修士学位論文

題目

構文木の差分を用いた細粒度版管理システムの提案と実現

指導教官
井上 克郎 教授

報告者
早瀬 康裕

平成 16 年 2 月 12 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

平成 15 年度 修士学位論文

構文木の差分を用いた細粒度版管理システムの提案と実現

早瀬 康裕

内容梗概

近年，インターネットの普及により，オープンソースソフトウェア開発が広まっている．オープンソースソフトウェア開発では，世界中に分散した開発者が，版管理システムを用いて並行して作業を行い，その結果互いに独立した形で版管理システムに登録することが頻繁に起こる．この時，後から自分の作業結果を登録する開発者は，既存の修正内容と自らの修正内容を版管理システムによってマージさせることが一般的である．既存の版管理システムでは，管理対象となるテキストの行を単位としたマージ処理を行っていたが，この方法をソースコードに適用する場合，間違った出力をしたり，マージ出来るべき変更をマージ出来なかったりする問題があった．この問題を解決するには，プログラミング言語の構文を理解したマージ処理が考えられる．しかし，プログラミング言語は数多く存在するため，それぞれについて，精度の高いマージ用アルゴリズムを定義するのは困難である．

本研究では，一般的なプログラミング言語が何らかの木構造を持つことに着目し，ソースコードを木構造を持つ中間言語に変換した上で，中間言語に対するマージ用アルゴリズムの提案を行う．本アルゴリズムでは中間言語に対して頂点のマッチングや差分計算を行うことで，その差分情報をツリー構造を編集する操作の列として定義する．そして定義された差分情報を，ツリーに適用してマージ処理を行う．

また本アルゴリズムを，実際の版管理システム上で実装した．本システムはプログラミング言語として Java を対象としている．本システムによるマージと行単位のマージとの比較実験を行った結果，行単位のマージでは正しくマージすることが出来なかった問題を解決することができ，本システムの有効性が確認できた．

主な用語

木構造 (Tree Structure)
差分計算 (Delta Calculation)
版管理システム (Revision Control System)

目 次

1 はじめに

近年，インターネットの普及により，オープンソースソフトウェア開発 [1] が広まっている．オープンソースソフトウェア開発では，世界中に分散した開発者により，協調かつ並行して開発作業が行なわれている．典型的なオープンソースソフトウェア開発では，開発者相互は電子メールやメーリングリストによって互いに情報交換を行いながら，版管理システムを用いて生成するプロダクトを一括して管理している [2].

このような開発では，複数の開発者によって並行して作業を行い，その結果互いに独立した形で版管理システムに登録することが頻繁に起こる．この時，後から自分の作業結果を登録する開発者は，既存の修正内容と自らの修正内容を統合する (マージする) 必要がある．このマージ処理は，人手によって行われる場合と，版管理システムなどによって行わせる場合が考えられるが，マージを行う回数が多いことや，作業の正確性，確実性を高めることを目的として，版管理システムに行わせることが一般的である．

既存の版管理システムである，CVS[3], RCS[4], subversion[5], BitKeeper[6], perforce[7] といったシステムでは，管理対象となるテキストの行を単位としたマージ処理を行っていた．しかし，この方法をソースコードに適用する場合，間違った出力をしたり，マージ出来るべき変更をマージ出来なかったりする問題があった．例えば，ある変数を利用する文を追加する変更と，その変数を削除する変更とは，マージ出来ない組み合わせであるが，既存のシステムではこれを発見することが出来ず，間違った出力を行っていた．この問題を解決するため，ソースコードのマージをより正確に行うための方法として，プログラミング言語の構文を理解したマージ処理が考えられる [8]．しかし，プログラミング言語は数多く存在するため，それぞれについて，精度の高いマージ用アルゴリズムを定義するのは困難である．

そこで本研究では，プログラミング言語から独立したマージ用アルゴリズムを定義することをめざす．具体的には，一般的なプログラミング言語が何らかの木構造を持つことに着目し，ソースコードを木構造を持つ中間言語に変換した上で，中間言語に対するマージ用アルゴリズムを提案する．中間言語の表現には，独自スキーマの XML を用いる．この中間言語に対して，ツリーの差分計算アルゴリズムである FMES [9] を改変したアルゴリズムにより，頂点のマッチングや差分計算を行い，その差分情報をツリー構造を編集する操作の列として表現する．そしてこの差分情報を，ツリーに適用してマージ処理を行うことで，言語依存の部分とマージ処理の部分を分離する．

また本アルゴリズムを，版管理システムである subversion [5] 上で実装した．本システムはプログラミング言語として Java を対象としており，変数の削除や，ソースコード断片の移動などを正しく扱うことが出来る．さらに，作成したシステムによるマージと行単位のマージとの比較実験を行う．具体的には，サンプルのソースコードと，実際のオープンソース開

発履歴から抽出した情報を用いて，作成したシステムと行単位のシステムで実際にマージを行い，その結果を比較する．

以下，2 節では，オープンソースソフトウェア開発と版管理システムのそれぞれについて説明し，既存の版管理システムの問題を示す．3 節では，問題の解決法として，ソースコードの構文を考慮したマージシステムを提案し，4 節ではそのシステムの実装について説明する．5 節では提案するシステムを用いた場合に解決する問題の例を示す．6 節では，作成したシステムの適用実験と考察を行う．最後に 7 節で本研究のまとめと，今後の課題について述べる．

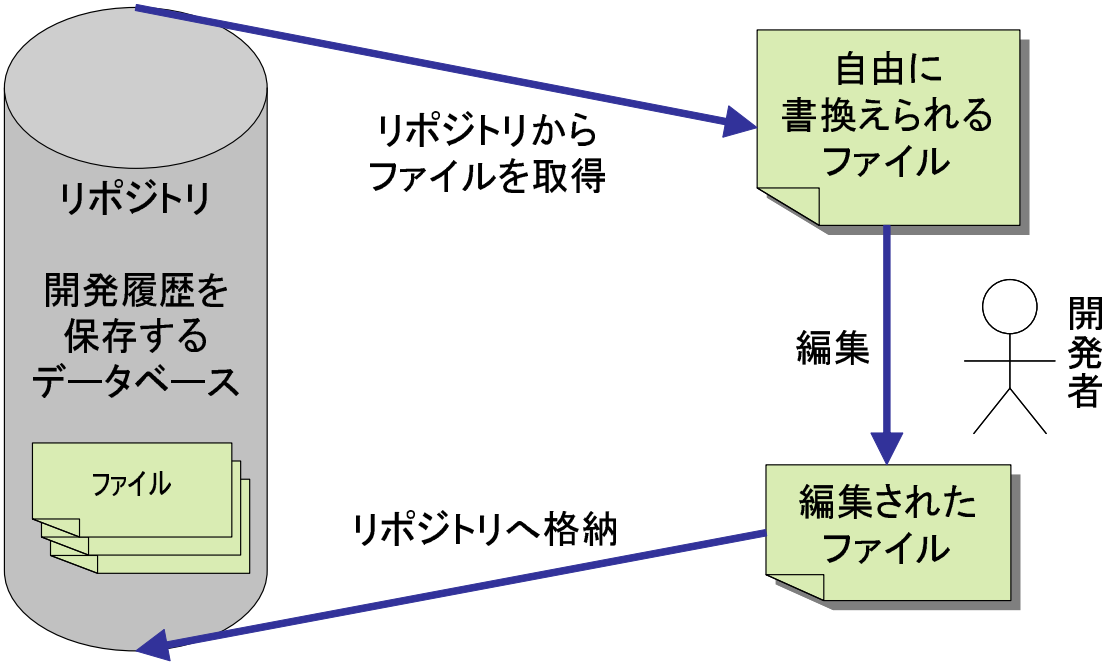


図 1: 版管理システム

2　オープンソースソフトウェア開発と版管理システム

2.1　オープンソースソフトウェア開発

オープンソースソフトウェア開発は近年急速に普及し，多くの有用なソフトウェアが開発されている．オープンソースソフトウェア開発は，多人数が自由に参加することが出来るという特徴がある．

一般的なオープンソースソフトウェア開発では，以下の道具を用いて開発を行う．まず，プロジェクトの成果物などを公開するためのウェブページがある．開発の方向性や，ソフトウェアの問題点などの議論にはメーリングリストを用いている．そして，ソースコードやドキュメントの変更履歴を保存し，多人数でソースコードを編集するために，版管理システムを用いている．

2.2　版管理システム

CVS [3] に代表される版管理システムは，ソースコードやドキュメント等のファイルの，変更履歴を保存するシステムである．

図 1 に，版管理システムを用いた開発を簡略化した図を示す．版管理システムにはリポジトリと呼ばれるデータベースがあり，開発対象のファイルは全てリポジトリに保存される．

開発者がファイルを編集する場合，リポジトリからファイルのコピーを取得する．取得したファイルはワークコピーと呼ばれる．開発者はワークコピーを編集した後に，リポジトリにファイルを格納する．新しいファイルが格納されても，リポジトリから古いファイルが失われることはなく，古いファイルをいつでも参照することが出来る．版管理システムを使ったソフトウェア開発は，この作業を繰り返す事で行われる．

2.2.1 複数人での開発

また，版管理システムは，複数人によるソフトウェア開発をサポートしている (図 2)．図 2(a) で示すように，開発者がファイルを取得する際には，他の開発者がそのファイルを取得していても良く，それぞれの開発者は，取得したワークコピーを自由に編集することが出来る．

このようにして，複数人によって同時並行的にファイルの編集が行われた場合，それぞれの開発者が自分の編集結果だけを格納すると，最後に格納したファイル以外に行われた編集が失われてしまう (図 2(b))．このため，複数人が行った変更を 1 つにまとめたファイルが欲しい．このファイルを得る処理を，マージと呼ぶ．マージは，版管理システムにより自動的に行われる場合 (2(c)) と，人間の手による解決が必要になる場合とがある．人間の手による解決が必要になることを，変更点の衝突と呼ぶ．

2.3 問題点

既存の版管理システムでは，マージを行単位で行っており，マージの時に同じ行が編集されている場合にだけ，衝突として開発者に解決を求める．これにより，不要な衝突を引き起こしたり，不正なマージ結果を得たりするなどと言う問題があった．以下，それぞれについて説明する．

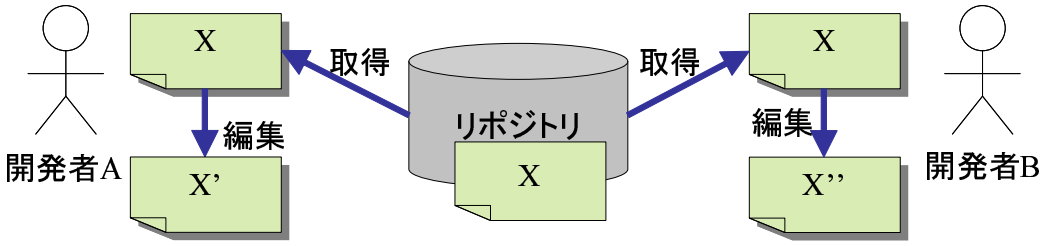
2.3.1 不要な衝突

編集されている行が同じである場合，本来はマージ可能な変更を，衝突としてしまう．

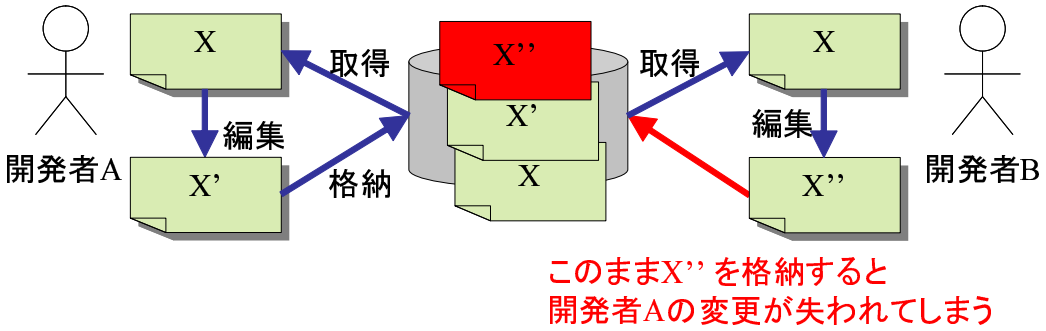
例えば，開発者 A と B が同じファイルを取得して，編集しているとする．そのファイルには以下のような行があった．

```
int refs;
```

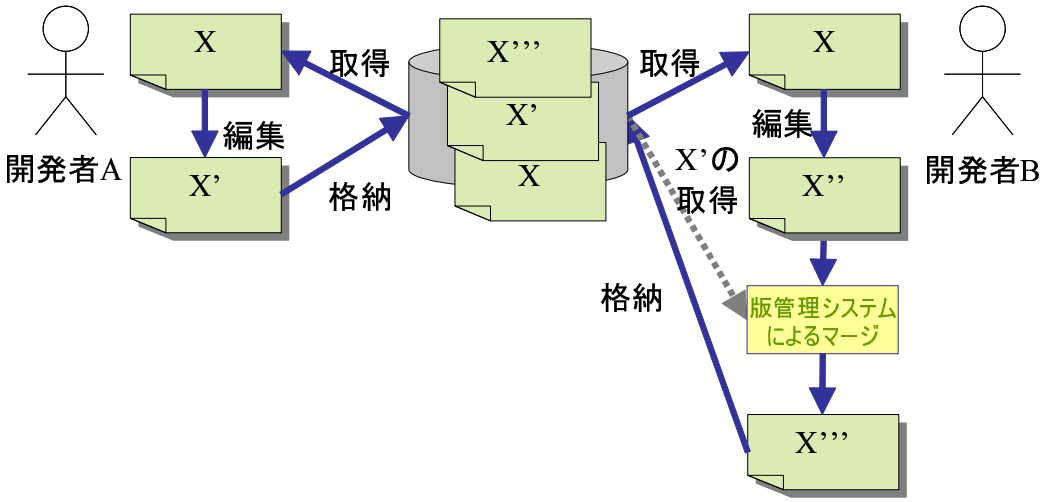
開発者 A は以下のように，初期値を設定する変更を加え，格納した．



(a) 複数の開発者による同時開発



(b) 格納



(c) マージした結果を格納

図 2: 版管理システムを使った同時開発

```
int refs=0;
```

開発者 B は A の変更を知る前に，以下のように変数についてのコメントを追加し，格納をしようとしたとする．

```
int refs; /* reference count */
```

開発者 A と B は同じ行を編集しており，版管理システムはこれを衝突と見做す．今の場合，衝突は，後から格納をした開発者 B が，解決しなければならない．

もし，変更点を正しく把握できれば，システムは，初期値とコメントの両方を含んだ，以下のようなコードを得ることが出来るはずである．

```
int refs=0; /* reference count */
```

2.3.2 不正なマージ結果

変更された行が違う場合，本来衝突すべき変更点を見逃がす場合がある．

開発者 A と B が同じファイルを取得して編集しているとする．ファイルには，以下のように変数を宣言する行が含まれていた．

```
int num, sum, avg;
```

開発者 A は変数 avg は不要であると考え，以下のように削除してコミットした．

```
int num, sum;
```

開発者 B は A が格納したこと知る前に，以下のように変数 avg を使う処理を追加した．

```
int num, sum, avg;

⋮

avg = num/sum;

⋮
```

B が格納する前には， A と B の変更点をマージする必要がある．版管理システムは，マージ結果として，以下のような出力を行う．

```
int num, sum;

⋮

avg = num/sum;

⋮
```

A と B は同じ行を変更しておらず，マージの際に衝突は起こらない．しかし，このマージ結果は，宣言されていない変数 avg を利用する不正なソースコードとなってしまう．もし，外のスコープで削除された変数と同名の変数が宣言されていたとすると，コンパイルエラーとならず，開発者が問題に気付かない可能性がある．

変数名が変更された場合にも，変数が削除された場合と同様の問題が起こる．

3　ソースコードの構文を考慮したマージシステムの提案

2.3 節で述べた問題を解決するため，ソースコードの構文を考慮したマージアルシステムを提案する．マージシステムの行う処理の手順は，以下のとおりである．

1. ソースコードを解析してツリーを作る
2. ツリーの差分を計算する
3. マージを行う

以下，それぞれについて説明する．

3.1　ツリーへの変換

まず，ソースコードを構文解析してツリー A を作成する．ツリーは順序木であり，子頂点の数に制限は無い．頂点には ID と，文字列の情報が記録されている．ツリーの形は，構文解析木に準じたものである．ツリーから元のソースコードに戻すことが出来るようにするために，ツリーには空白文字列やコメントを表す頂点を含める．ツリーのデータ構造は，プログラミング言語に依存しないので，マージシステムに汎用性を持たせることが出来る．

頂点に ID を付ける手順は，以下の通りである．版管理システムには，解析するソースコードの元となったバージョンのツリー B が，保存されている．B の頂点には既に ID が付けられている．まず，A と B を比較し，マッチングを計算を行う．マッチング計算とは，A, B から，それぞれの頂点集合の部分集合 A', B' と，A' から B' への全単射写像 f を見つけることである．写像 f では，頂点に格納されたデータと親子頂点や兄弟頂点から，似ていると判断された頂点同士が対応する．A の頂点のうち，A' に含まれた頂点には，対応する B' の頂点と同じ ID を付ける．A の頂点のうち，A' に含まれなかった頂点には，新しい ID を付ける．

マッチング計算には，3.2.2 節で説明する FMES アルゴリズム [9] を用いる．

次に，変数や関数を利用している頂点から，宣言している頂点へのリンクを張る．

3.2　差分計算

ツリーの差分は，編集スクリプトを用いて表す．編集スクリプトは，編集操作の列であり，編集操作は以下の 4 種類である．

1. 頂点の追加を表す insert 操作は，引数に，追加する頂点の ID，頂点が格納するデータ，親の ID，何番目の子供にするかを表す数値の 4 つを取る．

2. 頂点の削除を表す delete 操作は，引数に，削除する頂点の ID を取る．

3. 頂点に格納されたデータの更新を表す update 操作は，引数に，更新の対象となる頂点の ID と，その頂点に格納する新しいデータを取る．

4. 部分木の移動を表す move 操作は，引数に，移動する部分木の根の ID，移動先の親 ID，何番目の子供にするかを表す数値の 4 つを取る．

ツリーの編集は，対象となるツリーに上記の編集操作を適用することによって行う．編集スクリプトの先頭の操作から順に，ツリーに編集操作を適用することを，ツリーに編集スクリプトを適用すると呼ぶ．ツリー A に編集スクリプト S を適用するとツリー B が得られるとき，$A \xrightarrow{S} B$ と表すことにする．

任意のツリー A を B に変換する編集スクリプトは， insert 操作と delete 操作の組み合わせで表現可能である．例えば A の頂点を全て delete した後に，B の頂点を insert する編集スクリプトは，A を B に変換する．しかし，人間がソースコードを編集する時には， ソースコードの一部分を移動したり，文字列の一部を書換えるといった操作を行う．これらの操作を自然に表現するには，move 操作や update 操作を用いた方がよい．

ここで，あるツリー A と B があるとき， A を B に変換するスクリプトは無数に存在する．例えば， $A \xrightarrow{S} B$ である任意の S の末尾に， A にも B にも含まれない頂点を追加する操作と，その頂点を削除する操作を加えた編集スクリプト S' も， $A \xrightarrow{S'} B$ を満たす．

ツリーの差分を表すスクリプトとしては，このような無駄な編集操作が含まれているものは望ましくない．そこで，編集スクリプトにコストを定義し，コストが最も小さいスクリプトを，差分として採用することにする．編集スクリプトのコストは，含まれる編集操作のコストの総和であり，編集操作のコストは差分計算アルゴリズムによって定める．

ツリーの差分を計算するアルゴリズムとして， FMES [9] にリネーム解析を加えたものを採用する．以下， FMES アルゴリズムに用いる最長共通部分列 (LCS) と FMES アルゴリズム，リネーム解析について説明する．

3.2.1 最長共通部分列

ある列 a から， 0 個以上の要素を削除して出来る列を， a の部分列と呼ぶ．また，列 a, b, x があったとき， x が a の部分列であり，かつ x が b の部分列であるとき， x は a と b の共通部分列であると言う．ここで， A と B の共通部分列全体の集合のうち，長さが最大の要素を最長共通部分列 (LCS) と呼ぶ.

既存の版管理システムの差分計算は，テキスト行を要素とした列に対して， LCS を計算する処理に基いている．

LCS の計算方法としては， E. W. Myers [10] のアルゴリズムや，今回作成したシステムで採用した S. Wu, U. Manber ら [11] のアルゴリズムがある．

3.2.2 FMES アルゴリズム [9]

FMES はツリーの差分計算を行うアルゴリズムである．FMES は近似アルゴリズムであり，コスト最小の解を得られるとは限らない．ここで，ツリーの頂点数を n，比較するツリーの間の差の大きさを e と表すことにする．FMES で，良い近似解を得るためには， $e \ll n$ である必要がある．時間計算量は $O(ne + e^2)$ と表される．

insert, delete, move のコストは 1 であり， update のコストは，書換える前後の値によって 0 から 2 の値を取る．書換える前後の値が違うほど，update 操作のコストは大きな値を取るようにする．

提案するシステムでは，書換え前後の文字列の長さをそれぞれ l_1, l_2，書換え前後の文字列の共通部分の長さを l_c と表したとき， update のコストを $\frac{2(1-l_c)}{l_1+l_2}$ と定義する．

FMES アルゴリズムは，前後 2 つの段階に分かれている．

1 つ目の段階は，比較する二つのツリーの間で，頂点のマッチング計算を行う．ここでは，葉頂点，識別子の内部頂点，それ以外の内部頂点を区別し，それぞれ別にマッチング計算を行う．

まず，二つのツリーの葉頂点をそれぞれ深さ優先探索順に並べ， LCS を計算する．ここで，葉頂点が一致する条件は，それぞれの頂点に格納されたテキストの共通部分列がテキストの長さと比べて一定以上であることとする． LCS で対応した頂点の組をマッチングに加え，対応しなかった頂点のマッチングは総当たりで計算する．

次に，それぞれのツリーの内部頂点を，格納されたテキスト毎に分け，深さ優先探索順に並べた列を作る．同じテキストを格納した列同士で， LCS を計算する．ここで頂点が一致する条件は，それぞれの頂点の下にある葉頂点が，一定以上マッチングに入っていることとする． LCS で対応した頂点の組をマッチングに加え，対応しなかった頂点のマッチングは総当たりで計算する．

この葉頂点と内部頂点のマッチング計算では，類似度が閾値以下の頂点同士はマッチしないという仮定を置くことで，計算量を削減している．

2 つ目の段階は，マッチングの終わったツリーの間で編集スクリプトを計算する．この段階では厳密解を求める．

3.2.3 リネーム解析

FMES は，テキストの類似度に基いた，任意の構造化テキストに対する差分計算アルゴリズムである．そのため，そのままソースコードに適用すると問題が起こる．

最も大きな問題は，識別子のマッチングである．似た役割の変数を，末尾に付けた番号や名前で区別することは多い．`buffer1, buffer2...`)．このような変数は，テキストとして見た場合に十分似ているため，間違った組み合わせでも一致していると判定されてしまう．

また，テキストの類似度だけを用いてマッチングを計算していると，リファクタリング等で変数名が変更された場合に，正しくマッチングを計算することが出来ない．

このように，識別子のマッチングにテキストの類似度を用いるのは危険である．そこで，識別子のマッチングには，他の内部頂点のマッチングとは異なるアルゴリズムを用いる．

まず，葉頂点と識別子以外の内部頂点のマッチングを， FMES アルゴリズムを用いて計算しておく．次に，それぞれのツリーの識別子を，深さ優先探索順に並べた列を作り，この列の LCS を計算する．識別子の頂点が一致する条件は，以下に示す条件のうち，いずれかを満たすものとする．

1. 葉頂点の文字列が完全に一致している
2. その変数を利用している頂点の親が，一定以上の割合で共通している

LCS で対応した頂点の組をマッチングに加え，対応しなかった頂点のマッチングは総当たりで計算する．

3.3 マージ

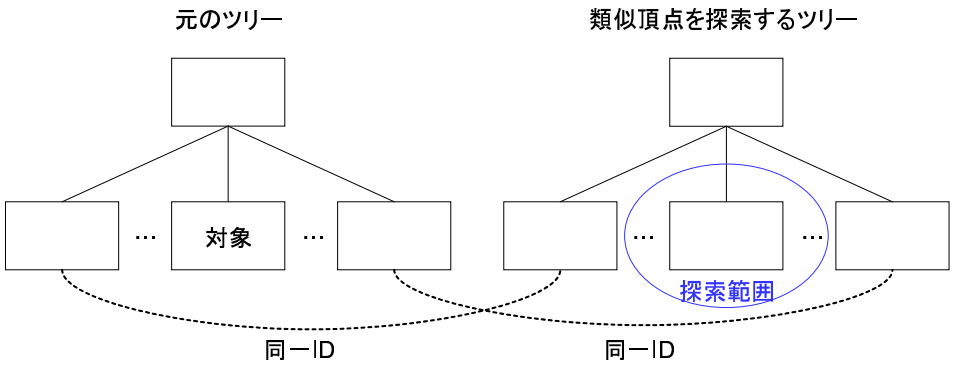
ツリー A と B の差分を表す編集スクリプト S を，A 以外のツリー C に適用し，新たなツリーを得ることを，マージと呼ぶ．

3.3.1 代替頂点と代替位置の探索

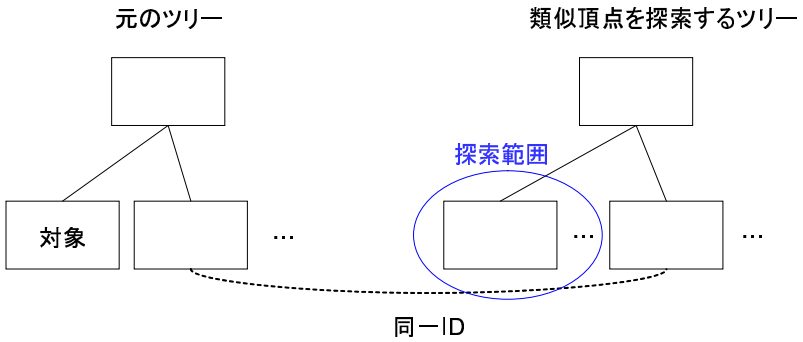
編集操作は，対象頂点の ID を引数に取る．しかし，C に S を適用する際に，編集操作の引数となる ID を持った頂点が C にあるとは限らない．

操作する ID を持った頂点が C に無い場合，操作対象に類似した頂点を探して代替する．代替頂点の検索は，以下の 2 つの方法で行う．

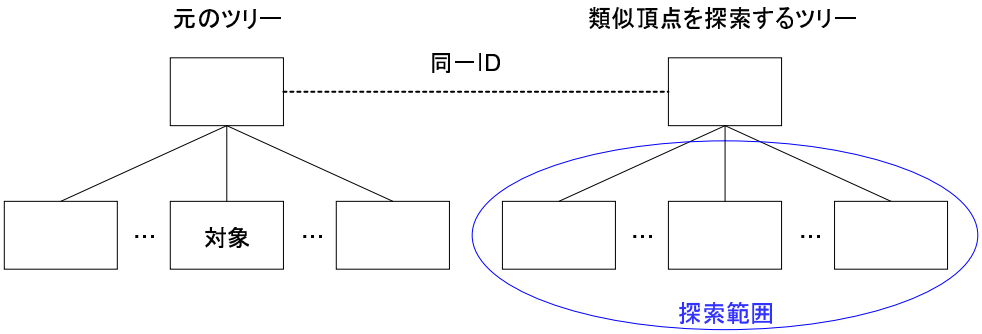
1 つめの方法では，兄弟の頂点から探索する (図 3(a))．対象の左右にある頂点の組の集合を考える．そのうち，探索するツリー中に同じ ID の頂点があり，その親が共通している頂点



(a) 兄弟頂点から探索



(b) 兄弟頂点から探索 (端にある場合)



(c) 親頂点から探索

図 3: 類似頂点の探索

の組だけの集合をつくる．そこから，元のツリーで，頂点間の距離が最も短い頂点の組だけを取り出す．探索するツリーで，その組と同じ ID を持つ頂点に挟まれた頂点の中から，同じ ID を持つ頂点が元のツリーに無く，テキストが類似している頂点を候補として取り出す．

対象頂点が，元のツリーで左端か右端にある場合は，図 3(b) のように，その反対側にある頂点だけを考えて，探索を行う．

2 つめの方法では，親の頂点から探索を行う．図 3(c) のように，親の頂点と同一の ID を持つ頂点の子供を候補として，同じ ID を持つ頂点が元のツリーに無く，テキストが類似している頂点を候補として取り出す．

2 つめの方法は，1 つめの方法で探した頂点の親の ID が元の頂点の ID と異なる場合にのみ行う．

これらの方法で探索した頂点の候補には点数を付ける．ID の一致する頂点が見つかった場合には 3 点を与える．代替頂点を探した場合には，その頂点の親頂点，左の頂点，右の頂点のそれぞれが一致している場合に 1 点ずつ与える．

編集操作の引数となる位置には ID が付けられていないため，毎回探索する必要がある．代替頂点の探索と同様に，左右と親の頂点から代替位置を探索し，得点を付ける．

3.3.2 適用

代替頂点と代替位置の探索で，複数の候補が見つかった場合には，頂点と位置の組み合わせについて，適用した場合のツリーを作成する．

操作を適用する際に，探索において，同一の ID の頂点や，良く類似した代替頂点・代替位置が見つからなかった場合には，その操作を適用しなかった場合のツリーも作成する．

ツリーには，適用した操作の数と，代替頂点や代替位置の得点合計を記録しておく．

3.4 マージ結果の選択

マージ処理の段階で，操作の対象となる頂点が存在しない場合の処理には選択の余地がある．そのため，マージ結果もまた複数存在しうる．

マージ結果が複数得られた場合は，どれが正しい結果なのかは一概には決められない．

そこで，操作の適用時に記録した得点と，出力結果のソースコードが構文制約に合致しているかどうかでマージ結果を整列し，開発者に提示する．開発者は提示されたソースコードのなかから，1 つを選択することで，マージ処理は完了する．

3.5 マージシステムの全体像

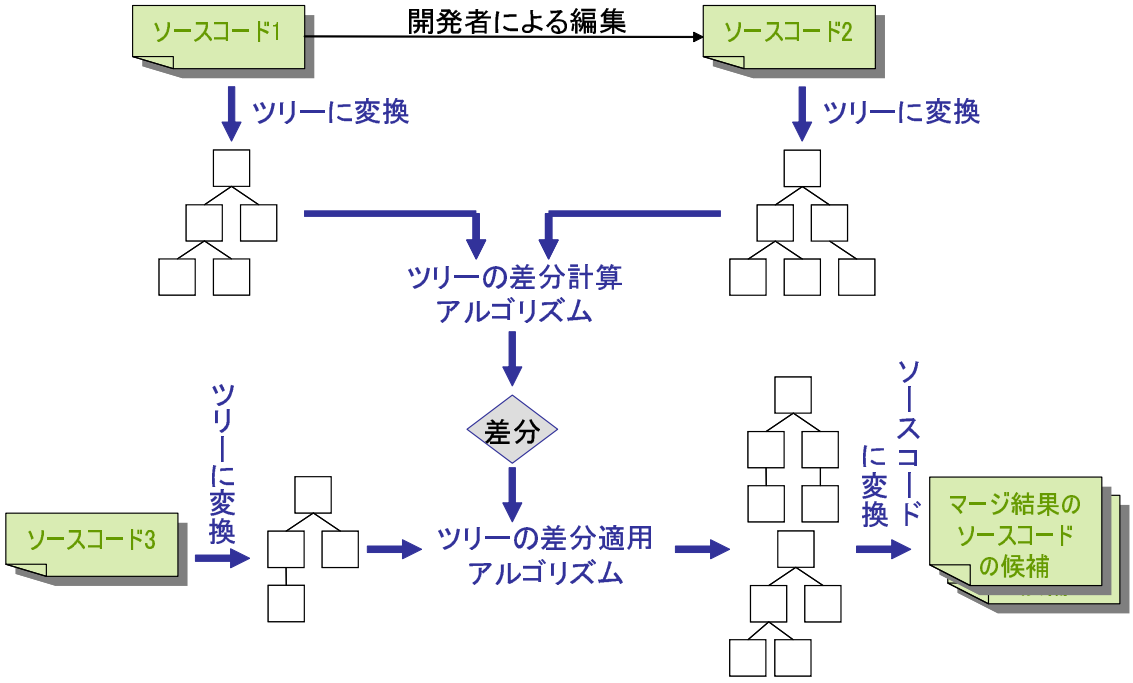


図 4: マージシステムの全体像

4　システムの実装

本節では，システムをどのように実装するかについて説明する．

システムの実装は，既存の版管理システムである subversion [5] を拡張して行った．

4.1　システムの概要と既存のシステムとの比較

図 5 が，既存の subversion システムの概略図であり，矢印はデータの流れを表している．subversion は，リポジトリを管理するサーバと，差分計算やマージ処理を行うクライアントに分かれている．開発者は，ソースコードを subversion クライアントに渡し，subversion クライアントは，開発者から渡されたファイルと，リポジトリに保存されたファイルを使って，テキストの行単位でマージを行ったり，差分を計算したりする．

図 6 が，拡張を施した subversion システムの概略図である．開発者が扱うファイルが，ソースコードである場合にだけ，特別な処理をする．システムの拡張は，クライアントだけに行い，開発者の操作と，サーバ側の処理には変更を加えない．サーバのリポジトリには，ソースコードを変換したツリーのデータが格納される．ツリーの表現には XML を用いる．

開発者が扱うファイルは全てソースコードであり， subversion クライアントがマージや差分計算を行う時には XML で行う．

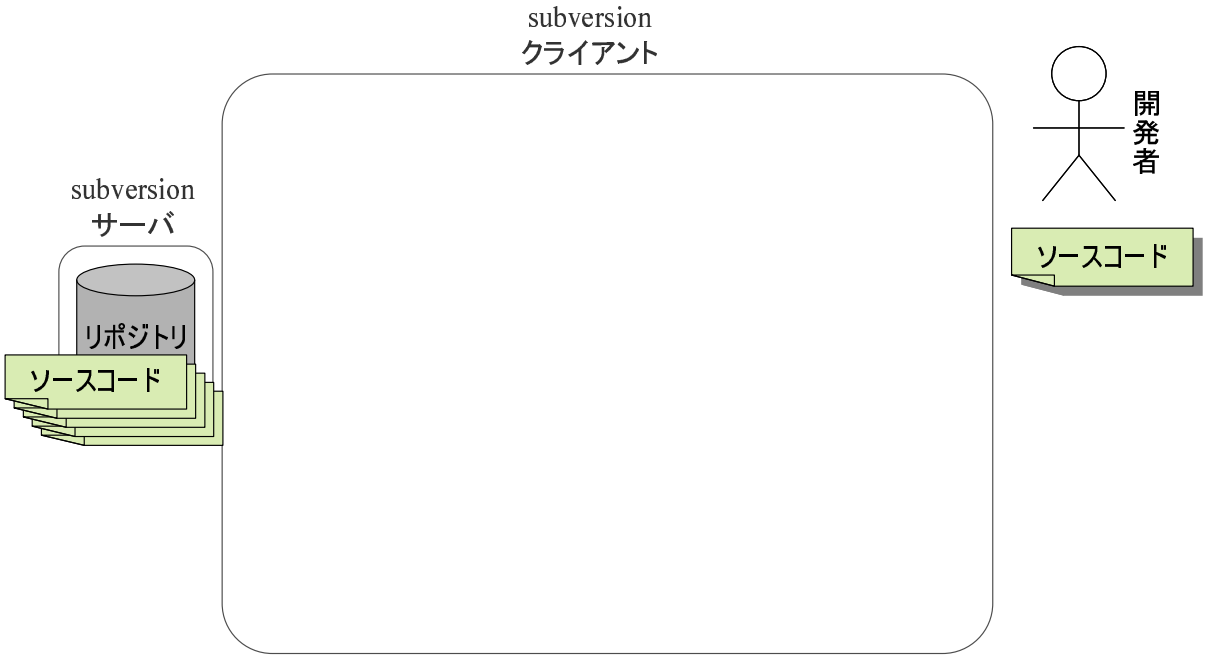


図 5: subversion の概要

4.2 ソースコードと XML の変換

4.3 ソースコードから XML への変換

Java ソースコードを解析し，空白文字やコメントの頂点を保ったまま構文解析木を作成する．全て構文解析木の頂点を XML の頂点に対応させ， XML 文書を作成する．

まず，XML ファイル上で，意味解析を行い変数を利用している識別子頂点から，識別子を宣言している頂点へリンクを張る．リンクされる頂点に一時的な ID を付与し，その ID でリンク元の頂点から参照する． ID とリンクの表現には，属性 id と ref を用いる．リンク元の頂点の ref 属性に，リンク先の頂点の ID を格納する．

次に，新しく作成した XML ファイルと，直前のバージョンの XML ファイルとの間で，頂点対応の計算を行う．直前のバージョンの XML ファイルの全ての要素には，既に ID が付けられている．計算には 3.2.2 節で説明した，FMES アルゴリズムの 1 段階目と， 3.2.3 節で説明したリネーム解析を用いる．

この計算により，作成した XML ファイルの頂点のうち，直前のバージョンの XML ファイル内の頂点と対応する頂点があった頂点には，対応した頂点と同じ ID を付ける．対応する頂点が無かった頂点には，新しい ID を付与する．

識別子に一時的付けた ID を変更する時には，その頂点にリンクしている頂点のリンク先

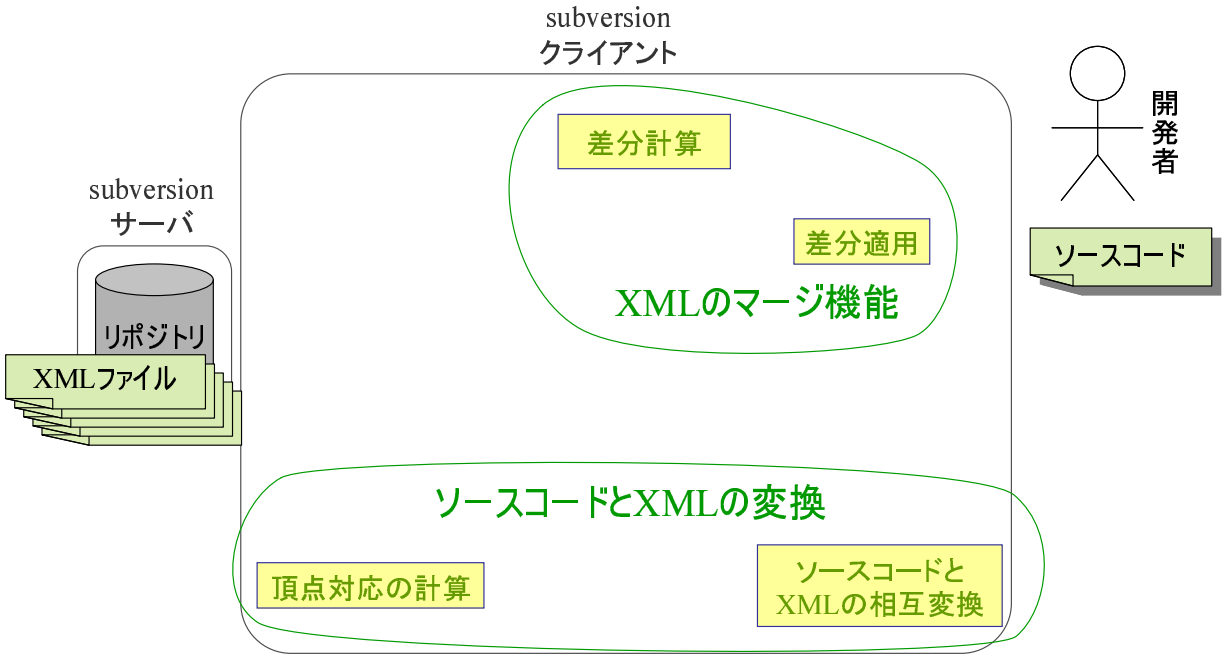


図 6: ソースコードの木構造を考慮した subversion の概略

も同様に変更する．

新しい ID を付ける際には， Universally Unique Identifier (UUID) を用いる．

新規にファイルを格納する場合で，直前のバージョンが無いときには，全ての頂点に新しい ID を付与する．

4.4 XML からソースコードへの変換

XML ファイルから，ソースコードへ変換する際には， XML ファイルから全てのタグを取り除けばよい．

4.5 取得と格納の実装

取得と格納を行う時の，データの流れを表したのが，図 7 と図 8 である．取得の時には，subversion クライアントが XML からソースコードへの変換を行い，開発者に渡す．逆に，リポジトリに格納する前には，入力されたソースコードから XML への変換を行ない，取得した時の XML と頂点の対応を計算する．

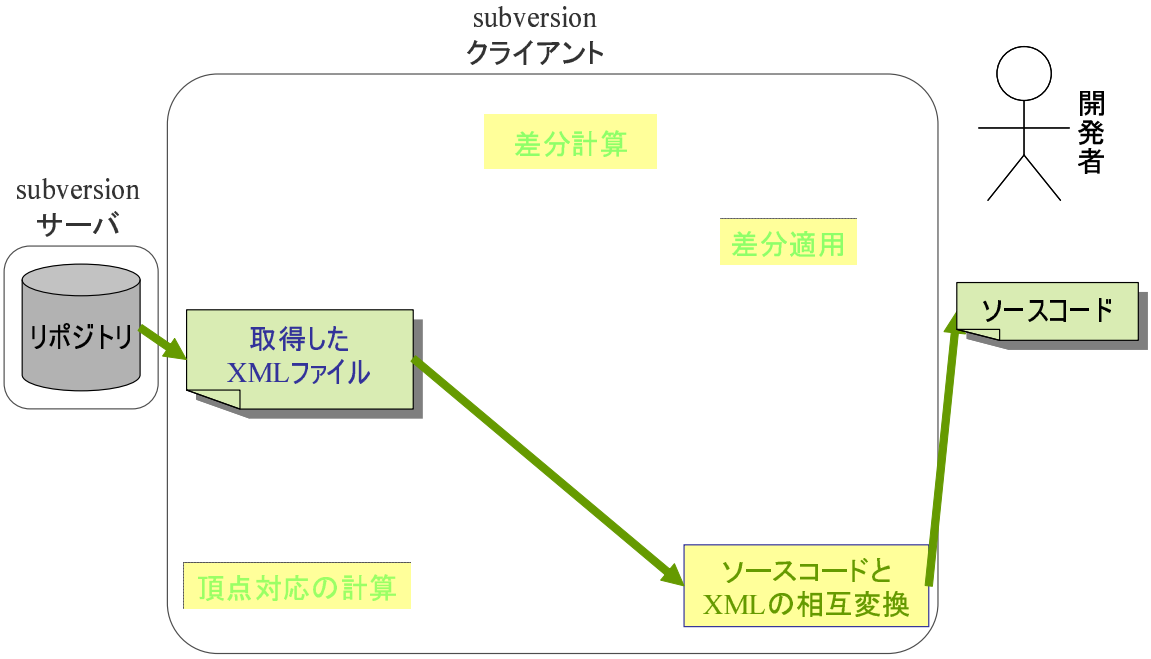


図 7: リポジトリからの取得

4.6 マージの実装

マージを行う時のデータの流れを表したのが，図 9 である．

まず，格納の時と同様に，開発者の編集したソースコードを XML に変換した後に，取得した時の XML ファイルと比較することで，頂点 ID 付きの XML ファイルを作る．

次に，リポジトリに格納されている最新の XML ファイルと，取得した時の XML ファイルとの間で差分計算を行う．差分は編集スクリプトであり，複数の差分が発見されることがある．

この差分を，ソースコードから作った XML ファイルに適用する．

4.6.1 差分の表現

XML 文書の差分を表現するのには，XML ファイルを用いる．

差分を表す XML の root は，`edit-script` 要素である．その直下に，以下の要素を，適用する順に置く．

編集操作には 3.2 節で説明した編集操作を， XML に対応させるために拡張したものを用いる．

以下に，編集操作一覧を示す．

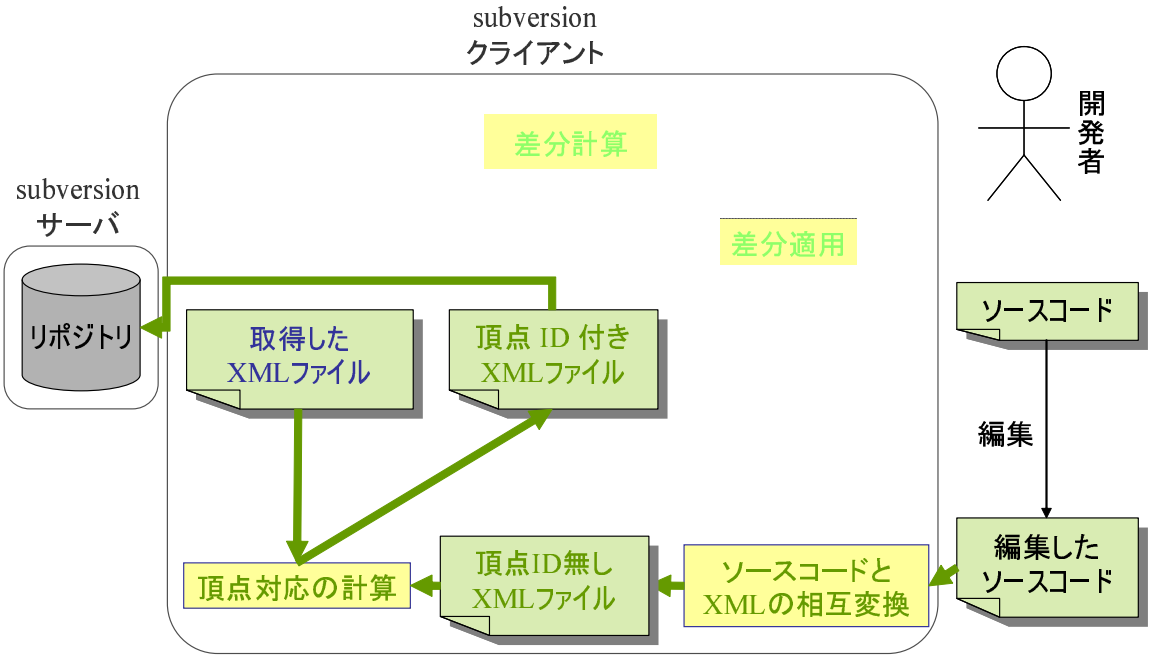


図 8: リポジトリへの格納

insert-element 要素 要素頂点の追加を表す操作である．子供に追加する要素を持つ．必ず以下の属性を持つ．

parent-id 属性 追加する先の親頂点が持つ ID を指定する

index 属性 追加する先の親頂点の，何番目の子供の後に追加するかを示す整数値を指定する． 0 を指定したときには，子供の先頭として追加する

insert-text 要素 テキスト頂点の追加を表す操作である．子供に追加するテキストを持つ．必ず以下の属性を持つ．

parent-id 属性 追加する先の親頂点が持つ ID を指定する

index 属性 追加する先の親頂点の，何番目の子供の後に追加するかを示す整数値を指定する． 0 を指定したときには，子供の先頭として追加する

insert-node-list 要素 要素頂点とテキスト頂点の列を追加を表す操作である．子供に追加する要素とテキストの列を持つ．必ず以下の属性を持つ．

parent-id 属性 追加する先の親頂点が持つ ID を指定する．

index 属性 追加する先の親頂点の，何番目の子供の後に追加するかを示す整数値を指定する． 0 を指定したときには，子供の先頭として追加する．

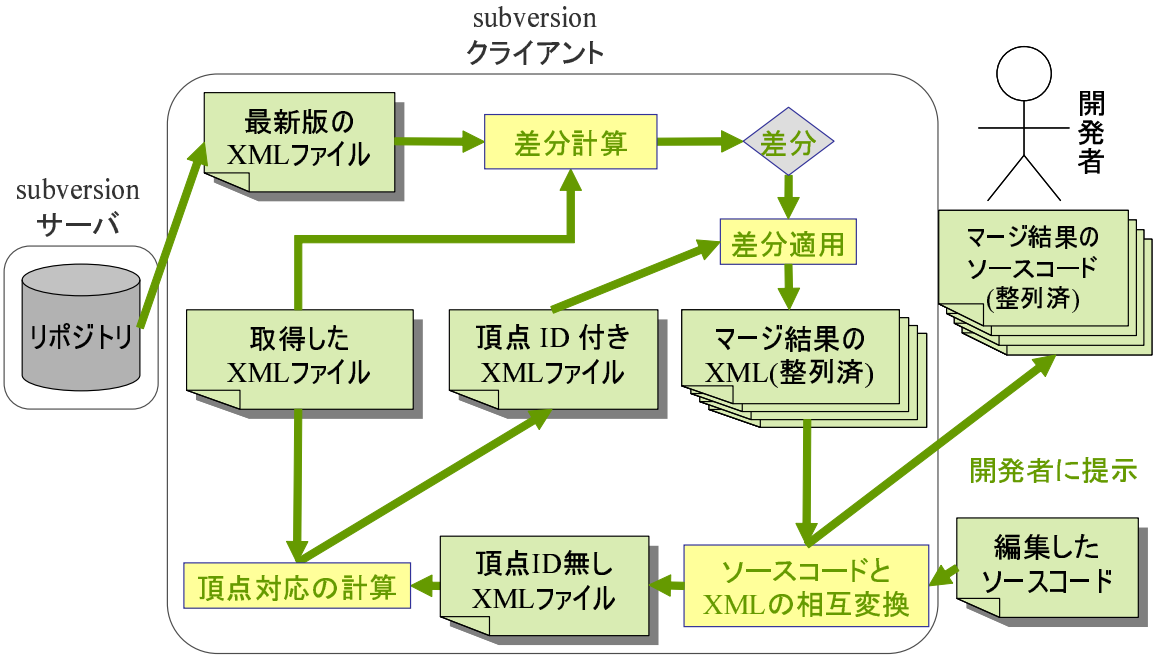


図 9: 木構造に対応したマージ

delete-element 要素 要素頂点の削除を表す操作である．必ず以下の属性を持つ．

id 属性 削除対象の要素頂点が持つ ID を指定する．

delete-text 要素 テキスト頂点の削除を表す操作である．必ず以下の属性を持つ．

parent-id 属性 削除対象のテキスト頂点の，親の頂点が持つ ID を指定する．

index 属性 削除対象のテキスト頂点が，その親の何番目の子供であるかを示す整数値を指定する．

move-element 要素 要素頂点の移動を表す操作である．必ず以下の属性を持つ．

id 属性 移動対象の要素頂点が持つ ID を指定する．

parent-id 属性 移動先の親頂点が持つ ID を指定する．

index 属性 移動する先の親頂点の，何番目の子供の後に追加するかを示す整数値を指定する． 0 を指定したときには，子供の先頭として追加する．

move-text 要素 テキスト頂点の移動を表す操作である．必ず以下の属性を持つ．

old-parent-id 属性 移動対象のテキスト頂点の，移動前の親の頂点が持つ ID を指定する．

old-index 属性 移動対象のテキスト頂点が，移動前にその親の何番目の子供であるかを示す整数値を指定する．

parent-id 属性 移動先の親頂点が持つ ID を指定する．

index 属性 移動する先の親頂点の，何番目の子供の後に追加するかを示す整数値を指定する． 0 を指定したときには，子供の先頭として追加する．

update-element 要素 要素頂点の更新を表す操作である．必ず以下の属性を持つ．

id 属性 更新対象の要素頂点が持つ ID を指定する．

また，更新操作によって，以下のような形式の属性を持つ．

insert- で始まる属性 更新対象の頂点に，先頭から接頭辞 “insert-” を削除した名前の属性を追加する．属性値は，追加する属性の属性値を表す．

delete- で始まる属性 更新対象の頂点から，先頭から接頭辞 “delete-” を削除した名前の属性を削除する．

update-old- で始まる属性 更新対象の頂点の持つ，先頭から接頭辞 “update-old-” を削除した名前の属性の値を更新する．属性値は，更新前の値を表す．

update-new- で始まる属性 更新対象の頂点の持つ，先頭から接頭辞 “update-old-” を削除した名前の属性の値を更新する．属性値は，更新後の値を表す．

update-text 要素 テキスト頂点の更新を表す操作である．以下の属性を必ず持つ．

parent-id 属性 更新対象の頂点の，親の頂点が持つ ID を指定する．

index 属性 更新対象の頂点が，その親の何番目の子供であるかを示す整数値を指定する．

また，以下の子要素を必ず持つ．

old-content 要素 子要素に，更新前のテキストを持つ要素頂点である．

new-content 要素 子要素に，更新後のテキストを持つ要素頂点である．

insert, delete, update, move の全ての操作には， -element で終わる名前の要素頂点を操作対象とする操作と， -text で終わる名前のテキスト頂点を操作対象とする操作がある．

テキスト頂点は属性を持てないため，頂点 ID を持たない．そのため，テキスト頂点を操作対象とする操作は，親頂点の ID と，その何番目の子供であるかを表す数値を指定することで，操作対象の頂点を特定する．

要素頂点を操作対象とする操作は，操作対象の頂点の ID を指定できるため，3.2 節で説明したものと同じである．

また， XML の差分表現には， 3.2 節で説明したのとは異なる `insert-node-list` 操作がある．これは，連続する `insert` 操作をまとめることで，差分適用時の代替頂点探索コストと，ツリーの分岐コストを減らすための操作である．詳しくは 節で説明する．

4.6.2 差分の適用

マージ処理で差分を適用する時には，差分を計算計算する元となったオリジナルのツリーにもまた，同じ差分を同時に適用する．オリジナルのツリーに差分を適用するときには，代替頂点や位置を探す必要は無い．

マージ処理で，テキスト操作の適用を行うときや，要素頂点の操作で，操作対象の頂点 ID を持つ無い場合には，代替頂点の探索を行う．また，位置を指定する操作である， `insert` 操作や `move` 操作を適用するときには，必ず代替位置の探索を行う．

代替頂点と位置の探索には， 3.3.1 節で説明した方法を用いる．

代替頂点の候補が複数見つかったために複数ツリーを作ることになった場合や，類似度の低い候補しか見つからなかったために操作を適用しない場合のツリーを作成することになった場合には，ツリーを複製して対処する．複製したツリーも，元のツリーと同時に，次の操作を適用する．

この方法では，複製が繰り返されると，ツリーの数は指数的に増えてしまう．ツリーの数が増え過ぎると，操作を適用するのに必要な時間が長くなるという問題と， メモリが不足するという問題が発生する．そこで，同時に操作を適用するツリーの数に上限を設ける．その数を越えた場合には， 3.3 節で説明した得点を基準とし，得点の高いツリーを残す．得点の低いツリーはファイルに書き出して，一旦適用処理の対象から外す．このとき，その時点でのオリジナルのツリーもファイルに書き出して保存しておく．

残った得点の高いツリーへの適用が終わったあとに，ファイルに書き出したツリーと，オリジナルのファイルを読み出して，適用を再開する．

全ての操作の適用が終わったツリーは以下の確認を行う．

- 識別子頂点のリンク先が存在しているかの確認．リンク先が存在していなかった場合は，警告を表示する．
- 識別子頂点のリンク元とリンク先で，子供となるテキストが一致しているかの確認．テキストが異なっていた場合は，変数がリネームされているので，リンク元のテキストを変更して，リンク先のテキストと同じにする．

5　解決する問題の例

2.3 節で述べた問題が，本システムによって，具体的にどのような形で解決するかを例として示す．

2.3.1 節で示したように，行単位のマージでは，同じ行が編集されているときには，自動的なマージ処理を行うことは出来なかった．

提案する手法では，図 10 のように，元のソースコードの行と，開発者 A の編集結果，開発者 B の編集結果をツリーにする．開発者 A と B が行った変更は，水平線と斜線で表示した頂点の追加である．開発者 A と B の変更をマージすると，マージ結果として，図の下に表示されているツリーが得られ，マージに成功する．マージ結果のツリーをソースコードに戻した結果は，図の一番下に表示されている．

このように，同じ行に対する変更であっても，ツリー上で操作した場所が異なれば，マージすることが出来る．

また， 2.3.2 節で示したように，行単位のマージでは，変数の定義と利用を考慮しておらず，宣言していない変数を使うマージ結果を生む場合があった．

提案する手法では，図 11 のように，元のソースコードと，開発者 A の編集結果，開発者 B の編集結果をツリーにする．B の編集結果では，B の追加した，変数 avg を利用している頂点から，変数を定義している頂点へのリンクが張られている．これらの変更点をまとめると，図の下のツリーになる．このツリーでは，変数へのリンクが切れており，不正なソースコードであることが分かる．

このように，提案手法では，異なる行に対する変更であっても，変数の定義と使用の関係が崩れる場合を発見することが出来る．

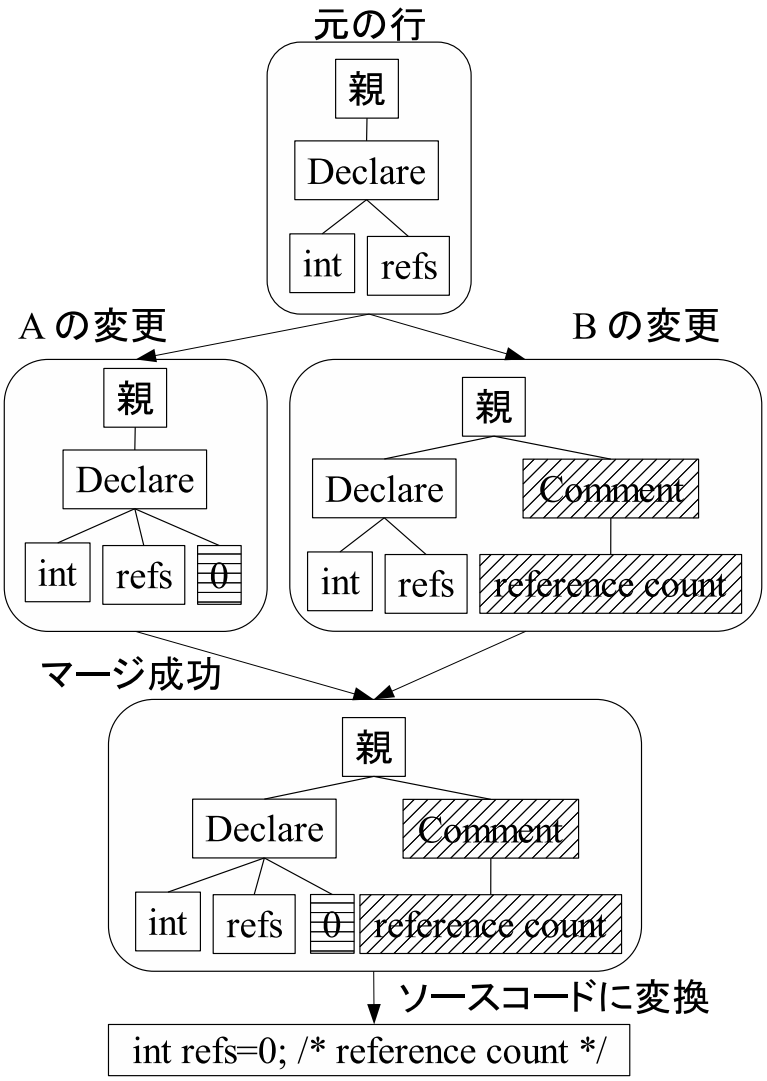


図 10: 同じ行のマージに成功する場合

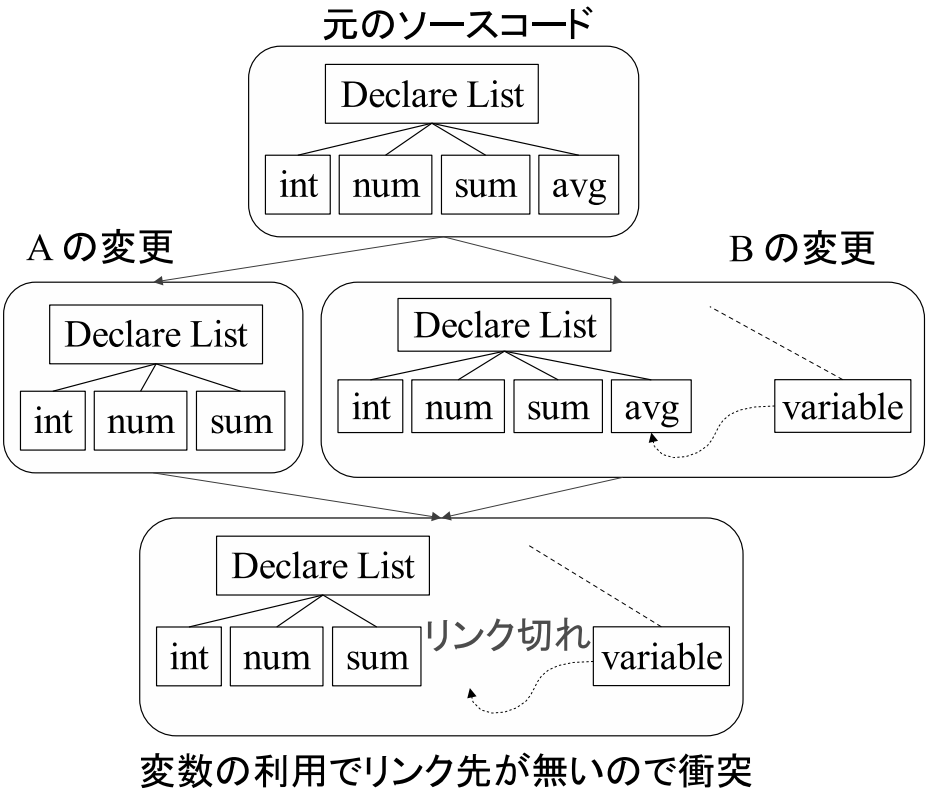


図 11: 未定義変数を発見して衝突する場合

6 考察

6.1 実験 1 サンプルに対する適用

作成したシステムを試すために，サンプルのソースコードを作成し，適用を行った．異なる変更を加えたソースコード 1, 2, 3 を作成した．それぞれに加えた変更について説明する．

ソースコード 1 オリジナルで宣言されていた変数 `avg` を削除し，代わりとなるメソッド `getAvg()` を追加した．

ソースコード 2 新しいメソッド `standardDeviation()` を追加した．メソッド standardDeviation の中では，変数 `avg` を利用している．

ソースコード 3 オリジナルで宣言されていた変数 `avg` を `average` に改名した．

作成したオリジナルのソースコードと，ソースコード 1, 2, 3 は，付録 A.1 節に示す．

オリジナルからソースコード 1, 2, 3 への，ツリーの差分をそれぞれ計算し，差分 1, 2, 3 として作成した．この差分を，ソースコード 1, 2, 3 へ適用し，その結果を観察した．結果を付

	オリジナルから 1	オリジナルから 2	オリジナルから 3
ソースコード 1		不正な出力	衝突
ソースコード 2	不正な出力		不正な出力
ソースコード 3	衝突	不正な出力	

表 1: 行単位のマージを試みた結果

	オリジナルから 1	オリジナルから 2	オリジナルから 3
ソースコード 1		候補が多過ぎて失敗	成功
ソースコード 2	デッドリンク検知		成功
ソースコード 3	成功	成功	

表 2: ツリーのマージを試みた結果

録 A に示す．また行単位のマージ処理でも，同様の適用実験を行い，ツリーのマージと比較した．実験結果を，表 1 と表 2 に示す．

表 1 に示すように，行単位のマージでは，全てのマージ結果で失敗している．不正な出力を得てしまったり，不要な衝突を起こしてしまうためである．

表 2 では，行単位のマージでは正しい結果を得られなかった組合せで，一件を除いてマージに成功するか，デッドリンクを発見することが出来た．失敗した一件では，代替位置の探索に失敗し，大量の候補を発生させてしまった．

このことから，代替頂点・位置の探索の精度を上げる必要があると考えられる．

6.2 実験 2 既存の開発履歴に対する擬似的な適用

6.2.1 実験対象

図 12 のように，ある 1 つのファイル X を，開発者 A と開発者 B が短時間に格納している場合を考える．このような格納が行われているもののうち，リビジョン n+1 での変更範囲が，リビジョン n での変更範囲に含まれているものは，開発者 B はリビジョン n を取得した後に編集行ったと考えられる．そうでないものは，開発者 B がリビジョン n を取得して編集したのではなく，リビジョン n-1 を取得して編集し，開発者 A の行った変更をマージした後に格納した可能性が高いと考えられる．

そこで，オープンソースソフトウェアのリポジトリから，マージが行われたと推測されるリビジョンを抽出し，開発者 B がマージを行う前の状態を擬似的に復元する実験を行った．

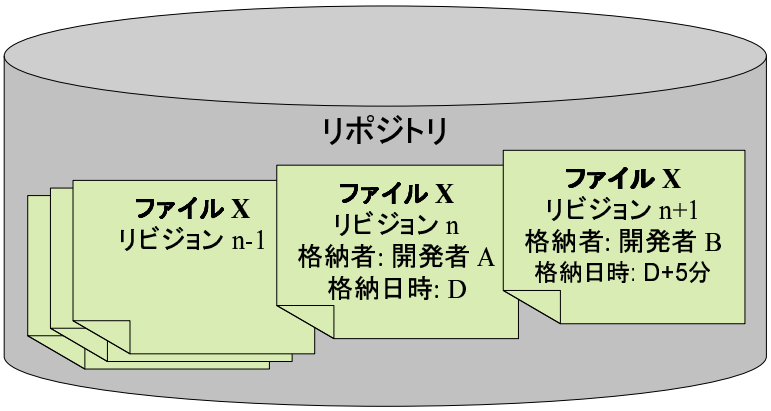


図 12: 短い時間に起こった複数人による格納

行単位のマージ	件数	ツリーのマージ	件数
成功	71	成功	71
失敗	13	成功	9
		失敗	4

表 3: 実験 2 の結果概要

実験に用いたデータは， Eclipse プロジェクトのリポジトリ (22606 ファイル，162683 リビジョン) と Jakarta プロジェクト (19420 ファイル，103358 リビジョン) から， 10 分以内に格納が起こった 84 件である．

6.2.2 結果

実験結果の概要を，表 3 に示す．行単位のマージで成功した 71 件は，ツリーのマージでも全て成功している．行単位のマージで失敗した 13 件のうち，ツリーのマージでは 9 件に成功している．

次に，行単位のマージで失敗した場合の詳細を，表 4 に示す．

ツリーのマージに失敗した 4 件のうち， 2 件は意味的にマージできない 2 つの編集をマージしようとしたためであった．残りの 1 件では，意味的にはマージできる編集であったのだが，編集操作の適用時に大量の候補が発生し，マージすることが出来なかった．

行単位のマージが失敗した原因	件数	ツリーのマージ	件数	備考
同じ行への空白文字の追加削除	4	成功	4	
意味的な変更と整形	1	成功	1	
改行コードの変更	1	成功	1	
重なりあった編集	2	成功	2	1 件で多量の候補が出現
後の変更が最初の変更を上書き	2	成功	1	
		失敗	1	大量の候補が出現
意味的な衝突	2	失敗	2	
編集に失敗したファイルの格納	1	失敗	1	構文的に正しくないソースコード

表 4: 実験 2 で，テキストのマージに失敗した場合の詳細

6.3 関連研究

一般的な XML 文書の差分を計算するシステムとして，diffxml [12]，XmlDiff [13]，DeltaXML [14]， XML TreeDiff [15]， diffmk [16] ， xydiff [17] などが挙げられる．本研究は XML の要素に ID を付与することにより，差分の適用を簡略化している点が異なる．

XML 文書をデータベースに格納してバージョン管理を行うシステムとして，XCoP [18]，Oracle XML DB [19] が挙げられる．これらのシステムは， XML の編集操作に専用の操作を用いるが，本研究ではソースコードの編集作業に利用するために，任意のアプリケーションを利用することが出来る．

Shu-Yao Chien ら [20] [21] は，XML 文書の変更を保存する際に，古いバージョンの取得コストとディスク使用量を両立させながら記録する方法を提案している．

Marc Shapiro [22] A. Kermarrec [23] らは，編集スクリプト内の操作間で依存関係を計算し，操作を並び換えることで，複数の編集スクリプトを合成して適用する手法を提案している．

7　まとめ

本研究では，版管理システムのマージ機能の問題を示し，問題への対処として，ソースコードの構文に従ったマージ処理を提案した．また，そのシステムの設計を示した．このシステムにより，マージを行う際の衝突を減らし，マージによって起こる問題を減らすことが出来る．

今後の課題は， Java 以外の言語に対応すること，マッチングの精度を向上させること，マージ処理の効率を改善すること，リポジトリ内の異なるファイル間でリンクを張る [24] ことである．

ソースコード以外の文書形式への対応も重要な課題である．

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情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝致します.

本論文の作成にあたり，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情報科学研究科コンピュータサイエンス専攻 楠本 真二 助教授に深く感謝致します.

本論文の作成にあたり，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情報科学研究科コンピュータサイエンス専攻 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情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝致します.

参考文献

[1] Open Source Initiative. The open source definition. http://www.opensource.org/docs/definition.php.

[2] Karl Fogel 著, でびあんぐる監訳, 竹内 里佳訳. CVS ―バージョン管理システム― Open Source Development with CVS. オーム社, 2000.

[3] Concurrent version system. http://www.cvshome.org/.

[4] Walter F. Tichy. Rcs — a system for version control. In *Software-Practice and Experience*, Vol. 15, pp. 637–654, July 1985.

[5] subversion. http://subversion.tigris.org/.

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

[7] perforce. http://www.perforce.com/.

[8] Susan Horwitz, Jan Prins, and Thomas Reps. Integrating noninterfering versions of programs. *ACM Trans. Program. Lang. Syst.*, Vol. 11, No. 3, pp. 345–387, 1989.

[9] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 493–504, 1996.

[10] E. W. Myers. An O(ND) difference algorithm and its variations. In *Algorithmica*, Vol. 1, pp. 251–256, 1986.

[11] U. Manber S. Wu, G. Myers, and W. Miller. An O(NP) sequence comparison algorithm. *Information Processing Letters*, Vol. 35, pp. 317–323, September 1990.

[12] Adrian Mouat. Xml diff and patch utilities. http://diffxml.sourceforge.net/, June 2002.

[13] XmlDiff. http://www.logilab.org/projects/xmldiff/.

[14] Monsell EDM Ltd. DeltaXML. http://www.deltaxml.com/.

[15] IBM. XML TreeDiff. http://alphaworks.ibm.com/tech/xmltreediff.

[16] Sun Microsystems. diffmk. http://www.sun.com/xml/developers/diffmk/.

[17] INRIA. XyDiff.
http://www-rocq.inria.fr/ cobena/cdrom/www/xydiff/eng.htm.

[18] Budi Surjanto, Norbert Ritter, and Henrik Loeser. XML content management based on object-relational database technology. In *Web Information Systems Engineering*, pp. 70–79, 2000.

[19] Oracle. Oracle XML DB.
http://otn.oracle.com/tech/xml/content.html.

[20] Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo. XML document versioning. *SIGMOD Record*, Vol. 30, No. 3, pp. 46–53, 2001.

[21] Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo. Version management of XML documents. *Lecture Notes in Computer Science*, Vol. 1997, pp. 184–189, 2001.

[22] Marc Shapiro, Antony Rowstron, and Anne-Marie Kermarrec. Application-independent reconciliation for nomadic applications. In *Proc. SIGOPS European Workshop: "Beyond the PC: New Challenges for the Operating System"*, Kolding (Denmark), 2000.

[23] A. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Twentieth ACM Symposium on Principles of Distributed Computing (PODC)*, August 2001.

[24] 中山崇. 関数の変更履歴と呼出し関係に基づいた開発履歴理解支援システムの実現. 大阪大学基礎工学部情報科学科 特別研究報告, 2004.

付録

A　実験 1 に用いたデータ

A.1　ソースコード

A.1.1　オリジナル

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum, avg;

    public Calculator(Set s) {
              set = s;
    }

    void calculate() {
              num = 0;
              sum = 0;
              for (Iterator i = set.iterator(); i.hasNext(); ) {
                    Number n = (Number)i.next();
                    sum += n.doubleValue();
                    num++;
              }
              avg = sum/num;
    }

    double max() {
              double rval = Double.NEGATIVE_INFINITY;
              for (Iterator i = set.iterator(); i.hasNext(); ) {
                    Double d = (Double)i.next();
                    if (rval < d.doubleValue()) {
                    rval = d.doubleValue();
                    }
              }
              return rval;
    }
}
```

A.1.2　ソースコード 1

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
```

```
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
    }

    double getAvg() {
        return sum/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.1.3 ソースコード 2

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum, avg;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
        avg = sum/num;
    }

    double standardDeviation() {
        double v = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            v += (n.doubleValue()-avg)*(n.doubleValue()-avg);
        }
        return Math.sqrt(v)/num;
    }
```

```
    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.1.4 ソースコード 3

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum, average;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
        average = sum/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.2 ソースコードから作成した XML ファイル

オリジナルのソースコードを XML ファイル変換した結果を示す。
分量が多いため、先頭 4 行だけを抜粋した。

```
<?xml version="1.0"?>
<program id="uuid:729d4992-15d3-0310-afe1-f3d4c8afbea3"><imp
```

ort id="uuid:a39d4992-15d3-0310-afe1-f3d4c8afbea3">import <q
ualified-name id="uuid:ac9d4992-15d3-0310-afe1-f3d4c8afbea3"
><qualified-name id="uuid:b69d4992-15d3-0310-afe1-f3d4c8afbe
a3"><qualified-name id="uuid:c29d4992-15d3-0310-afe1-f3d4c8a
fbea3"><identifier id="uuid:cc9d4992-15d3-0310-afe1-f3d4c8af
bea3">java</identifier></qualified-name><dot id="uuid:d59d49
92-15d3-0310-afe1-f3d4c8afbea3">.</dot><identifier id="uuid:
de9d4992-15d3-0310-afe1-f3d4c8afbea3">util</identifier></qua
lified-name><dot id="uuid:e79d4992-15d3-0310-afe1-f3d4c8afbe
a3">.</dot><identifier id="uuid:f09d4992-15d3-0310-afe1-f3d4
c8afbea3">Set</identifier></qualified-name><semicolon id="uu
id:009e4992-15d3-0310-afe1-f3d4c8afbea3">;</semicolon></impo
rt>
<import id="uuid:0b9e4992-15d3-0310-afe1-f3d4c8afbea3">impor
t <qualified-name id="uuid:129e4992-15d3-0310-afe1-f3d4c8afb
ea3"><qualified-name id="uuid:1c9e4992-15d3-0310-afe1-f3d4c8
afbea3"><qualified-name id="uuid:239e4992-15d3-0310-afe1-f3d
4c8afbea3"><identifier id="uuid:2d9e4992-15d3-0310-afe1-f3d4
c8afbea3">java</identifier></qualified-name><dot id="uuid:34
9e4992-15d3-0310-afe1-f3d4c8afbea3">.</dot><identifier id="u
uid:3e9e4992-15d3-0310-afe1-f3d4c8afbea3">util</identifier><
/qualified-name><dot id="uuid:469e4992-15d3-0310-afe1-f3d4c8
afbea3">.</dot><identifier id="uuid:4e9e4992-15d3-0310-afe1-
f3d4c8afbea3">Iterator</identifier></qualified-name><semicol
on id="uuid:5b9e4992-15d3-0310-afe1-f3d4c8afbea3">;</semicol
on></import>

<type-declaration id="uuid:659e4992-15d3-0310-afe1-f3d4c8afb
ea3"><class-header id="uuid:6d9e4992-15d3-0310-afe1-f3d4c8af
bea3"><class id="uuid:769e4992-15d3-0310-afe1-f3d4c8afbea3">
class</class> <identifier id="uuid:7d9e4992-15d3-0310-afe1-f
3d4c8afbea3">Calculator</identifier></class-header> <lbrace
id="uuid:849e4992-15d3-0310-afe1-f3d4c8afbea3">{</lbrace>

A.3 提案手法でのマージ結果

A.3.1 ソースコード 1 に差分 3 を適用した結果

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
```

```
        }
    }

    double getAvg() {
        return sum/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.3.2 ソースコード 2 に差分 1 を適用した結果

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
    }

    double getAvg() {
        return sum/num;
    }

    double standardDeviation() {
        double v = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            v += (n.doubleValue()-avg)*(n.doubleValue()-avg);
        }
        return Math.sqrt(v)/num;
    }

    double max() {
```

```
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.3.3　ソースコード 2 に差分 3 を適用した結果

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum, avg;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
        avg = sum/num;
    }

    double standardDeviation() {
        double v = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            v += (n.doubleValue()-avg)*(n.doubleValue()-avg);
        }
        return Math.sqrt(v)/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.3.4　ソースコード 3 に差分 1 を適用した結果

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
    }

    double getAvg() {
        return sum/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```

A.3.5　ソースコード 3 に差分 2 を適用した結果

```
import java.util.Set;
import java.util.Iterator;

class Calculator {
    Set set;
    double num, sum, average;

    public Calculator(Set s) {
        set = s;
    }

    void calculate() {
        num = 0;
        sum = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
```

```
            Number n = (Number)i.next();
            sum += n.doubleValue();
            num++;
        }
        average = sum/num;
    }

    double standardDeviation() {
        double v = 0;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Number n = (Number)i.next();
            v += (n.doubleValue()-average)*(n.doubleValue()-average);
        }
        return Math.sqrt(v)/num;
    }

    double max() {
        double rval = Double.NEGATIVE_INFINITY;
        for (Iterator i = set.iterator(); i.hasNext(); ) {
            Double d = (Double)i.next();
            if (rval < d.doubleValue()) {
                rval = d.doubleValue();
            }
        }
        return rval;
    }
}
```