ソフトウェア工学 129－7
（2000．11．16）

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

田中義己† 松下誠† 井上克郎†‡

† 大阪大学大学院基礎工学研究科
〒560-8531 大阪府豊中市待兼山町 1-3
‡ 奈良先端科学技術大学院大学情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5

近年，ソフトウェア開発の大規模化に伴ない，コンポーネント単位の開発や分散環境での開発が広く行われている．一方，開発されたプロダクトを効率よく管理するため，バージョン管理システムを利用することが多い．しかし，バージョン管理手法やそのシステム自体は，近年の開発環境に十分適応していないと考えられる．本論文では，分散環境にあるソフトウェアコンポーネントを，各々の開発に応じて利用できるバージョン管理手法を提案する．また，本手法に基づいたシステムを構築することにより，より柔軟にソフトウェアリポジトリを構成することが可能となり，個人を単位としたソフトウェアのバージョン管理を効果的に行うことが可能となる．

Virtually Controlled Revision Management System with Combination of External Software Repositories

Yoshiki Tanaka†, Makoto Matsushita† and Katsuro Inoue†‡

† Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka,
Osaka 560-8531, Japan
‡ Graduate School of Infomation Science
Nara Institute of Science and Technology,
8916-5 Takayama-cho, Ikoma, Nara 630-0101, Japan

Recently, software development tends to be based with conponents and in a distributed environment, as a scale of software is larger. On the other hand, configuration management system is often employed, for efficient management of product. But, a method or a system for configuration management is not suitable for recent software development environment. In this paper, we propose a method of configuration management, for sharing software components to fit each engineer's environment. We construct a system with this method. It is possible to have a management of personal development process as well as a more flexible management of software.

1 まえがき

ソフトウェア開発をする上で，ソフトウェアの管理方法は１つの問題である．最近では，それを解決する方法として，ClearCase や SourceSafe といった，バージョン管理機能の組み込まれた開発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では，開発履歴を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の際に，以前の開発工程を閲覧することで，より深い理解を得ることができる [11]．また，開発プロセスを作成するとき等に非常に役立つ．

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が肥大化し，開発規模も拡大している．その為，コンポーネントやモジュールといった，特定の意味のある単位でソフトウェアが開発されることが多い [6]．こういった開発手法が採用される背景には，ソフトウェアのシェアリングや再利用を考慮した開発が行われていることがある．さらに，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も行われる為，ますますコンポーネント単位での開発が進んでいる．また，これとは別に，一部オープンソース [5] でのソフトウェア開発も行われている．この場合にも作成したソフトウェアをシェアリングや再利用することが多く，その所在場所がより分散化している傾向にある．

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なうことが多い．その評価基準として，ISO 9000 シリーズ [7] や，CMM といったものがある．これらにより，ソフトウェア開発室単位での評価を行なうことが可能となった．さらに，ソフトウェアの開発単位ではなく，PSP などを利用して，個人単位での開発過程の品質評価もする動きもある．従って，ソフトウェア開発単位だけではなく，個人レベルにおいても，開発過程を蓄積しておく必要がある．

こういった背景がある一方，現在使われているバージョン管理システムでは，個々のマシン上にリポジトリを作成している．管理を行うには非常に有効であるが，上で述べたような分散環境に点在しているリポジトリを統括するには有効とは言えない．また，複数リポジトリが存在する中で，必要に応じて共有する場合，都合が良いとは言えない．

そこで，本研究では複数存在するリポジトリに対し，開発者が各々の開発に応じた形の管理が行える手法を提案する．さらに，それに基に，実際にシステムの構築を行う．

2 バージョン管理

2.1 バージョン管理システム

ソフトウェアの開発の際に，その過程を履歴として管理する機構が，バージョン管理システムである．また，ソフトウェアだけではなく，一般の文書作成等においても利用が可能である．従って，ファイルコンポーネント全般に適応できる．

多少，システムにより異なる部分もあるが，基本的な管理方法は，以下の通りである．

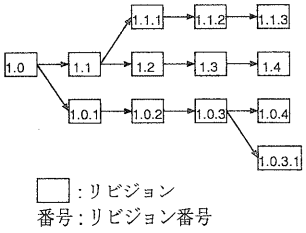

図 1: リビジョンツリーの例

- コンポーネントを格納しておく場所，リポジトリ (Repository) の作成を行なう．
- １ファイルのある一状態をリビジョンとして，リポジトリに格納する (Check in)．このとき，前のリビジョンにおける次のリビジョンとして格納されたり，ブランチを作成して派生したリビジョンとして格納したりもする．従って，リビジョンの関係は木のようになる (リビジョンツリー，図 1)．
- 作業用に，リポジトリから格納されているリビジョンを取り出す (Check out)．

バージョン管理システムと呼ばれるものは，多数存在する．UNIX 系 OS では，多くの場合，RCS[15] や CVS[2][9] といったシステムが標準で利用可能となっている．ClearCase[3] のように商用ものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，SourceSafe[13] や PVCS[12] をはじめ，数多く存在する．さらに，ローカルネットワーク内のみではなく，よりグローバルなネットワークを介したシステム [10] も存在する．

近年ハードウェアの進化と共に GUI 化が進んだ為，リポジトリ内部情報の視覚的な把握が可能となった．具体例として，以下のものが挙げられる．

- ネットスケープ等，ウェブブラウザを通して閲覧する．GUI の為の新たなシステム導入が不必要である．
- バージョン管理システムそのものが GUI 化されている．コマンドラインからの操作に比べ，間違いが少ない．

2.2 バージョン管理システムの現状

ソフトウェアの規模が肥大化したため，分散環境での開発が進んでいる．現在のところ，分散環境におけるソフトウェアの管理形態は，開発単位内のネットワーク環境上に存在するファイルサーバでの一括管理となることが多い．従って，このファイルサーバで，バージョン管理システムを利用し，ソフトウェアの履歴を採ることになる．この場合，以下のような問題が存在する．

(1) 作成されたソフトウェアの履歴を採るのは可能だが，その履歴から開発者個人個人の履歴を得るのは手間が掛かる．また，個人の履歴を採る為に，別のシステムを利用する方法もある．

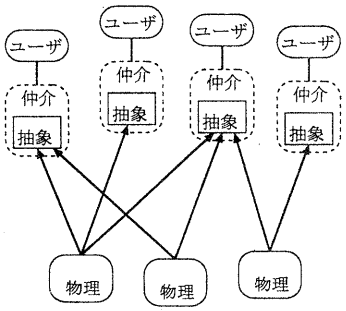

図 2: 概念的関係

しかし，異なるシステムを利用することにより，システム間での同期やインターフェイスの面で問題が生じる可能性がある．

(2) 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，リポジトリが１つという状況は少ない．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，複数のリポジトリから，ファイルを取り出して利用する場合，使用するマシン側でリポジトリ毎に分けて管理しなければならない．

(3) ファイルの再利用を行なう場合，ファイルを複製すると，その時点以降に複製元のファイルに対して行なわれた変更点が分からず，反映もできない．新しいファイルを得るには，再び複製しなければならない．

3 新たな管理手法

前の章の問題を解決するために，開発単位が単一の形態を採らない開発環境に対して利用可能な管理手法を提案する．この手法での解決方法は，以下の通りである．

(1) ソフトウェアの履歴を残すため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を設ける．これにより，ソフトウェアと開発者の履歴を平行して取ることが可能となる．

(2) 複数のリポジトリから取り出したファイルを，開発者を単位として一元管理を行う．これは，(1) の開発者用のリポジトリで平行して行うことが可能となる．

(3) リポジトリ内に存在するファイルを再利用する際，その複製を作成するのではなく，共有という形で利用する．ここでいう共有は，極めて参照に近い概念である．従って，元のファイル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．

これらを実現するために，物理リポジトリおよび抽象リポジトリという，概念的に異なる２種類のリポジトリを導入する．物理リポジトリは，ソフトウェアの実データを管理する．一方，抽象リポジトリは，開発者単位での管理に利用する仮想的なリポジトリとなっている．また，これら２種類のリポジトリの間に位置する，仲介レイヤーが存在する．これらの概念的な関係を図２に示す．この図おいて，矢印は抽象リポジトリは物理リポジトリからの実データ参照を示している．また，ユーザと抽象リポジトリは，基本的に１対１の関係であることも示している．

このように，２種類のリポジトリを導入しているが，開発者は抽象リポジトリのみ扱うので，既存の手法と変わらない．異なるのは，実データを別のリポジトリに格納する点だけである．ただし，リポジトリに対するオペレーションの動作は，既存のバージョン管理システムと比べ，変更が必要となる部分が存在する．また，２種類のリポジトリの特性を生かしたオペレーションを新たに付加する．これらについても，次章以降で詳しく説明を行う．

3.1 物理リポジトリ

このリポジトリには，バージョン管理システムの管理下で，実際のファイルが格納されている．具体的には，分散環境におけるファイルサーバ的な役割を担うことになる．また，実データ以外に，ファイルや各リビジョンの利用者に関する属性情報も管理する．属性に関しては，表１に示している．この情報は，主に物理リポジトリ内部のファイル・各リビジョンへのアクセス権の有無をチェックする時や，同一のリビジョンの利用者を調べる時などに利用する．この物理リポジトリは複数存在しても問題はなく，むしろ以下の文章では，複数あること，それも，その数に特に制限を設けないことを前提として話を進める．

物理リポジトリとのインタラクションは後で述べる仲介レイヤーを通して行ない，基本的に直接さわることは許可しない．物理リポジトリに対する操作を，すべて仲介レイヤー越しに行なうことにより，物理リポジトリの信頼性を高めることがねらいである．また，本手法による目的を果たすためには，物理リポジトリは，単純にデータをバージョン管理しておくための領域でさえあれば良い．仮に，物理リポジトリとして，RCS など，既存のシステムを利用した場合，機能的に様々なオペレーションが可能である場合でも，仲介システムにより利用するオペレーションはファイルの格納・取り出しに関する程度である．逆に物理リポジトリとしては，上記の機能さえ備えていれば問題はない．ログ取りなどのようにリビジョンツリー構成やデータ自体に触れない機能に関しての付加や利用は自由である．

表 1: 物理リポジトリに関する属性

リポジトリ	ファイル	リビジョン
・リポジトリ名	・ファイル名	・リビジョン番号
・作成日時	・作成日時	・作成日時
・作成者	・作成者	・作成者
・利用者	・最終更新日	・利用者
・操作可能者	・最終更新者	・アクセス可能者
・存在するファイル名	・利用者	・メッセージ
・メッセージ	・メッセージ	

表 2: 抽象リポジトリに関する属性

プロジェクト	ファイル	リビジョン
・プロジェクト名	・ファイル名	・リビジョン番号
・作成日時	・作成日時	・作成日時
・作成者	・作成者	・作成者
・存在するファイル名	・最終更新日	・リンク先のリビジョン
・メッセージ	・最終更新者	・メッセージ
	・メッセージ	

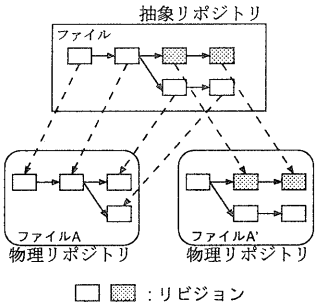

図 3: 異なるファイルへのリビジョン参照

3.2 抽象リポジトリ

このリポジトリには，ファイルの実態は存在しない．存在するのは，実際のファイルへのパス等，管理する上で必要となる属性 (表 2 参照) である．つまり，抽象リポジトリは，物理リポジトリに存在するリビジョンを参照している，仮想的なリポジトリである．しかし，この部分もリポジトリであるには相違なく，リビジョン管理されており，外見は物理リポジトリと変わりない．この抽象リポジトリは個人的，もしくは，小規模な開発単位における管理部となる．物理リポジトリと同様に，複数存在することを前提としている．

抽象リポジトリ中では，複数のプロジェクトを作成することができる．通常は，このプロジェクト単位で利用する．そして，このプロジェクトの中に複数のファイルが存在する．

この抽象リポジトリ内部では 1 ファイルに対して 1 つリビジョンツリーを持つことになる．しかし，同一のリビジョンツリーに存在する任意の 2 つのリビジョンに関して，それぞれが指している物理リポジトリ内のリビジョンは，互いに異なるファイルに属していても良い．このことを示したのが図 3 である．これにより，類似した複数のコンポーネントが異なるファイルとして存在し，その 1 つから別のものへの移行を行なう場合，新しいコンポーネントを別ファイルとしてインポートしなくても，単純に次のリビジョンとして取り込むことが可能となり，よりスムーズな移行が可能となる．また，移行時前後の状況を把握するのも容易である．

3.3 仲介レイヤー

上記の 2 種類のリポジトリとユーザとの間を仲介するの部分である．主な役割として以下のようなものがある．

(1) ユーザ側に，抽象リポジトリと物理リポジトリとを，統合した 1 つのリポジトリとして見せる．これにより，ユーザは既存ものと同様の感覚で，リポジトリへのアクセスが可能となる．
(2) 物理リポジトリへの直接アクセスを避ける．これは，物理リポジトリの内容の信頼性を高めた為である．
(3) ユーザが抽象リポジトリに対して行ったオペレーションの解析を行う．その結果に応じて，抽象リポジトリ及び物理リポジトリにアクセスし，データの格納・取り出しを行う．同時に，変更のあったファイルや追加されたリポジトリ等の属性更新も行う．この場合の各アクセス権は，オペレーションを実行したユーザと同じ権限である．
(4) 物理リポジトリに変更が生じた場合に，その更新内容を適当なユーザに送信する．具体的には，あるユーザが，リビジョンを 1 つ追加した際，

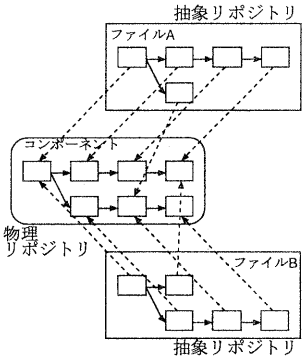

図 4: 実データに対する物理リポジトリと抽象リポジトリの関係

その派生元のリビジョンを利用しているユーザ (または，そのリビジョンを含むファイルのユーザ) に，追加されたリビジョンの属性情報を送信する．送信先のリストは，物理リポジトリ内のファイルやリビジョンの属性情報より得ることが可能である．

4 新手法の詳細

4.1 リビジョン単位でのシェアリング

先で述べたように，新手法では2種類のリポジトリを使い分ける．実データを基準とする物理リポジトリと抽象リポジトリとの関係は図4となる．この図では，物理リポジトリ内のファイルを複数の抽象リポジトリで共有している状態を示している．抽象リポジトリ内部のファイルにおける各リビジョンは，物理リポジトリ内部のいずれかファイルのリビジョンを指す．従って，図4でも示しているように，リビジョン単位でのシェアリングを行うことことが可能となる．この"リビジョン単位のシェアリング"により，以下のような利点がある．

- 必要なリビジョンのみの共有が可能となるので，同一ファイル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，個人レベルでのソフトウェア管理において有効である．
- 同じリビジョンを共有している他のユーザにより行われた更新事項が，物理リポジトリに格納されることになる．従って，その変更部分を容易に反映することが可能である．また，逆も同様で，自分の行った変更を他のユーザへ適応するもの簡単である．
- ファイルを複製するわけではないので，共有した (引用した) ファイルの履歴を物理リポジトリから参照可能である．よって，ソフトウェアを再利用する時などに，その開発経過を簡単に知ることができる．

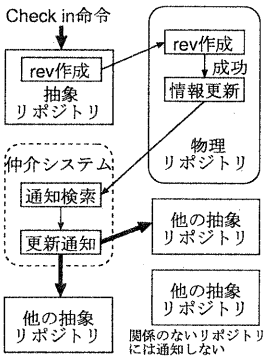

図 5: Check in の流れ (成功時)

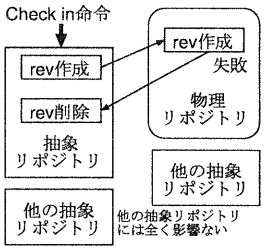

図 6: Check in の流れ (失敗時)

4.2 オペレーション

本手法でのオペレーションとは，抽象リポジトリに対するものである．しかし，実際のデータは物理リポジトリの内部に存在するので，既存のシステムにおけるオペレーションよりも複雑になる．以下で，主なオペレーションの手法を説明する．

4.2.1 Check in

抽象リポジトリに対し，データを格納するとき利用するオペレーションである．成功時および失敗時の動作を，それぞれ図5，6で示している．ここで失敗時というのは，抽象リポジトリ側では受理されるが，物理リポジトリ側で非受理となった場合のことである．

この Check in オペレーションには，以下の2種類が存在する．

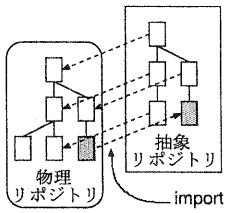

図 7: 物理リポジトリのリビジョンを利用した Check in の流れ

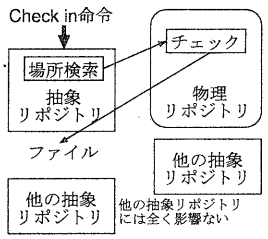

図 8: Check out の流れ

- 1つは，ユーザがファイルを編集して，それを次のリビジョンとして Check in するというオペレーションである．これは，既存の check in とほとんど変わりない．オペレーションの流れは，抽象リポジトリにリビジョンを作成，アクセス権などのチェックを行なったあと，対応するリビジョンを物理リポジトリ内部に作成する．そして，更新情報を適当なユーザにへの送信となる (図 5)．
- もう1つは，物理リポジトリ内の1つのリビジョンをそのまま抽象リポジトリ内のリビジョンとして利用するオペレーションである．後者の方の流れを図 7 に示す．この場合，物理リポジトリでのリビジョン変更はないので，更新情報を送信しない．一見 Check in には思えないが，抽象リポジトリにリビジョンが1つ増えるという点で，Check in といえる．しかも，そのリビジョンの属性として格納されるデータの種類は，前者，後者ともに同じである．従って，この2つは抽象リポジトリに対して，全く同一のオペレーションである．

4.2.2 Check out

リポジトリからファイルを取り出すためのオペレーション．動作の流れを図 8 に示すが，既存のオペレーションとはほとんど変わりはない．Check out のオペレーションを実行すると，抽象リポジトリと物理リポジトリとでアクセス権のチェックを行い，その後，物理リポジトリの属性情報を更新する．リビジョン属性の更新情報を送信することはない．また，抽象リポジトリに関しては何の変化も生じない．

4.2.3 Diff

2つのリビジョン間の差分情報を得るときに利用するオペレーションである．既存の Diff オペレーションとは，インターフェイス的には，ほぼ同じである．入力が2つのリビジョンで，出力が差分情報である．しかし，物理リポジトリで差分計算をするのではなく，仲介レイヤーで行なう．このことを図 9 に示す．この図からも判るように，物理リポジトリ側の Diff オペレーションを直接利用しない．それは抽象リポジトリの1ファイルにおける異なる2つのリビジョンが，それぞれ，物理

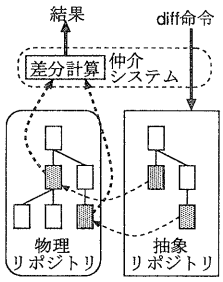

図 9: Diff の流れ

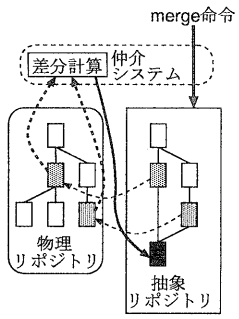

図 10: Merge の流れ

リポジトリ内の同一ファイルのリポジトリであるという保証が無いからである (抽象リポジトリの項を参照)．従って，Diff オペレーションの入力として与えた2つのリビジョンを物理リポジトリから取り出し，仲介レイヤーで差分計算する必要がある．

4.2.4 Merge

あるリビジョンに別のリビジョンを結合して新たなリビジョンを生成するオペレーションである．このオペレーションを図 10 に示すが，Diff オペレーションと Check in オペレーションを組み合わせと言って間違いない．このオペレーションでも Diff と同じく，差分計算は仲介レイヤーで行なう．理由も同様である．

ここで，計算した差分情報が適応可能であるかという問題が存在する．しかし，本システムでは，単純に適応可能であれば差分情報を結合し新たなリビジョンを作成する．また，適応不可能であれば，そのまま差分情報を破棄し，このオペレーションが失敗したことを告げる．

4.2.5 Extended Check in

これは，物理リポジトリ内の複数のリビジョンを，抽象リポジトリに取り込むオペレーションである．Check in オペレーションの拡張であり，抽象リポジトリにリビジョンが追加されることに変わりはない．このオペレーションにより，ファイルの派生過程を取り込むことが可能となる．この

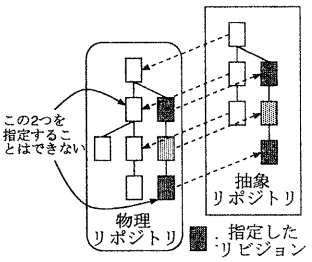

図 11: Extended Check in の流れ (その 1)

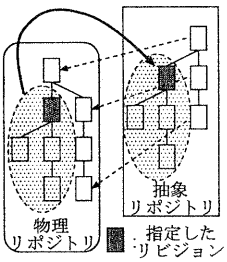

図 12: Extended Check in の流れ (その 2)

ことは，他のユーザが作成した，もしくは，過去に作成したファイルを再利用する時に，非常に有効である．このオペレーションには，以下のように，2種類ある．

- 1つは，物理リポジトリ内の1ファイルにおける2つのリビジョンを指定し，その間に存在するすべてのリビジョンを抽象リポジトリのリビジョンとして Check in するというオペレーションである (図 11 参照)．ただし，指定する2つのリビジョンに対して『1つのリビジョンは，もう1つのリビジョンから派生したもの』という制約を設ける．この条件があるのは，指定された2つのリビジョン番号のみで check in されるリビジョンの認識を可能にするためである．また，この条件下にないリビジョンを指定すると，抽象リポジトリにおいて，安定したリビジョン番号を付けることが不可能なので，これを防ぐということもある．

- もう1つのオペレーションは，物理リポジトリ内の1つのリビジョンを指定し，そのリビジョンをルートとするリビジョンツリーを抽象リポジトリへ Check in するというものである (図 12 参照)．このオペレーションには上記のような制約はない．

5 システムの試作

現在，前章で述べた手法に元に，分散環境において有効であるバージョン管理システムの構築を行っている．システムの構成を図 13 に示す．以下

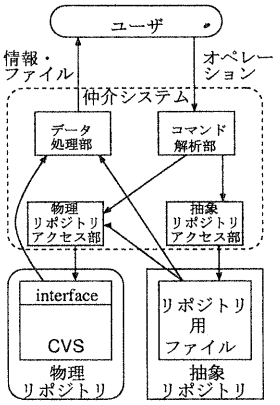

図 13: システムの構成

で個々の部分の実装に関して述べる．

5.1 物理リポジトリ

実際にファイルの実態を管理している部分である．本システムでは，純粋にファイルのバージョン管理を行う部分に関しては，CVS を利用する．そして，物理リポジトリへのアクセスは，CVS が備えているインターフェイスを，仲介システムを通して利用する形式で行う．物理リポジトリに既存のシステムを利用した理由は2つある．1つは，単体としてのバージョン管理システムの作成は，本研究の本意とは異なるためである．もう1つは，既存のバージョン管理システムを利用することで，本システムへの移行がスムーズに行えるからである．

基本的には物理リポジトリを直接操作することはできない．しかし，リポジトリに問題が発生した場合の対処策として，特別にリポジトリの操作可能権限を得ることで直接操作を可能としている．ただし，すべてのユーザが得られるわけではなく，物理リポジトリ毎に，操作可能権限を得ることのできるユーザを管理している．

5.2 抽象リポジトリ

物理リポジトリとは異なり，既存のバージョン管理システムを利用しない．理由は，抽象リポジトリは属性情報しか持たないからである．実装としては，抽象リポジトリは属性情報が格納されたファイルの集合となる．1つのリポジトリ内のすべてのリビジョンを1つのファイルにまとめると，ファイルサイズが増大し，必要なリビジョン情報を検索するのに時間が要する．また，1つのリビジョンを1つのファイルとすると，異なるリビジョンの情報を得るたびにファイルにアクセスすることになる為，この場合も，時間を必要とすることとなる．

また，抽象リポジトリの内部では，プロジェクト単位での管理も可能となっている．1つのリポジトリに対し，プロジェクトが複数作成することができる．

5.3 仲介システム

仲介システムは，以下の各構成要素からなる．

- コマンド解析部
  ユーザが入力したオペレーションを解析する．この解析結果により，以下の３つの部分の動作が決まる．
- 物理リポジトリアクセス部
  実データが必要な場合に，物理リポジトリにアクセスし，適当な情報を得る．実際は，物理リポジトリにCVSを利用しているので，CVSのコマンドを実行することになる．
- 抽象リポジトリアクセス部
  上と同様，抽象リポジトリ対して，情報の格納や取得を行う．実装は，抽象リポジトリ自体は単純なファイルの集合に過ぎないので，そのファイルに直接アクセスしている．
- データ処理部
  物理・抽象リポジトリより得られた情報を処理する．差分計算などが，この部分に当たる．

6 まとめ

本論文では，物理リポジトリ，抽象リポジトリという２種類リポジトリ，および，それらの間に位置する仲介レイヤーという概念を導入し，分散環境に有効であるバージョン管理システムを構築するための手法を提案した．本手法に基づいたシステムにより，既存のバージョン管理システムと比べ，各々の開発に適応したファイル管理が可能となる．また，ソフトウェアの開発において，そのソフトウェアの全体の管理だけでなく，それに携わっている開発者個々の管理も可能となる．従って，開発者単位でのソフトウェア開発履歴を得ることができるので，それらの情報を利用して，個人レベルでのソフトウェアプロセスの改善が可能となる．

現段階，システムは構築中であるが，評価に際しては，現在以下のような方法を検討している．

- 各オペレーション実行に必要となる時間を計測し，既存のシステムのそれと比較する．計測対象となるオペレーションは，新システムおよび既存のシステムに共通するもので行なう．これにより，新システムの相対的なパフォーマンスを知ることができる．

- 物理リポジトリおよび抽象リポジトリ内のファイルサイズの計測を行ない，そのうち抽象リポジトリ内に存在するファイルのサイズの割合を算出する．この値により，抽象リポジトリとして存在するデータの実態を持たないファイルのオーバーヘッドの相対量を得ることができる．

参考文献

[1] bonsai
http://www.mozilla.org/bonsai.html

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

[3] Clear Case
http://www.rational.com/products/clearcase/

[4] CVSWeb
http://stud.fh−heilbronn.de/ zeller/cgi/cvsweb.cgi/

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

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

[7] 飯塚悦功, "ソフトウェア ISO 9000", 日科技連, 1996.

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

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

[10] Peter Fröhlich and Woflgang Nejdl, "WebRC Configuration Management for a Cooperation Tool", SCM-7, LNCS 1235, pp175−185, 1997.

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

[12] PVCS
http://www.merant.co.jp/pvcs/

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

[14] Tapani Kilpi, "Product Management Requirement for SCM Discipline", SCM-7, LNCS 1235, pp175−185, 1997.

[15] WALTER F.TICHY, "RCS - A System for Version Control" SOFTWARE - PRACTICE AND EXPERIENCE, VOL.15(7), pp.637−654, 1985.

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