情報処理学会論文誌 Vol. 49 No. 7 2349–2359 (July 2008)

開発者による編集操作に基づくソースコード変更抽出

大 森 隆 行[†1] 丸 山 勝 久[†2]

ソフトウェアの保守作業におけるプログラム理解のためには，現時点のソースコードを見るだけでは不十分であり，版管理システムに格納されている過去のソースコードや，版間の差分情報を利用することが多い．しかしながら，従来手法ではリポジトリに存在するソースコードしか扱えないため，実際に行われた変更内容の詳細を知ることはできない．変更の意図をより深く把握するためには，より詳細に変更が追跡できることが望ましい．本論文では，統合開発環境上で行われた開発者の編集操作をすべて記録し，データベースにその履歴情報を格納する手法およびツールを提案する．ユーザは，本手法により記録された操作履歴情報を利用することで，プログラム変更の理解を支援するツールを容易に構築可能である．本論文では，提案手法の応用ツールとして作成した，操作履歴の一覧表示，フィルタリング表示，ソースコードの復元を行うツールを紹介する．

A Method for Extracting Source Code Modifications from Recorded Editing Operations

Takayuki Omori[†1] and Katsuhisa Maruyama[†2]

In software maintenance, it is insufficient for understanding programs to look at only an existing snapshot of source code. Therefore, a maintainer often examines differences between its snapshots stored in repositories of version control systems. Unfortunately, these differences do not contain all past modifications although they can help the maintainer grasp the intention of the modifications. This paper proposes a method for recording all editing operations a developer has applied to source code on an integrated development environment. It also shows a running implementation of the proposed method, which is called Operation Recorder built as an Eclipse plug-in. It facilitates tool venders creating respective applications that utilize source code modifications of a program. To demonstrate ease of tool development, we have developed three tools: an operation history viewer, an operation history filter, and a source code restorer, by using the Operation Recorder.

1. はじめに

ソフトウェアの保守作業においては，プログラム理解が重要である．これは，Concurrent Versions System（CVS）[1)] や Subversion [2)] などの版管理システムを用いて行われるソフトウェア開発においても同様であり，過去の開発内容を知るために，特定の版のソースコードだけでなく，そのソースコードがどのように変更されてきたのかを知る必要がある．しかしながら，この作業は，過去のプログラムにさかのぼり理解しなければならない，あるいは他の開発者により開発されているなどの要因により，一般的に困難である[3)]．

この問題に対して，保守者は，版間の変更内容を推測するために，ソースコードを版管理システムのリポジトリに格納する際に付加されるコメント（ログメッセージ）を利用することができる．しかしながら，このコメントは必ずしも付加されるとは限らない．また，付加されていても，それが変更内容を適切に説明しているとも限らない．

従来から，より詳細に変更内容を知るために，版間の差分をとることが行われてきた[4)]．最も基本的な差分抽出ツールとして，Unix のユーティリティである diff があげられる．しかしながら，diff は 2 つのテキストファイルを比較し，その差分を行ベースで出力するのみであるため，プログラムに適用すると，構文構造を無視した結果となり，理解性の高い差分とはならない．

行ベースの差分抽出に対して，プログラムを解析して得られる構文情報を用いて，よりプログラムに適した差分情報の抽出を試みる研究も数多く行われている[3)–5)]．しかしながら，それらの手法においても，実際に行われた変更の詳細を直接的に知ることは不可能である．たとえば，版間に何度も同じ箇所を変更した場合，その経過を見ることで開発者の意図を探ることが可能であると考えられるが，コミット時点のソースコードの差分からは，そのような情報は得られない．また，版間に多くの変更が含まれ，それらが互いに干渉している場合，変更内容の推測が難しい．具体的には，あるメソッドの名前を A から B に変更（リネーム）し，その後メソッド A を追加した場合，メソッドの内部構造まで調査しなければ，メソッド B が新規に作成されたように見えてしまう．このように，従来手法では，差分の理解性を高めるために，詳細な構文構造や意味の解析が必要となる．

†1 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†2 立命館大学情報理工学部
Department of Computer Science, Ritsumeikan University

© 2008 Information Processing Society of Japan

既存の版管理システムを用いた手法に共通の問題点は，リポジトリに存在しない状態のソースコードを扱えないことに起因する．筆者らは，このような既存手法の限界を超えるために，開発者の編集操作に基づくソースコード変更抽出が重要であると考えている．本論文では，開発者が統合開発環境（IDE: Integrated Development Environment）のエディタ上で行った編集操作をすべて記録し，その情報をデータベースに格納することで，プログラム変更理解のためのツール構築を支援する手法を提案する．本手法では，変更前のソースコードが変更後のソースコードに書き換えられるまでのすべての編集操作が記録される．さらに，本論文では，実際に広く用いられている IDE である Eclipse[6] のプラグインとして提案手法を実現したソースコード編集操作記録ツール **Operation Recorder** と，その応用ツールを紹介する．

Operation Recorder を用いることで，特定のソースコードに対して過去に行われた編集操作を，その種類や時刻に応じて容易に取得可能となるため，過去に行われた変更の意図を理解することが容易になると考えられる．この効果を確認するために，操作履歴の一覧表示ツールおよびフィルタリングツールを作成した．また，Operation Recorder は，特定のソースコードに対して過去に行われた編集操作をすべて記録しているため，開発過程の詳細な追跡が可能である．この効果を実証するために，リポジトリ内のソースコードを任意の状態に復元するツールを作成した．

以降，2 章では本研究で提案するソースコード編集操作記録ツール Operation Recorder について述べる．3 章では実際に本ツールをプログラム開発に適用した評価実験について述べる．4 章では応用ツールの実現と実行結果について述べる．5 章では既存研究と本研究の比較を行い，有効性について考察する．6 章でまとめと今後の課題を述べる．

2. ソースコード編集操作記録ツール

本章では，本研究で提案するソースコード編集操作記録ツール Operation Recorder の概要を述べ，その後，実装手法の詳細を述べる．

2.1 システム概要

Operation Recorder のシステム概要を**図 1** に示す．図 1 において「Operation Recorder」として示されている部分が本ツールである．本ツールは約 2,000 行の Java ソースコード（空行，コメントは除く）により構築される Eclipse プラグインとして実装されている*1．

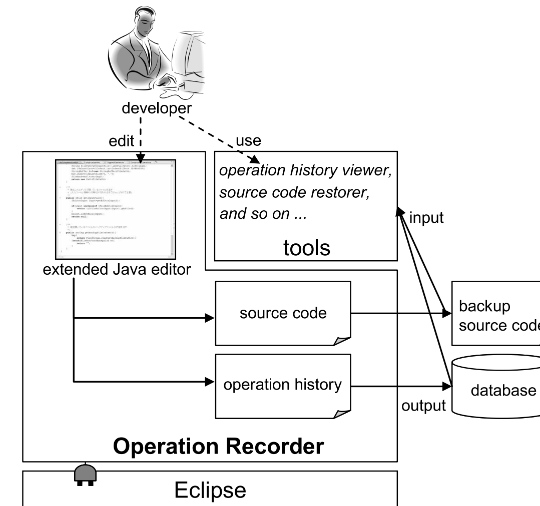

図 1　システム概要
Fig. 1　System overview.

本ツールは Eclipse の標準 Java エディタを拡張したエディタを持っており，開発者が拡張 Java エディタ上で操作を行うと，その操作履歴（operation history）をバックグラウンドですべて記録し，それを MySQL データベースに格納する．編集操作の記録に関して，ユーザは特別な作業を別途行う必要はない*2．また，本ツールはソースコードをコピーしたバックアップソースコードを作成する．バックアップのファイル名は元のソースファイルの名前の前に「.（ドット）」を付けたものであり，元のファイルとの対応がとれるようになっている．

出力されたバックアップソースコード，データベースに記録された操作履歴の内容は，これらを応用するツールの入力として利用可能である．

なお，本ツールで記録する対象の操作は，ソースコードに直接影響を与える文字の入力・削除および Eclipse のメニューから行われる自動編集である．カーソルの移動や文字の選択，マウスの動き，ファイルを開く・閉じる，エディタの切替えなどは，現時点では対象としていない．

ここで，エディタ上で以下の各操作が行われたときに，本ツールがどのように動作するか

*1 現時点では，Eclipse 3.2 において動作を確認している．

*2 MySQL サーバの起動や設定はあらかじめユーザが行っておく必要がある．

© 2008 Information Processing Society of Japan

の詳細について述べる．

- エディタを開いたとき
  前回エディタを閉じたときのソースコード（バックアップソースコードの内容）と現在のソースコードを比較し，変化していれば，それらの差分を操作履歴に登録する．このチェックは，IDE外でソースコードが編集された場合に，ソースコードの内容と操作履歴に矛盾が生じることを回避するために行っている．
- セーブ時
  Eclipseのアンドゥ履歴から，データベースに書き込む形式の操作履歴を生成し，記憶しておく．また，現在のソースコードを記憶しておく．ここで，バックアップソースコードの内容にファイルを開いてからこれまでに記録された操作を適用し，現在のソースコードと一致するかを確かめる．もし一致しなければ，それらの差分を操作履歴に登録する．このチェックは，本ツールにおいて操作履歴の取得が失敗した場合の対処のために行っている．
- エディタを閉じたとき
  前回のセーブ時に記憶された操作履歴をデータベースに書き込む．また，前回のセーブ時に記憶されたソースコードを，バックアップソースコードとして出力する．

2.2 Eclipseのアンドゥ履歴の活用

Operation Recorderが生成する操作履歴の元となるEclipseのアンドゥ履歴の内容は次の3種類のクラスのインスタンスから構成される．なお，これらのクラスは共通のインタフェース`IUndoableOperation`を実装継承している．

- `UndoableTextChange`クラス
  最も基本的な文字列操作を表現する．操作箇所を示すオフセット，挿入文字列，削除文字列などを保持している．単純な挿入操作の場合，削除文字列は空文字列となる（削除操作の場合は挿入文字列が空文字列となる）．EclipseのJavaエディタでは，文字列を選択した状態で文字を入力すると，選択した範囲の文字列が入力した文字で置換される．このような置換操作の場合，挿入文字列としては新しく入力された文字が保持され，削除文字列としては入力前に選択していた文字列が保持される．
- `UndoableCompoundTextChange`クラス
  Eclipseの自動編集機能などによって行われた操作を表現する．自動編集は`IUndoableOperation`を複数回適用することで実現されており，このクラスはそれらの`IUndoableOperation`オブジェクトを保持している．
- `TriggeredOperations`クラス
  Eclipseの自動編集機能のうち，メニュー項目から行われたリファクタリング操作などの特殊なものを表現する．`UndoableCompoundTextChange`と同様，具体的な操作の内容を`IUndoableOperation`オブジェクトとして保持している．また，操作の内容を示すラベル（リファクタリングの種類など）も保持している．

Operation Recorderは，Eclipseが保持しているアンドゥ履歴から上記のクラスのインスタンスを取得し，そこから必要な情報のみを抽出することで，データベースに格納する操作履歴を生成する．`UndoableCompoundTextChange`と`TriggeredOperations`は`IUndoableOperation`オブジェクトを保持するため，操作情報は木構造となるが，3段階以上のネストは操作履歴の理解において有用であるとは考えられないため，操作履歴生成の際，ルートと葉の2段階に変換する．葉はすべて`UndoableTextChange`オブジェクトである．

2.3 構文要素情報の付加

構文情報に基づく変更の追跡を可能とするには，編集された箇所がどの構文要素に対応するかを明らかにしなければならない．このため，本ツールは，Eclipseのアンドゥ履歴情報には含まれていない構文要素情報を，`UndoableTextChange`が保持する挿入文字列と削除文字列それぞれについて付加する．

この情報を抽出するために，ソースコードを解析して得られる抽象構文木（AST）と`UndoableTextChange`が保持するオフセット情報を利用する．ここで，編集された時点ではソースコードにエラーがあり，ASTが構築できないことがある．このため，挿入箇所については，当該操作の後に生成されるAST（エラーを含まないソースコードから生成）のうち最も近いものを使用する．後のASTを用いる理由は，挿入要素に対応する構文要素が存在している可能性が高いためである．削除箇所については，当該操作の前に生成されたASTのうち最も近いものを使用する．前のASTを用いる理由は，削除箇所に対応する構文要素が削除後のASTには存在していないためである．実際には，操作が行われた時点とASTが生成された時点でオフセットがずれることがあるため，当該操作からそのASTが構築される時点までに行われた操作に基づき，オフセットを適切に加減する．

この計算について，**図2**の例を用いて説明する．たとえば，図2において，operation-Bの挿入要素のオフセットは，

$$100 + 3(\text{mno の長さ}) - 6(\text{pqrstu の長さ}) = 97$$

と計算する．すなわち，AST-Bにおける97文字目の構文要素がoperation-Bで挿入した構文要素となる．また，operation-Bの削除要素のオフセットは，

© 2008 Information Processing Society of Japan

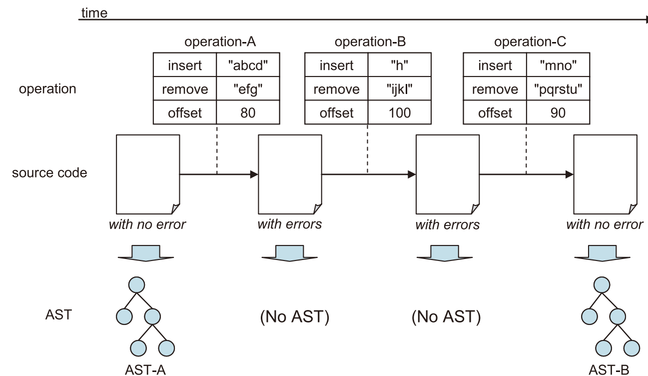

図 2　構文情報取得のためのオフセット計算
Fig. 2　Offset calculation for obtaining syntactic information.

$$100 + 3(\text{efg の長さ}) - 4(\text{abcd の長さ}) = 99$$

と計算する．すなわち，AST-A における 99 文字目の構文要素が operation-B で削除した構文要素となる．このようなオフセットの加減算は，前後の AST に至るまでに当該操作より前のオフセットにおいて操作が行われた場合にのみ必要となる．このため，図 2 において，operation-A の挿入要素のオフセットは AST-B において 80 文字目となる．

現在の実装では，オフセット計算の回数を最低限に抑えるため，構文要素情報生成の際に用いるオフセット値を，各編集箇所の先頭位置のもののみとしている．つまり，ある操作において挿入された文字列が「abcd」であった場合，付加される構文要素情報は，その先頭文字「a」がどの構文要素に属するかを示すものとなる．このため，複数の構文要素にまたがる挿入や削除が行われた場合，先頭箇所以外の構文情報が欠落する．ユーザにとって，より直感的な情報とするために，現在，操作箇所に含まれるすべての文字列に共通する構文要素を示すように改善することを検討している．

また，挿入が行われたが，すぐに削除された場合などに，操作対象の文字列が AST に存在しないことが起こりうる．このような場合，構文要素情報は直感的なものとならないため，情報を出力しないように改善することを検討している．

2.4　操作時刻情報の付加

本ツールでは，編集操作間の時間的関係を解析可能とするため，データベースに出力する操作履歴情報に，操作を行った時刻を付加する．

ここで，操作を行った時刻と，データベースに格納するための操作履歴情報を生成する時刻が異なることに注意しなければならない．Eclipse のアンドゥ履歴は，アンドゥ操作によって一部が消去されたり，後から操作の一部が変化したりすることがある．よって，アンドゥ履歴の変化に合わせて操作履歴を作り変えることは効率が悪いうえ，実装が難しく，操作履歴の堅牢性に問題が生じる可能性が高い．この問題に対処するために，Operation Recorder では，操作履歴の生成をソースコードの保存時に一括して行っている．実際には，操作が行われたことを検知すると，その時刻とアンドゥ履歴内のオブジェクトの対応を保存しておき，操作履歴構築の際に，保存した対応情報を用いて，各操作に時刻情報を付加する．

2.5　操作履歴スキーマ

データベースに格納される操作履歴のスキーマを**表 1** に示す．操作履歴は，operation テーブル，composite テーブルの 2 つのテーブルに保管される．

operation テーブルは，2.2 節で述べた `UndoableTextChange` に相当する情報を保管する．composite テーブルは，2 段階のネストに変換した操作のルートに該当するデータを保管する．すべてのデータには ID（主キー）が割り振られている．operation テーブルのデータのうち，親操作を持つものは，parent カラムで親操作の ID を参照している．

表 1 (a) および表 1 (b) の time が操作時刻情報である．また，表 1 (b) の下 6 つのデータが，構文要素情報を表す．

3.　評 価 実 験

本手法の実現可能性と Operation Recorder の実用性を示すために，提案システムを利用して，実際にプログラム開発を行い，操作履歴の記録を行った．本章では，この評価実験の結果を述べる．

評価実験に用いたプログラムは，Java アプレットとして作成したリバーシゲーム（4 ソースファイル，合計 618 行）である．作成したプログラムのファイル行数，バイト数を**表 2** に示す．これらの数値には，空行やコメントも含んでいる．

取得した操作履歴のレコード数は，composite テーブルが 278，operation テーブルが 2,023 であった．また，MySQL 4.1 リファレンスマニュアル[7] の「6.2.6 各カラム型に必要な記憶容量」により必要なデータ量を計算すると，**表 3** のようになる．表 3 において，size は各データ型により定められた容量であり，L は実データのバイト長を表す．data size は評価実験で実際に記録したデータの容量を示す．size および data size の単位はすべてバイトである．

© 2008 Information Processing Society of Japan

表 1　操作履歴データベーススキーマ
Table 1　Database schema of operation history.

(a) composite table

カラム名	型	説明
compositeID	bigint	ID．主キー
time	bigint	操作が行われた時刻
refactoringName	bigint	操作の種類．リファクタリングの名前などを保持

(b) operation table

カラム名	型	説明
time	bigint	操作が行われた時間．time と xth を合わせて主キーとしている
xth	bigint	同じ時間に複数の操作がある場合にキーの重複を避けるためのインデックス
developer	varchar(50)	操作を行った開発者
file	varchar(255)	操作が行われたファイルのパス（Eclipse ワークスペースからの相対パス）
sort	varchar(20)	操作の種類を表す．通常は「inserted」,「removed」,「modified」のいずれか
start	bigint	挿入・削除文字列の開始オフセット
end	bigint	削除文字列の終了オフセットに 1 を加えた数．挿入の場合，start と同じ値
inserted	longtext	挿入文字列
removed	longtext	削除文字列
parent	bigint	親操作を参照（composite table のキーを参照）
classNameForward	varchar(255)	挿入が行われたクラス
methodNameForward	varchar(255)	挿入が行われたメソッド
nodePathForward	text	挿入が行われた AST 上のノードパス *
classNameBackward	varchar(255)	削除が行われたクラス
methodNameBackward	varchar(255)	削除が行われたメソッド
nodePathBackward	text	削除が行われた AST 上のノードパス *

*Eclipse AST に含まれる，各ノード（`org.eclipse.core.dom.ASTNode` のサブクラス）のクラス名をルートから順にスラッシュ（/）で連結した文字列．ただし，最上位要素である `CompilationUnit` は自明であるため省略している．

表 2　実験において作成したプログラム
Table 2　Program in the experiment.

file name	lines	size (byte)
Ai.java	27	522
Board.java	249	5,511
Game.java	130	2,231
Main.java	212	4,975
Total	618	13,239

ここでは，(1) 現実的な記憶容量で操作履歴の記録を行えるか，(2) 操作履歴を正確に記録できるか，(3) 開発者の IDE 上での操作を妨げないかの 3 点について考察する．

表 2 および表 3 から分かるとおり，今回の評価実験において，本手法で必要な（本ツールで扱う）データ量は，以下のようになっている[★1]．

- ソースコード　13,239 バイト
- バックアップソースコード　13,239 バイト
- 操作履歴データ　478,236 バイト

これらの合計は 504,714 バイトであり，これは元のソースコードのデータ量 13,239 バイトに対し，約 38 倍となっている．ソースコードを何度も書き換えるような長期間の開発においては，より多くのデータ容量が必要となると考えられるが，近年のハードディスク容量を考えれば，この程度のオーダであれば，データ容量の問題は生じないといえる．

★1 データベースのテーブル定義など，個別のプロジェクトに関連しないデータの容量は除いている．

© 2008 Information Processing Society of Japan

表 3　実験において記録された操作のサイズ
Table 3　Size of information on recorded operations in the experiment.

(a) composite table

column name	size	data size
compositeID	8	2,224
time	8	2,224
refactoringName	L+1	1,946
Total		6,394

(b) operation table

column name	size	data size
time	8	16,184
xth	8	16,184
developer	L+1	18,207
file	L+1	79,469
sort	L+1	17,888
start	8	16,184
end	8	16,184
inserted	L+4	26,584
removed	L+4	15,665
parent	8	16,184
classNameForward	L+1	9,957
methodNameForward	L+1	14,648
nodePathForward	L+2	138,390
classNameBackward	L+1	5,252
methodNameBackward	L+1	7,341
nodePathBackward	L+2	57,521
Total		471,842

次に，操作履歴記録の正確性に関して述べる．4.3 節で述べるソースコード復元ツールでは，復元途中の操作に矛盾が発生すると，エラーを表示する．表 2 にあげた 4 つのソースファイルすべてについて，このツールにより復元を試みたところ，矛盾なくソースファイル生成直後の状態に復元可能であることを確認できた．これにより，今回 Operation Recorder が記録した 2,023 エントリの操作履歴は，ソースコードの初期状態復元という観点において正確であるといえる．

最後に，本評価実験を通して感じた実用性に関する考察を述べる．今回の実験では，エディタにおける編集作業中については，操作性に問題はなかった．しかしながら，操作履歴が長い場合，エディタを閉じるときに時間がかかることがあった．そこで，簡易実験として，「`System.out.println("hello");`」という文字列を何度も連続して挿入（ペースト）するという操作を行った．この実験において，100 回程度の挿入を行うと，エディタを閉じる処理の遅れが体感できる（約 1 秒程度）ようになることが判明した．なお，実験に用いた計算機は CPU Pentium4 3.2 GHz，メモリ 2 GB である．また，MySQL サーバは，Operation Recorder と同じホストで稼動している．以上のことから，現在の実装では，短期間でエディタを閉じる開発スタイルでは実用性に問題はないと思われる．ただし，長期間エディタを閉じずに開発を行う場合は，エディタを閉じたときにユーザが長時間待たされることになり，操作性に悪影響を与える恐れがある．現在，操作履歴の書き出しを，エディタを閉じるとき以外にも行うことで，実用性に関する問題を解決することを考えている．

4. 応用ツール

提案手法およびその適用例を示すために，本章では，Operation Recorder を用いて実現した応用ツールを 3 つ紹介する．

4.1 操作履歴表示ツール

このツールは，データベースから操作履歴を取得し，それをファイルごとに一覧表示する機能を持つ．具体的には，データベースからそのソースファイルについてのデータを取得する SQL 文を生成し，取得したデータを Eclipse のビュー上にテーブル形式で表示する．2.2 節で述べたとおり，操作情報は深さ 2 の木構造として表現されるため，composite テーブルに属するデータは 2 段階のツリーで表示される．それ以外はツリーでなく単独で表示される．なお，操作は時間順にソートされる．このツールは約 650 行の Java ソースコードにより実現可能であった．

図 3 に，本ツールの実行例を示す．図 3 に示すとおり，通常の操作については，オフセット，挿入・削除文字列，操作が行われた時間，開発者，操作が行われたクラス名，メソッド，構文要素の種類（AST の葉にあたる要素）を表示する．たとえば，図 3 において最上位に表示されている操作では，オフセット 365 において，「4」という文字を削除し，「3」という文字を挿入している．また，この操作は，開発者 takayuki によって 2007/10/06 12:17:24 に行われたものであり，Board クラスの Board メソッドの NumberLiteral が操作対象となっていることも分かる．

リファクタリングの場合，ツリーの上位にリファクタリング名が表示され，その子に通常の操作と同様の情報が表示される．また，自動編集機能やコンテンツアシストを利用した箇所もツリー形式（上位には「Typing」と表示される）で表示される．

© 2008 Information Processing Society of Japan

operation	time	developer	class.method
offset:365[挿入]3[削除]4	2007/10/06 12:17:24	takayuki	Board.Board//NumberLiteral Board.Board//NumberLiteral
offset:389[挿入]¥r¥n¥t¥r¥n¥t/**	2007/10/06 12:18:21	takayuki	Board.(unknown)//TypeDeclaration
offset:398[挿入]¥r¥n¥t */	2007/10/06 12:18:23	takayuki	Board.(unknown)//TypeDeclaration
offset:398[挿入]¥r¥n¥t *	2007/10/06 12:18:23	takayuki	Board.(unknown)//TypeDeclaration
offset:404[挿入]石の取得	2007/10/06 12:18:26	takayuki	Board.(unknown)//TypeDeclaration
offset:414[挿入]¥r¥n¥tpublic CellStatus getCell	2007/10/06 12:18:26	takayuki	Board.getCell//MethodDeclaration
offset:442[挿入]()	2007/10/06 12:18:37	takayuki	Board.getCell//MethodDeclaration
Typing	2007/10/06 12:18:37		
offset:443[挿入]int x, iny	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType
offset:452[削除]y	2007/10/06 12:18:37	takayuki	Board.(unknown)//TypeDeclaration
offset:452[挿入]t y, CellS	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType
offset:457[挿入]CellStatus[削除]CellS	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType Board.(unknown)//TypeDeclaration
offset:455[削除], CellStatus	2007/10/06 12:18:45	takayuki	Board.(unknown)//TypeDeclaration
offset:456[挿入]{	2007/10/06 12:18:47	takayuki	Board.getCell//Block
offset:457[挿入]¥r¥n¥t¥t¥r¥n¥t}	2007/10/06 12:18:47	takayuki	Board.getCell//Block
offset:461[挿入]assert 0<=x && x<=8	2007/10/06 12:18:48	takayuki	Board.getCell//AssertStatement

図 3　操作履歴表示ツールの実行例

Fig. 3　Screenshot of the operation history viewer.

operation	time	developer	class.method
offset:414[挿入]¥r¥n¥tpublic CellStatus getCell	2007/10/06 12:18:26	takayuki	Board.getCell//MethodDeclaration
offset:442[挿入]()	2007/10/06 12:18:37	takayuki	Board.getCell//MethodDeclaration
Typing	2007/10/06 12:18:37		
offset:443[挿入]int x, iny	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType
offset:452[挿入]t y, CellS	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType
offset:457[挿入]CellStatus[削除]CellS	2007/10/06 12:18:37	takayuki	Board.getCell//PrimitiveType Board.(unknown)//TypeDeclaration
offset:456[挿入]{	2007/10/06 12:18:47	takayuki	Board.getCell//Block
offset:457[挿入]¥r¥n¥t¥t¥r¥n¥t}	2007/10/06 12:18:47	takayuki	Board.getCell//Block
offset:461[挿入]assert 0<=x && x<=8	2007/10/06 12:18:48	takayuki	Board.getCell//AssertStatement
offset:478[挿入]7[削除]=	2007/10/06 12:18:58	takayuki	Board.getCell//NumberLiteral Board.getCell//Block
offset:479[削除]8	2007/10/06 12:18:59	takayuki	Board.getCell//Block
offset:478[挿入]=	2007/10/06 12:19:00	takayuki	Board.getCell//InfixExpression
offset:480[挿入] && 0<=y <=y<=7	2007/10/06 12:19:01	takayuki	Board.getCell//InfixExpression
offset:495[挿入];	2007/10/06 12:19:09	takayuki	Board.getCell//AssertStatement

図 4　操作履歴のフィルタリングツールの実行例

Fig. 4　Screenshot of the operation history filter.

4.2　操作履歴のフィルタリングツール

4.1 節で実現した操作履歴表示ツールに対して，ユーザが条件を指定することで，それに合致する操作のみを表示するフィルタリング機能を追加した．ユーザが指定可能な条件は，ファイル，時間範囲，変更の種類，構文要素（クラス，メソッド，AST 末端の構文要素の種類），開発者である．現在のプロトタイプでは，条件の OR 指定や NOT 指定などは実装しておらず，ユーザが指定した条件にすべて合致するもののみを表示する．このツールは 4.1 節のツールに約 350 行の Java ソースコードを加えることで実現可能であった．

図 3 で示した操作履歴に対して，「時間 2007/10/06 12:18:20 から 2007/10/06 12:19:10，メソッド名 getCell」という条件を適用した結果を**図 4** に示す．図 4 で表示されている操作情報を見ると，time 欄が 2007/10/06 12:18:26 から 2007/10/06 12:19:09 までとなっている（図 3 には存在していた，時刻 12:17:24 の操作などが表示されていない）．また，class.method 欄に注目すると，すべての操作が Board クラスの getCell メソッドで行われたものとなっている．これらのことより，操作履歴が前述の条件のとおりにフィルタリングされたことが確認できる．さらに，フィルタリングされた操作履歴の最上位を見ると，「public

© 2008 Information Processing Society of Japan

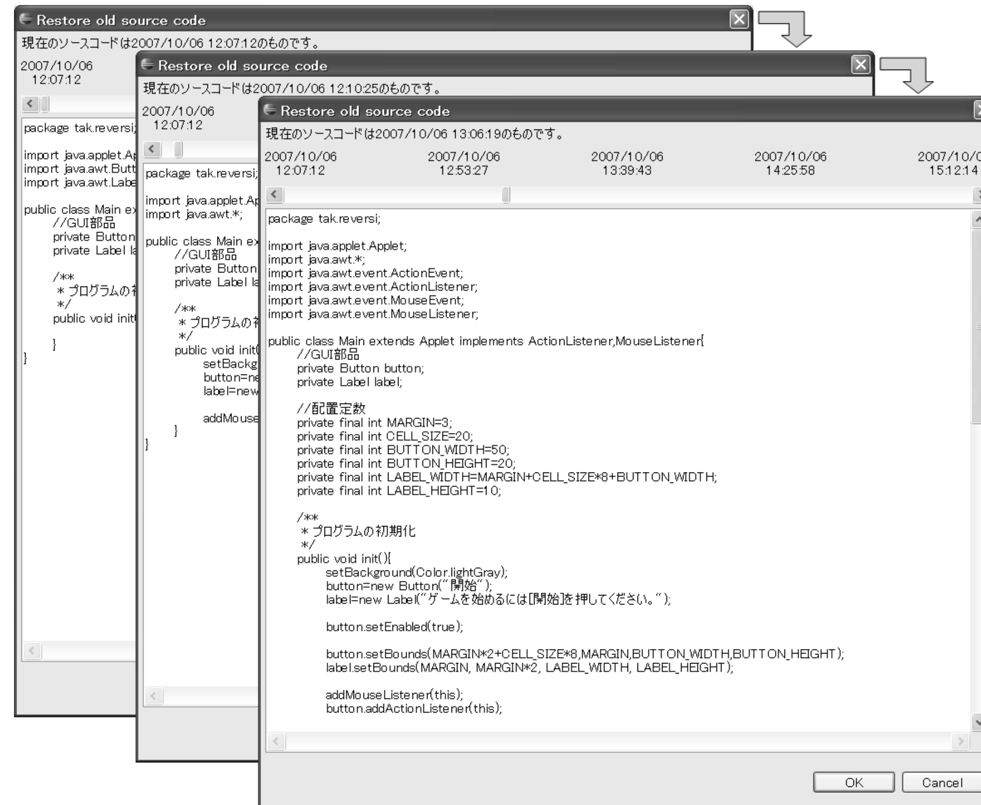

図 5　ソースコード復元ツールの実行例
Fig. 5　Screenshots of the source code restorer.

CellStatus getCell」という文字列が挿入され，ここでメソッド getCell を作成したことが分かる．

本ツールにおいてメソッド指定を利用すると，特定のメソッドに対して，その作成や削除が行われたタイミングを容易に知ることができる．また，ある特定の時間に行われた編集操作のみを抽出し，それらに着目することで，変更の意図を知ることができる可能性は高い．膨大な変更情報のうち，目的に応じて必要なもののみを表示することは変更理解のために重要であるといわれており[8]，本ツールは，このことを支援しているといえる．

4.3　ソースコードの復元ツール

本ツールは，Operation Recorder が出力する操作履歴情報とバックアップソースコードを利用して，過去の任意の状態のソースコードを表示・復元する．具体的には，操作履歴情報をすべて取得し，バックアップソースコードにその操作列を逆適用する（挿入文字列を削除，削除文字列を入力とする）ことで，ソースコードの復元を実現している．このツールは，4.1 節のツールに約 250 行の Java ソースコードを加えることで実現可能であった．

ソースコードの復元の様子を**図 5** に示す．図 5 のフォームにおいて，ユーザは，画面上

　© 2008 Information Processing Society of Japan

部のスライダを左右に動かすことで，どの時刻のソースコードに戻すかを自由に指定することができる．画面中央のテキストボックスには，ユーザが指定した時刻のソースコードが表示されている．図 5 では，1 つのソースコードについて，3 つの異なる時間の状態を復元し，時間の経過順にスクリーンショットを並べて示している．これらのスクリーンショットを見ることで，フィールドの数や init メソッドの中身が時間を追うごとに増えている様子を確認できる．

適当なソースファイルが表示されている状態において，フォーム上の OK ボタンを押すことで，Java エディタにそのソースファイルが反映される．また，スライダを連続的に動かすことで，そのソースファイルのこれまでの変遷をアニメーションのように見ることもできる．これをプログラミング教育に応用することで，学習者に模範的な開発手順を示し，学習に役立てることも可能であると考えている．

現在のプロトタイプでは単純に復元したい時刻を指定することしかできない．今後の予定として，変更がまったく行われていない時間帯を非表示にしたり，時間のスケールを日単位，時間単位，分単位などと変更可能にしたりするなどの改善を考えている．

5. 関連研究

本章では，本手法と関連研究とを比較し，本手法の有効性について考察する．

5.1 SpyWare

本研究と同様，IDE 上の開発者の操作履歴に着目した研究に，Robbes らによる SpyWare[9)–11)] がある．SpyWare は，Squeak を対象とした IDE 上に実現されており，Operation Recorder と同様，操作履歴の取得を行う．この結果をもとに，開発中の各種メトリクス値の変化の表示，開発プロジェクトの品質評価実験などを行っている．SpyWare も本手法も，IDE 上での操作履歴を取得する点は同一であり，その応用の方向性も類似している．文献 10) では，本研究で実現したソースコード復元ツールに類似した Change Replayer のほか，System Change Visualizer，Change Merger などの多彩な応用アイデアが紹介されており，本研究でもこれらの応用を実現することを考えている．しかし，文献 9)–11) を見る限り，それらのツールの実装や実現メカニズムの詳細については不明である．

また，Robbes らは SpyWare を Eclipse 上で動作させることについて論文中で触れているが，これはまだ実現していない．本研究では，広く用いられている Eclipse 環境において，Java 言語についての操作履歴を取得する手法を示し，さらにそのツールを提供している点で，意義があるといえる．

5.2 版間差分抽出手法

ここでは，プログラム変更理解支援に関して，版間の差分抽出手法および構文要素の版間対応付け手法に着目して既存手法との比較を行う．

版間の構文要素の対応付けを行うために，木構造の比較により AST のノードを対応付ける手法が提案されている[5),12)]．文献 12) の手法は AST の構造比較により型と名前の対応付けを行うことで，構文上変化している箇所（たとえば，変数名が変化している箇所）でも対応付けが行えることが特長である．しかしながら，この手法では構造が大きく変化した場合に対応付けができないという問題がある．これに対して，本手法では，取得した操作履歴からソースコード上の不変箇所を明確にし，それに対応する AST 上の要素を調べることで，正確に対応付けを行うことが可能である．

構文要素の対応付けが不可欠な問題に，XML 化されたソースコード（XSDML[13)] など）における，構文要素に対する ID 割振りがある．XSDML では更新作業を想定していないため，異なる版間で ID を引き継ぐことができず，構文要素の同一性を維持できないという問題があった．XSDML Diff[14)] を用いることで，この問題を回避することができるが，版間の構文要素のマッピングは XSDML Diff が利用している差分抽出アルゴリズム LaDiff[15)] に依存する．LaDiff は木構造差分を抽出するアルゴリズムの中で高速であるが，挿入と削除を葉要素のみに限定しているという制限がある．本手法では，すべての操作を記録しているため，版間のソースコード上の不変箇所を明確にできる．これにより，構文要素の版間の対応付けが可能である．すなわち，本手法では上記のような木構造差分アルゴリズムの制限にとらわれず，構文要素の ID を確実に次の版に引き継ぐことができる．同様に，文献 16)，17) の手法において構文要素に付加された付箋も，本手法により版間の引き継ぎが可能である．

1 章でも述べたとおり，diff などのテキストレベルのツールはプログラムの版間差分抽出には適さない．また，AST に基づいた差分抽出手法は構文木が生成可能な状態でなければ適用できない．このような，差分抽出手法どうしの制限や欠点を補うため，版間の構文要素対応付けにおいては，複数の手法を目的に応じて相補的に利用することになる[5)]．これに対し，本手法では，テキストレベルの操作をすべて記録しており，構文要素についての情報も保持しているため，テキストレベルと AST レベルの両方の長所を持ち合わせている．このため，本手法によりそれらの手法の多くが代替可能であり，複数の手法を切り替える手間を省くことができる．

プログラムの構文レベルにとどまらず，意味までを考慮した差分抽出手法に，Jackson ら

© 2008 Information Processing Society of Japan

の Semantic Diff[18]，吉田らの Semantic Diff[4] がある．これらの手法は，プログラムの入出力が変化していなければ，差分として出力しないという特長がある．このことは，プログラムの動作に着目した変更理解において有効である．本研究では，操作履歴を正確に提示することで，ソースコードを記述したときの開発者の意図の推測などを支援できると考えており，プログラムの意味の変化の有無にかかわらず，テキストの変更をすべて正確に抽出することに注力している点で，上記のような手法と方向性が異なる．

5.3 ソフトウェア進化情報の可視化

版管理システムを用いて開発されているソフトウェアの進化を視覚化することでソースコードの理解性を高める研究がある．たとえば，リポジトリを解析することにより，メトリクスの変化をグラフ化したり，変更箇所を色付けしたりする手法が提案されている[19)–21)]．このような手法は，ソースコードの行数の変化履歴を調査したり，頻繁に変更されている箇所を特定したりするなど，ソースプログラム全体を広く概観することでプログラム変化の概略を把握することに適している．これに対し，本手法では，ソースコードに対して行われた個々の変更操作を詳細に閲覧することで，エラーの原因などを詳細に追究することに適している．

6. おわりに

本研究では，IDE 上で開発者が行った編集操作をすべて記録することで，操作履歴の追跡を可能とする手法およびその実現ツールを提案した．また，それを基にした応用ツールとして，操作履歴表示ツール，操作履歴フィルタリングツール，ソースコード復元ツールを実現し，それらの有効性について述べた．評価実験では，操作履歴記録ツールが，データ量や実行速度の観点から，実用性を持つことを示した．

現在は，変更履歴情報を Eclipse のアンドゥ履歴から直接的に生成している．このようにして得られた編集操作情報は，プログラム理解のためには細かすぎることがある．これに対して，操作どうしの空間的距離（コード片の距離）や時間的距離（変更の時間の差）を用いることで，複数の操作をグループ化できる可能性がある．また，変更に含まれる構文要素の参照関係などの意味的情報を利用することで，複数の変更を抽象化して 1 つの変更として扱うことも考えられる．

実用性という観点からは，巨大なプロジェクトに対して本手法が適用可能であるかを早急に実験する必要がある．また，実際に版管理システムを用いた多人数での開発への適用も今後の課題である．本手法のさらなる応用として，現在，5.2 節で述べた，XML 化されたプログラム要素への ID 割振り，4.3 節で述べたプログラミング教育への適用のほか，リファクタリング抽出への応用などを考えている．

謝辞　本論文の執筆にあたり，有益なコメントをいただきました立命館大学情報理工学部山本哲男講師に感謝いたします．

参　考　文　献

1) Ximbiot: Ximbiot - Your Source for CVS Support. http://www.cvshome.org/
2) CollabNet, Inc.: subversion.tigris.org. http://subversion.tigris.org/
3) Yang, W.: Identifying Syntactic Differences Between Two Programs, *Software–Practice and Experience*, Vol.21, No.7, pp.739–755 (1991).
4) 吉田　敦，山本晋一郎，阿草清滋：意味を考慮した差分抽出ツール，情報処理学会論文誌，Vol.38, No.6, pp.1163–1171 (1997).
5) Kim, M. and Notkin, D.: Program Element Matching for Multi-Version Program Analyses, *Proc. 2006 International Workshop on Mining Software Repositries*, pp.58–64 (2006).
6) The Eclipse Foundation: Eclipse.org home. http//www.eclipse.org/
7) MySQL AB: MySQL AB :: MySQL 4.1 リファレンスマニュアル. http://dev.mysql.com/doc/refman/4.1/ja/
8) Neuwirth, C.M., Chandhok, R., Kaufer, D.S., Erison, P., Morris, J. and Miller, D.: Flexible Diff-ing in a Collaborative Writing System, *Proc. 1992 ACM Conference on Computer-Supported Cooperative Work*, pp.147–154 (1992).
9) Robbes, R. and Lanza, M.: A Change-based Approach to Software Evolution, *Electronic Notes in Theoretical Computer Science 166*, pp.93–109 (2007).
10) Robbes, R. and Lanza, M.: Towards Change-aware Development Tools, Technical Report, University of Lugano, Faculty of Informatics Technical Report No.2007/06 (2007).
11) Robbes, R.: Mining a Change-Based Software Repository, *Proc. 4th International Workshop on Mining Software Repositories*, p.15 (2007).
12) Neamtiu, I., Foster, J.S. and Hicks, M.: Understanding Source Code Evolution Using Abstract Syntax Tree Matching, *ACM SIGSOFT Software Engineering Notes*, Vol.30, No.4, pp.1–5 (2005).
13) 吉田　一，山本晋一郎，阿草清滋：XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装，情報処理学会論文誌，Vol.44, No.6, pp.1509–1516 (2003).
14) 高橋　透，大久保弘崇，粕谷英人，山本晋一郎：構文木に着目して XML マークアップされたソースプログラム間の差分抽出，情報処理学会研究報告，Vol.2006, No.35, pp.41–48 (2006).
15) Chawathe, S.S., Rajaraman, A., Garcia-Molina, H. and Widom, J.: Change De-

© 2008 Information Processing Society of Japan

tection in Hierarchically Structured Information, *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pp.493–504 (1996).
16) 沢田洋平，大久保弘崇，粕谷英人，山本晋一郎：付箋によるコミュニケーション機能を備えたソフトウェアブラウザ，電子情報通信学会ソフトウェアサイエンス研究会，Vol.103, No.189, pp.13–18 (2003).
17) 山本晋一郎，丸山勝久：Java を対象とした細粒度リポジトリと付箋機能を備えたブラウザ，情報処理学会 OO2003 シンポジウム，pp.189–190 (2003).
18) Jackson, D. and Ladd, D.A.: Semantic Diff: A Tool for Summarizing the Effects of Modifications, *Proc. International Conference on Software Maintenance*, pp.243–252 (1994).
19) Xie, X., Poshyvanyk, D. and Marcus, A.: Visualization of CVS Repository Information, *Proc. 13th Working Conference on Reverse Engineering* (*WCRE 2006*), pp.231–242 (2006).
20) Voinea, L. and Telea, A.: CVSgrab: Mining the History of Large Software Projects, *Eurographics/ IEEE-VGTC Symposium on Visualization*, pp.187–194 (2006).
21) Voinea, L., Telea, A. and van Wijk, J.J.: CVSscan: Visualization of Code Evolution, *Proc. 2005 ACM Symposium on Software Visualization*, pp.47–56 (2005).

(平成 19 年 10 月 18 日受付)
(平成 20 年 1 月 8 日採録)

大森　隆行（学生会員）

2003 年立命館大学理工学部情報学科卒業．2005 年同大学院理工学研究科博士前期課程修了．現在，同研究科博士後期課程在学中．ソフトウェア開発環境の研究に従事．

丸山　勝久（正会員）

1991 年早稲田大学理工学部電気工学科卒業．1993 年同大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．2000 年 4 月より立命館大学理工学部助教授．2004 年 4 月より同大学情報理工学部助教授．2007 年 4 月より同学部教授．2003 年 9 月〜2004 年 9 月 University of California，Irvine 客員研究員．ソフトウェア保守，プログラム解析，ソフトウェア開発環境の研究に従事．博士（情報科学）．電子情報通信学会，日本ソフトウェア科学会，IEEE-CS，ACM 各会員．

© 2008 Information Processing Society of Japan