修士学位論文

題目

複数のリポジトリを統合できる
バージョン管理システムの提案と試作

指導教官
井上 克郎 教授

報告者
田中 義己

平成 13 年 2 月 14 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成 12 年度 修士学位論文

複数のリポジトリを統合できる
バージョン管理システムの提案と試作

田中 義己

内容梗概

ソフトウェア開発中で生成されるプロダクトの管理を行う際には，一般的にバージョン管理システムが用いられる．バージョン管理システムでは，ソフトウェア開発の各段階において生成されたプロダクトの登録・変更・削除などの作業をリポジトリと呼ばれるデータベースを用いて管理している．しかし，既存のシステムはプロダクトに対する作業履歴を不可分な物として扱っている．このため，プロダクト履歴を把握することは可能であるが，開発者に着目した開発履歴を取得することが困難である．また，開発時に利用するコンポーネントが複数のリポジトリに跨る場合，開発者は，それぞれのコンポーネントをリポジトリ単位で扱うことになり，煩雑な作業が必要となる．さらに，再利用する際，他のリポジトリ内にあるコンポーネントをリポジトリ間で複製を行うが，複製された後の更新を反映する際には，再度複製を行う必要性がある．

本研究では，従来のバージョン管理システムの持つ欠点を解決する目的として，プロダクト自身の変更履歴とプロダクトに対する作業履歴を独立して管理する手法 DiRM/VR の提案を行う．本手法により，開発者単位の開発履歴を容易に取得することができる．複数のリポジトリに存在する既存のコンポーネントを自由に組み合わせることにより，単一のリポジトリとして管理することができる．コンポーネント自体の複製を行なうことなく，他のリポジトリ内のコンポーネントを利用することで，更新部分を自動的に反映させることができる．また，本手法に基づいたバージョン管理システム DLCM の試作を行い，DiRM/VR の有理性を確認する実験を行った．本実験により，既存のバージョン管理システムの機能を損ねることなく，利用できることが判った．

主な用語

バージョン管理 (Version Control System)
リポジトリ (Repository)
分散環境 (Distributed Environment)

目 次

1 まえがき

ソフトウェア開発中で生成されたプロダクトの管理方法は古くからある課題の１つである．最近では，それを解決する１つの方法として，バージョン管理システムを利用する傾向がある．バージョン管理システム [9] では，作成されたプロダクトの登録・変更・削除などの作業を開発履歴として残すことが可能である．その履歴を利用し，ソフトウェアの再利用時に，以前の開発過程を閲覧することで，より深い理解を得ることができる．

一方，ハードウェアの発達に伴い，作成されるソフトウェアも肥大化し，その開発規模が拡大している．その為，コンポーネントやモジュールといった，特定の意味を持つブロック単位でソフトウェアを開発することが有効である [6][7][16][18][24][26]．こういった開発手法が採用される背景として，ソフトウェアの共有や再利用を考慮したソフトウェア開発の普及が挙げられる．また，ネットワーク技術の発展により，分散環境でのソフトウェア開発が一般的となってきており，コンポーネント単位を意識したソフトウェア開発に拍車がかかっている．最近では，FreeBSD や Linux などに代表されるオープンソース型のソフトウェア開発形態も広く用いられるようになっている [21][22][23]．オープンソース型のソフトウェア開発では，作成したソフトウェアの共有や再利用が頻繁に行われ，その所在場所がより分散化する傾向にある．

さらに，ここ数年の傾向として，ソフトウェア開発過程の品質評価を行うことが多い．その評価にあっても，開発組織単位だけでなく，開発者単位でも行われる．従って，ソフトウェアレベルだけではなく，開発者レベルにおいても，その開発過程を蓄積しておく必要がある．

既存のシステムではプロダクトに対する作業履歴を不可分な物として管理している．従って，プロダクト履歴の把握は可能であるが，開発者を単位として開発履歴を取得することが困難となる．また，開発時に利用するコンポーネントが複数のリポジトリに跨る場合，それぞれのコンポーネントをリポジトリ単位で扱うことになる為，開発者に煩雑な作業を強いることになる．さらに，他のリポジトリ内にあるコンポーネントを再利用する際，リポジトリ間で複製を行うが，複製された後の複製元への更新を反映するには，再度複製を行う必要がある．このように，既存のバージョン管理システムは，現在のソフトウェア開発形態に対応しているとは言えない．

本研究では，先の問題点を解決する為のバージョン管理手法 DiRM/VR を提案する．本手法では，プロダクトデータと開発者データの双方をまとめた管理を行わず，プロダクトデータ管理部と開発者データ管理部に分離して管理する抽象リポジトリを導入している．また，リビジョン作成 (データ格納)，データ取得，差分情報取得の３つのデータ操作を定義する．これにより，２つの管理部で個別に管理されているデータとの正確な授受が可能となる．

提案した手法に基づいたバージョン管理システム DLCM および，データ閲覧システム

DLCMView を試作した．また，これらを利用した３つの適応実験を行った．それにより，開発者単位の開発履歴は容易に取得することができ，また，複数のリポジトリに存在するコンポーネントの，格納元とは独立な一元管理が可能であることを確認した．本実験により，提案した手法に基づくバージョン管理システムが，既存のシステムの機能を損ねることなく，利用できることが判った．

以降，2. では，準備としてバージョン管理おける基本事項の説明および，バージョン管理モデルとシステムの紹介を行う．また，既存のシステムにおける問題点を提起する．3. では，問題を解決する為の手法 DiRM/VR の提案を行う．4. において，先の手法に基づいた試作したバージョン管理システム DLCM および，閲覧システム DLCM View についての説明を行い，5. でシステムを利用した適応実験の内容およびその考察を記す．そして，最後に 6. で，本研究のまとめと今後の課題について述べる．

2 バージョン管理

2.1 バージョン管理とモデル

バージョン管理とは，以下３つの役割を持つ機構である．

- プロダクトに対して施された追加・削除・変更などの作業を履歴として蓄積する．
- 蓄積した履歴を開発者に提供する．
- 蓄積したデータを編集する．

各プロダクト (ソースコード，リソースなど) の履歴データは，**リポジトリ**と呼ばれるデータ格納庫に蓄積される．その内部では，プロダクトのある時点における状態**リビジョン (Revision)** を単位として管理する．１つのリビジョンには，ソースコードやリソースなどの実データと，作成日時やメッセージログなどの属性データが格納されている．

また，リポジトリとのデータ授受をする為に，開発者はシステムに依存した**オペレーション (operation)** を利用する必要がある．

バージョン管理手法を述べるにあたり，その基礎となるモデルが数多く存在する [2][8][17]．本節では，多くのバージョン管理システムが採用している４つの概念モデルの紹介を行う [10][29]．尚，以降，本文において，プロダクトのある時点における状態のことを**リビジョン (Revision)** という呼ぶ．

2.1.1 The Checkout/Checkin Model

このモデルは，ファイルを単位としたリビジョン制御に関して定義されている (図 1 参照)．

リビジョン管理下にあるコンポーネントはシステムに依存したフォーマット形式のファイルとしてリポジトリに格納されている．開発者のそれらのファイルを直接操作するのでなく，各システムに実装されているオペレーションを介して，リポジトリとのデータ授受を行う．リポジトリより特定のリビジョンのコンポーネントを取得する操作を チェックアウト (Checkout) という．逆に，データをリビジョン に格納し，新たなリビジョンを作成する操作を チェックイン (Checkin) という．

単純にリビジョンを作成するのみでは，シーケンシャルなリビジョン列を生成することになる．しかし，過去のリビジョンに遡り，別の工程で開発を行う場合 (例えば，デバッグ) 等の為，リビジョン列を分岐させるには，ブランチ (Branch) という操作により，ブランチを生成し，その上にリビジョンを作成するという手法を採る．このブランチに対して，元のリビジョン列のことを**トランク (Trunk)** という．また，ブランチ上での作業内容 (デバッグ修

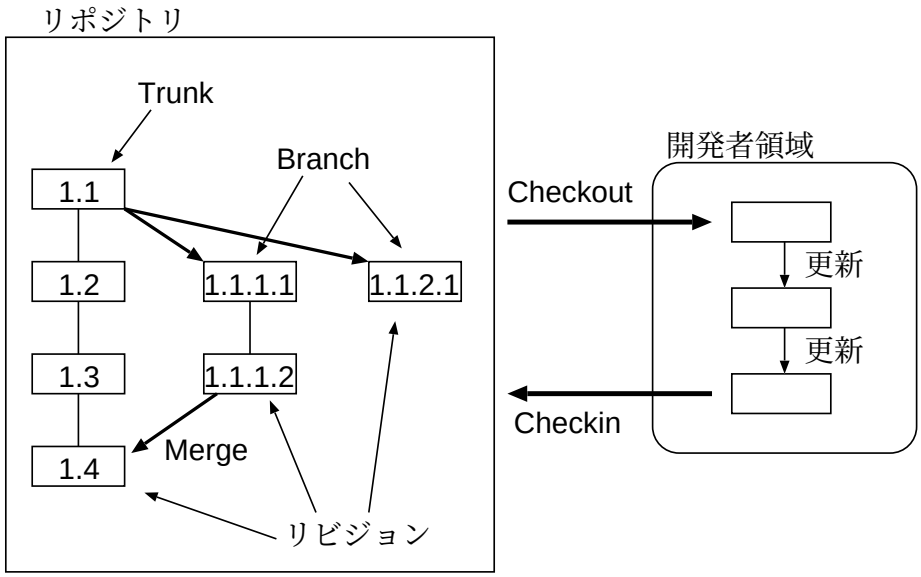


図 1: Checkout/Checkin Model

正部分等) を，別のリビジョンに統合する作業を マージ (Merge) という．このように，リビジョン列は木構造になることから リビジョンツリー (RevisionTree) という．

2.1.2 The Composition Model

このモデルはリポジトリ内部に存在する複数のコンポーネントに対して，それらのリビジョンを効果的に指定するための機構に関する定義をしている．

実際には，組織モデル (system model) と 抽出規則 (selection rules) とから，このモデルは成り立っている．前者はシステム管理下にあるコンポーネント名をリストとして持っている．単純にコンポーネント名を持っているだけではなく，ディレクトリ階層に関する情報も保持している．これにより複数のコンポーネントをモジュールとして管理できるので，開発者が無駄なコンポーネントに関与することの回避が可能である．後者は，開発者がリポジトリに対してオペレーションを行う際に，リビジョンを指定する為の機構である．具体的には，各リビジョンが保持する属性値でのフィルタリングにより，特定のリビジョンを指定する．

2.1.3 The Long Transaction Model

このモデルは複数の開発者でリポジトリを共有し開発するという状況における問題を回避するための定義をしている．

具体的には，作業領域 (workspace) および 並列処理規則 (concurrency control scheme) と

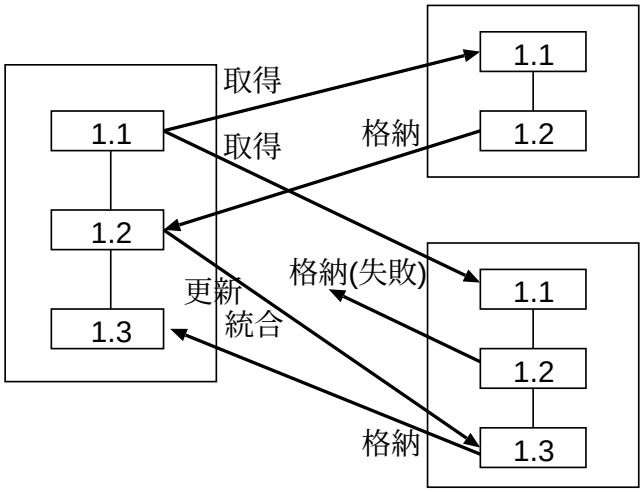


図 2: Concurrency Control Scheme の例

いう２つの概念を定義している．前者は，リポジトリとは別の開発者の専用の領域を意味し，リポジトリよりこの領域へファイルを取り出し，開発者はこのスペース開発を行う．開発終了後，本来のリポジトリへデータの格納を行う．後者は，複数の開発者が関与する場合に生じる問題 (競合の回避，同期など) を解決するための枠組である (図 2 参照).

2.1.4 The Change Set Model

リビジョン間の差分情報集合に関する記述をしているのが，このモデルである．

ファイルデータを管理するに当たり，初期リビジョンのデータはそのまま格納し，以降のデータ格納の際は，前のリビジョンとの差分データを１ブロックとして保持する．従って，各コンポーネントには，１つのベースとなるデータおよび，複数の差分データブロックが存在する．しかし，リポジトリの内部には複数のコンポーネントが存在する．

差分データ集合 (change set) とは，これら複数のコンポーネントに対する差分情報集合である (図 3 参照)．これにより，複数のコンポーネントに対するデータ操作が容易になる．

2.2 バージョン管理システム

バージョン管理システムと呼ばれるものは多数あり，OS 付属のものから，商用の大規模なシステムまで存在する．また，ローカル環境で利用システムから，ネットワークを介したものまである [12][14][20]．以下，主な物として，RCS, CVS, Visual SourceSafe がある．

- RCS

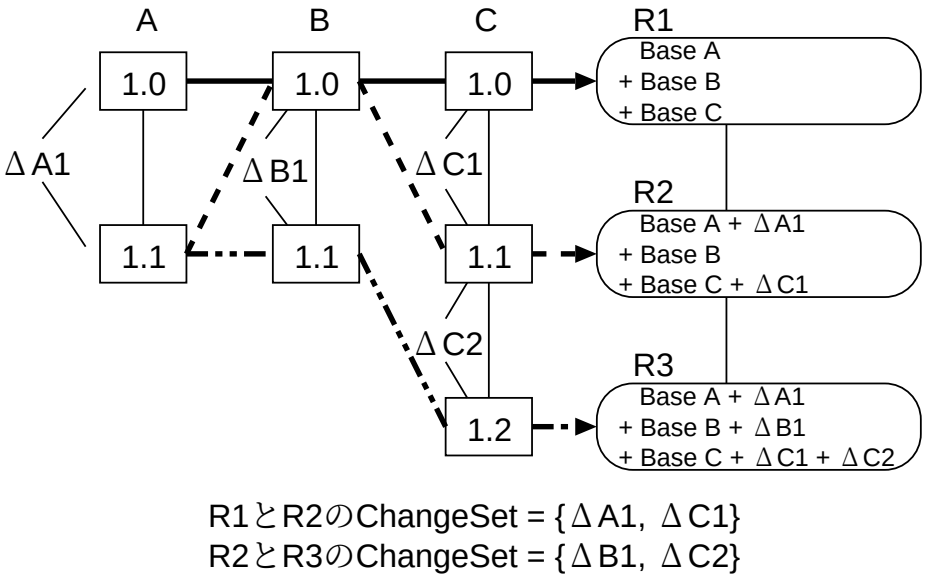


図 3: Change Set の例

RCS[27] は UNIX 上で動作するツールとして作成されたバージョン管理システムである．現在でもよく使われいているシステムである．単体で使用される他，システム内部に組み込み，バージョン管理機構を持たせる場合などの用途もある．RCS ではプロダクトをそれぞれ UNIX 上のファイルとして扱い，１ファイルに対する記録は１つのファイルに行われる．

RCS におけるリビジョンは，管理対象となるファイルの中身がそれ自身によって定義され，リビジョン間の差分は diff コマンドの出力として定義される．各リビジョンに対する識別子は数字の組で表記され，数え上げ可能な識別子である．新規リビジョンの登録や，任意のリビジョンの取り出しは，RCS の持つツールを利用する．

- CVS

CVS[4][11][15] は RCS 同様，UNIX 上で動作するシステムとして構築されたバージョン管理システムであり，近年最も良く使われるシステムの１つである．RCS と大きく異なるのは，複数のファイルを処理する点である．また，リポジトリを複数の開発者で利用することも考慮し，開発者間の競合にも対処可能となっている．さらに，ネットワーク環境 (ssh, rsh 等) を利用することも可能である為，オープンソフト開発やグループウェアの場面で活躍の場が多い．その最たる例が，FreeBSD や OpenBSD 等のオペレーティングシステムの開発である．

- Visual SourceSafe

Microsoft 社から出ているソフトウェア開発環境 Visual Studio における，ソースコードなどのリソースを管理しているのが，Visual SourceSafe[28] である．このように，バージョン管理システムを，開発環境に組み込むことで，ビルドシステムなど他の部分との連系がスマートになる．

2.3 バージョン管理システムの問題点

ネットワーク技術の発展により，分散環境でのソフトウェア開発が進んでいる [13]．また，オープンソフト開発も行われ，分散化に拍車がかかっている．この場合のソフトウェア管理おいて，バージョン管理システムを採用した際に，以下のような問題が生じる．

2.3.1 コンポーネント中心のバージョン管理

既存のシステムにおいて，バージョン管理される対象は主に，作成されるコンポーネント (ファイルなど) である．開発者に関する情報は，各リビジョンの実データの付属データという位置付けである．従って，ソフトウェアの履歴を得るのは容易であるが，開発者個人の履歴を取得するのは困難である．個人の履歴を蓄積する為に，別のシステムを利用する方法もあるが，バージョン管理システムとの間が非同期となる．

2.3.2 複数あるリポジトリの処理

開発環境が拡大すると，特に，オープンソースでの開発となると，元々単一のリポジトリでの開発が，複数のリポジトリへ分散する状況がある．複数のマシンに分散している場合が多々存在する．しかし，１プロジェクトで利用するコンポーネントが複数のリポジトリに跨る場合，各々のリポジトリよりファイルを取り出すことになり，開発者側の負担が多くなる．

2.3.3 コンポーネントの複製

コンポーネントの再利用を行う際，リポジトリ間での複製を行われる．この場合，データが冗長になるだけでなく, 複製時以降に元のコンポーネントに対して施された変更が複製先のものに反映されず，再度明示的な複製の必要性が存在する問題がある．

3　提案するバージョン管理手法

3.1　キーアイデア

前節の問題を解決するために，新たなバージョン管理手法**DiRM/VR** を提案する [25]. 本手法では，**抽象リポジトリ**という新たなリポジトリを導入する．これにより以下の３つ事項が実現可能となる．

- 開発者単位の履歴蓄積
- 点在するリポジトリの一元管理
- コンポーネントの複製回避

この抽象リポジトリでは，プロダクトデータと開発者依存の属性データを個別に管理する．従って，開発単位が単一の形態を採らない開発環境に対して有効なソフトウェア管理の実現が可能である．本手法による問題解決方法は，以下の通りである．

1. プロダクトの履歴を蓄積する為のリポジトリとは別に，個々の開発者の履歴を蓄積するためのリポジトリが存在するので，ソフトウェアと開発者双方の履歴の採取が可能となる．
2. 複数のリポジトリから取り出したコンポーネントを，開発者単位で一元管理ができる．これは，先に述べたの開発者用のリポジトリで同時に行うことが可能である．
3. リポジトリ内に存在するコンポーネントを再利用する際，複製するのではなく，共有する．ここでの共有とは，極めて参照に近い概念である．従って，元のコンポーネントに変更点が生じた場合，その内容を容易に閲覧できる．これは，共有しているリビジョンと，更新されたリビジョンとは，同一のリビジョンツリーに存在するものとして認識可能だからである．

3.2　抽象リポジトリ

「抽象リポジトリ」は，**実データ管理部**，**属性データ管理部**，**ユーザインタフェース部**，**制御部**の４要素から構成される．これらの関係を，図 4 に示す．以下，各要素について説明する．

- 実データ管理部

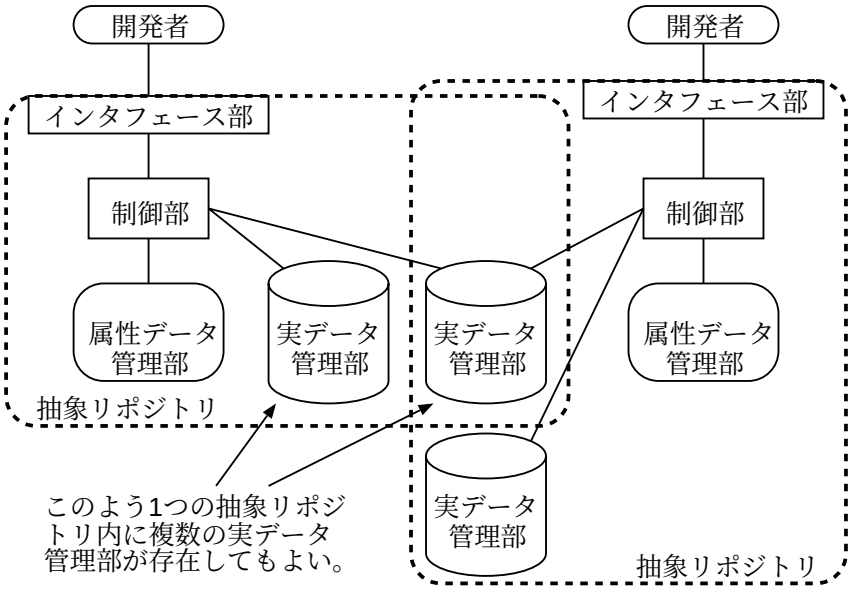


図 4: 抽象リポジトリの構成図

実データ管理部は，プロダクトデータをバージョン管理する部分である．ここでのプロダクトデータとは，ソースコードやリソースなどを指す．

抽象リポジトリの一構成要素ではあるが，これ自体は独立に存在するものである．これは実データ管理部が既存の開発環境におけるプロダクト管理部と同等の役割を担うからである．

この部分は複数の開発者での共有が可能である．従って，ある開発者の変更点を他の開発者にも簡単に反映させることが可能である．さらに，実データ管理部は複数のプロダクト管理用のリポジトリから構成されること前提としている．

また，開発者がデータ格納および変更を行う際，実データ管理部に直接的なアクセスすることを禁止する．この部分はあくまでも，プロダクトデータの管理部であり，開発者単位のデータ管理部ではない．開発者は属性データ管理部にあるデータへアクセスし，その内容に応じて，制御部によりアクセスすることになる．これにより，実データ管理部に存在するプロダクトデータの信頼性を高めることが可能となる．

- 属性データ管理部

開発者個人に依存したデータをバージョン管理するのが属性データ管理部である．開発者は抽象リポジトリへのアクセスにあたり，オペレーションを利用するが，その際に指定するコンポーネント名やリビジョン番号などは，この属性データ管理部が保持しているデータや値となる．

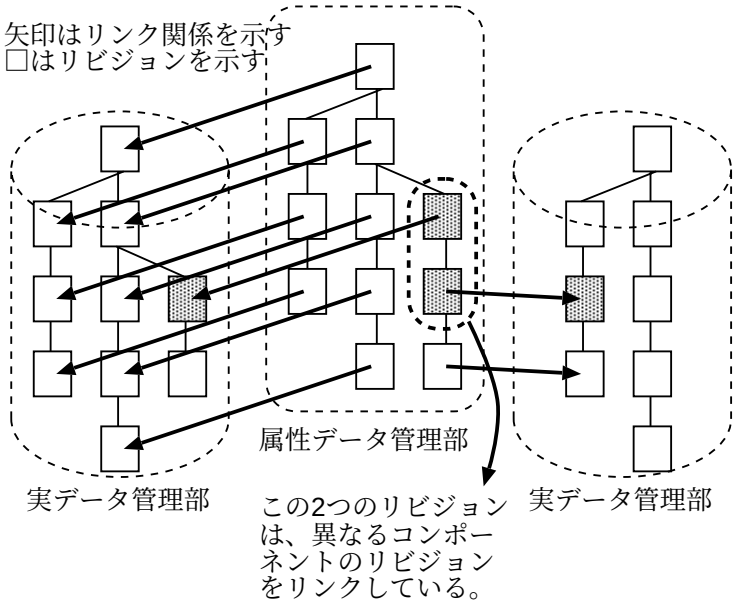


図 5: 異なるコンポーネントへのリンク

実データと同様に，バージョン管理システム下にある為，リビジョン単位で構成されている．しかし，実データ管理部と決定的に異なるのは，実データ保持せず，代りに実データへのリンク情報を保持する点である．つまり，属性データ管理部内に格納されている開発者毎のデータと，このリンク情報で示された実データを合わせることにより，従来のバージョン管理の１つにリビジョンに相当する．従って，開発者はプロダクトデータのリビジョン構成と独立した，自己の開発に依存する管理が可能となる．

上記の様に，属性データ管理部の１つのリビジョンに対して，実データ管理部の１リビジョンが対応するわけであるが，属性データ管理部内の１コンポーネントが保持する複数のリビジョンは，互いに，実データ管理部における異なるコンポーネントのリビジョンを指し示すことを許す (図 5)．これにより，開発者は実データ管理部内のコンポーネントを，より自由度の高いレベルで組み合わせて利用することが可能となる．

- ユーザインタフェース部

開発者が，リポジトリとのデータ操作を行なう為には，オペレーションを発行する必要がある．ユーザインタフェース部ではこのオペレーションを受け取り，解析する役割を果たす．

- 制御部

ユーザインタフェース部より渡されたオペレーション解析結果を分析し，それに応じたデータ操作を行うのが，この部分の役割である．データ操作については 3.3 また，他に以下

のような役割も担っている．

- 開発者に，実データ管理部と属性データ管理部を統合した１つのリポジトリとして認識させる．これにより，開発者は既存のバージョン管理システムと同様の感覚で，リポジトリへのアクセスが可能となる．
- 実データ管理部内への直接アクセスを避ける．先述してあるが，これは実データの信頼性を高めるための処置である．

3.3 データ操作

データ操作には．以下の３つが存在する．

- リビジョンの作成 (データ格納)
- データの取得
- リビジョン間の差分データ計算

本手法では２種類の性質の異なるデータ管理部に対してデータの追加・削除・変更を行う為，既存のシステムのデータ操作とは異なる．次節より順に説明する．

3.3.1 リビジョン作成 (データの格納)

開発者がリポジトリへデータを格納を行うと，新たにリビジョンが作成される．これは従来のバージョン管理システムにおける概念である．本手法においては，実データ管理部内へのリビジョン作成の有無に依存せず，属性データ管理部にリビジョン作成する操作をデータ格納と呼ぶ．データ格納には３方式ある．以下，順に説明する．

- 従来方式

これは，開発者が各々の作業領域においてコンポーネントに変更を行った後，そのデータをリポジトリへ格納することにより，新しいリビジョンを作成する方法である．まず，実データ管理部の適当なリビジョン (これは，実データ管理部の実装に依存) として格納される．この時の情報 (コンポーネント名，リビジョン番号など) を元に，そのリビジョンへのリンク情報を含むリビジョンを属性データ管理部に作成する (図 6 の左参照)．ただし，属性データ管理部内の各リビジョンは，必ず実データ管理部の１リビジョンを指すので，実データ管理部でのリビジョン作成に失敗した場合は，属性データ管理部においてもリビジョン作成失敗ということになる．

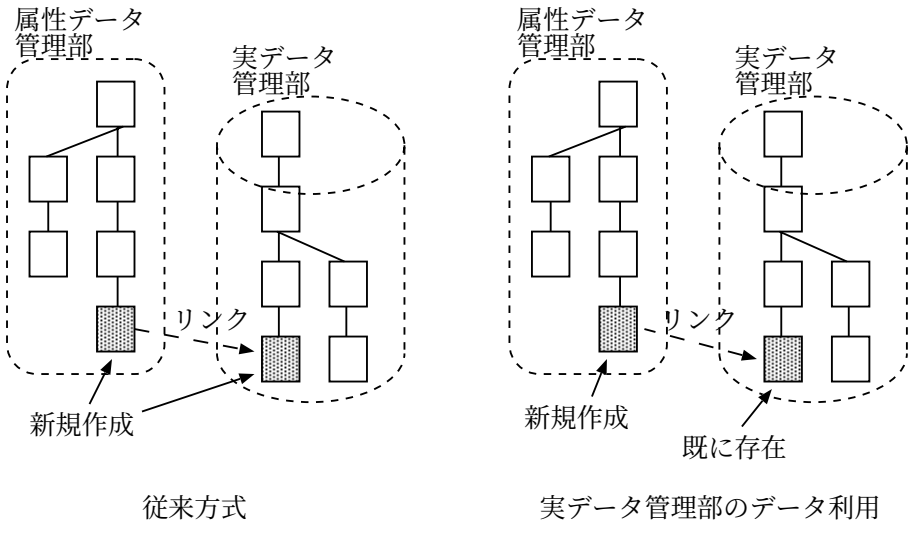


図 6: リビジョン作成の方式

- 実データ管理部のデータ利用

本手法特有のリビジョン作成方法である．上記の手法との相違点は，既に実データ管理部に存在するデータを利用する部分である．この方法では，実データ管理部内のコンポーネントの１リビジョンへのリンク情報を持ったリポジトリデータが属性データ管理部に作成されることになる (図 6 の右参照)．属性データ管理部の側から観測すれば，結果として，リビジョンが１つ追加されるだけであり，この点は従来方式と同じである．

- リビジョンツリー参照

これも，本手法特有の操作である．この手法では，実データ管理部内にある１つのコンポーネントのリビジョンツリーにおいて，その全体もしくは部分ツリー上にあるすべてのリビジョンを，属性データ管理部でのリビジョンとして利用する (図 7 参照)．このとき，それぞれの部分リビジョンツリーの構成は完全に同期させることにする．従って，実データ管理部において，取り込み時以降にリビジョンが作成された場合，明示的にそのリビジョンを取り込むことなく利用することが可能となる．逆に，属性データ管理部においては，派生リビジョンの作成を禁止する．これは，実データ管理部内のリビジョン構成と属性データ管理部内のリビジョン構成に矛盾を生む可能性があるからである．

3.3.2 データの取得

開発者はデータを取得するにあたり，属性データ管理部内におけるリビジョンを指定する．そして，そのリビジョンが保持しているリンク情報をもとに，実データ管理部内に存在

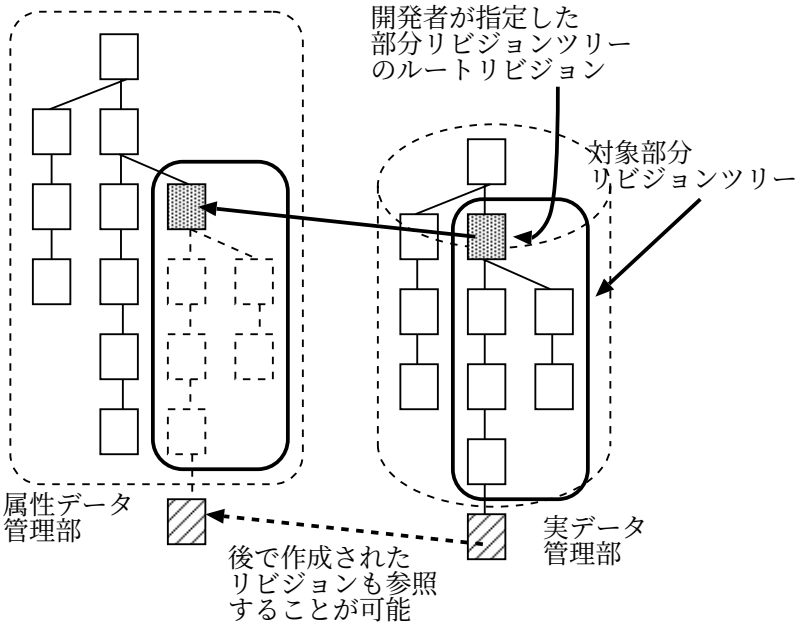


図 7: リビジョンツリー参照

する目的のリビジョンデータを取り出す．制御部において，それぞれの管理部から取り出したデータを編集して，開発者側へ渡す．当然のことであるが，開発者が指定したリビジョンが存在しない，もしくは，リンク情報で指定した実データ管理部側のリビジョンが存在しない場合は，データ取得できない．

3.3.3 リビジョン間の差分データ計算

差分情報の閲覧時だけでなく，リビジョン統合時も必要となるのが，２つのリビジョン間の差分データ情報である．開発者は２つのリビジョンを指定し，それぞれのリンク情報が指す実データ管理部内のそれぞれのリビジョンの差分情報を得ることなる．実データ管理部でのデータを保持するにあたり，前後のリビジョン間の差分情報を保持していることが多い (実際は，実データ管理部の実装に依存)．よって，本来ならこれらのデータを利用するべきであるが，本手法においては利用不可能である．それは，属性データ管理部の１コンポーネントにおける異なるリビジョンの実体が，互いに，実データ管理部において，別々のコンポーネントのリビジョンである可能性があるからである (図 8 参照)．そのため，指定した２つのリビジョンの実体をそれぞれ取り出し，明示的に差分データを計算する必要がある．また，データ取得と同様，差分データ計算するに際に必要なデータが欠落している場合には，計算されない．

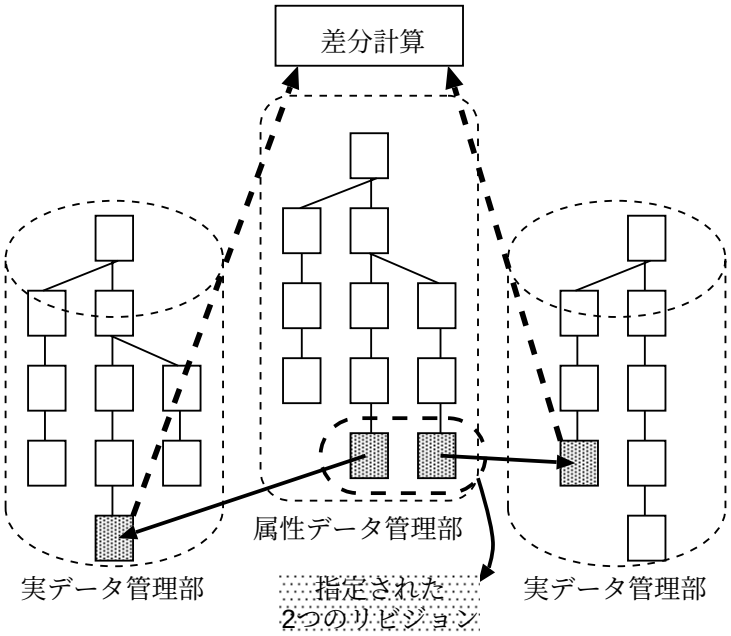


図 8: 差分情報取得方法

3.3.4 リビジョンの共有

これまでに述べたデータ操作手法により，複数の開発者間で，コンポーネントのリビジョン単位での共有が可能となる．これは，データの格納の部分に示した「実データ管理部のデータ利用」でのリビジョン作成が可能となるからである．これにより，以下のような利点が得られる．

1. 必要なリビジョンのみの共有が可能となるので，同一コンポーネントにおける他の不必要なリビジョンを引用しなくて済む．このことは，開発者レベルでのソフトウェア管理において有効である．

2. 同じリビジョンを共有している他の開発者により行われた更新事項を容易に反映することが可能となる．また，逆も同様で，自分の行った変更を他の開発者へ適応するのも簡単である．

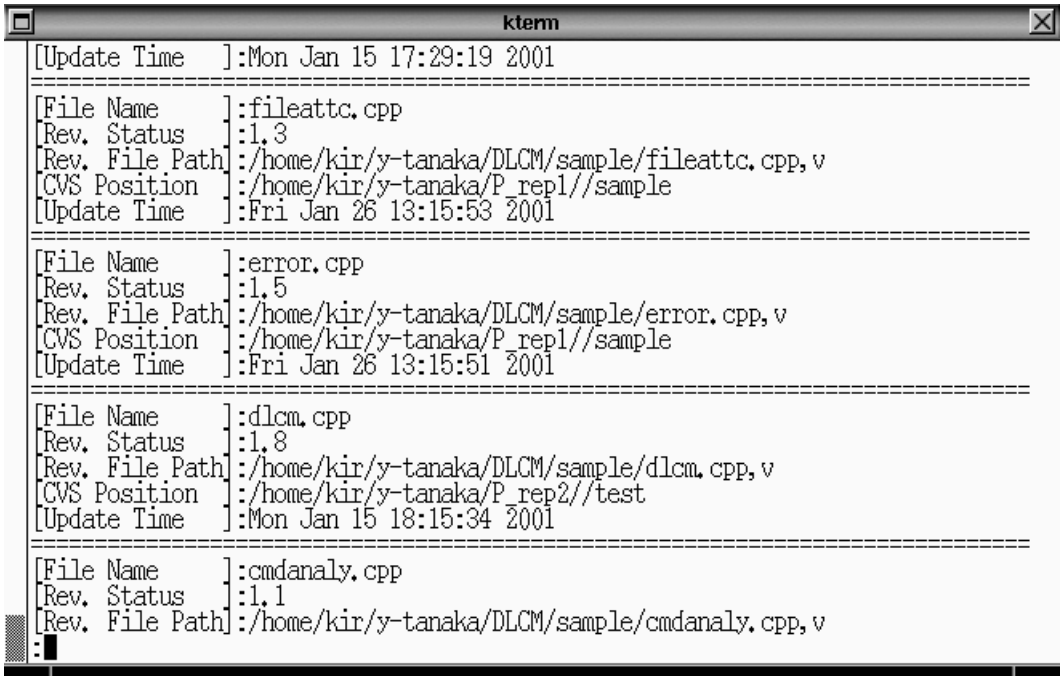


図 9: DLCM の画面 (”dlcm status”を実行)

4 システムの試作

4.1 DLCM

先に述べた，バージョン管理手法，データ操作手法に基づいたバージョン管理システム DLCM の試作を行った (画面写真：図 9,10)．開発環境は以下の通りである．

- CPU : Pentium3 1GHz ×2
- RAM : 1Gbyte
- OS : FreeBSD 4.2-STABLE
- 言語 : C++
  コード量は 8500 行程度 + ライブラリ (STL)

4.1.1 DLCM の構成

本システムの構成を図 11 に示す．以下で，各部の実装説明を行う．

- 実データ管理部

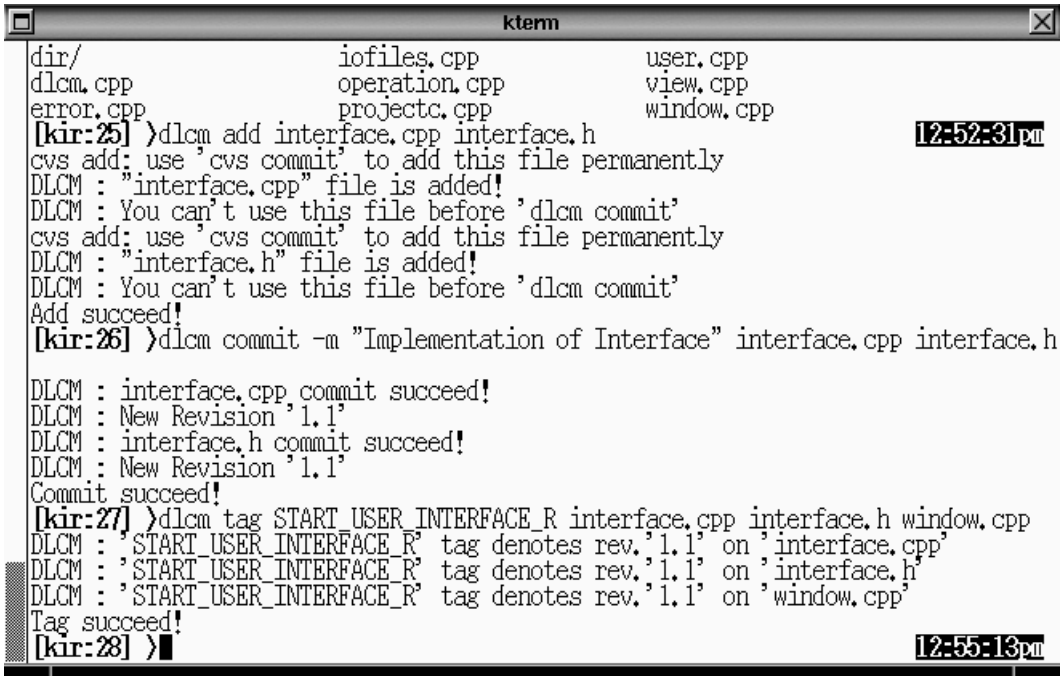


図 10: DLCM の画面 (種々のオペレーション実行)

本システムでは，既存のシステムである CVS のリポジトリを利用する．従って，この部分の実装は CVS に依存である．

- 属性データ管理部

この部分には，各コンポーネントのリビジョンデータ (実データ以外) が格納される．各リビジョンで保持する属性データは次の通りである．

- リビジョン番号
- タグデータ
- 作成日時
- 作成方法
- 実データ位置
- ブランチ
- メッセージ

１コンポーネントは複数のリビジョンデータを持つが，それらは１ファイルで管理する (ファイルの中身は後に記す)．ただし，タグデータのみ別ファイルに格納している．属性データ管理部には，このファイルが複数存在し，ディレクトリを構成している．従って，作業領域でディレクトリがネストして存在している場合，属性データ管理部においても，その構成と一致するディレクトリ構成をしている．

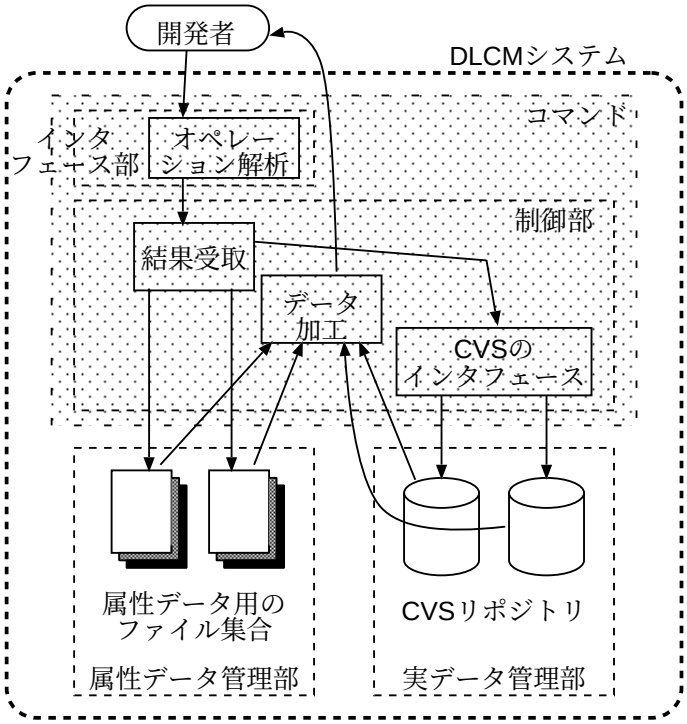


図 11: DLCM の構成

---------- 属性データ管理部のファイル内容 ----------

'#' 以降は，解説.

```
@@@@@                                   #--- ファイル開始
filename:scall.cpp                            # ファイル名
makeuser:y-tanaka                             # 作成者
makedate:Mon Jan 15 17:29:15 2001             # 作成日時

@@@                                     #--- リビジョンデータ開始
1.4                                           # リビジョン番号
Mon Jan 15 20:42:09 2001                      # 作成日時
commit                                        # 作成方法 (オペレーション)
/home/kir/y-tanaka/P_rep2/                    # リンク先の場所およびファイル名
test
scall.cpp
1.9                                           # リンク先でのリビジョン番号
```

```
@@                                      #--- ブランチデータ開始
1.4.1/Tue Jan 16 16:57:38 2001                  # ブランチ番号/作成日時
1.4.2/Tue Jan 16 16:57:40 2001
1.4.3/Tue Jan 16 16:57:41 2001
@                                       #--- メッセージログ開始
                                                # メッセージログ (無し)

@@@
1.3
Mon Jan 15 20:14:25 2001
commit
/home/kir/y-tanaka/P_rep2/
test
scall.cpp
1.4
@@                                              # ブランチデータ (無し)
@

                    ----------- (中略) ------------

@@@
1.1
Mon Jan 15 17:29:03 2001
commit
/home/kir/y-tanaka/P_rep2/
test
scall.cpp
1.1

@@
@
This is test                                    # メッセージログ

@@@@@                                           # ファイル終了
```

- インタフェース部

この部分は開発者が，データ操作を行う為に発行したオペレーションの解析を行う部分である．解析結果は制御部へ渡す．

- 制御部

この制御部は大きく３つの部分から成り立つ．１つ目は，開発者が発行したオペレーションの解析結果をインタフェース部から受け取り判断する**判断部**である．受け取った結果に基づいて，必要となるデータ操作を行う．２つ目が，**CVS インタフェース部**である．これは，実データ管理部の実装として利用している CVS リポジトリへのアクセスを行う部分である．最後３つ目が，各データ管理部より取り出したデータの操作を行う**データ解析部**である．具体的には，差分情報の計算や，リビジョンの統合，開発者へ渡すデータの編集などを行う．差分計算にあたっては，diff コマンドを利用し，リビジョンの統合には patch コマンドを利用している．

4.1.2 オペレーション

開発者がデータ操作を行うためには，オペレーションの発行する必要がある．本システムで実装した 17 のオペレーションを以下で簡潔に示す．尚，オペレーションの具体的な仕様は付録「DLCM のオペレーション」参照．

1. データ格納に関するオペレーション

    - commit
      データの格納を行う．前もって add や delete 実行されている場合，それぞれデータの新規格納，削除が行われる．
    - pcommit
      実データ管理部内のリビジョンを利用したデータ格納を行う．実データ管理部には何の変化も生じない．
    - sqcommit
      実データ管理部内における複数のリビジョンを利用したデータ格納を行う．実際には，２つのリビジョンを指定し，その間および両端のリビジョンすべてを取り出す．このとき，指定する２つのリビジョンは,「一方が他方から派生したもの」という制約を満たす必要がある．

- rtcommit
  実データ管理部内の部分リビジョンツリーを利用したデータ格納を行う．取り込んだ部分は実データ管理部のリビジョンツリー構成と全く同じになる．

2. データ取得に関するオペレーション

- checkout
  データの取り出しを行う．プロジェクト名のディレクトリを作成し，その下に取り出したデータを保存する．
- update
  データの更新を行う．
- log
  コンポーネントのリビジョンデータを出力する．ブランチ番号でのフィルタリングも可能である．

3. 差分情報取得に関するオペレーション

- diff
  リビジョン間の差分情報を出力する．差分の計算には，diff コマンドを内部で利用している．
- merge
  リビジョンの統合を行う．差分計算には UNIX の diff コマンドを用い，統合には UNIX の patch コマンドを利用する．従って，patch コマンドの実行が失敗した場合は無効となる．また，統合されたデータの保証に関しては別問題なので，これに関しては認知しないことにする．

4. 上記の３つのいずれにも属さないオペレーション

- begin
  属性データ管理用リポジトリの新規作成を行う．最初に必ず実行するオペレーションとなる．
- proj
  プロジェクトの作成を行う．属性データ管理部内にプロジェクトを作成し．デフォルトで使用する CVS リポジトリを実データ管理部内に作成する．
- regp
  実データ管理部の所在位置をエイリアス登録する．これにより，実データ管理部の位置指定をエイリアス名で行うことが可能となる．

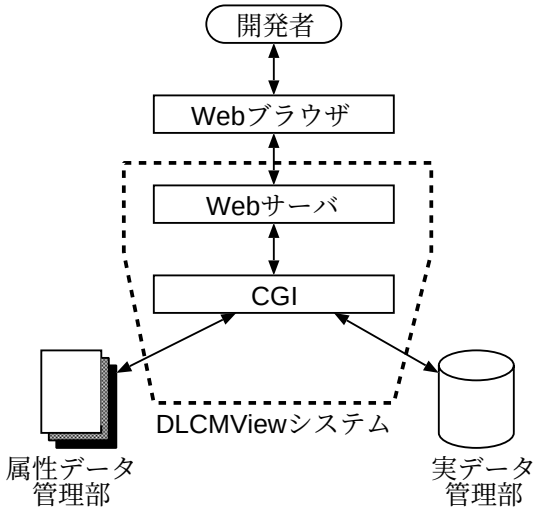


図 12: DLCM View の構成

- add
  コンポーネントの追加を行う．ローカルに保持しているファイルのエントリーリストにファイル名が追加されるだけで，実際にデータが格納されるわけではない．
- delete
  コンポーネントの削除を行う．add と同様，この時点ではデータ削除されない．
- branch
  ブランチの作成を行う．作成と同時に，そのブランチに対するタグを付けることが可能である．
- tag
  リビジョンにタグを付ける．
- status
  作業中のコンポーネントの状態を出力する．

4.2 DLCM View

前節で述べたバージョン管理システム DLCM が管理している開発者単位の情報を閲覧するためのシステム DLCM View も作成した．開発環境は以下の通りである．

- CPU : Pentium3 1GHz ×2
- RAM : 1Gbyte

- OS：FreeBSD 4.2-STABLE
- 言語：Perl
  コード量は 1600 行程度 + ライブラリ (jcode.pl)

このシステムは，CGI エンジンを利用しており，既存の Web ブラウザでの情報閲覧が可能である (図 12 参照)．閲覧機能として４種類のモードを備えている (画面写真：図 13〜16)．

- コンポーネント一覧モード
  属性データ管理部に格納されているコンポーネントを表示する．各コンポーネントに対して５つの項目が記される．また，項目でのソートも可能である．
- リビジョン一覧モード
  各コンポーネントに存在するリビジョンを表示する．ブランチやトランクなどの条件でフィルタリングが可能である．
- リビジョンデータモード
  各リビジョンのデータを表示する．実データ，属性データ共に出力される．
- 差分データモード
  指定したリビジョン間の差分情報を表示する．

この中で，リビジョンデータモードおよび差分データモードでは，実データを必要とするため，RCS の co オペレーションを利用して実データ管理部からデータの取り出しを行っている．また，差分情報に関しては diff コマンドを内部的に利用している．

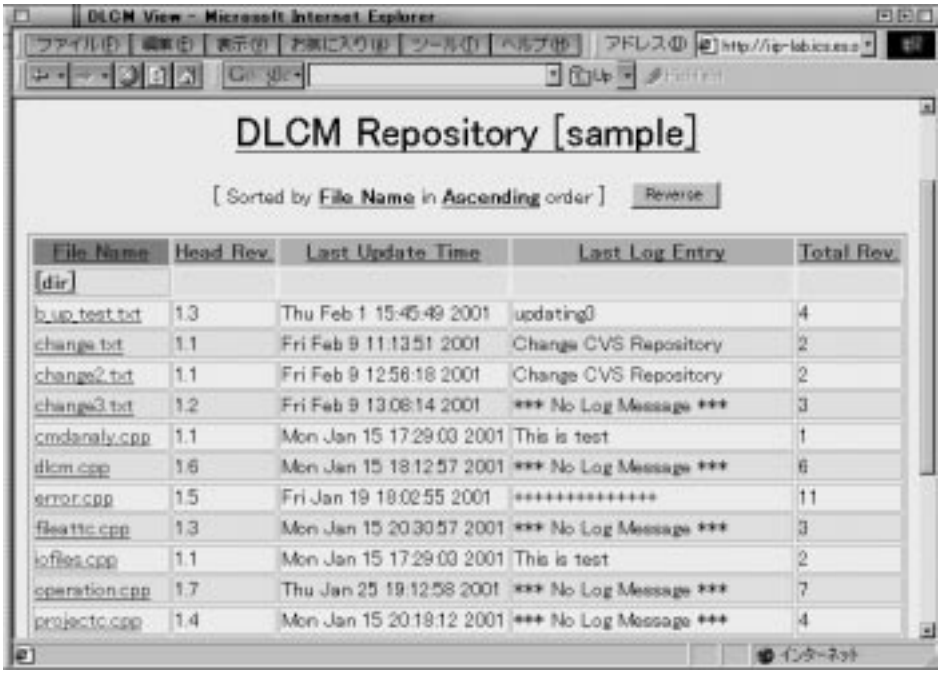


図 13: コンポーネント一覧モード

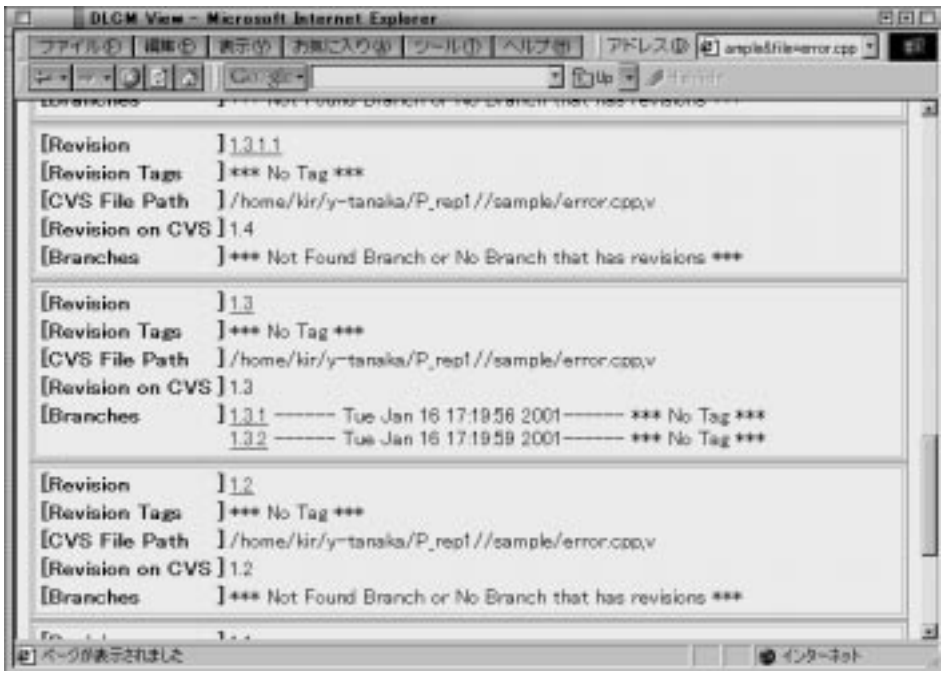


図 14: リビジョン一覧モード

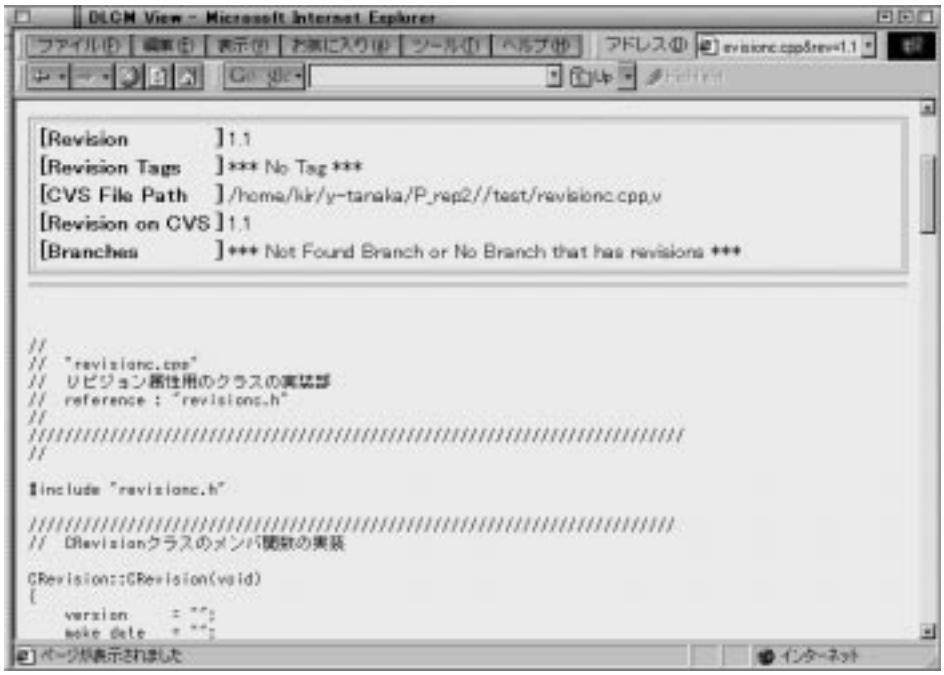


図 15: リビジョンデータモード

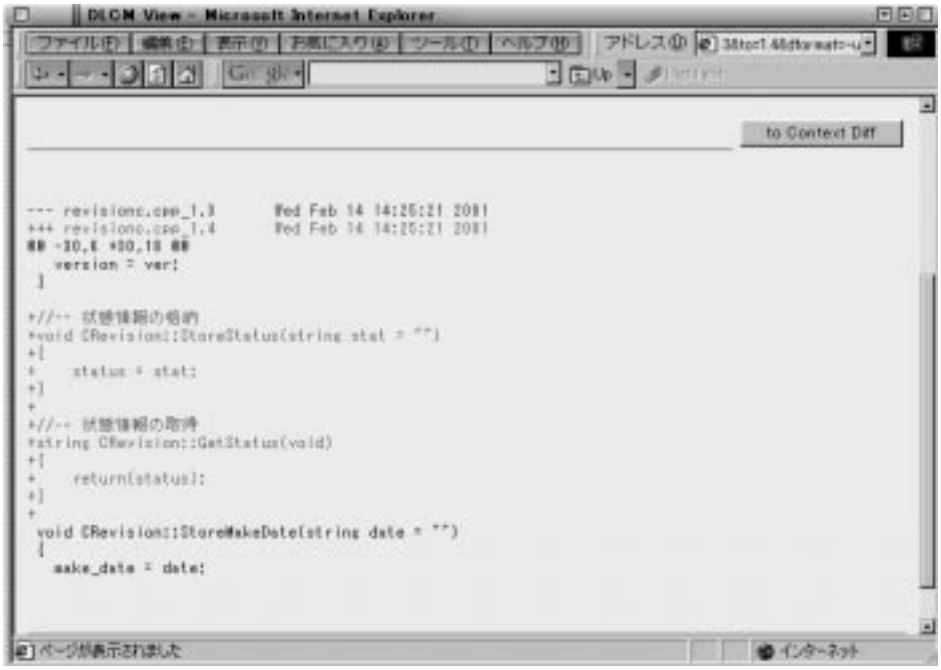


図 16: 差分データモード

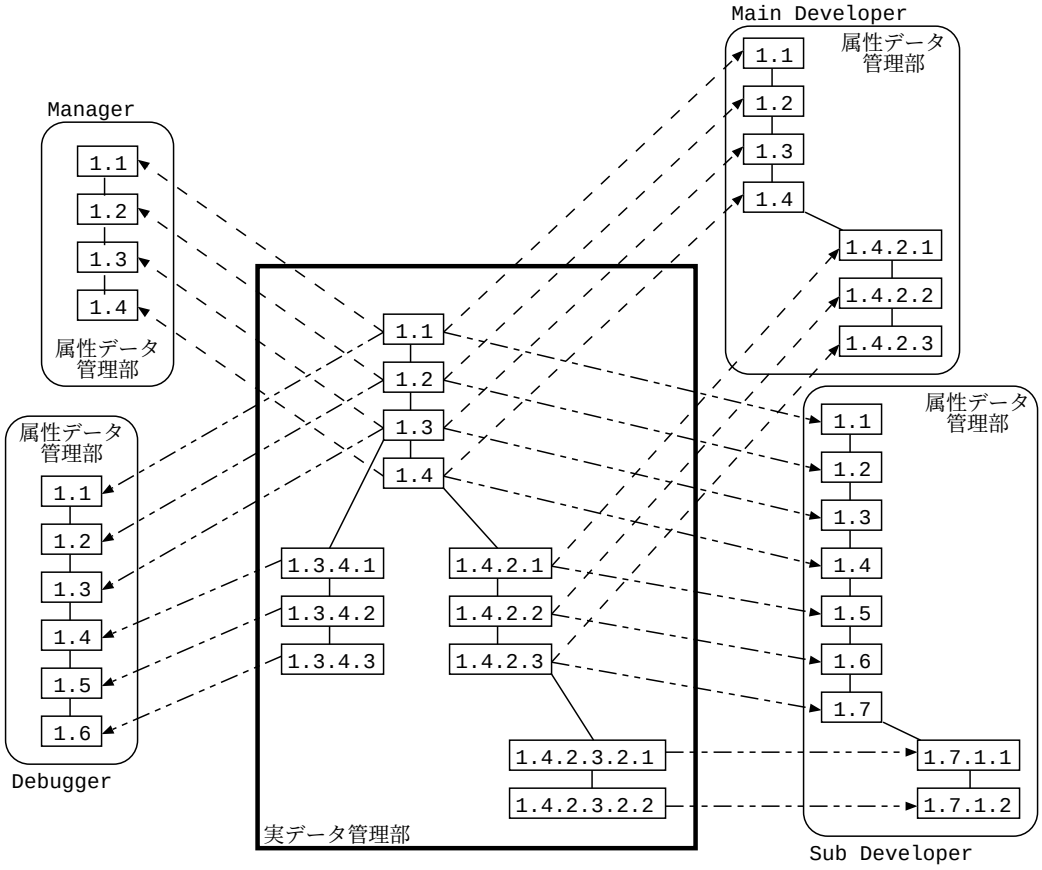


図 17: 各開発者の履歴

5 実験・評価

5.1 適応実験

本論文で述べたバージョン管理手法により，2.3 で述べた問題点の解決が可能であること確認する為，試作したシステムを利用して実験を行った．それぞれについて以下で示す．

5.1.1 適応実験１

各開発者毎の履歴取得の例として，コンポーネントに対する開発者各々の視点での開発に関して簡単に行った．開発者は Manager, Main Developer, Sub Developer, Debugger の４人であり．それぞれに視点は以下の通りでる．

- Manager
  このコンポーネントの責任監督者．トランク上に存在するリビジョンの監視をする．

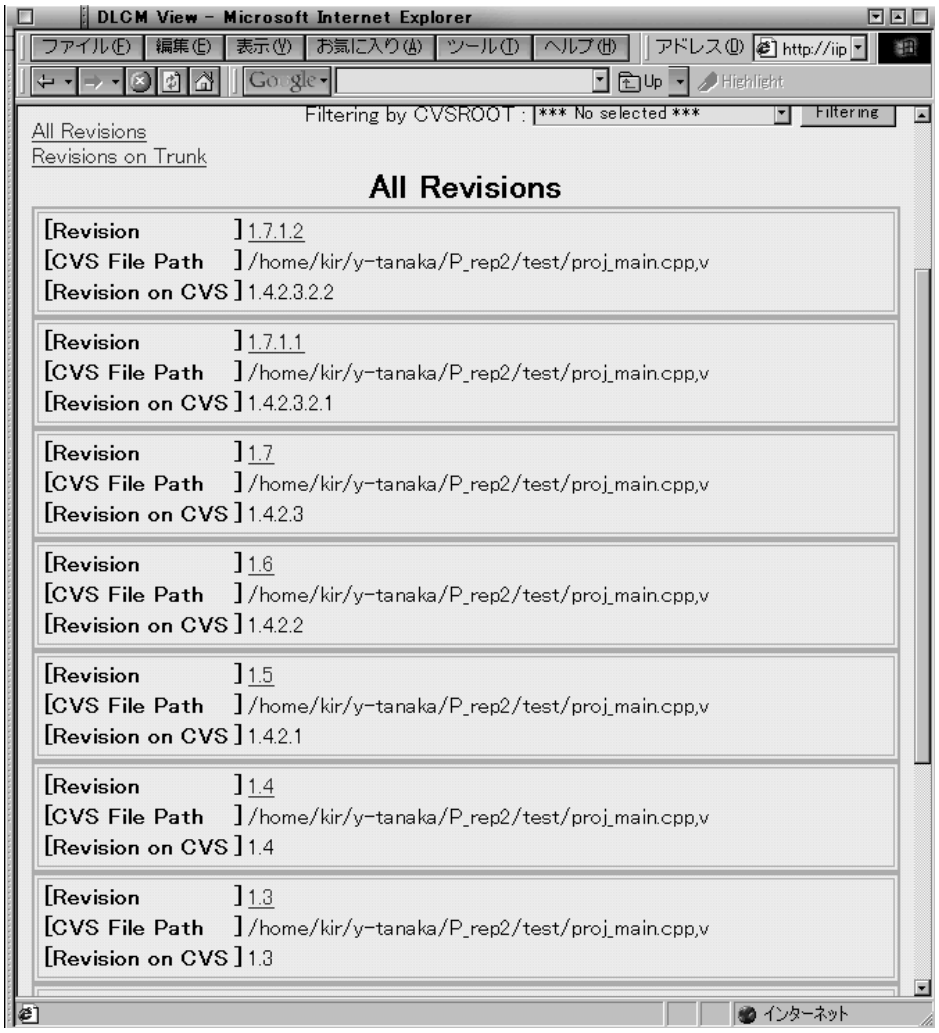
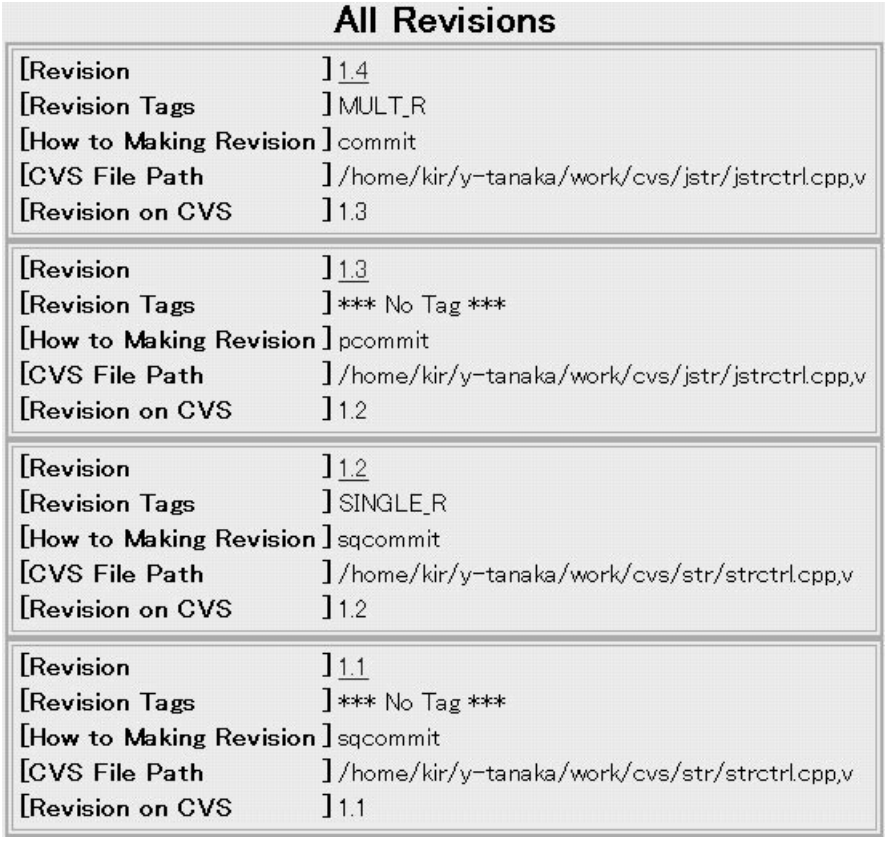


図 18: Sub Developer の開発履歴

- Main Developer
  実際の開発において中心となる開発者．トランク上のリビジョンから派生したブランチ上で開発を行い，トランクへのマージを行う．

- Sub Developer
  Main Developer の補佐的な仕事をする開発者．Main Developer の開発ブランチを参照しながら，さらに派生したブランチで担当の開発を行う．

- Debugger
  過去のリビジョンにおいて，発見されたバグを修正する開発者．



All Revisions

[Revision	] 1.4
[Revision Tags	] MULT_R
[How to Making Revision	] commit
[CVS File Path	] /home/kir/y-tanaka/work/cvs/jstr/jstrctrl.cpp,v
[Revision on CVS	] 1.3
[Revision	] 1.3
[Revision Tags	] *** No Tag ***
[How to Making Revision	] pcommit
[CVS File Path	] /home/kir/y-tanaka/work/cvs/jstr/jstrctrl.cpp,v
[Revision on CVS	] 1.2
[Revision	] 1.2
[Revision Tags	] SINGLE_R
[How to Making Revision	] sqcommit
[CVS File Path	] /home/kir/y-tanaka/work/cvs/str/strctrl.cpp,v
[Revision on CVS	] 1.2
[Revision	] 1.1
[Revision Tags	] *** No Tag ***
[How to Making Revision	] sqcommit
[CVS File Path	] /home/kir/y-tanaka/work/cvs/str/strctrl.cpp,v
[Revision on CVS	] 1.1

図 19: ”strctrl.cpp”のリビジョンデータ

ある時点での，コンポーネントのリビジョンツリーおよび，各開発者の状態を図 17 に示す．本手法において，開発者単位の開発履歴は属性データ管理部に蓄積されることになるので，開発者毎にそのデータを取り出すことで，収集することが可能である．実際の収集の際には，log オペレーションによって出力できるほか，DLCM View システムによっても閲覧が可能である (図 18 に，Sub Developer の履歴を示す．ただし，ここでは，属性データ管理部でのリビジョン番号と実データ管理部の所在位置およびリビジョン番号のみ示す)．従って，開発者単位での品質評価を行う場合，それに必要となるデータを，メッセージログとして格納しておくことにより，上記の手法で容易に取り出すことができる．

5.1.2 適応実験２

複数のリポジトリの一元管理の例として，日本語 (マルチバイト文字) を含む文字列処理に未対応のライブラリと対応したライブラリの差し替えを行った (実行結果は，図 20)．ライブラリの名前は，”strctrl.h”および”strctrl.cpp”である．図 19 に”strctrl.cpp”の DLCM View

図 20: 実行画面 (tty 出力)

での画面写真を示す．このデータにおいて，リビジョン 1.2 から 1.3 への移行時に，実データとして利用するファイルが”strctrl.cpp”から，”jstrctrl.cpp”に入れ替わっていることが分かる．さらに 1.4 が存在するのは，インクルードするファイルなどを一部書き替えが必要だからである．従って，異なるリポジトリに存在するコンポーネントであっても，上記の様に一元管理することが可能となり効果的である．

5.1.3 適応実験３

コンポーネントの複製を行わずに参照とする例して，FreeBSD と OpenBSD を利用した実験を行った．FreeBSD の openssh のファイルの大半は OpenBSD の ssh のファイルを複製している．そこで，FreeBSD と OpenBSD の CVS リポジトリ，および，ローカルに変更した内容を格納するための CVS リポジトリの計３つを実データ管理部とする，抽象リポジトリを作成し，ssh の実行ファイルを作成するという実験を行った (図 21 参照．画面では，FreeBSD，OpenBSD からのファイルの例として，それぞれ”dsa.h”,”getput.h”を挙げている)．この実験するにあたり，数十のファイルを扱うことになるが，その内５つのファイルの変更を必要とした (内３つは，パスの変更のみ)．変更したファイルは先の３つ目に述べた

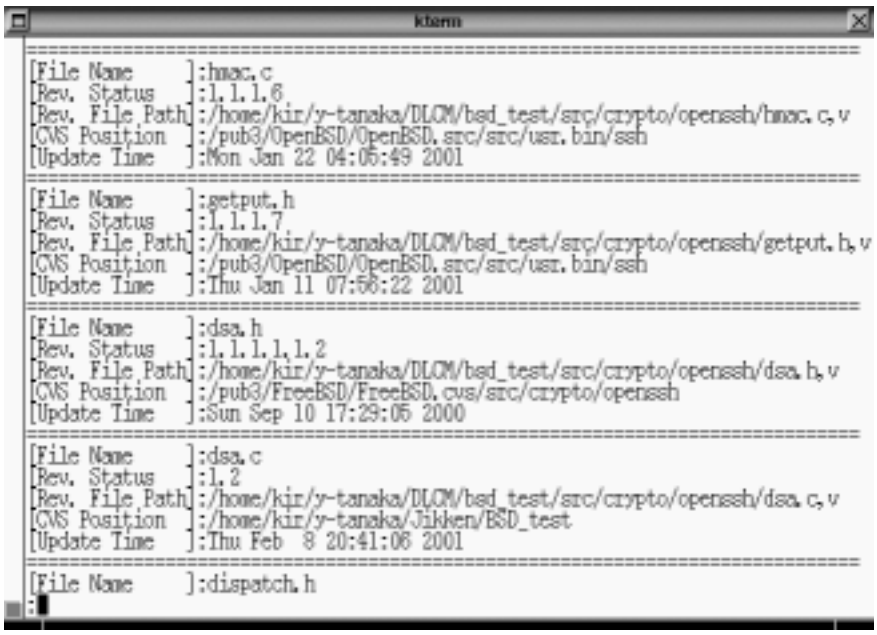


図 21: 実験後の各ファイルのステータス

CVS リポジトリに格納した (図 21 の”dsa.c”参照). OpenBSD 側でリビジョンが作成されても，必要な時のみ update オペレーションにより，ファイルを更新することが可能であり，複製する必要はなくなる.

5.2 評価

上記の様に，３パターンの適応実験を行ったが，既存のシステムで同様のことを行うためには以下のような手順を踏む必要がある (尚，番号は先の適応実験の番号と対応している).

1. 開発者が関与したリポジトリおよび，それぞれの内部に存在するコンポーネントを調べる.
   それらのファイルのすべてのリビジョン情報を取り出し，対象となっている開発者が操作したリビジョンを検出する.
   該当するリビジョンのみのデータを収集する.

2. 異なるリポジトリであるので，作業領域へファイルの取り出しを行う.
   適当なファイル名に変更する.
   変更を施した後は，所定の作業領域へファイルを移動させる．同時にファイル名も元に戻す.
   リポジトリへのデータ格納を行う.

3. 元のリポジトリにおける更新の有無を調べる．

更新がある場合，そのリポジトリデータを複製する．(これらのことを繰り返し行う)

このように，既存のシステムでは手間がかかるのだけではなく，開発者の中心ではなく，ソフトウェア中心のデータ管理であることが問題となる．しかし，提案した手法では，ソフトウェアと開発者の双方を考慮したデータ管理を行っているため，これらの問題に対して，スマートな解決が可能となる．

6 まとめ

本論文では，まず既存のバージョン管理システムでの３つの問題点を述べた．

- 開発者単位の履歴蓄積が困難
- 点在するリポジトリの一元管理が困難
- コンポーネント複製による問題

これらを解決する為，新たなバージョン管理手法 DiRM/VR を提案した．本手法では，プロダクトデータだけではなく，開発者に依存したデータも考慮している．実際には，プロダクトデータと開発者データを個別のリポジトリで管理を行い，それらを同期させることで，問題の解決を行った．

この手法に基づいたバージョン管理システム DLCM，および，データ閲覧システム DLCMView を試作した．また，これらのシステムを利用した３つの適応実験を行い，提案手法により問題点の解決が可能であることを確認した．

本手法により，開発者単位で履歴の蓄積が可能となる為，各開発者の開発に関する情報を利用して，個人レベルでのソフトウェア品質評価を行うことができる．また，複数存在する開発者間においてリビジョンを単位としたコンポーネント共有を行える．従って，開発者は各々の開発に沿った形で，必要に応じたリビジョンのみ参照・引用が可能となり，開発効率が向上につながる．

本手法における実データ管理部は複数の開発者によって共有されるため，リビジョン競合などが起こる．本システムでは，単純に別のリビジョンに格納することで競合を避けているが，実データの蓄積されているリビジョンツリーのみを閲覧すると，整合性がとれていない場合が存在する．一方，バージョン管理におけるブランチリビジョンおよびリビジョン統合の効果的な適応法に関する研究 [5] がされている．また，効果的なリビジョンの作成に関する研究 [1][3][19] も行われている．それらを参考に実データ管理部内でのデータ格納手法を定義することで，秩序のあるバージョン管理システムの構築が可能となると思われる．

また，試作したシステムのオペレーションを実行するにあたり，既存のシステムと比較して，長時間かかる．これは，データ操作を行うために必要となるファイル入出力の回数が多いからであることが判明している．システム試作において CVS を利用しており，CVS リポジトリへアクセスする場合には特に時間が必要となる．従って，ファイルアクセス回数を制限することにより，オペレーション実行時間を削減することが必要であると思われる．

謝辞

本論文を作成するにあたり，常に適切な御指導を賜わりました大阪大学大学院基礎工学研究科情報数理系専攻 井上 克郎 教授に心より深く感謝致します.

本論文の作成において，適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 楠本 真二 助教授に深く感謝致します.

本論文の作成において，適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 松下 誠 助手に深く感謝致します.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院基礎工学研究科情報数理系専攻井上研究室の皆様に深く感謝致します.

参考文献

[1] Makram Abu-Shakra and Gene L. Fisher, "Multi-Grain Version Control in the Historian System", ECOOP 98, SCM-8, LNCS1439, pp.46–56, 1998.

[2] Ulf Asklund, Lars Bendix, Henrik B. Christensen, and Boris Magnusson, "The Unified Extensional Versioning Model", 9th International Symposium, SCM-9, LNCS1675, pp.100–122, 1999.

[3] David L. Atkins, "Version Sensitive Editing: Change History as a Programming Tool", ECOOP 98, SCM-8, LNCS1439, pp.146–157, 1998.

[4] Brain Berliner, "CVS II:Parallelizing Software Development", In USENIX, Washinton D.C., 1990.

[5] Jim Buffenbarger and Kirk Gruell, "A Branching/Merging Strategy for Parallel Software Development", 9th International Symposium, SCM-9, LNCS1675, pp.86–99, 1999.

[6] Henrik Bærbak Christensen, "The Ragnarok Architectural Software Configuration Management Model", In Proceedings of the 32nd Hawaii International Conference on System Science, 1999.

[7] Henrik Bærbak Christensen, "Experiences with Architectural Configuration Management in Ragnarok", ECOOP 98, SCM-8, LNCS1439, pp.67–74, 1998.

[8] Reidar Conradi and Berbard Westfechtel, "Version models for software configuration management", ACM Computing Surveys, Vol. 30, No.2, pp.232–280, June 1998.

[9] Jacky Estublier, "Software Configuration Management:A Roadmap", The Future of Software Engineering in 22nd ICSE, pp.281–289, 2000.

[10] Peter H. Feiler, "Configuration Management Models in Commercial Environments", CMU/SEI-91-TR-7 ESD-9-TR-7, March, 1991.

[11] Karl Fogel, "Open Source Development with CVS", The Coriolis Group, 2000.

[12] Peter Fröhlich and Woflgang Nejdl, "WebRC Configuration Management for a Cooperation Tool", SCM-7, LNCS 1235, pp.175–185, 1997.

[13] André van der Hoek, Dennis Heimbigner, and Alexander L. Wolf, "A Generic, Peer-toPeer Repository for Distributed Configuration Management", Proceedings of ICSE 18, pp.308–317, March, 1996.

[14] James J. Hunt, Frank Lamers, Jürgen Reuter, and Walter F. Tichy, "Distributed Configuration Management via Java and the World Wide Web", ICSE'97, SCM-7, LNCS1235, pp.161–174, 1997.

[15] 鯉江 英隆, 西本 卓也, 馬場 肇 著, "バージョン管理システム (CVS) の導入と活用", SOFT BANK, December, 2000

[16] Magnus Larsson, Ivica Crnkovic, "New Challenges for Configuration Management", 9th International Symposium, SCM-9, LNCS1675, pp.232–243.

[17] Jyhjong Lin and Chunchou Yeh, "An Object-Oriented Formal Model for Software Project Management", Proceedings of APSEC, pp.552–559, December, 1999

[18] Yi-Jing Lin and Steven P.Reiss, "Configutarion Management with Logical Structures", Proceedings of ICSE-18, pp.298–307, 1996.

[19] Peter Lindsay and Owen Traynor, "Supporting Fine-Grained Traceability in Software Development Environments", ECOOP 98, SCM-8, LNCS1439, pp.67–74, 1998.

[20] Bartosz Milewski, "Distributed Source Control System", ICSE'97, SCM-7, LNCS1235, pp.98–107, 1997.

[21] 落水浩一郎, "分散共同ソフトウェア開発に対するソフトウェアプロセスモデルに関する基礎考察", 電子情報通信学会, SS2000-48(2001-01), pp.49–56, 2001.

[22] Koichiro Ochimizu, Hiroyuki Murakoshi, Kazuhiro Fujieda, Mitsunori Fujita, "Sharing Instability of a Distributed Cooperative Work", ISPSE2000, pp.33–42, 2000.

[23] Eric S. Raymond, "The Cathedral & the Bazaar", O'REILLY, 1999.

[24] Bradley R. Schmerl and Chris D. Marlin, "Versioning and Consistency for Dynamically Composed Configuration", ICSE'97, SCM-7, LNCS1235, pp.49–65, 1997.

[25] 田中義己, 松下誠, 井上克郎, "複数のリポジトリを共有できる仮想的なバージョン管理システムの提案", 情報処理学研究会, 2000-SE-129, pp.49–56, Nov, 2000.

[26] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr., "N Degrees of Separation: Multi-Dimensional Separation of Concerns", Proceedings of ICSE 21, pp.107–119, May, 1999.

[27] Walter F. Tichy, "RCS - A System for Version Control", SOFTWARE - PRACTICE AND EXPERIENCE, VOL.15(7), pp.637–654, 1985.

[28] Visual Source Safe
http://www.microsoft.com/japan/developer/ssafe/

[29] Darcy Wiborg Weber, "Change Sets Versus Change Packages: Comparing Implementations of Change-Based SCM", ICSE'97, SCM-7, LNCS1235, pp.25–35, 1997.

付録

DLCM のオペレーション

● begin [-d path]

*** 属性データ管理用リポジトリの作成

-d : 属性データ管理用リポジトリとして使用するディレクトリを指定する．このオプションを指定しない場合は環境変数 DLCMPATH で指定しているパスを用いる．

● proj projname cvsroot cvsrep [-d path]

*** プロジェクトの作成

projname : 作成するプロジェクト名

cvsroot : デフォルトで利用する CVSROOT へのパス

cvsrep : デフォルトで利用するプロジェクトへのパス．ただし，指定するパスは cvsroot からの相対パスとする．

-d : 属性データ管理用リポジトリとして使用するディレクトリを指定する．このオプションを指定しない場合は環境変数 DLCMPATH で指定しているパスを用いる．

● regp aliasname cvsroot cvsrep

*** CVS リポジトリのエイリアス登録

aliasname : エイリアスとして使用する名前

cvsroot : デフォルトで利用する CVSROOT へのパス

cvsrep : デフォルトで利用するプロジェクトへのパス．ただし，指定するパスは cvsroot からの相対パスとする．

● add [-a cvsroot cvsrep] [-l aliasname] files...

*** リポジトリへのファイル追加

files... : リポジトリへのファイル追加

-a : 追加先の CVS リポジトリのエイリアス名を指定する．指定のない場合は，プロジェクト作成時に指定した場所になる．また，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-l : 追加先の CVS リポジトリの CVSROOT へのパスおよび，その位置からのプロ

ジェクトへの相対パスを指定する．指定のない場合は，プロジェクト作成時に指定した場所になる．また，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

● delete　files...

*** ファイルの削除（実際に削除されるのは commit されたときである）

files...　：リポジトリからのファイルの削除

● commit　[-r num] [-a cvsroot cvsrep] [-l aliasname]
[-m message] [files...]

*** ファイルのチェックイン

files...　：リポジトリへのチェックインするファイル名．指定しない場合は，./CVS/DLCM_Entries にリストアップされているすべてのファイルを対象とする

-a　：格納先の CVS リポジトリのエイリアス名を指定する．指定のない場合は，作成されるリビジョンの親のリビジョンと同じ場所への格納を行う．"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-l　：格納先の CVS リポジトリの CVSROOT へのパスおよび，その位置からのプロジェクトへの相対パスを指定する．指定のない場合は，作成されるリビジョンの親のリビジョンと同じ場所への格納を行う．"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-m　：一緒に格納するログメッセージを書く．指定されない場合，カレントディレクトリに MSGFILE という名前のファイルが存在すれば，その内容をログメッセージとして使用し，存在しない場合は，vi を起動しそこに書かれた内容をログメッセージとして使用する．

-r　：格納先のブランチ番号およびブランチタグを指定する．トランクを指定する場合は，"-r tr" と入力することに可能である．また，このオプションが指定されない場合は，作業中のリビジョンと同じブランチまたはトランクに格納される．

● pcommit　[-r num] l_file [-a cvsroot cvsrep] [-l aliasname] cvsfile cvsnum [-m message]

*** 実データ管理部のファイルを利用したチェックイン

l_file　　: 開発者が利用するときのファイル名

cvsfile　: CVS リポジトリ内部でのファイル名

cvsnum　　: cvsfileで指定したファイルにおけるリビジョン番号

-a　: 格納先の CVS リポジトリのエイリアス名を指定する．"-a","-l"少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-l　: 格納先の CVS リポジトリの CVSROOT へのパスおよび，その位置からのプロジェクトへの相対パスを指定する．"-a"，"-l"の少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-m　: 一緒に格納するログメッセージを書く．指定されない場合，カレントディレクトリに MSGFILE という名前のファイルが存在すれば，その内容をログメッセージとして使用し，存在しない場合は，vi を起動しそこに書かれた内容をログメッセージとして使用する．

-r　: 格納先のブランチ番号およびブランチタグを指定する．このオプションが指定されない場合は，トランク上にリビジョンが作成される．

● rtcommit　[-r num] l_file [-a cvsroot cvsrep] [-l aliasname] cvsfile cvsnum [-m message]

*** 実データ管理部からのリビジョンツリーのチェックイン

l_file　　: 開発者が利用するときのファイル名

cvsfile　: CVS リポジトリ内部でのファイル名

cvsnum　　: cvsfileで指定したファイルにおけるリビジョン番号であり，この部分で示したリビジョンをルートとするリビジョンツリーが取り込まれることになる．

-a　: 格納先の CVS リポジトリのエイリアス名を指定する．"-a","-l"少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-l　: 格納先の CVS リポジトリの CVSROOT へのパスおよび，その位置からのプロジェクトへの相対パスを指定する．"-a"，"-l"の少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-m : 一緒に格納するログメッセージを書く．指定されない場合，カレントディレクトリにMSGFILEという名前のファイルが存在すれば， その内容をログメッセージとして使用し，存在しない場合は，viを起動しそこに書かれた内容をログメッセージとして使用する．

-r : 格納先のブランチ番号およびブランチタグを指定する．このオプションが指定されない場合は，トランク上にリビジョンが作成される．

● sqcommit [-r num] l_file [-a cvsroot cvsrep] [-l aliasname] cvsfile fromnum tonum [-m message]

*** 実データ管理部からのシーケンスなリビジョン列のチェックイン

l_file : 開発者が利用するときのファイル名

cvsfile : CVSリポジトリ内部でのファイル名

fromnum : このリビジョンを起点とするリビジョンシーケンスを取り込む

tonum : このリビジョンを終点とするリビジョンシーケンスを取り込む

-a : 格納先のCVSリポジトリのエイリアス名を指定する．"-a","-l"少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-l : 格納先のCVSリポジトリのCVSROOTへのパスおよび，その位置からのプロジェクトへの相対パスを指定する．"-a"，"-l"の少なくとも片方は存在しなければならす，"-l"と両方を指定した場合はコマンドラインで後に書いた方が優先される．

-m : 一緒に格納するログメッセージを書く．指定されない場合，カレントディレクトリにMSGFILEという名前のファイルが存在すれば， その内容をログメッセージとして使用し，存在しない場合は，viを起動しそこに書かれた内容をログメッセージとして使用する．

-r : 格納先のブランチ番号およびブランチタグを指定する．このオプションが指定されない場合は，トランク上にリビジョンが作成される．

● checkout projname

*** チェックアウト

projname : チェックアウトするプロジェクト名

● update [-r num] file

*** ファイルのアップデート

file　　: アップデートをする対象となるファイル.

-r　: アップデートするリビジョン番号もしくはタグを指定する．また，リビジョン番号を指定した場合は，その最新リビジョンを取り出す．このオプションをしてしない場合は，トランクリビジョンの先頭のリビジョンとなる.

● branch　[-t tag] num file

*** ブランチを作成する

num　　: ブランチの派生元のリビジョンを指定する.

file　　: ブランチを作成するファイルを指定する.

-t　: 作成したブランチに付けるタグを指定する．このオプションを指定しない場合はタグを付けない.

● diff　[-rf num] [-rt num] file

*** リビジョン間の差分情報を取得する

file　　: 差分情報を求める対象のファイル名

-rf : 差分を計算において，その基準となるリビジョンを指定する．指定されない場合は，作業中のファイルが対象となる．また，"-rf"と"-rt"とは少なくとも片方は指定されなければならない.

-rt : 差分を計算において，その対象となるリビジョンを指定する．指定されない場合は，作業中のファイルが対象となる．また，"-rf"と"-rt"とは少なくとも片方は指定されなければならない.

● merge　[-rf num] [-rt num] file

*** リビジョンをマージする.

file　　: リビジョンをマージする対象のファイル名

-rf : リビジョンの統合において，その基準となるリビジョンを指定する．指定されない場合は，作業中のファイルが対象となる．また，"-rf"と"-rt"双方共に指定されなければならない.

-rt : リビジョンの統合において，その対象となるリビジョンを指定する．指定されない場合は，作業中のファイルが対象となる．また，"-rf"と"-rt"双方共に指定されなければならない.

● tag　[-r num] tag [files...]

*** リビジョンにタグを付ける．

tag : リビジョンに付けるタグ名

files... : タグを付ける対象となるファイル名．指定されない場合は，./CVS/DLCM_Entries にリストアップされているすべてのファイル名が対象となる．

-r : タグを付ける対象となるリビジョン番号を指定する．これを指定しない場合は，作業中のリビジョンにタグがつけられる．

● log [-h] [-r num] [files...]

*** ファイルのリビジョン情報を出力する

files... : リビジョン情報出力の対象となるファイル名．指定されない場合は，./CVS/DLCM_Entries にリストアップされているすべてのファイル名が対象となる．

-h : ヘッダーリビジョンの情報のみを出力する．このオプションを指定した場合，"-r"オプションは無効となる．

-r : 出力対象のリビジョン番号およびタグを指定する．ブランチ番号およびタグを指定した場合は，そのブランチ上に存在するリビジョンのみを出力する．ただし，トランクを指定する場合は，tr とすること．また，このオプションを指定しない場合は，すべてのリビジョンについて出力する．

● status [files...]

*** 作業中のファイルの状態を出力する

files... : ファイル状態出力の対象となるファイル名．指定されない場合は，./CVS/DLCM_Entries にリストアップされているすべてのファイル名が対象となる．