2005－SE－147（4）
2005／3／17

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

中島 健至† **藤枝 和宏**† **鈴木 正人**† **落水 浩一郎**†‡

†北陸先端科学技術大学院大学 情報科学研究科

‡国立情報学研究所

〒 923-1292 石川県能美市辰口町旭台 1-1

e-mail: {n-takesi, fujieda, suzuki, ochimizu}@jaist.ac.jp

概要

オープンソースソフトウェアの開発プロジェクトでは, 規模が大きくなるにつれ, 目的の異なる複数の子プロジェクトを持つことがある. これらの開発で広く用いられている CVS は, リポジトリが単一構成のため, 子プロジェクトごとに異なる管理方針の設定が難しい. 本稿では, この問題を解決するリポジトリ分散の支援機構を提案する. この機構により, リポジトリ間の関連を維持しながら, リポジトリごとに独自の管理方針が設定可能となる.

和文キーワード

オープンソースソフトウェア開発, 管理方針, リポジトリ, 構成管理システム, CVS

Repository Decentralization Support Mechanism for Open Source Software Development

Takeshi NAKASHIMA† **Kazuhiro FUJIEDA**† **Masato SUZUKI**† **Koichiro OCHIMIZU**†‡

†School of Information Science, JAIST

‡National Institute of Informatics

Asahidai 1-1, Nomi, Ishikawa 923-1292

Abstract

Open source software development often has several sub-projects with different goals along with its augmenting scale. CVS used in such development can have only one repository. So it is difficult to support various policy of operation and management for these sub-projects. In this paper, we propose a distribution of CVS repository to solve this problem. It allows us to set policy proper for each of repositories and to manage various relationships of them.

英文 key words

Open Source Software Development, Management Policy, Repository, Software Configuration Management Systems, CVS

1 はじめに

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 つの形態に, オープンソースソフトウェア開発 (OSSD) がある. OSSD では, 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. このようなプロジェクトにおいては成果物と変更履歴を管理する 1 つのリポジトリを共有することは現実的ではないため, リポジトリの分散構成が必要となる.

既存の構成管理システムや外部ツールの分散機能では, 粗粒度で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た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子から親へ反映できない場合がある. そのため, 既存の OSSD 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, OSSD を適切に支援可能なリポジトリ分散の支援機構を提案する. 本研究では, OSSD で広く使用されている CVS 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.

本稿の構成は, 2 節で既存の構成管理システムと外部ツールの問題点を指摘する. 3 節では, 問題点を解決する本システムについて述べる. 4 節では, 本システムを用いて FreeBSD などの OSSD で使用されている CVSup の模倣と問題解決を行う. 5 節では, 現実のプロジェクトに対して適用例を示す. 最後に, 6 節でまとめと今後の課題について述べる.

2 既存のリポジトリ分散モデルの問題点

既存の構成管理システムや外部ツールは, 成果物と変更履歴を過剰に転送するため. 転送先のリポジトリは転送元と同一のリポジトリになる [12]. そのため, 独自のリポジトリの構築と運用を行う開発プロジェクトを適切に支援することができない.

2.1 CVSup

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, 他拠点のリポジトリから手動で複製を行う.

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 では図 1 に示すように, 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.

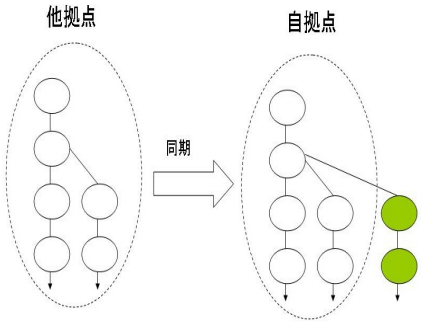

図 1: CVSup

2.2 ClearCase Multisite

ClearCase [3] 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 [4] 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 は, 図 2 の問題がある. ClearCase Multisite は, 他拠点の変更は, 常に自拠

点に反映される．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．

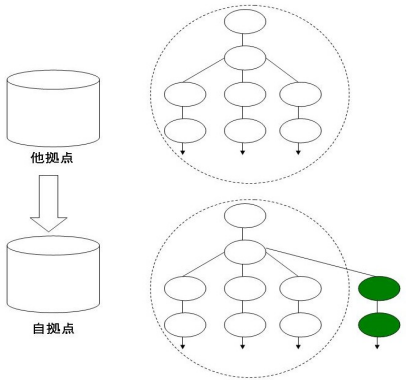

図 2: ClearCase Multisite

2.3 Bitkeeper

BitKeeper [5] 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．その転送方式の概要を図 3 に示す．

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制限される．

3 リポジトリ分散の支援機構

3.1 概要

前章で述べたリポジトリ分散の問題を解決するために，CVS リポジトリ間の支援機構を提案する．支

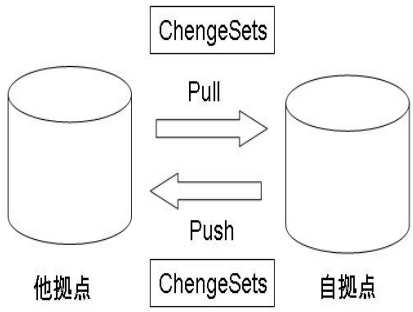

図 3: Bitkeeper

援機構は，図 4 に示す枠内のデータ転送管理部，細粒度の操作，上位操作からなる．

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．

細粒度の操作は転送元のリビジョンツリーから任意の差分またはリビジョン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．

上位操作はリポジトリ間の同一内容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．

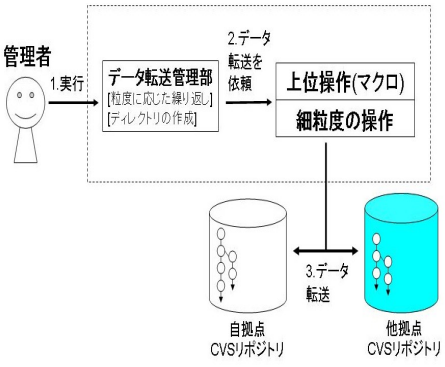

図 4: リポジトリ分散の支援機構

3.2 データ転送管理部

データ転送管理部は, 開発プロジェクトが必要とする粒度のデータ転送を実現する. 管理者により指定される粒度, すなわち FILES(1 つ以上のファイル), DIRECTORYS(1 つ以上のディレクトリ), FILES+DIRECTORYS, REPOSITORY(リポジトリ全体) に対して, 細粒度の操作, 上位操作を繰り返し実行する. また, データ転送管理部は必要に応じて転送先にディレクトリを作成する.

3.3 細粒度の操作

細粒度の操作は root と exchange からなる. 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. 方向は自拠点を基準とする.

3.3.1 root

root は転送元の任意のリビジョンを元に, 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 転送元のリビジョン番号
> 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, 他拠点から自拠点となる. go ならば, 自拠点から他拠点となる. 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 5 を用いて説明する. 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.

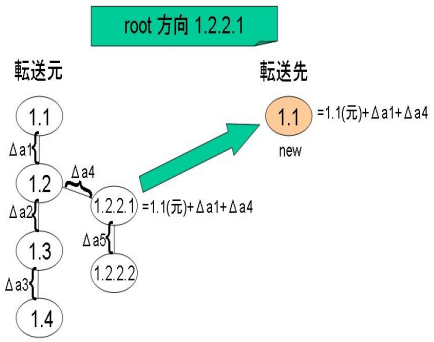

図 5: root

3.3.2 exchange

exchange は転送元の任意の差分または, リビジョン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. なお, 転送先の指定したリビジョンの先にリビジョンがすでに存在する場合は, その先のリビジョンを全て削除して, 新たにリビジョンを 1 つ作成する. 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. root と同じ意味である. 2 つ目の引数は転送方式である. delta(差分) と file(リビジョン) 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 はブランチを作成する場合と作成しない場合がある. まず, 作成しない場合について図

6 を用いて説明する．今, 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．

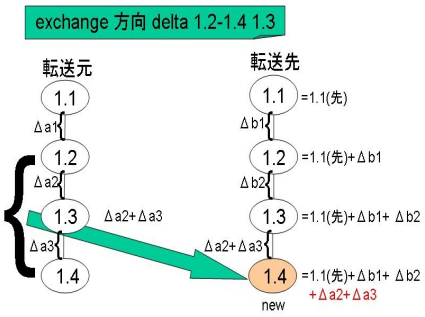

図 6: exchange

ブランチを作成する場合の exchange について, 図 7 を用いて説明する．今, 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．

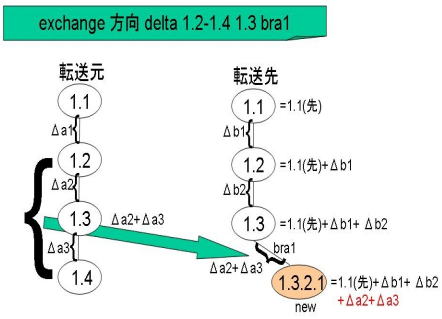

図 7: exchange ブランチ作成

3.4 上位操作

上位操作は複数の細粒度の操作をまとめて実行するマクロである．上位操作を表 1 に掲載する．リポジトリの管理者は, 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．

表 1: 上位操作

操作名	引数	説明
graft	方向 転送方式 転送元の範囲指定 転送先リビジョン番号 ブランチ名	リビジョンツリーの一部を転送
replicate	方向	リビジョンツリー全体を複製
add	方向 転送方式 自拠点の枝 他拠点の枝	トランクまたは, あるブランチの開発進展部分を転送
add_tree	方向 転送方式	リビジョンツリー全体の開発進展部分を転送

各操作の引数の詳細:
方向, 転送方式, 転送先リビジョン番号, ブランチ名は exchange と同様である．
転送元の範囲指定:転送元の差分またはリビジョンを範囲で指定する．
自拠点の枝:trunk/任意のブランチ名
他拠点の枝:trunk/任意のブランチ名

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 はデータベースを用いる．以下にデータベースのデータ構造を示す．
(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) 自拠点のファイルが属するディレクトリパス
(4) 他拠点のファイルが属するディレクトリパス
(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
(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．

4　CVSup の模倣と問題解決

4.1　模倣

本研究の細粒度の操作の記述力を検討するために, FreeBSD などの OSSD で使用されている CVSup の 2 つの動作モード exact, non_exact を模倣した.

4.1.1　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. 図 8 を元に説明する. 今, 図 8 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と仮定する. ここで, 自拠点は他拠点の開発成果を取り込むために exact モードで CVSup を実行する. すると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.

提案手法で exact モードを模倣する場合, 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を他拠点と完全に一致させることができる.

4.1.2　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. 図 9 で今, 他拠点のトランクは 1.4 まで, ブランチは 1.2.2.2 まで, 自拠点のトランクは 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と仮定する. ここで, 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 non_exact モードで CVSup を実行する. すると図 9 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できる. この結果新たに, 自拠点と他拠点の 1.3, 1.4 は一致する.

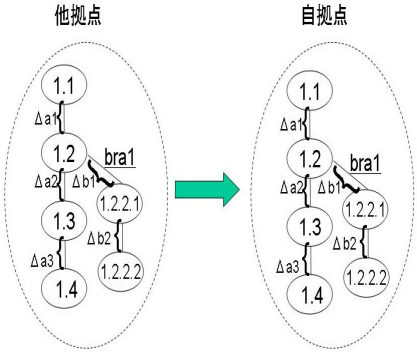

図 8: exact モード

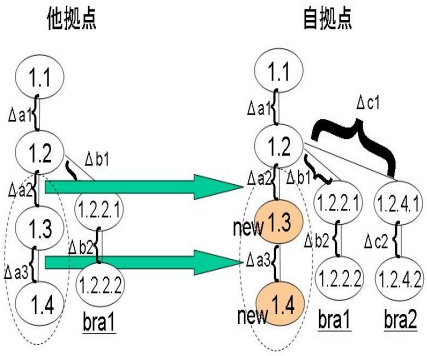

図 9: non_exact モード

non_exact モードを模倣する場合, 以下のように記述する.

```
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 を一致させることができる.

4.2 問題解決

CVSup は原則 exact モードで動作する．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．

今，図 10 の上半分において FreeBSD では 1.3 まで，子プロジェクトでは 1.3 と 1.2.2.1 まで開発が進んでいる．ここで，FreeBSD と子プロジェクトの 1.2 は同一のリビジョンであるが，1.3 は異なるリビジョンである．子プロジェクトは non_exact モードで FreeBSD の差分を取り込もうとする．しかし，この場合トランクでコンフリクトが発生して，強制的に exact モードで実行される．そのため，子プロジェクトのリビジョンツリーは FreeBSD と完全一致になる．このとき，子プロジェクトのトランク 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．

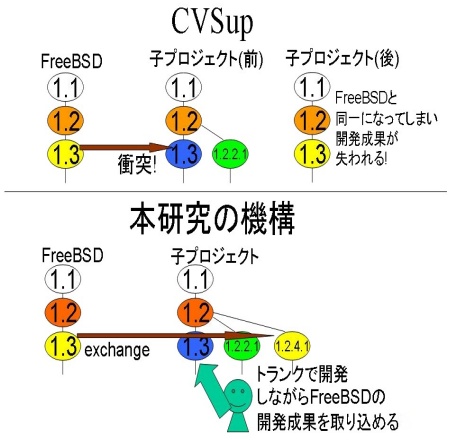

図 10: CVSup の衝突と exchange を用いた回避

本研究の機構を用いると，例えば，子プロジェクトは図 10 の下半分のように他拠点の 1.2 から 1.3 の差分を exchange 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，子プロジェクトは開発の自由度を維持しながら，FreeBSD と関連を持つことができる．

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 FreeBSD の子プロジェクトは図 11 のように行

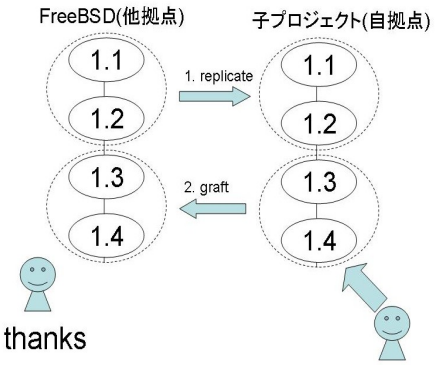

図 11: 他拠点への開発成果の反映

うことができる．まず，FreeBSD の 1.2 までの成果物と変更履歴を上位操作の replicate を使って複製する．その後，子プロジェクトは 1.4 まで開発を進める．子プロジェクトは 1.2 から 1.4 までの開発成果を上位操作の graft を用いて FreeBSD に開発成果を送ることができる．FreeBSD はこの開発成果を利用して開発を進めることができる．

5 適用例

ここでは，本研究の適用例として，1 つの親を持つ Cygwin [6] プロジェクトと複数の親を持つ KAME [8] プロジェクトを例に挙げる．

5.1 Cygwin

図 12 に示す Cygwin 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 [7] リポジトリの w32api を複製して開発を進めている．また，Cygwin プロジェクトは w32api のバグの修正を MinGW 側へ転送する．

本システムはディレクトリ単位の転送を可能としている．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．

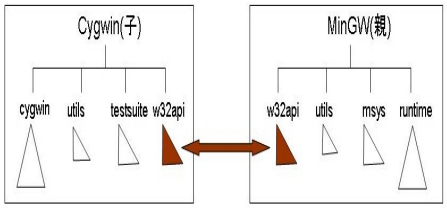

図 12: Cygwin

5.2 KAME

図 13 に示す KAME は IPv6 プロトコルスタックの開発を行うプロジェクトである． KAME は OpenBSD [9] リポジトリの部分的複製をディレクトリ/openbsd 以下に, NetBSD [10] リポジトリの部分的複製を/netbsd 以下に, FreeBSD [11] リポジトリの部分的複製を/freebsd 以下に取り込んでいる．そのため, 親プロジェクトとは異なるディレクトリ構造を持つ．KAME の開発成果は snapshot として公開され, OpenBSD, NetBSD, FreeBSD のリポジトリに格納される．

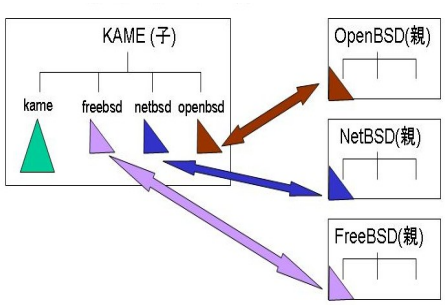

図 13: KAME

本システムは, 必要な粒度で部分的複製と双方向の転送を支援できる．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．

6 おわりに

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, 適切な粒度のデータ転送を行う CVS リポジトリの分散支援機構の提案を行った．これにより, 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．

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, 今後の課題として, 時間的なリポジトリ間のずれを解消する機構や転送ミスを防ぐ機構の開発が必要となる．

参考文献

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

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

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

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

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

[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] 渡辺憲介:管理ポリシーの分離可能なリポジトリの複製管理機能に関する研究, Master's Thesis, 北陸先端科学技術大学院大学. March 2004.