コンポーネント技術に基づく Undo/Redo 機能実装手法
An Approach to Component-Based Undoable Extension Technology

鷲崎 弘宜　　白銀 純子　　深澤 良彰
Hironori Washizaki　Junko Shirogane　Yoshiaki Fukazawa
早稲田大学理工学部
School of Science & Engineering, Waseda University

2000 年 3 月 9 日

概要

アプリケーションへの Undo/Redo 機能の実装を、アプリケーションフレームワークを用いたクラス群への実装ではなく、構成する各コンポーネント単位での実装を行うことで、拡張性や再利用性に優れたものとする手法とその Java 言語についての実装である UndoableBean を示す。

1　はじめに

実行時に GUI を伴う GUI アプリケーションには、ユーザが行った操作の取消し (Undo)/ 再実行 (Redo) を実現する機能が備わっていることが多い。特に GUI を用いてユーザに複雑な操作を提供可能なアプリケーションにとって、Undo/Redo 機能は必須であると言える。

従来、アプリケーションに Undo/Redo 機能を持たせるためには、ターゲットとなるシステム標準のアプリケーションフレームワークの機能を用いて、アプリケーション全体についての実装を行う必要があり、例えば、MacApp framework や WFC(Windows Foundation Classes)、InterViews、JFC(Java Foundation Classes) 等は、Undo/Redo 機能を提供している。しかしこのような、全体への実装は実装時における開発コストが高く、またアプリケーション中の各クラスが独立して Undo/Redo 機能を実装しないため、再利用性が極めて低いという問題を抱えている。

アプリケーションを構成する基本要素を全てコンポーネントとすれば、アプリケーション全体の状態変化はコンポーネントの状態変化の集合と合致し、コンポーネントは保持すべきデータをプロパティとして明確に規定しているため、公開されるプロパティアクセサを通じて、全てのコンポーネントについて共通的に Undo/Redo 操作が可能となり、アプリケーション全体への Undo/Redo 機能実装が可能であると同時に、各コンポーネントが Undo/Redo 機能を持って独立して再利用可能となる。またその Undo/Redo 操作の対象はユーザの操作に限らず、アプリケーションにおけるあらゆるコンポーネントレベルでの動作であり、アプリケーションのユーザが直接関与しない内部動作についても Undo/Redo 操作を実現する。

本論文では、Java 言語におけるソフトウェアコンポーネント (JavaBeans) について、共通的な Undo/Redo 操作を可能とする UndoableBean フレームワーク (パッケージ) の機能、実装法およびそれらの効果的な利用法を提示する。

2　Undo/Redo 機能

以下の全てを満たす機能を一般に Undo/Redo 機能と呼び、同機能をユーザに提供するアプリケーションはユーザ操作について Undoable であるという [1]。

- 直近の動作（ユーザ操作）を取り消し可能である。
- 直近の動作を再実行可能である。
- 最近の動作を任意のまとまった量で取り消し / 再実行可能である。

アプリケーションが Undoable であるとき、使用するユーザにとって以下のメリットが得られる。

(1) ユーザビリティの向上

アプリケーションに対するユーザ操作は、人が行う以上、間違いのないことはありえず、誤った操作を常に取消し / 取消しの取消し可能であることはユーザビリティの劇的な向上につながる。従って、Undoable なアプリケーションは Undoable でないアプリケーションに対して、ユーザに高い生産性をもたらす。

(2) 複雑な GUI 設計の提供

ユーザに提供される GUI が複雑であるとき、アプリケーションはユーザが戸惑うことのないように

Undo/ Redo 機能を持つ必要がある。逆に、Undoable でないアプリケーションは複雑な GUI をユーザに提供すべきではない。

(3) アプリケーションの信頼性向上

使用するアプリケーションが Undoable であることは、誤った操作をしてもかまわないという潜在的な安心感をユーザに与える。

アプリケーションに Undo/Redo 機能を付加する必要性は、アプリケーションの新規開発段階と、開発後の仕様変更に伴う変更拡張段階のどちらにおいても生じる。

3 従来の実装手法と問題点

我々は、コンポーネント技術に基づく新たな Undo/Redo 機能実装手法を Java 言語 (Java2SE) について実装したため、以下において、その比較対象として Java 言語に関し Undo/ Redo 機能を実現するアプリケーションフレームワークである undo フレームワーク (JFC) [2] を取り上げ、その仕組みと問題点を指摘する。

歴史的に、Undo/Redo 操作を実現するための背景技術として、ソフトウェアデザインパターンである Command パターンと Observer パターン [3]、及びそれらを実行スケジューリングについて特化させた Command Processor パターンが広く用いられる [4]。

(1) Command パターン (図 1)

オブジェクトに対する要求をカプセル化することによって、要求のパラメータ化・バッファリングを実現するデザインパターンであり、ConcreteCommand を編集オブジェクトとして Undo 用メソッド及び Redo 用メソッドを用意することで、共通的な Undo/Redo 操作が可能となる。

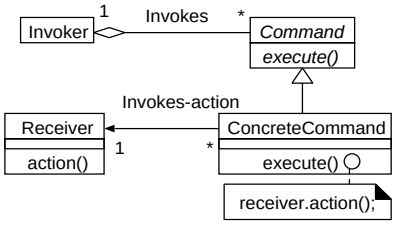

図 1: Command パターンの構造図

(2) Observer パターン (図 2)

一対多の依存関係を定義し、オブジェクトの変化を他の依存オブジェクトに自動通知する仕組みであり、Undo/Redo 操作を行いたい対象のオブジェクト本体と、それらのオブジェクトから発生する編集オブジェクトを管理する管理オブジェクト (Manager) との間の、依存性の低減及び編集オブジェクト発生の自動通知のために用いられる。

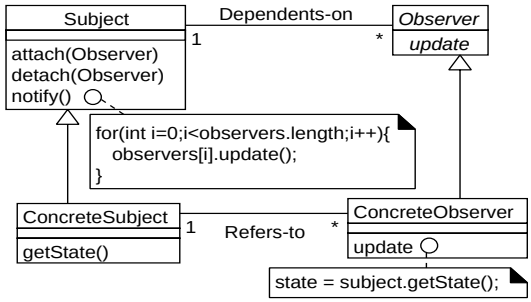

図 2: Observer パターンの構造図

javax.swing.undo フレームワークは、これらのデザインパターンを用いて共通的な Undo/Redo 機能を任意のクラス構成に実装するためのもので、対象アプリケーションに関し、取消し / 再実行を行いたい全ての編集動作を決定し、各編集クラス (ConcreteUndoableEdit) を作成して、その動作の発生元クラス (FooEditSource) をイベントソースとする。発生元クラスについて、全ての編集動作時に必ず編集オブジェクトをカプセル化した UndoableEditEvent を発火するように設定する。構成するクラス関連図を図 3に示す。

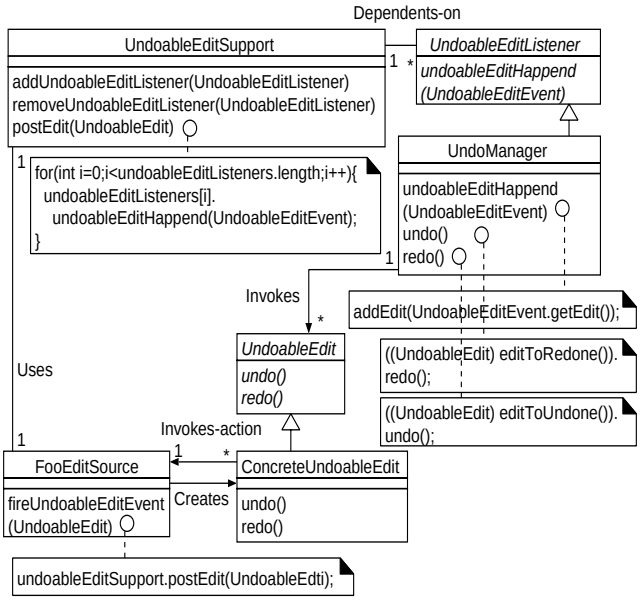

図 3: undo フレームワークのクラス・インタフェース関連構造図

undo フレームワークに従ったアプリケーションへの Undo/ Redo 機能実装は、修正時及び再利用時において以下の問題を生じる。

(1) 発生元クラス修正に伴う開発コストの増大

Undo/Redo 機能を実装することを前提とせずに作成されたクラスは、自身の編集動作について undo フレームワークの利用を前提とした設計になっていない可能性が高い。従って、既存のクラ

スへの undo フレームワークの適用による Undo/Redo 機能の追加実装設定は、以下に挙げる問題によりコストの増大をもたらす。

- クラスの Undo/Redo 操作時に復元すべきフィールド選択の必要性

  既存のクラスへ Undo/Redo 機能の実装を試みる開発者は、直接アクセスされているフィールドを全て見つけ出し、全ての変数について、 Undo/ Redo 操作を行う際に元の状態を復元すべきデータかどうか、クラスにとって状態を表す本質的なデータであるかどうかを、判定しなければならない。

- 復元すべきフィールドへの入出力操作の発見・復元設定追加の必要性

  直接アクセスされる復元対象データの判別の後に、各クラスへの Undo/Redo 機能実装を試みる開発者は、そのデータへ専用のアクセサを介さずに直接のアクセスを行っている全ての箇所をそのクラスのプログラムソース中から全て見つけ出し、その全ての箇所において、 Undo/Redo 操作についての必要な設定記述を追加する必要がある。例えば、図 4のようなクラス FooEditSource において、 int 型変数 bar を復元対象データと判断するなら、 (*1) 及び (*2) における値の直接更新時において、必ず (*X) のような UndoableEditEvent を発火・通知する動作を追加する必要がある。

```
public class FooEditSource extends JPanel{
  protected int bar;
  protected String hoge;

  public void setHoge(String hoge) {
    this.hoge = hoge;
    fireUndoableEditEvent(new HogeEdit(hoge));
  }
  public String getHoge() { return hoge; }

  public int doAnything() { return ++bar; }  // (*1)
  public void doSomething(int boo) {
    bar = bar^2 + boo;                       // (*2)
    setHoge((boo > 0) ? ‘‘Plus’’ : ‘‘Minus’’);
  }
}
  fireUndoableEditEvent(new BarEdit(bar));// (*X)
```

図 4: プロパティ”hoge” 及び直接アクセスされるフィールド”bar” についての Undo/Redo 機能実装例

(2) 発生元クラス内部改変の危険性

前述のように、アプリケーション中のクラスへ Undo/Redo 機能の実装を行う時、クラス内の専用アクセサを持たないフィールドへの入出力処理を、内部において改変する危険性がある。

(3) クラス単位での再利用性低減

- クラス毎の Undo/Redo 機能実装レベルの差異

  対象アプリケーション中の任意のクラスへの Undo/ Redo 機能の実装に関して、実装の明確な規定が存在しないため、詳細は実装者に一任される。従って各クラスの実装レベルが、実装時の単一アプリケーション中で共通に統一されていても、個別に再利用する場合において他と差異が生じる可能性が大きい。

- アプリケーション内部における相互依存性

  注意深く全体の設計を行わない限り、対象アプリケーション内部において、発生元クラスは他のクラスと直接参照等により相互に依存性を持つ。内部における依存性の強さは、アプリケーションからの目的とする発生元クラスの切り出しにくさをもたらし、Undo/Redo 機能実装済クラスの再利用を妨げる要因となる。

そこで、実装対象アプリケーションの構成要素を、クラス単位ではなく自己完備されたコンポーネントに位置づけることで、それらの問題を克服するコンポーネント指向 Undo/Redo 機能実装支援手法を導く。

4 コンポーネント技術

ソフトウェアコンポーネントの定義は、開発時と利用時における観点から最大公約数的に、「意味のある機能性を持ち（自己完備）、他と依存せずに統合するための情報（メタ情報）を提供するソフトウェア部品」とされる [6]。

(1) 自己完備されたソフトウェア部品

コンポーネントが保持するデータのうち、外部への作用として意味があると考えられるものはプロパティとして位置づけられ、外部からアクセスするためのインタフェースとしてプロパティアクセサが用意される [7]。現在のコンポーネントフレームワークは、コンポーネントがインタフェースとして、プロパティアクセサ以外の任意のメソッドを外部に公開することを可能とするが、コンポーネント内の凝縮度（cohesion）の最大化を考えたとき、望ましくない [8]。 JavaSoft(Sun Microsystems) が以前 JDK1.0 においてコンポーネントを表示するメソッドを Foo#show() としていたところを、コンポーネントフレームワーク JavaBeans の導入に伴い Foo#setVisible(boolean) に変更したように、コンポーネントへの操作は一般に全てプロパティアクセスの形をとる。

(2) メタ情報の提供

他との依存性が排除されたコンポーネントは、組み込まれたアプリケーションシステムにおいて他

のコンポーネントもしくはシステム環境と適切に通信を行うために、イントロスペクション機構を用いて自身のインタフェース情報を提供し実装の詳細を隠蔽する。各環境は内部の実装を知ることなく得られるメタ情報のみを元に、各コンポーネントを共通的に扱うことが可能である [8]。コンポーネントメタ情報としては、WindowsDNA/COM+/ActiveX における IUnknow や CORBA における IDL、及び JavaBeans における BeansInfo 等がある。

本研究では、コンポーネントフレームワークとして、Java 言語におけるコンポーネントフレームワーク JavaBeans を取り上げ、Undo/Redo 機能実装の対象とする。

JavaBeans フレームワーク（java.beans.*）は、フレームワークに従って開発されたコンポーネントクラス Bean を、ビジュアルな開発環境において容易に配置、設定することを目的とするため [5]、Bean は自身が置かれた開発・実行環境に対してメタ情報を BeanInfo オブジェクトとして提供する。

BeanInfo オブジェクトの取得には Introspector クラスを用い、提供されるクラスについて、Bean として既に BeanInfo が用意されている場合はそれを利用し、用意されていない場合は、JavaBeans におけるプロパティ・イベントについての各コーディングパターンに従った自動取得がなされる [9]。

従って、例え厳密に Bean として設計されていないクラス (javax.swing.DefaultListModel 等) であっても、メタ情報を提供するコンポーネントクラスとして扱うことが可能である。

5 コンポーネント単位での Undo/Redo 機能実現

Undo/Redo 機能の実装単位をコンポーネントに限定することで、アプリケーション全体での Undo/Redo 動作が構成する各コンポーネントの状態変化の集合により実現する（表 1）。

表 1: Undo/Redo 機能実現の差異

手法	実装対象要素	全体としての機能実現
アプリケーションフレームワーク	関連するクラス群	クラス間における通信状況
コンポーネント	コンポーネント単体	各コンポーネントの状態変化の集合

ユーザ操作を含むアプリケーションの動作を取消し / 再実行することは、アプリケーション全体の状態をその動作以前に戻す / 戻すことの取消しを意味する。

従って、アプリケーションの本質的な動作および状態保持の役割を全てコンポーネント群が担うとき、アプリケーション全体の状態変化は、コンポーネントの状態変化の集合と合致し、個々のコンポーネントの状態変化を戻す / 戻すことの取消しの積み重ねが全体における Undo/Redo 機能を実現する。

(1) Undo/Redo 操作時におけるプロパティ情報の復元

各コンポーネントの状態変化は、コンポーネントの定義に基づきプロパティアクセサを用いたプロパティの変更状況を監視することで取得することができる。

コンポーネントへの外部からのアクセスは全てプロパティアクセサを通してのみ行われるため、コンポーネントの状態を元に戻すという動作は、コンポーネント内のもっとも最近に変更されたプロパティを元の値に戻すことである。

従って、プロパティアクセサをコンポーネントクラスのサブクラスにおいてオーバーライドし、フックメソッドとして変更イベントを必ず外部に発火するように実装すれば、そのサブクラスからのプロパティ変更イベントを全て受け取り、公開されたアクセサを元にそのプロパティを元の値に変更することで、元のコンポーネントとして Undo/Redo 操作が正しくなされることになる。これは、ユーザ操作に限らず、あらゆるコンポーネントを主体とした動作を取消し / 再実行可能であることを意味する。

またプロパティはコンポーネントが最低限保持すべきデータであるため、Undo/Redo 機能実装者はデータの復元について、プロパティのみに注意すれば良く、従来手法における復元すべきデータの判別作業が不要となる。

(2) Undoable なサブクラスの自動生成

コンポーネントの定義より、Undo/Redo 機能を実装するための Undoable コンポーネント作成ツールは、コンポーネントよりメタ情報を取得可能である。コンポーネントメタ情報には、プロパティと対応するプロパティアクセサについての情報をはじめとして、対象とするコンポーネントフレームワーク固有の情報が定型的に含まれることが保証される。従って、作成ツールに各固有情報毎の共通実装方法を規定しておくことで、完全に自動的な Undoable なコンポーネントサブクラスを生成可能である。

作成ツールは実装の詳細を知ることなく、コンポーネントが環境（ここでは作成ツール）に提供するメタ情報のみを用いてサブクラスとして機能を追加するため、元のコンポーネントの内部を改

変する必要がなく安全な機能追加を実現し、追加機能以外のコンポーネント本来の動作の一致を保証する。

6 UndoableBean 概要

Java 言語におけるコンポーネントフレームワーク JavaBeans の仕組みを用いて、各コンポーネント Bean に対し Undo/Redo 機能を共通に実装する手法を提供する基本フレームワーク UndoableBean を開発した。

6.1 Undo/Redo 動作の流れ

UndoableBean 基本フレームワークにおいて、Undo/Redo 機能を実装したコンポーネントを実現するための共通インタフェース UndoableBean と、UndoableBean 実装クラスが生成する各種編集クラス（UndoableBean Edit）が提供される。UndoableBean 基本フレームワークで提供される主要クラス・インタフェースの相関関係を図 5に示す。

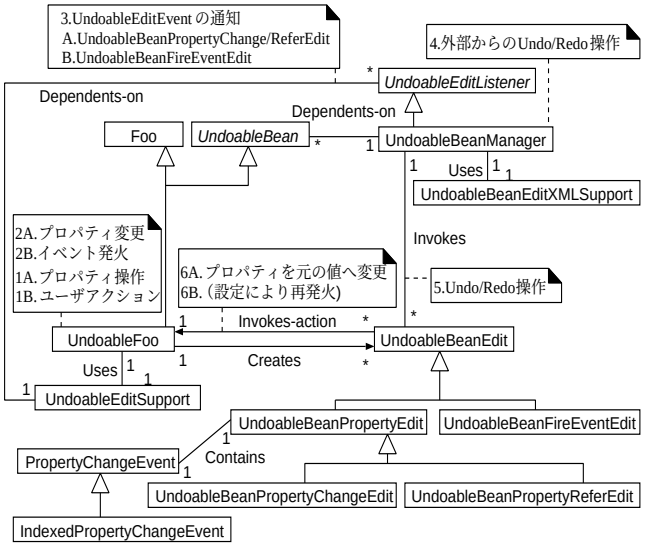

図 5: UndoableBean 主要構成要素

UndoableBean 実装クラス単体（UndoableFoo）と、UndoableBeanManager の関係による Undo/Redo 動作の流れを図 5を用いて説明する。

UndoableFoo へのプロパティ操作 (1A)、ユーザからのイベント発火 (1B) は、UndoableFoo 自身へのプロパティ変更 (2A)、イベント発火 (2B) の各動作を導く。動作は、UndoableEditListener として登録された UndoableBeanMangaer へ、UndoableBeanEdit オブジェクトを格納した UndoableEditEvent オブジェクトをパラメータとしてイベント通知される (3)。その際にプロパティは元の値と新しい値の両方が格納される。UndoableBeanManager は通知された Event オブジェクトより Edit オブジェクトを取り出してリストに格納する。その後、UndoableBeanManger への Undo/Redo 操作により (4)、リスト中からもっとも最近の Edit オブジェクトが取り出され、その Edit オブジェクトへの Undo/Redo 操作を行う (5)。その操作は、UndoableBeanPropertyChangeEdit/ReferEdit オブジェクトであれば、発生元（UndoableFoo）に対してプロパティの値を元に戻す (6A)。UndoableBeanFire EventEdit オブジェクトであれば、設定により発生元に対してイベントの再発火を行う (6B)。

6.2 アプリケーション全体での Undo/Redo 機能実現

対象アプリケーションが、全て UndoableBean 実装オブジェクトとイベントアダプタのみから構成されるとき、アプリケーションの挙動は各 UndoableBean のプロパティ変更（及びイベント発火）としてアプリケーション内唯一の UndoableBeanManager に伝わり、各プロパティの再変更・イベント再発火により、全体としての Undo/Redo 動作を実現する。

アプリケーションが全て UndoableBean 実装オブジェクトで構成された場合の概念図を図 6に示す。

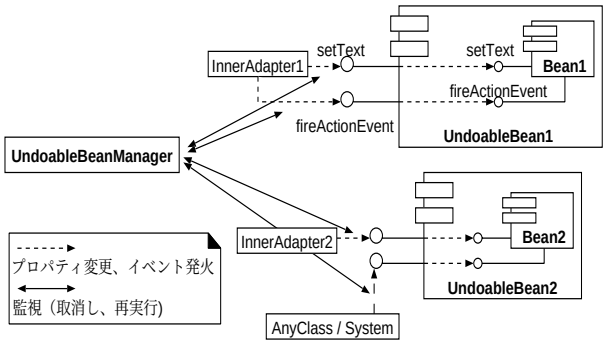

図 6: UndoableBean によるアプリケーション実装

各 Bean における Undo/Redo 機能の共通実装により、アプリケーションとして組み合わせた際に、全体として正しく想定する通りに Undo/Redo 機能が働くことが保証される。ただし以下の条件を満たす必要がある。

- アプリケーションが UndoableBean 実装オブジェクトとそれらのイベント接続用 Adapter からのみ構成されている必要がある。
- 各 UndoableBean 実装への操作は、プロパティアクセスを通じてのみ行われる必要がある。ただしこれはコンポーネント指向の基本概念として推奨されるものである。すなわちコンポーネントは保持すべきデータをプロパティとして持ち、外部に対しプロパティアクセサのみを公開する。

7 UndoableBean 基本フレームワークの構成

7.1 UndoableBean 実装クラス

Undo/Redo 機能を追加実装したい JavaBeans コンポーネント（Bean）について、 UndoableBean インタフェースを実装したクラスが UndoableBean 実装クラス（Undoable な JavaBeans コンポーネント）となる。

一度 Undo/Redo 機能を実装した UndoableBean 実装クラスは、あらゆるオブジェクト環境下において、独立したコンポーネント（Bean）として再利用可能である。

(1) UndoableBean インタフェース

動作履歴保存ファイルの設定、及び、 Undo/Redo 操作進行状況の設定機能を提供する。

(2) UndoableBeanSupport クラス

UndoableBean 実装クラスに共通して実装すべき機能を実装したユーティリティクラスである。 Java 言語はクラスの多重継承を許さないため、スーパークラスを本来 Undo/Redo 機能を追加したい Bean とし、 UndoableBean としての機能をインタフェースで提供する。

UndoableBean インタフェースの主な機能は全てこのユーティリティクラスで実装し、このクラスのインスタンスを UndoableBean 実装オブジェクトに保持させて機能の転送を行う。

(3) プロパティアクセス

各プロパティアクセサをオーバーライドして、アクセス時に必ず外部にその旨を Undo/Redo 可能編集イベント (UndoableBeanPropertyChange /ReferEdit のカプセル化) の発火として通知する（図 7）。

Setter メソッドでは UndoableBeanPropertyChangeEdit、 Getter メソッドでは UndoableBeanPropertyReferEdit をそれぞれ用いる。

Edit オブジェクト中には、将来の Undo 操作に備えてプロパティの元の値が、 PropertyChangeEvent オブジェクトとして格納され、 UndoableEditEvent 通知の後、元の本来のメソッドを起動する。

プロパティアクセサには添字つきプロパティを扱うために index パラメータをとるものもあり、その場合は IndexedPropertyChangeEvent オブジェクトが用いられる。

(4) イベント発火

イベントの発火メソッド (通常 fireXXXEvent) をフックメソッドとしてオーバーライドし、自身からの発火メソッド呼出し時において必ず外部にその旨を Undo/Redo 可能編集イベント (UndoableBeanFireEventEdit のカプセル化) の発火として通知する (図 8)。

Edit オブジェクトには、発火メソッド名と発火パラメータが格納され、将来の Undo 操作によるイベント再発火に備える。

```
public void setText(java.lang.String text){
 if(!isUndoableEditInProgress()) {
  PropertyChangeEvent pce = new PropertyChangeEvent
              (this,''text'',super.getText(),text);
  UndoableBeanPropertyChangeEdit edit =
           new UndoableBeanPropertyChangeEdit(pce);
  undoableEditSupport.postEdit(edit);
 } else {
  setUndoableEditInProgress(false);
 }
 super.setText(text);
}
```

図 7: プロパティアクセサ (Setter メソッド) のオーバーライド

```
protected void fireActionPerformed() {
 if(!isUndoableEditInProgress()) {
  UndoableBeanFireEventEdit edit =
                  new UndoableBeanFireEventEdit(this);
  edit.setEventName(``action'');
  edit.setUserAction(true);
  edit.setFireEventMethodName(``fireActionPerformed'');
  Object[] params = {};
  edit.setParamObjects(params);
  undoableEditSupport.postEdit(edit);
 } else {
  setUndoableEditInProgress(false);
 }
 super.fireActionPerformed();
}
```

図 8: イベント発火メソッドのオーバーライド

7.2 編集クラス： UndoableBeanEdit

全ての編集クラス（Edit クラス）は、 UndoableBean Edit クラスを継承し、 UndoableBeanManager から統一的に扱われる。

全て共通して、 undo/redo メソッド起動時に Undo/Redo 動作するかどうかの設定フィールド recall と、ユーザアクションかどうかの設定フィールド userAction を持つ。

(1) UndoableBeanPropertyChangeEdit

プロパティのオーバーライドされた Setter メソッド（setXXX）内で生成される。内部にプロパティの元の値と生成元への参照を格納した PropertyChangeEvent オブジェクトを保持し、 UndoablePropertyDescriptor クラスを用いて発生元（UndoableFoo）の Undo 用 /Redo 用メソッドが取得される。

UndoableBeanManager からの undo/redo メソッド起動時に、取得済の Undo 用 /Redo 用メソッド

を用いて発生元に対して値の復元動作を行う (図 9)。

```
public void undo() throws CannotUndoException {
 super.undo();
 if(isRecall() && undoMethod != null) {
  synchronized(this) {
   ((UndoableBean) getSource()).
                    setUndoableEditInProgress(true);
   PropertyChangeEvent pce = getPropertyChangeEvent();
   try {
    if(isPropertyIndexed()) {
     Object[] oldValues = { new Integer(((Indexed
PropertyChangeEvent)pce).getIndex()),pce.getOldValue()};
     undoMethod.invoke(getSource(),oldValues);
    } else {
     Object[] oldValues = { pce.getOldValue() };
     undoMethod.invoke(getSource(),oldValues);
    }
   } catch(InvocationTargetException ite) {
    throw new RuntimeException();
   } catch(IllegalAccessException iae) {
    throw new RuntimeException();
   }
  }
 }
}
```

図 9: UndoableBeanPropertyChangeEdit の undo メソッド

(2) UndoableBeanPropertyReferEdit

オーバーライドされたプロパティの Getter メソッド（getXXX）内で生成される。UndoableBeanEditViewer（後述）のような動作履歴視覚化ツールにおいて、プロパティの取得動作を表示するために用いられ、実際の Undo/Redo 動作には関わらない。

(3) UndoableBeanFireEventEdit

オーバーライドされたイベントの発火メソッド内で生成される。生成時に、発火メソッド名と発火パラメータが格納される。

(4) 一括指定方法

区切りとして isUserAction フィールドを各 UndoableBeanEdit に用意することで、一連の動作としての複数の UndoableBeanEdit オブジェクトについて一括した Undo/Redo 操作（undoCompounds/ redoCompounds）を可能とし、同時に動作履歴の視覚化におけるより意味的に正しい表示を実現する。

図 10において、[1] および [4] が $userAction = true$ な Edit であるため、[1]+[2]+[3] と [4]+[5] がそれぞれ一連の動作として UndoCompounds/ RedoCompounds 操作の対象となる。

また、userAction フィールドの他に、生成された createTime を必ず各 UndoableBeanEdit オブジェクトに持たせ、連続した UndoableBeanEdit 間での createTime の差が、ある Interval 値内であれば一括して Undo/Redo するという方法も合わせて提供する。

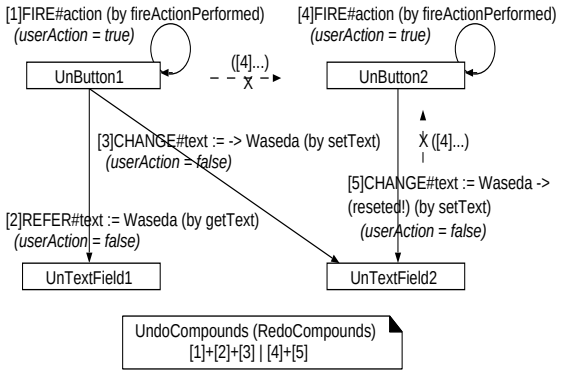

図 10: EditViewer における一括指定の認識

7.3 管理マネージャ：UndoableBeanManager

UndoableBeanManager は、UndoableBean 及び UndoableBeanEdit の管理を行う中核処理部である。主な役割を以下に挙げる。

(1) UndoableBean の管理

各 UndoableBean へ自身を UndoableEditListener として登録し、各 UndoableBean におけるプロパティの変更状況及びイベントの発火を見張る。その際に、Singleton パターン [3] の適用により、UndoableBeanManager はシステム内で唯一であることが保証されるため、UndoableBean 実装オブジェクトを宣言（new）するだけで、UndoableBeanManager との関連付けがなされる。

また、UndoableBeanManager が存在可能な個数を limit として設定できるため、将来システムにおいて、部分毎に UndoableBeanManger を配置するといった使用も可能である。

(2) UndoableBeanEdit の管理

自身に登録されている全ての UndoableBean 実装オブジェクトからのプロパティアクセス編集オブジェクト（UndoableBeanProperty[Change/Refer] Edit）及びイベント発火編集オブジェクト（UndoableBeanFire EventEdit）を UndoableBeanEdit イベントの通知という形で受け取る。

それらの Edit オブジェクトをリストに管理し、外部からの Undo/Redo 操作に応じて、Edit オブジェクトを取り出してその Edit オブジェクトへの Undo/Redo 操作を行う (図 11)。

(3) 動作履歴セットの複数管理・保存

プロパティアクセス及びイベント発火について蓄積される Edit オブジェクト群は、履歴セットとし

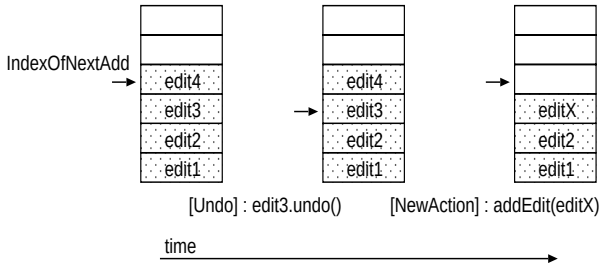

図 11: UndoableBeanManager による Edit 管理

てユニークな名前を設定し、 UndoableBeanManager 内部で複数の履歴セットを持つことを可能とする (図 12)。

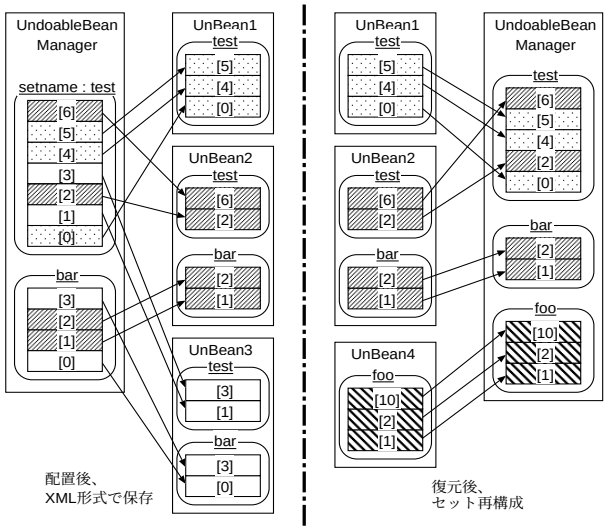

図 12: UndoableBeanManager による履歴セット管理

7.4 XML 形式での動作履歴保存

Edit オブジェクト群により構成される履歴セットの保存形式として、 Java 言語やコンポーネント技術との親和性と扱いの多様性から XML(eXtensible Markup Language) 形式を選択した。 UndoableBeanManager はユーザからの履歴セット保存命令に対して、 UndoableBeanEditXMLSupport クラスを用いて、各 UndoableBean 単位で XML 外部ファイルを生成する。

(1) UnBML(UndoableBeanEdit-MarkupLanguage)

UndoableBeanEdit/EditSet 専用の記述言語 UnBML を考案し、 XML パーサによる処理用に DTD(Document Type Definition) を策定した (図 13)。

(2) UndoableBeanXMLSupport クラス

インスタンスを生成しないユーティリティクラスであり、内部で XML パーサとして IBM 社の XML Parser for Java[10] を用いている。

外部から UnBML ファイルをパラメータとした静的メソッドの起動により、一時的に DOM(Document Object Model) ツリーを生成した後、デフォルトコンストラクタを用いた Edit オブジェクトの生成と各プロパティの再帰的な代入を行い、最後に UndoableBeanEdit の履歴セットを返す。

逆に Edit オブジェクトの入力により、各オブジェクトのプロパティを再帰的に調査し、 UnBML に従った DOM ツリーの生成・外部ファイルとしての出力を行う（図 14）。

```
<!ELEMENT UndoableBeanEditSets (SourceClassName,(Undoable
BeanEditSet*)>
<!ELEMENT SourceClassName (#PCDATA)>
<!ELEMENT UndoableBeanEditSet (UndoableBeanPropertyChange
Edit | UndoableBeanPropertyReferEdit | UndoableBeanFireEv
entEdit)*>
<!ELEMENT UndoableBeanPropertyChangeEdit (Recall,UserActi
on,CreateTime,(PropertyChangeEvent | IndexedPropertyChang
eEvent))>
<!ATTLIST UndoableBeanEditSet SETNAME CDATA #REQUIRED>
<!ATTLIST UndoableBeanPropertyChangeEdit INDEX CDATA #REQ
UIRED>
```

図 13: UnBML の DTD 抜粋

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE UndoableBeanEditSets SYSTEM "UnBML.dtd">
<!--This file uses UnBML(UndoableBeanEdit-MarkupLanguage)
1.06 format-->
<!--Contact author : washi@fuka.info.waseda.ac.jp-->
<UndoableBeanEditSets>
  <SourceClassName>jp.gr.java_conf.washi.undoablebean.be
ans.UndoableJTextField</SourceClassName>
  <UndoableBeanEditSet SETNAME="Thu Nov 04 03:41:49 JST">
    <UndoableBeanPropertyChangeEdit INDEX="10">
      <Recall>true</Recall>
      <UserAction>false</UserAction>
      <CreateTime>941654524674</CreateTime>
      <PropertyChangeEvent NAME="text">
        <NewValue>Waseda</NewValue>
        <OldValue>test</OldValue>
      </PropertyChangeEvent>
    </UndoableBeanPropertyChangeEdit>
```

図 14: UnBML に従った EditSet 保存ファイル例

(3) 履歴編集とアプリケーション挙動変更

UndoableBeanXMLSupport は内部において、入力・出力時に一度 DOM ツリーを経由した処理を行うため、 DOM ツリーに対して Edit オブジェクトの順序や値を外部から修正・変更することを可能とする。

また、 UnBML ファイルは単純なテキストファイルであるため、 UndoableBeanManager 以外の他のクラスや汎用 XML エディタやテキストエディタ等の各種ツールを用いて編集を行うことが可能である。

これらの履歴情報の編集作業は、 UndoableBeanManager を介してアプリケーション本体の動作につながるため、 Undo/Redo 機能に伴うアプリケーションの静的・動的な挙動変更をアプリケー

ション本体の修正を行わずに安全に実現し、高度なオートデモ環境やコンポーネントプレイヤ [11] の開発を容易なものとする。

7.5 イントロスペクション機構の拡張

JavaBeans により標準で提供されるイントロスペクション機構では、 Bean のプロパティ情報の取得について柔軟に拡張できないため、拡張可能なイントロスペクション機能を UndoableIntrospector クラスを中心に新たに提供している (図 15)。

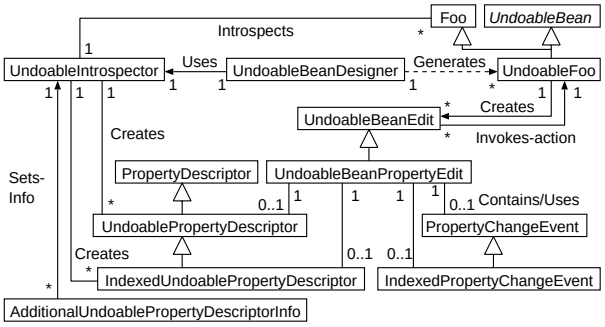

図 15: UndoableIntrospector クラス関連図

UndoableIntrospector は、 AdditionalUndoablePropertyDescriptorInfo オブジェクトを与えることで、プロパティ情報についての低レベルリフレクションの挙動を容易に変更可能である。例えば、 AddtionalUndoablePropertyDescriptorInfo(“uAdd”,”remove”, ”add”,0) と同 (“uRemove”,”add”, ”remove”,0) をペアで与えることで、 Bean#addElement(Object) や Bean#removeElement(Object) をプロパティアクセサとして追加的に認識し、 UndoablePropertyDescriptor から Undo /Redo 用のメソッドを取得可能となる。

8 開発・利用ツールの提供

UndoableBean をより効果的に利用するために、本研究ではいくつかのツールを提供している。

8.1 開発ツール： UndoableBeanDesigner

UndoableBeanDesigner は、 UndoableBean 実装クラスを生成する。

(1) UndoableBean 実装クラスソース自動生成

元の Bean に UndoableBean インタフェースを実装させたサブクラスとしての UndoableBean 実装クラスのソースコードを、プロパティ及びイベントのビジュアルな設定を元に自動生成する (図 16)。コンストラクタは全てオーバーライドして提供される。

(2) プロパティ、イベントの設定

ビジュアルにプロパティ・イベントについて、 Undo/Redo 操作の対象にするかどうか、および isUserAction、 isRecall の各フィールドの設定を可能とする。

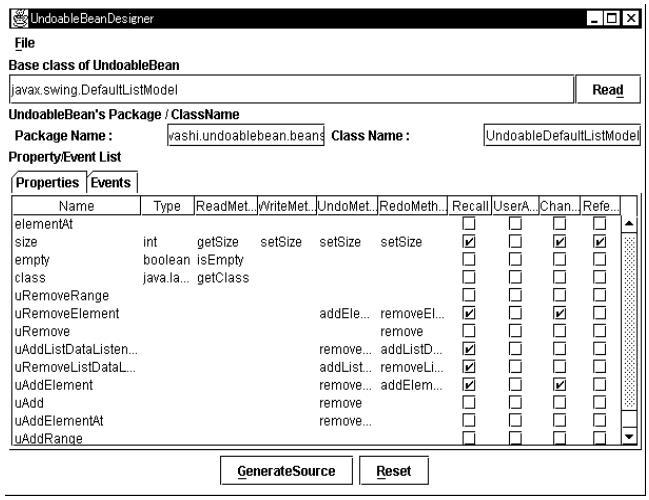

図 16: UndoableBeanDesigner の実行画面

(3) 設定の修正、追加

UndoableBean 実装クラスの生成は、元の Bean の各プロパティ・イベント発火について UndoableIntrospector クラスを用いたイントロスペクションを行い、開発者が Undo/Redo 機能実装の必要選択を自由に行うことができる。従って既 UndoableBean 実装クラスについて、仕様変更に伴う Undo/Redo 機能実装の修正・拡張を柔軟に行うことを可能とし、この修正は他に影響が及ばないため、OCP(Open-Closed-Principle[12]) に基づく修正箇所の局所化に成功している。

8.2 履歴利用ツール

UndoableBeanManager と協調動作する、履歴操作ツール及び履歴視覚化ツールを提供している。

各ツールの将来にわたる拡張性・再利用性を考慮した MVC パターンの適用により、各ツールの依存性は極めて低く、それぞれ独自に開発することを可能とする (図 17)。

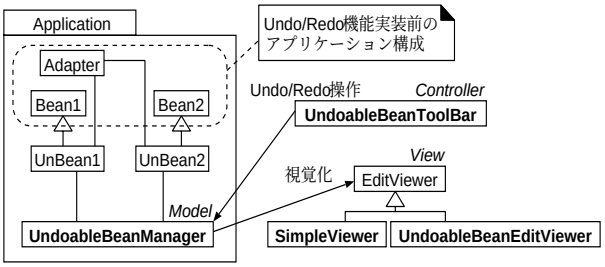

図 17: 各種ツールの関係

(1) 履歴操作ツール： UndoableBeanToolBar

直接 UndoableBeanManager を Undo/Redo 操作する実装も可能であるが、 UndoableBeanManager への操作は共通的であるため、履歴操作ユーティリティツールとして UndoableBeanToolBar を利用できる。

自動認識された UndoableBeanManager への、 Undo/Redo、 UndoCompounds/RedoCompounds、 UndoAll/RedoAll、及び履歴セットの選択・保存・復元の各操作インタフェースを提供する (図 18)。 javax.swing.JToolBar クラスを継承することで、あらゆる GUI アプリケーションに着脱可能であり、利便性が高い。

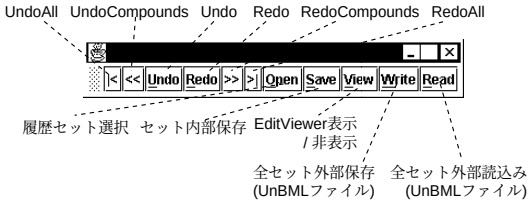

図 18: UndoableBeanToolBar の実行画面

(2) 履歴視覚化ツール： UndoableBeanEditViewer 他

ログを表示する機能のみを持つ SimpleViewer と、より高度な UndoableBeanEditViewer の両者を提供する。

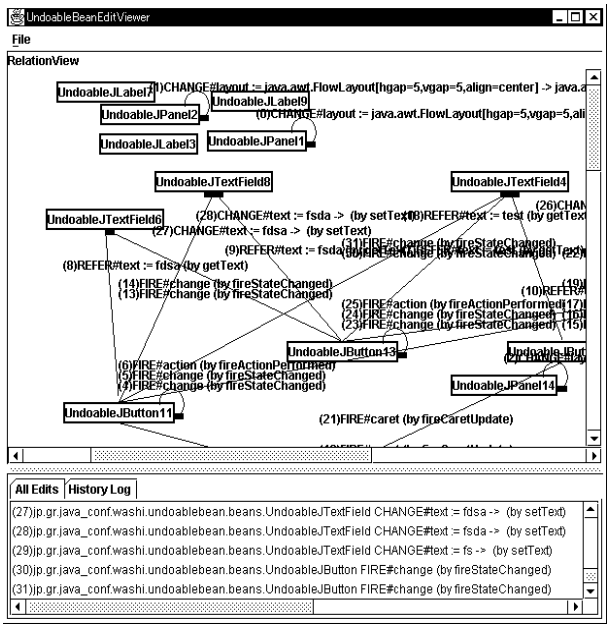

図 19: UndoableBeanEditViewer の実行画面

UndoableBeanEditViewer は、全ての UndoableBean とそれらの間のプロパティアクセス及びイベント発火をビジュアルに表示し、一般的な RAD ツール同様に、利用者がマウスを用いて各表示位置を変更して見やすくすることが可能であり、利用者に対するアプリケーション動作理解支援を実現する (図 19)。

UML 協調図 [13] により図 20のように表されるアプリケーション構成は、 Viewer により図 21のように表示される。各 UndoableBean 間の関係図示は各 Edit の発生順序に従うが、各 Edit クラスの userAction フィールド及び createTime を用いた一括指定の仕組みにより、アプリケーションの動作として意味的に正しい表示を行う。

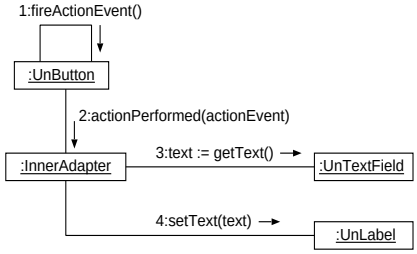

図 20: UML による協調図

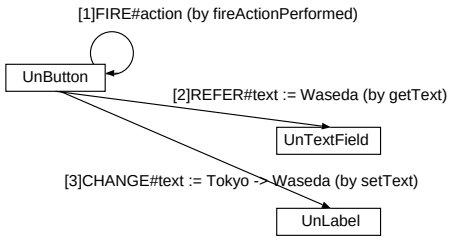

図 21: UndoableBeanEditViewer による表示

Java アプリケーションのクラス間通信の状況を実行時に視覚化する手法は、 BDK(BeanBox) への MethodTrace サービスの付加 [14] や Jinsight[15]、VisiComp Visualization Tool for Java[16] 等により実現されているが、 UndoableBean に基づく各種 EditViewer は、デバッガのように実行時に Undo/Redo 動作に伴うステップ実行状況を把握できる点において優位である。

9 アプリケーション開発における評価

9.1 開発コストの削減と再利用性の向上

アプリケーションの新規開発時における Undo/Redo 機能実装コストを、従来手法と UndoableBean を用いた手法で比較するために、文献 [17] におけるサンプルアプリケーションである LunchApplet(図 22) について、 UndoableBean による実装・再構成を行なった。

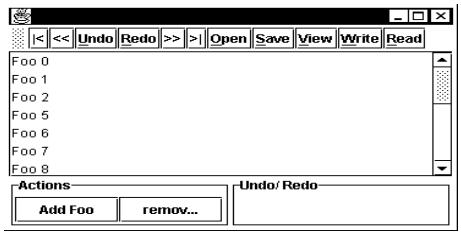

図 22: LunchApplet 実行画面

各手法において、アプリケーション全体について Undo/Redo 機能実現に必要なコード量の比較を表 2に示す。

表 2: Undo/Redo 機能実現コード量 (単位: ライン数)

構成クラス	従来手法	UndoableBean
LunchApplet	0	0
UndoPanel	24	10
AddAction	2	0
RemoveAction	2	0
UndoAction	7	—
RedoAction	7	—
UndoAdapter	7	—
AddEdit	33	—
RemoveEdit	33	—
ListSelection-		
-Adapter	0	0
UpdateAdapter	—	17
UndoableDefault-		α
-ListModel	—	(72 ～ 102)
(UndoableJButton)		α
	—	(112 ～ 152)
総実現コード量	124	27+2 α
		(211 ～ 281)
総コード量	381	479 ～ 549

(α は UndoableBeanDesigner によるビジュアルな設定コスト, α ≅ 0)

(1) 実作業の大幅な削減

表 2において、 Undo/Redo 機能の実現にかかるコストは、単純に実現のためのコード量を比較して 124 : 27(27+2 α) となり、圧倒的に UndoableBean を用いる手法が優れることが分かる。

これは、従来手法で作成する必要のあったいくつかのクラス (Add/RemoveEdit 他) を、 UndoableBean を用いた手法においては全く作成する必要のないことが大きな開発コストダウンにつながっている。

α は UndoableBeanDesigner による UndoableBean 生成時の設定コストであり、全ての設定作業をビジュアルに行うことが可能なため、ほぼ 0 に等しいと言える。

最終的なコード量に関しては、従来手法が少ないが (124 < 211)、各 UndoableBean 実装クラスのコードは UndoableBeanDesigner により自動生成されるため、開発コストの対象とはならない。

(2) 保守コストの削減

従来手法による実装作業はハンドコーディングを主体とするため、開発者による誤りの可能性が存在し、より大規模な開発において誤りの修正・デバッグコストは莫大なものとなる。対して UndoableBean を用いれば、 UndoableBean 実装クラスのソースコードは自動生成されるため誤りはありえず、将来の保守過程においてもコスト的に優れている。

(3) 高い再利用性

一度 Undo/Redo 機能を実装した UndoableBean 実装オブジェクトは、あらゆるオブジェクト環境下において、独立したコンポーネント（Bean）として再利用可能である。

また UndoableBeanDesigner による UndoableBean 実装オブジェクトクラスの生成は、元の Bean の各プロパティ・イベント発火について、開発者が Undo/Redo 機能実装の必要選択を自由に行うことができる。

従って、既 UndoableBean 実装クラスについて、仕様変更に伴う Undo/Redo 機能実装の修正・拡張を柔軟に行うことを可能とし、高い再利用性を実現している。

また、この再利用性の向上は、機能実現対象とするアプリケーション本体についてもいえる。 UndoableBean を用いた Undo/Redo 機能の実現は、アプリケーション中の Bean 定義を UndoableBean に差し換えるだけでほとんどの修正作業が済むため、機能実現後に仕様変更や動作が重い等の利用から Undo/Redo 機能が不要になるとき、 Undo/Redo 機能実現済みアプリケーションを用意に元の状態に戻すことが可能である。

9.2 実行効率の検証

UndoableBean において、再利用性向上と開発コスト削減の代償としての実行効率の低下が、従来手法に比べてどの程度のものかを検証するため、各手法によ

る LunchApplet について、ユーザ入力・ Undo 操作・ Redo 操作という一連の動作の自動繰り返しテストを行い、実行時間を計測し表 3の結果を得た。

表 3: 自動繰り返しテストの実行時間評価 (単位:ms)

回数 (パターン)	従来手法	UndoableBean (non-Viewer)	UndoableBean (with-Viewer)
100 (1)	17,395	17,054	31,124
100 (2)	44,113	42,401	68,148
1000 (1)	164,677	162,984	372,916
1000 (2)	489,995	465,790	1,628,562

パターン (1):$[Action] + ([Undo] + [Redo]) * n$
パターン (2):$([Action] * 2 + [Undo] + [Redo]) * n$
計測環境: Java2SEv1.3, PentiumIII 500MHz, Memory 256M, WindowsNT4.0 Server

UndoableBeanEditViewer を用いない純粋な Undo/Redo 機能の実現に関しては、 UndoableBean を用いた手法がわずかながら従来手法より効率が良く、 Viewer を用いても通常の使用における許容範囲内であると言える。

特にパターン (2) は Edit オブジェクトが逐次増加するため、長時間の実行に従い実行環境のメモリ領域を圧迫するが、 UndoableBean を用いた手法においても問題なく安定動作した。

10 おわりに

コンポーネント技術に基づく Undo/Redo 機能実装を提案・実装し、その有用性を示した。 UndoableBean は現時点で、アプリケーション動作理解支援や、オートマクロ・オートデモ機能拡張といった用途へ応用が可能であり、今後さらに基本フレームワークの洗練と各種ツール群の拡充を目指す。本論文で述べた UndoableBean フレームワーク及び関連ツールは、http://www.fuka.info.waseda.ac.jp/~washi/undobean/ からダウンロード可能である。

UndoableBeanManager は、実行時に複数の履歴セットを保持可能であるため、長時間のアプリケーション利用に関し、実行環境のメモリ領域を圧迫する恐れがあり、今後、古く利用頻度の低い Edit オブジェクト・履歴セットを、一時的に外部ファイルに書き出す機能を追加することで解決する。

参考文献

[1] A.Cooper, テクニカルコア訳, “ユーザインタフェースデザイン” , 翔泳社, 1996

[2] G.McCluskey, “Undoing Text Edits”, JDC Tech Tips, Sun Microsystems , July 14, 1999

[3] E.Gamma, R.Helm, R.Johnson, J.Vlissides , 本位田真一, 吉田和樹監訳, ” デザインパターン”, ソフトバンク, 1995

[4] F.Buschmann, R.Meunier, H.Rohnert, P.Sommerlad, M.Stal , 金澤典子, 水野貴之, 桜井麻里, 関富登志, 千葉寛之訳, “ソフトウェアアーキテクチャ”, トッパン, 1999

[5] G.Hamilton, ”JavaBeans 1.01 Specification” , Sun Microsystems, 1997

[6] 青山幹雄, 中所武司, 向山博編, ” コンポーネントウェア” , 共立出版, 1998

[7] Q.Ning, ”A Component-Based Software Development Model” , Proc. of IEEE COMPSAC ’ 96, August 1996

[8] A.W.Brown, K.C.Wallnau, ”Engineering of Component-Based Systems” , Proc. of the 2nd IEEE International Conference on Engineering of Complex Systems, 1996

[9] R.Englander, 鷲見豊訳, ”JavaBeans 基礎から開発まで” , オライリージャパン, 1997

[10] IBM Corp., “XML Parser for Java”, http://www.alphaworks.ibm.com/formula/xml

[11] 斎木太郎, 村松亮智, 長田洪司, 青山幹雄, “コンポーネントプレイヤの試作について” , 情報処理学会第 58 回全国大会, 1999

[12] B.Meyer, ”Object-Oriented Software Construction” , Prentice Hall, 1997

[13] C.Larman, 今野 睦, 依田 智夫監訳, 依田 光江訳, ” 実践 UML” , ピアソンエデュケーション, 1998

[14] O.Kluyt, “JavaBeans Technology: Unlocking The BeanContext API”, JDC Articles, Sun Microsystems, 1999

[15] IBM Corp., “Jinsight”, http://www.alphaworks.ibm.com/formula/jinsight

[16] VisiComp Inc., “VisiComp Visualization Tool for Java”, http://www.visicomp.com/

[17] T.Meshorer, ”Add an undo/redo function to your Java apps with Swing” , JavaWorld, June, 1998