ＋Δ a3 となり, 転送元の 1.4 と転送先の 1.3.2.1 は同一の内容となる．

以上, 4 つの場合それぞれが上書きを行うことが出来る．その例を図 3.18 を元に説明する．転送元では 1.4, 転送先では 1.4 と 1.3.2.1 まで開発が進行している．exchange を実行するとまず, 1.1 より下に存在する全てのリビジョンを削除する．この場合, 1.2, 1.3, 1.4, 1.3.2.1 が削除される．その後, 転送元のΔ a1 とΔ a2 を転送して, 転送先にリビジョン 1.2 が作成される．転送先のリビジョン 1.2 は操作前は 1.1(先)+Δ b1 であったが, 操作後は 1.1(先)+Δ a1+Δ a2 となる．

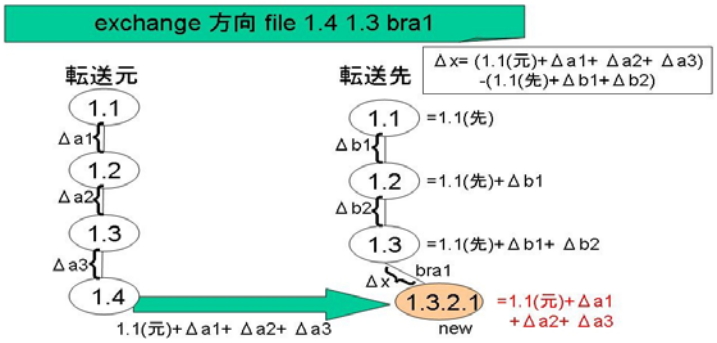

図 3.17: exchange ブランチ作成 file 転送

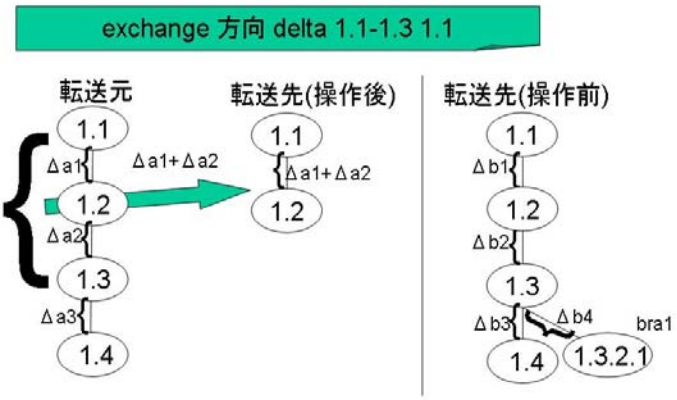

図 3.18: exchange 上書き

3.4 上位操作

上位操作はデータ転送に細粒度の操作を用いる. 開発プロジェクトは, この上位操作を用いることでリビジョンツリーの部分転送, 全体転送を効率よく行うことが出来る. 上位操作は graft, replicate, add, add_tree からなる.

このうち, 転送元の開発進展部分だけを転送先に反映するため add, add_tree は DB を用いる. 表 3.1 の (1) と (2) を対にして保持することで, そのリビジョンからの開発進展部分を取り込むことができる. (3),(4),(5),(6),(7) は (1)+(2) を補助するためにある. 自拠点のどこのディレクトリに存在するファイルが, どの他拠点のリポジトリのどのディレクトリのどのファイルと同じかということを調べる.

表 3.1: データベースのデータ構造

(1) 自拠点のリビジョン番号
(2) 他拠点のリビジョン番号
(3) 自拠点ファイルが属するディレクトリ名
(4) 他拠点ファイルが属するディレクトリ名
(5) 自拠点のファイル名
(6) 他拠点のファイル名
(7) 他拠点のリポジトリの場所

表 3.2 にデータベースの情報例を示す. この例では, 自拠点の mywork/EleControl.java の 1.3 が他拠点の elevator/ElevatorControl.java の 1.4 と同じであることを表している.

表 3.2: データベースの情報例

1.3
1.4
mywork
elevator
EleControl.java
ElevatorControl.java
:pserver:i310076@i451.jaist.ac.jp:/cvsroot/econ03

3.4.1 graft

graft は転送元のリビジョンツリーの一部を転送先の指定したリビジョンに接木する.

・graft 方向, 転送方式, 転送元のデータ指定, 転送先のリビジョン番号, ブランチ名
方向:come/go
転送方式:delta/file
転送元のデータ指定:rev-rev
転送先のリビジョン番号:rev
ブランチ名:branch_name

graft は 2 つの特徴を持ち, それぞれの場合で delta 転送, file 転送を行うことが出来る. 1 つ目の特徴を図 3.19 を元に説明する. 方向は指定していないが come ならば, 転送元が他

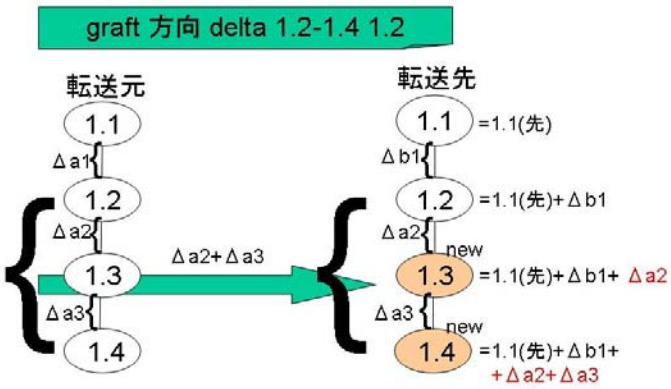

図 3.19: graft

拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる. 今, 転送元で 1.4 まで, 転送先で 1.2 まで開発が進んでいる. graft を実行すると, 内部で exchange が 2 回実行され, 転送先にリビジョン 1.3, リビジョン 1.4 が作成される. なお, exchange は以下のように実行される.
exchange 方向 delta 1.2-1.3 1.2
exchange 方向 delta 1.3-1.4 1.3

2 つ目の特徴を図 3.20 を元に説明する. 方向は指定していないが come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる. 今, 転送元で 1.4 まで, 転送先で 1.2 まで開発が進んでいる. graft を実行すると, 転送先にリビジョン 1.3.2.1, リビジョン 1.3.2.2 が作成される. なお, exchange は以下のように実行される.

exchange 方向 delta 1.2-1.3 1.2 bra1
exchange 方向 delta 1.3-1.4 1.2.2.1

この graft は転送元の n 個の delta/file の指定があると exchange を n 回行う．なおこの例では, 2 つの delta が指定されており, 内部で exchange が 2 回実行されている．

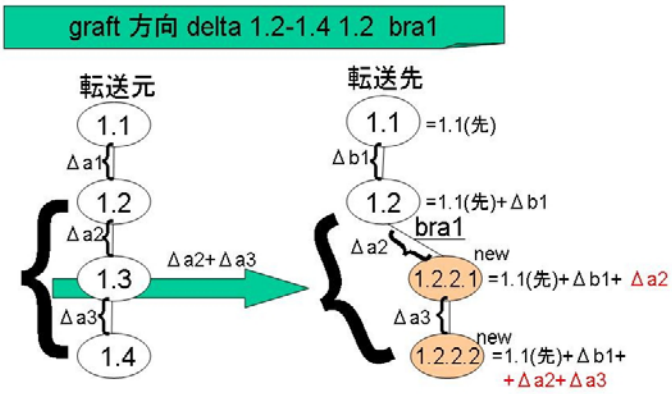

図 3.20: ブランチ作成 graft

3.4.2 replicate

replicate は転送元のリビジョンツリー全体を転送先に複製する.

・replicate 方向
方向 come/go

図 3.21 を元に replicate を説明する. 方向は指定していないが come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる. いま, 転送元では 1.4, 1.2.2.2 まで開発が進んでいる. replicate を実行すると, 転送先に転送元と完全に同じリビジョンツリーが作成される.

このオペレーションは, 転送元に存在するリビジョンを n 個とすると, root を 1 回, exchange を n-1 回行う. 図 3.21 の場合, root を 1 回, exchange を 5 回行う. なお, root と exchange は以下のように実行される.
root 方向 1.1
exchange 方向 delta 1.1-1.2 1.1
exchange 方向 delta 1.2-1.3 1.2
exchange 方向 delta 1.3-1.4 1.3
exchange 方向 delta 1.2-1.2.2.1 1.2 bra1
exchange 方向 delta 1.2.2.1-1.2.2.2 1.2.2.1

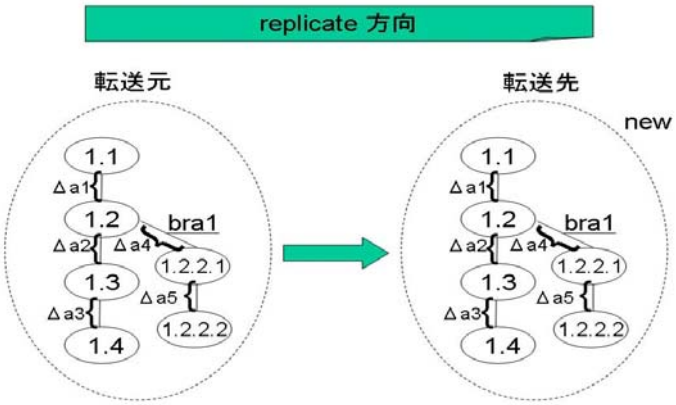

図 3.21: replicate

3.4.3 add

add は自拠点の枝 (トランクまたは, あるブランチ) と, 他拠点の枝 (トランクまたは, あるブランチ) を指定すると, 転送元の開発進展部分を転送先に接木する. この操作はデータベースを用いて実行を行う.

・add 方向, 転送方式, 自拠点の枝, 他拠点の枝
方向 come/go
転送方式 delta/file
自拠点の枝 trunk/branch_name
他拠点の枝 trunk/branch_name

add を図 3.22 を元に説明する. なお, 方向は特に指定しないが, come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる. 今, 転送元では 1.4 まで, 転送先では 1.2 まで開発が進んでいる. ここで転送元の 1.2 と転送先の 1.2 は同一の内容であり, これは DB に記録されている.

add を実行すると, DB から転送先のリビジョン番号 1.2 と同一の内容である転送元のリビジョン番号 1.2 が取得される. すると, 転送元の 1.2 から 1.4 の差分 Δ a2, Δ a3 が転送され, 新たにリビジョン 1.3, 1.4 が転送先に作成される. add は転送元の枝で開発が進展したリビジョンを L 個とすると, exchange を L 回行う. なお, このデータ転送では exchange が 2 回実行されている. exchange は以下のように実行される.
exchange 方向 delta 1.2-1.3 1.2
exchange 方向 delta 1.3-1.4 1.2

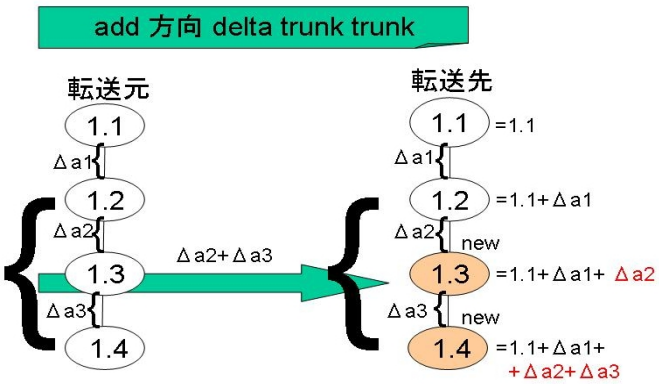

図 3.22: add

3.4.4 add_tree

add_tree は転送元のリビジョンツリー全体の開発進展部分を, 転送先の同じリビジョンツリーの位置に接木する. この操作はデータベースを用いて実行を行う.

・add_tree 方向 転送方式
方向 come/go
転送方式 delta/file

add_tree を図 3.23 を元に説明する. なおここでは方向は指定しないが, come ならば, 転送元が他拠点で転送先が自拠点となる. go ならば, 転送元が自拠点で転送先が他拠点となる.

いま, 転送元では 1.4, 1.2.2.2 まで, 転送先は 1.3, 1.2.2.1 まで開発が進んでいる. 転送元の 1.3 と転送先の 1.3, 及び転送元の 1.2.2.1 と転送先の 1.2.2.1 は同一の内容である.

add_tree を行うと, 転送元の Δ a3 と Δ a5 が転送され, 転送先に 1.4, 1.2.2.2 が作成される. なお, 転送元の Δ a3 は転送先のリビジョン 1.3 に, 転送元の Δ a5 は転送先のリビジョン 1.2.2.1 につく. add_tree は転送元で開発が進展したリビジョンを k 個とすると, exchange を k 回行う. なお, exchange は以下のように 2 回実行される.
exchange 方向 delta 1.3-1.4 1.3
exchange 方向 delta 1.2.2.1-1.2.2.2 1.2.2.1

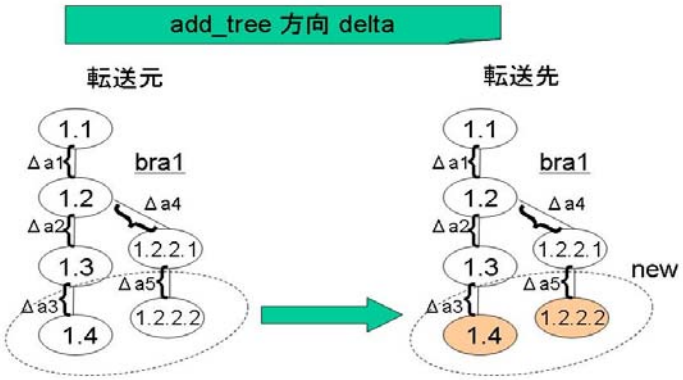

図 3.23: add_tree

3.5　システムの配置

本システムは CVS リポジトリが存在するコンピュータに配置されることを想定している. 以下に典型的な 2 つの場合について述べる.

1 つ目は, 一方のプロジェクトが本システムの機構 (Tcvs) を用いる場合である. この場合, 図 3.24 に示す A プロジェクトは B プロジェクトからリポジトリの読み込みと書き込みの許可を得る必要がある. A プロジェクトのコンピュータの構成は開発者が 1 人の場合はクライアント, 複数人の場合はサーバ (pserver) を想定している.

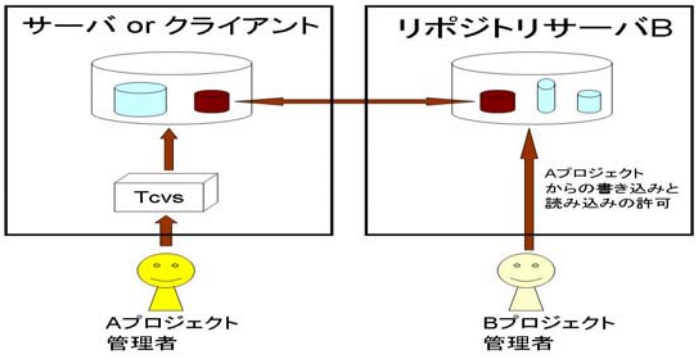

図 3.24: 一方のプロジェクトが使用

2 つ目は, 双方のプロジェクトが本システムの機構を用いる場合である. この場合, 図 3.25 に示す双方のプロジェクトがリポジトリの読み込みと書き込みの許可を他方のプロジェクトに対して行う必要がある. なお, A プロジェクト, B プロジェクトともにサーバを立てる必要がある.

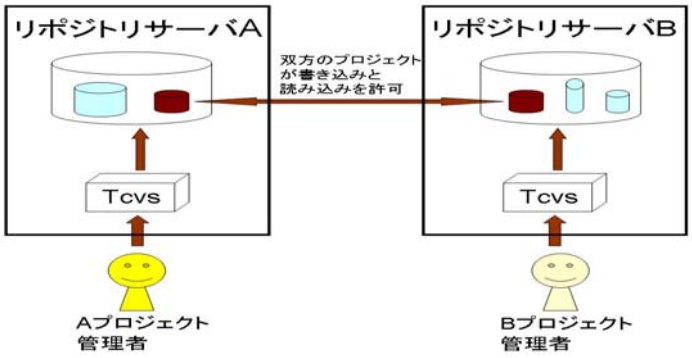

図 3.25: 双方のプロジェクトが使用

第4章　実装

本章では, 前章の設計に基づき, 実装を行ったこのシステムを Tcvs(Transfer CVS) と呼ぶ. このシステムは, 独自のリポジトリの構築と運用を可能にする.

4.1　実装環境

以下の実装環境となる. プログラミング言語はオブジェクト指向言語 Ruby を利用した.

- OS: FreeBSD 5.1-Release
- 構成管理システム: CVS1.11.5-FreeBSD
- プログラミング言語: Ruby1.8.1
- データベース: MySQL4.0.18
- 使用ライブラリ: ruby-mysql-2.4.5

4.2　動作方法

本システムはコマンドライン上で以下のように入力することで, リポジトリ間のデータ転送を行うことが出来る.

% ruby Tcvs.rb ファイル名

Tcvs を実行するにはまず, データ転送記述を行う. データ転送記述をコンフィグレーションファイルとして用意した理由は, 引数が多いという問題を解決するためと, 複数回のデータ転送操作を可能とするためである. 書式は以下のようになっている.

```
[def_obj]..[def_obj_end]
[OP]..[OP_end]
ループ
[EOF]
```

[def_obj]..[def_obj_end] にはリポジトリパス, 粒度, ディレクトリ構造, ファイル名などを記述する. 次に, [OP]..[OP_end] の間にはリビジョンツリーの転送操作を記述する. [OP]..[OP_end] の中には複数回のリビジョンツリーの転送操作を記述することが可能である. [def_obj]..[def_obj_end] と [OP]..[OP_end] は [EOF] を記述するまで [def_obj]..[def_obj_end], [OP]..[OP_end], [def_obj]..[def_obj_end], [OP]..[OP_end] と何度でも記述できる.

ここでは他拠点から ElevatorControl.java という 1 つのファイルのリビジョンツリーを完全に複製するという場合を考え. 以下にファイルの例として op.conf をあげる.

```
[def_obj]
#your_repository_name
econ03
#your_repository
:pserver:i310076@i451.jaist.ac.jp:/cvsroot/econ03
#my_repository
/usr/hoge
#my_directory_name
elevator
#your_directory_name
elevator
#my_file_name
ElevatorControl.java
#your_file_name
ElevatorControl.java
[def_obj_end]

[OP]
replicate come
[OP_end]

[EOF]
```

#your_repository_name の 1 行下にある econ03 は
他拠点のリポジトリ名である.
#your_repository の 1 行下にある :pserver:i310076@i451.jaist.ac.jp:/cvsroot/econ03
は他拠点のリポジトリパスである.
#my_repository の 1 行下にある /usr/hoge は
自拠点のリポジトリパスである.
#my_directory_name の 1 行下にある elevator は
自拠点のリポジトリのディレクトリ名である.

#your_directory_name の 1 行下にある elevator は
他拠点のリポジトリのディレクトリ名である .
#my_file_name の 1 行下にある ElevatorControl.java
は自拠点の elevator ディレクトリ以下に作成されるファイル名である .
#your_file_name の 1 行下にある ElevatorControl.java
は他拠点の elevator ディレクトリ以下に存在するファイル名である .

[OP]..[OP_end] の間にはリビジョンツリーのオペレーションとして replicate come が記述してある. これは他拠点のリビジョンツリー全てを複製する操作である. [EOF] はファイルの END である. 以上のデータ転送記述ファイル op.conf を以下のように

% ruby Tcvs.rb op.conf

実行すると, 他拠点のリポジトリの elevator ディレクトリ以下にある ElevatorControl.java が自拠点のリポジトリの elevator ディレクトリ以下に複製される.

4.2.1 粒度の指定方法

ここでは, 方向を come として説明する. ファイルを指定する場合には, 以下のように記述を行う.

```
#my_directory_name
elevator
#my_file_name
ElevatorControl.java
#your_directory_name
elevator
#your_file_name
ElevatorControl.java
```

ディレクトリを指定する場合には，以下のように記述を行う .

```
#my_directory_name
elevator
#your_directory_name
elevator
#your_file_name
*
```

リポジトリ全体を指定する場合は以下のように記述する .

```
#your_directory_name
*
```

複数のファイルや複数のディレクトリを指定する場合には, [def_obj]..[def_obj_end] と [OP]..[OP_end] を繰り返し記述する.

4.2.2 異なるディレクトリの指定方法

#my_directory_name と#your_directory_name 以下を記述する .

```
#my_directory_name
elevator
#your_directory_name
econ
```

4.2.3 異なるファイル名間の指定方法

#my_file_name と#your_file_name 以下を記述する .

```
#my_file_name
foofoo.java
#your_file_name
hogehoge.java
```

4.2.4 複数拠点との転送方法

[def_obj]..[def_obj_end] と [OP]..[OP_end] を繰り返し記述する.

4.2.5 リビジョンツリーの操作

[OP]..[OP_end] の間に root, exchange, graft, replicate, add, add_tree を記述する. 複数回の記述が可能である.

```
[OP]
root come 1.1
exchange come delta 1.1-1.2 1.1
graft come delta 1.2-1.5 1.2
[OP_end]
```

4.2.6 記述の省略

[def_obj]..[def_obj_end] は 2 度目からは全て記述する必要はなく, 変更したいとこだけを記述すればよい. 例えばファイルの指定だけを変えたい場合は以下を記述する.

```
[def\_obj]
#my_file_name
ファイル名
#your_file_name
ファイル名
[def_obj_end]
```

第5章　考察

5.1　記述力の検討

本研究の細粒度の操作の記述力を検討するために, FreeBSD プロジェクトで使用されている CVSup のリポジトリ間のリビジョンツリーの扱いを記述した.

CVSup には exact, non_exact という 2 つの動作モードがある. exact は自拠点のリビジョンツリーを, 他拠点のリビジョンツリーと同一にする. non_exact は自拠点に独自に存在するブランチを維持したまま他拠点の差分を取り込む.

5.1.1　exact

exact は自拠点のリビジョンツリーを, 他拠点のリビジョンツリーと同一にする動作モードである. exact モードを図 5.1 と図 5.2 を元に説明する. 今, 図 5.1 では他拠点のトランクは 1.4 まで, ブランチは 1.2.2.2 まで開発が進んでいる. 自拠点はトランクは 1.3, ブランチは 1.2.2.3 まで開発が進んでいる. ここで, 自拠点は他拠点の開発成果を取り込むために CVSup を exact モードで実行する. すると実行後の図 5.2 では自拠点は他拠点と同一のリビジョンツリーになる. ここで, 自拠点と他拠点の 1.1, 1.2, 1.3, 1.4, 1.2.2.1, 1.2.2.2 は同一のリビジョンである.

本研究の細粒度の操作で exact モードを記述する場合, 以下のように記述することで, 他拠点と自拠点のリビジョンツリーを完全に一致させることができる.

```
root come 1.1
exchange come delta 1.1-1.2 1.1
exchange come delta 1.2-1.3 1.2
exchange come delta 1.3-1.4 1.3
exchange come delta 1.2-1.2.2.1 1.2 bra1
exchange come delta 1.2.2.1-1.2.2.2 1.2.2.1
```

まず, 他拠点の 1.1 を root によって複製する. 次に, exchange によって他拠点の 1.1-1.2 の差分を取り込んで 1.2 を複製する. 同様のことを繰り返すことで, トランクの 1.4 まで複製する. さらに, 他拠点の 1.2-1.2.2.1 差分を取り込んで 1.2.2.1 を複製する. 最後に, 1.2.2.1-1.2.2.2 の差分を取り込んで 1.2.2.2 を複製する. これにより自拠点のリビジョンツリーは他拠点と完全に一致する.

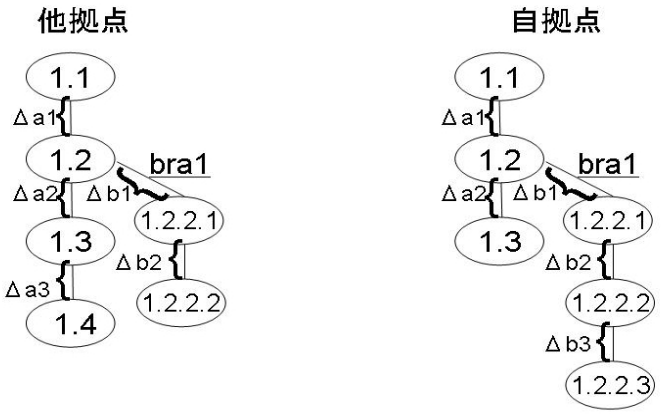

図 5.1: 実行前

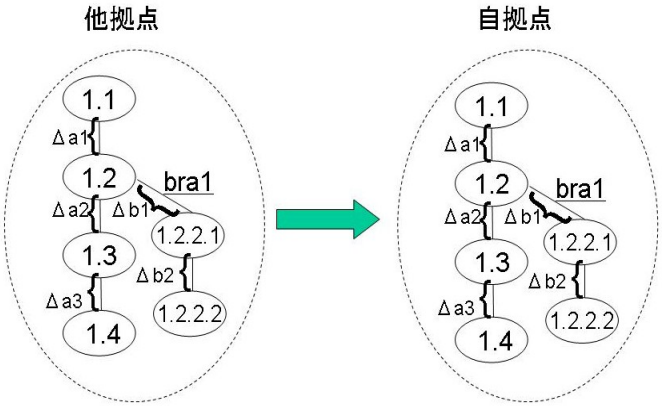

図 5.2: 実行後

5.1.2 non_exact

non_exact は自拠点に独自に存在するブランチを維持したまま他拠点の差分を取り込む動作モードである.

non_exact モードを, 図 5.3 を元に説明する. 今, 他拠点のトランクは 1.4 まで, ブランチは 1.2.2.2 まで開発が進んでいる. 自拠点はトランクは 1.2, ブランチは 1.2.2.2, 1.2.4.2 まで開発が進んでいる. 自拠点と他拠点の 1.1, 1.2, 1.2.2.1, 1.2.2.2 は一致する. 自拠点は他拠点の開発成果を取り込むために CVSup を non_exact モードで実行する. すると図 5.3 のように自拠点は独自のブランチ bra2 を維持したまま他拠点の開発成果を取り込むことが出来る. ここで, 自拠点と他拠点の 1.3, 1.4 は一致する.

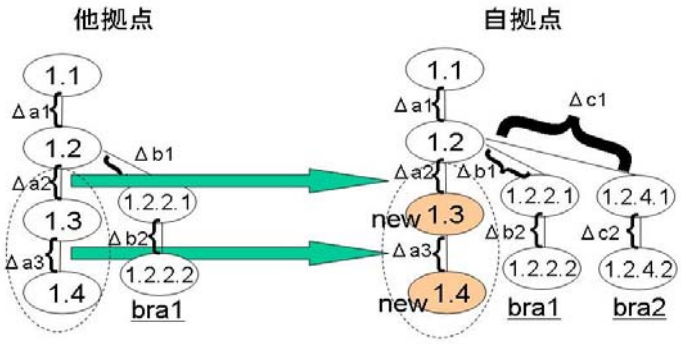

図 5.3: non_exact

本研究の細粒度の操作で non_exact を記述する場合, 以下のように記述できる.
exchange come delta 1.2-1.3 1.2
exchange come delta 1.3-1.4 1.3

まず, exchange によって他拠点の 1.2-1.3 の差分を取り込んで 1.3 を複製する. 次に, 他拠点の 1.3-1.4 の差分を取り込んで 1.4 を複製する. これにより, 自拠点と他拠点の 1.3, 1.4 は一致する.

5.1.3 実行回数

本研究の細粒度の操作で CVSup の exact モードと non_exact モードを記述した場合, 以下の実行回数となる.
exact モードは他拠点に存在するリビジョン数を n とすると, root を 1 回, exchange を n-1 回行う. non_exact モードは他拠点で進行したリビジョン数を m とすると, exchange を m 回行う.

5.2 CVSupの問題解決

CVSupの問題点を図5.4を元に説明する．CVSupは原則exactモードで動作する．つま

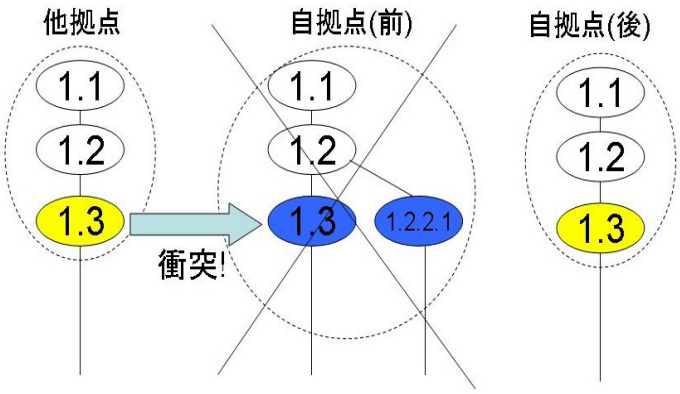

図 5.4: 成果物と変更履歴の消失

り, 完全一致となる．non_exactモードは例外的な動作である．そのため, non_exactモードで衝突が発生すると強制的にexactモードで実行される．つまり, 自拠点のリビジョンツリーは他拠点と完全一致となる．例えば, 自拠点でトランク1,3と独自のブランチ1.2.2.1で開発を進めていて, 他拠点の成果物と変更履歴を取り込むために, non_exactモードを実行しようとしても, トランクでコンフリクトすると, exactモードが呼び出され自拠点と他拠点は完全一致となる．このとき, 自拠点のトランク1,3と独自のブランチ1.2.2.1で開発していた成果物と変更履歴は全て失われる．

本研究の機構を用いると, 例えば, 自拠点は図5.5のように開発を進めることが出来る．他拠点の1.2から1.3の差分をexchangeコマンドを使ってブランチ1.2.4.1に取り込むことができる．これにより, 自拠点は開発の自由度を維持しながら, 他拠点と関連を持つことができる．

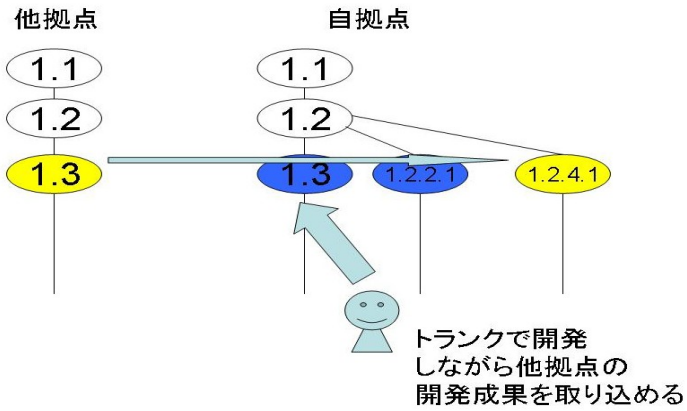

図 5.5: トランクでの開発

また, CVSup は他拠点から自拠点へのデータ転送はできるが, 自拠点から他拠点に開発成果を送ることができない. しかし, 本研究の機構を用いると, 例えば子プロジェクトは図 5.6 のように行うことが出来る. まず, 他拠点の 1.2 までの成果物と変更履歴を replicate コマンドを使って複製する. その後自拠点で 1.4 まで開発を進める. 自拠点は 1.2 から 1.4 までの開発成果を graft コマンドを用いて他拠点に開発成果を送ることができる. 他拠点はこの開発成果を利用して開発を進めることが出来る.

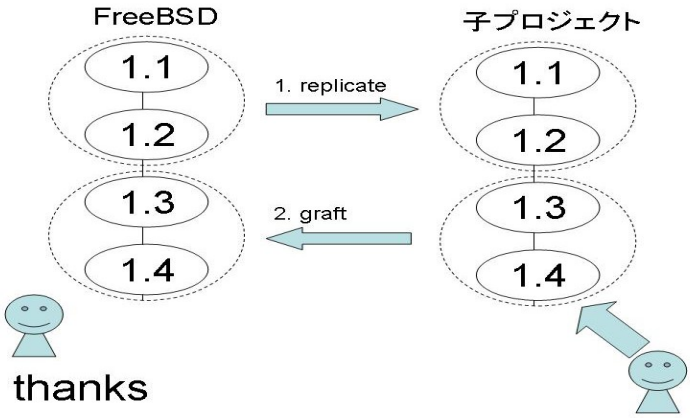

図 5.6: 他拠点への開発成果の反映

第6章　適用例

本章では, 本研究の適用例として, CVS を用いられて開発が進められているプロジェクトの中で, 一つの親を持つ Cygwin プロジェクトと, 複数の親を持つ KAME プロジェクトを例に挙げる. これらのプロジェクトは独自のリポジトリ構造を持ちながら親プロジェクトと双方向で成果物と変更履歴の反映を行っている. しかし CVS は, リポジトリ分散の機能が無い. また, CVSup では転送の方向が単方向, いわゆる Pull モデルのため, これらの開発プロジェクトを適切に支援できない.

6.1　Cygwin

Cygwin [6] は, UNIX 系 OS では一般に広く普及している GNU プロジェクトによる開発ツール群を Windows 環境用に移植するプロジェクトである. Cygwin プロジェクトは MinGW リポジトリの w32api を複製して利用しているだけでなく, w32api のバグなどの修正を MinGW 側へ転送を行う. このように, Cygwin は独自のリポジトリ構造を保ちながら, MinGW と双方向でデータ転送を行っている.

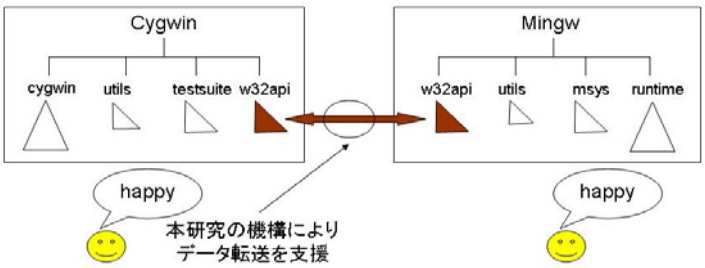

図 6.1: Cygwin

本システムは様々なリビジョンツリー操作, ファイル単位, ディレクトリ単位の双方向の転送を可能としている. 図 6.1 に示すように, 本システムを Cygwin プロジェクトに適用した場合, Cygwin は MinGW の w32api に対して上位操作の replicate を適用することで部分複製ができる. また, w32api のバグの修正を上位操作の add_tree を用いて MinGW 側へ反映することができる. このように, 本システムは独自のリポジトリの構築と運用を支援できる.

6.2 KAME

KAME [8] は FreeBSD [11] リポジトリ, OpenBSD [9] リポジトリ, NetBSD [10] リポジトリの部分的複製をおこない開発を進めている. KAME は OpenBSD リポジトリの部分的

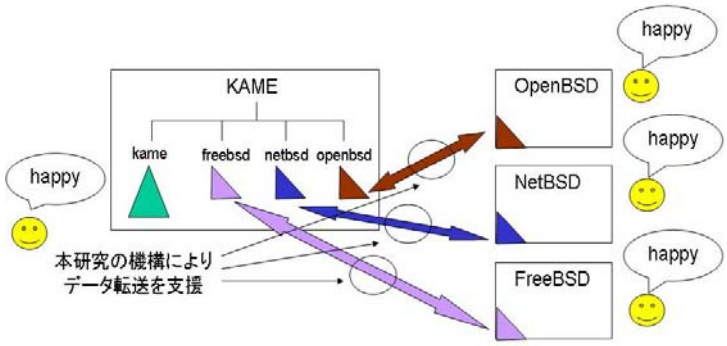

図 6.2: KAME

複製をディレクトリ /openbsd に, NetBSD リポジトリの部分的複製を /netbsd に, FreeBSD リポジトリの部分的複製を /freebsd に取り込んでいる. このため, 親プロジェクトとは異なるディレクトリ構造を持つ. KAME は IPv6 対応 TCP/IP ソフトの開発を行い, この成果は適宜 OpenBSD, NetBSD 及び FreeBSD のリポジトリに格納される.

図 6.2 に示すように, 本システムは, 複数リポジトリとの関係, 異なるディレクトリ構造間の関係を記述できる. 本システムを KAME プロジェクトに適用した場合, KAME は OpenBSD, NetBSD, FreeBSD 毎に必要な粒度 (FILES+DIRECTORYS) を定義して replicate を用いることで部分的複製ができる. また, KAME で開発が進展した場合, 成果物と変更履歴を OpenBSD, NetBSD, FreeBSD に add_tree を用いて転送できる. このように, 本システムは独自のリポジトリの構築と運用を支援できる.

第7章　おわりに

7.1　まとめ

本研究では, 独自のリポジトリ構造を持つオープンソースソフトウェアの開発プロジェクトに対し適切な支援を行う, CVS リポジトリの分散支援機構の設計と実装を行った. これは任意の粒度のデータ転送と管理方針の分離という特徴を持つ. これらにより, プロジェクトに応じたリポジトリの構築と運用が可能となる. 各拠点のプロジェクトは他拠点と関連を維持しつつ, 独自の管理方針に基づいてリポジトリを運用できるので, 開発効率の向上が期待できる.

7.2　今後の課題

今後の課題として, 異なる構成管理システム間のデータ転送と協調支援機構の実現が挙げられる.

- **異なる構成管理システム間のデータ転送**
  例えば, CVS-Bitkeeper 間や CVS-Subversion 間などでデータ転送を可能とすることにより, 拠点間で異なる構成管理システムを用いた開発を行える.

- **協調支援機構**
  本研究では拠点間で独自のリポジトリ構成を持つことを可能としたが, 拠点間で矛盾が生じることを防ぐ機構を提供していない. そこで, 協調支援機構として現在, 即時反映と矛盾防止機構というものが挙げられる.
  ・即時反映
  時間がたつにつれて開発が進み, リポジトリ間で成果物と変更履歴にずれが発生する. これらを解消するために, コミットされたら即座にリポジトリ間で成果物と変更履歴の反映を必要とする場合がある. 現在, 以下の方法が考えられる.
  ・自拠点から他拠点
  コミット時に専用スクリプトで Tcvs を呼び出す.
  定期的に Tcvs を実行する.
  ・他拠点から自拠点
  定期的に Tcvs を実行する.

・矛盾防止機構
ディレクトリやファイルが増えた場合, 人間の認知能力には限界があるため, データ転送の操作ミスをしてしまう場合がある. それにより, 無用なリビジョンの増加, ファイルの増加, コンフリクトなどが発生してしまう. このような操作ミスを防ぐため, 現在, 以下の方法が考えられる.

自拠点のディレクトリとファイルを禁止条件として設定可能にし, 原則として転送できないようにする. そして例外条件として, 他拠点のリポジトリ, ディレクトリ名, ファイル名, 操作名, 操作の引数を指定可能とする.

これにより, 例えば自拠点のファイルは, あるリポジトリとだけ転送を行ったり, あるディレクトリに存在するファイルとだけ転送を行ったりできる. また, 操作名, 操作の引数を指定することでリビジョンツリー全体の一致, トランクやあるブランチの一致, あるリビジョンとあるリビジョンの一致など様々な支援ができる.

謝辞

本研究を行うにあたり, 終始御指導賜りました落水 浩一郎教授に心より深く感謝致します.

本研究を行うにあたり, 藤枝 和宏助手には, 終始御指導, 御助言いただき厚く感謝致します.

また, 日頃より御指導, 御助言頂きました鈴木 正人助教授に深く感謝致します.

そして, 服部 哲助手, 早坂 良氏には, 本研究に関する貴重なご意見を頂き心より感謝致します.

本研究に関して多くの有意義な助言を頂きました落水研究室の諸氏に感謝申し上げます.

最後に, 北陸での生活を支援してくれた両親, ならびに生活面でお世話になった友人達に深く感謝申し上げます.

参考文献

[1] CVS, http://www.cvshome.org/

[2] CVSup, http://www.cvsup.org/

[3] BitKeeper, http://www.bitkeeper.com/

[4] ClearCase, http://www.rational.com/products/clearcase/index.jsp/

[5] ClearCase Multisite, http://www.rational.com/products/cc_multisite/index.jsp/

[6] Cygwin, http://cygwin.com/

[7] MinGW, http://sourceforge.net/projects/mingw/

[8] KAME, http://www.kame.net/

[9] OpenBSD, http://www.openbsd.org/

[10] NetBSD, http://www.netbsd.org/

[11] FreeBSD, http://www.freebsd.org/

[12] FreeBSD Japan, http://www.jp.freebsd.org/

[13] Apache, http://www.apache.org/

本研究に関する発表論文

- 中島 健至, 藤枝 和宏, 落水 浩一郎: オープンソースソフトウェア開発に適したリポジトリ分散の支援機構, 情報処理学会 ソフトウェア工学研究会 March 2005.