細粒度コンポーネント環境における動的かつ階層的 Undo 機構の実現

鷲崎弘宜† 深澤良彰†

GUIを基本とした対話型アプリケーションシステムにおいて，ユーザが行った操作の取消し/再実行を実現するUndo機構は必要不可欠である．従来のオブジェクト指向開発におけるUndo機構の実装は，Undoフレームワークを開発時に利用することでなされてきたが，開発・保守コストが高く迅速な実装を行うのが困難であった．本論文では，細粒度なコンポーネント群によって構成されるすでに開発されたアプリケーションに対して，コンポーネントが持つ状態値の変化を中心としてUndo機構を実装し，開発・保守コストの大幅な低減を実現する手法を提案する．細粒度コンポーネントに着目した本手法では，アプリケーションに対してユーザ操作によってもたらされる影響が複数の履歴群によって表現されるので，ユーザが想定する1つの取消し/再実行を形成する履歴群のまとまりを適宜作成する必要がある．そこで，履歴リスト中の履歴群を実行時に複合化することで，利便性の高い適切な大きさの履歴群に再構成する履歴階層化手法を提案し，実装とともにその有用性を示す．

Dynamic Hierarchical Undo Facility in a Fine-grained Component Environment

Hironori Washizaki† and Yoshiaki Fukazawa†

The undo facility is essential for interactive application systems. In conventional object-oriented software development, undo facilities have been implemented based on undo frameworks. However, the use of undo frameworks costs a great deal in both the development and maintenance stages. In this paper, we propose a new technique by which an undo facility can be easily implemented in component-based applications using changes of the component properties. Since the granularity of the actions corresponding to the change of the properties is small, this technique requires the user to perform more undo/redo operations than conventional techniques would. Thus, we also propose a technique for organizing the commands, which represent the real actions internally, into a hierarchy dynamically in order to compose commands with the appropriate granularity for users.

1. はじめに

GUIを基本とした対話型アプリケーションシステムの普及によって，ユーザに要求される操作は複雑化しつつあり，ユーザが行った操作の取消し（Undo）/再実行（Redo）を実現するUndo機構は，その重要性を増している．Undo機構は一般に，アプリケーションの実行中に発生したユーザ操作に対応する履歴の集合から構成される履歴リストによって実現される[1)]．

従来のオブジェクト指向開発において，Undo機構はオブジェクト指向Undoフレームワークを利用することで実装されてきた．しかしながら，Undoフレームワークの利用は開発時と保守時におけるコストの高さを招く．我々は，複数のコンポーネントから構成されるアプリケーションに対して，細粒度なコンポーネントの状態値変化と実行時の履歴階層化処理によって，Undo機構を容易に実装する手法を提案し，開発・保守コストの大幅な削減を実現する．

† 早稲田大学理工学部
School of Science and Engineering, Waseda University

2. Undo機構とその実装手法

2.1 Undo機構の種類

以下の3つを満す機能をUndo/Redo機能と呼ぶ[2)]．

- 直近のユーザ操作を取消し可能である．
- 直近のユーザ操作を再実行可能である．
- 複数の直近ユーザ操作をまとめて取消し/再実行可能である．

これらの機能を実現するために，Undo機構はユーザ操作を履歴オブジェクト（以下，履歴と呼ぶ）として記録する．実行されたユーザ操作を表す履歴群は時系列に沿って履歴リストに格納される．

既存のUndo機構は実装上の仕組みから，制限線形

的 Undo 機構・非線形的 Undo 機構・選択的 Undo 機構・階層的 Undo 機構に分類される．

制限線形的 Undo 機構（単純履歴 Undo 機構）は，利用者が行った操作を順に線形の履歴リストに保存し，操作の順に従った Undo/Redo 操作を利用者に認める機構である[1]．希望する状態に復帰するために，利用者は直前・直後の Undo/Redo 操作の連続実行を強いられるが，実装が容易であり，また利用者にとって直感的に分かりやすいという利点を持つ[3]．履歴リスト上のある時点にまで戻った段階で，新しいユーザ操作が起こったときに，以降の Redo 操作用の履歴は削除される．

非線形的 Undo 機構は，履歴リストに保存された履歴群のうちで，取消しを望まない履歴をとばして任意の履歴を Undo 操作可能にする機構である[4]．とばされた履歴は将来の Redo 操作のために履歴リスト中に存在し続けるので，履歴リスト中の履歴群が表すアプリケーション状態の関係が複雑となり，利用者が把握困難となることから利便性に優れない[3]．

選択的 Undo 機構は，実行された順序にとらわれずに任意の履歴を Undo/Redo 操作可能にする機構である[5]．同機構は，選択的な Undo/Redo 操作によって，ユーザ操作時には存在しなかった新しいアプリケーションの状態を作り出すことができる．しかし，実行時にどの履歴を選択すべきかを利用者に対して適切に表示する機構がないので，利便性に優れない[3]．各アプリケーション状態ごとの画面出力を蓄積した履歴ブラウザ[6]を用いて選択の支援を行う試みもあるが，ユーザ操作の影響が画面出力上に反映されない場合に有効ではない．

階層的 Undo 機構は，線形の履歴リストを用いるが，実行時に階層化可能な履歴を用いて，低い階層もしくは高い階層での Undo/Redo 操作を認める機構である[7]．ユーザ操作に対応した階層化可能な履歴クラスの準備が必要であり，実装コストが大きい．一方，共通に利用可能な定義済みの履歴クラスを組み合わせることで，ユーザ操作に対応した履歴クラスの新規開発を避け，階層的 Undo 機構を実現する試みに JCommands[8]がある．しかしながら JCommands では，既存のアプリケーションへ Undo 機構を実装する場合に，履歴クラスを利用するための追加プログラムコードを元のアプリケーションを修正して埋め込む必要があり，保守性の低下を招く．

2.2 Undo フレームワーク

多くのオブジェクト指向アプリケーションフレームワークは，制限線形的 Undo 機構を実現する Undo フレームワークを提供する．歴史的に，Undo フレームワークは，Command Processor パターン[9]を基本設計指針とする．Command Processor パターンは，要求に対する応答とその実行を分離し，実行の管理や取消しなどの実現を目的とするパターンである．MacApp や ET++などのアプリケーションフレームワークは，Command Processor パターンの適用された Undo フレームワークを提供する．

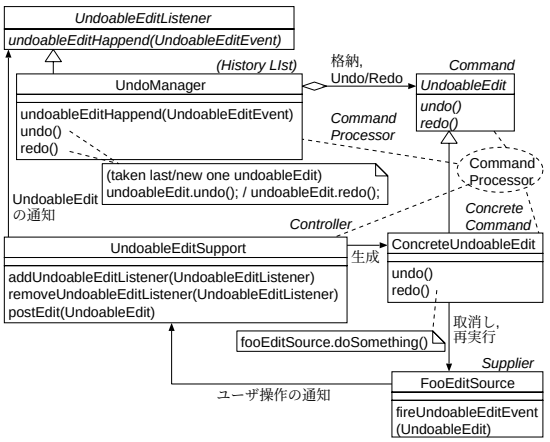

図 1 javax.swing.undo パッケージ主要構成関連
Fig. 1 Class diagram of javax.swing.undo.

Java 言語による開発では Undo フレームワークとして，javax.swing.undo パッケージ（以下，undo パッケージと呼ぶ）[2]が利用可能である．undo パッケージにおける主要構成クラスを，**図 1** に示す．図 1 において，対象とするアプリケーション中でユーザ操作から影響を受ける部分を FooEditSource クラス（編集元クラスと呼ぶ）とする．編集元クラスへのユーザ操作の影響は，その影響の取消し/再実行機能を持つ Undo/Redo メソッドを提供する履歴クラス UndoableEdit に格納される．開発者は，影響の取消し/再実行を実現したいすべてのユーザ操作について，それぞれ対応する履歴クラスのサブクラス ConcreteUndoableEdit を定義する必要がある．実行時に履歴オブジェクト群は，履歴リストを持つ履歴マネージャ UndoManager によって管理される．

2.3 実装上の問題

Undo フレームワークを用いた従来の Undo 機構実装手法は，開発・保守過程において以下の問題を生じる．

- 履歴クラス定義の増大

対象とするアプリケーションが豊富な機能を持つ場合に，ユーザ操作に対応する履歴クラスを数多く定義する必要がある[9]．たとえば，Undo 機構を実装する対

象がテキスト編集アプリケーションであるとき，開発者はコピーや貼付け，削除といった各ユーザ操作に応じてそれぞれ履歴クラスを定義する必要がある．

- 編集元クラスの内部修正

ユーザ操作により影響を受ける編集元クラスについて，影響を受け将来の Undo 操作時に復元すべきデータ群を特定し，ユーザ操作時にそのデータ群の状態を適切に履歴リストを管理する履歴マネージャへ通知するための処理手続きを追加する必要がある．そのため，編集元クラス内部に，編集元クラスが本来持つアプリケーションロジック部分と，Undo フレームワーク対応部分が混在することとなり，保守過程においてどちらかの仕様変更時に修正箇所が局所化されない．

3. 細粒度コンポーネント環境

コンポーネントは，ある機能を提供する独立して交換可能なソフトウェア構成単位である．狭義には，内部構造が公開されず利用時のインタフェースが公開され，オブジェクトコードの配布を基本とするソフトウェア部品を指す．本論文ではこの狭義のコンポーネントを対象とする．コンポーネントの実装はオブジェクト指向言語で行うことが自然であり[10]，本論文では実装をオブジェクト指向言語に限定する．

コンポーネント間の制御方針には，メソッドの直接駆動にあたる能動的制御と，Multicast パターン[11]に基づくイベントを起点とした受動的制御がある[12]．コンポーネントの粒度を，コンポーネントが提供する機能の概念的な大きさと定義し，制御方針と関連して粗粒度・中粒度・細粒度の 3 種に分類できる．

(1) 粗粒度

ビジネスロジックがカプセル化されたコンポーネントであり，能動的制御を基本とする．実現可能システムとして，CORBA や Enterprise JavaBeans がある[13]．

(2) 中粒度

細粒度なコンポーネントを組み合わせ，アプリケーションロジックを付加し，特定のドメインで再利用することを目的とした複合化コンポーネントが中粒度である．

(3) 細粒度

RAD ツールなど付属のコンポーネントライブラリにより提供され，GUI の最小構成要素となるウィジェットや，汎用的なデータベースアクセス機能などの最小のロジックを持ったコンポーネントが細粒度であり，受動的制御を基本とする．実現可能システムとして，ActiveX/COM や JavaBeans などがある[14]．RAD ツールの成功にともない，細粒度コンポーネントが現状として最も普及している．細粒度コンポーネントが持つ要素を以下にあげる．

- 状態（プロパティ）：外部から値を観測可能な属性
- 観測操作：状態の値を外部から観測する操作メソッド
- 設定操作：状態の値を新たに設定する操作メソッド
- イベント発火操作：受動的制御のきっかけ

コンポーネントが持つ属性の観測可能性の高さは，コンポーネントの可試験性の高さに直結する[15]．そこで一般にコンポーネントは，持つ属性の多くを状態として観測可能なものとして公開するように実装される[15]．一方，コンポーネントが公開する状態のうちで，組立て段階および実行時に設定可能とする必要があると判断されるものについて，設定操作が実装される[12]．

JavaBeans はコンポーネントを特に状態について統一的に扱うための命名規則と内観機構（イントロスペクション機構）を提供する．たとえば “setXX”（“getXX”）という名のメソッドは，内観機構により状態 “XX” に対応した設定操作（観測操作）として認識される[14]．

4. Undo 機構追加実装

我々は，利用者への分かりやすさから制限線形的 Undo 機構を基礎とし，細粒度コンポーネント環境の特徴を利用した Undo 機構実装手法を提案する．

4.1 コンポーネントの状態値変化

ユーザ操作によるアプリケーションへの影響は，アプリケーションを構成する細粒度コンポーネントが持つ属性群の値の変化として表現される．細粒度コンポーネントの性質から，コンポーネントが持つ属性の多くは一般に観測可能かつ設定可能として設計されるので，ユーザ操作にともない影響を受ける部分を，観測可能かつ設定可能な状態で網羅できる．

このとき，状態の値を設定操作によって変更する際に，新たに設定する状態の値を取得し，同時に，変更する前の状態の値を観測操作によって取得できる．状態の値が変化する際に変化する前後での状態の値を取得し，時系列に沿って保存しておけば，その後，設定操作を用いて変化する前の値を設定することで，コンポーネントが持つ状態の値を以前のものに復元できる．逆に，復元した後で再度，設定操作を用いて変化した後の値を設定することで，コンポーネントが持つ状態の値を復元以前のものに再設定できる．このような動作をコンポーネントの状態値変換に関する復元/再設

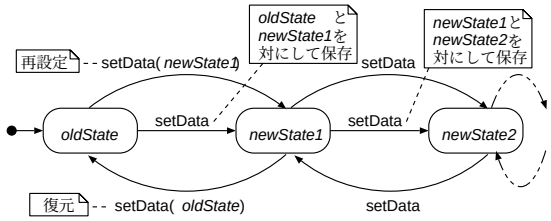

図 2 コンポーネントの状態値変化
Fig. 2 Change of component's state value.

定動作と呼ぶ.

復元/再設定動作の仕組みを**図 2** に示す．図 2 の例では，あるコンポーネントが data という名前の状態を持つ．コンポーネントの外部またはコンポーネント自身によって状態 data の値を変更する設定操作 setData が呼ばれたときに，元の値 *oldState* と新しい値 *newState*1 を対にして保存する．その後，Undo 操作が行われたときは，元の値 *oldState* を引数として設定操作 setState を呼び出すことで，状態 data の値を復元する．Undo 操作後に Redo 操作が行われたときは，元の値 *newState*1 を引数として設定操作 setData を呼び出すことで，状態 data の値を再設定する．

状態値を保存するにあたり，状態の型が int や boolean などの単純な値をとるプリミティブ型である場合は，型が表す値が保存対象となる．状態の型がプリミティブ型以外の一般的なオブジェクト型（Object など）である場合は，状態の値はあるオブジェクトの実体への参照として表現されるので，参照が保存対象となる．

4.2 Undoable 拡張と原子履歴

コンポーネントに対して自身の状態値変化を外部に伝える機能を持たせるために，コンポーネントを継承したサブクラスを作成し，元のコンポーネントの設定操作をすべて再定義（オーバライド）する．再定義後の設定操作には，基礎として用いる Undo フレームワークに対応して，状態値変化履歴オブジェクトを外部に通知する機能を追加する．状態値変化履歴オブジェクトには，設定操作の起動時に，状態の設定前の状態値および設定後の状態値の対を格納する．状態値変化履歴オブジェクトの Undo/Redo メソッドを実行することで，対応するコンポーネントの状態値の復元/再設定を実現する．

再定義された設定操作は，履歴オブジェクトの通知後に継承元のコンポーネントの設定操作を起動することで，拡張前後における振舞いを同一に保つ．設定操作の構造（シグネチャ）を拡張前後で同一に保つことで，拡張前のコンポーネントを利用しているアプリケーション中で，拡張済のものへ置き換えることを容易とする．設定前の状態値は，対象とする状態の観測操作の自動実行により得られる．この機能追加を行う拡張を Undoable 拡張と呼ぶ.

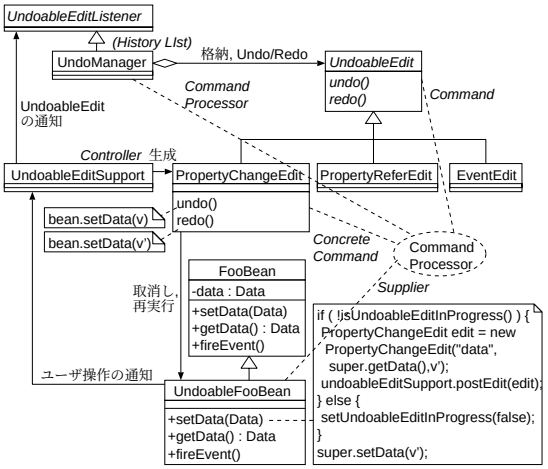

図 3 コンポーネントのメソッド再定義
Fig. 3 Overriding of components' methods.

undo パッケージを用いて Undoable 拡張を行う際の，メソッド再定義の例を**図 3** に示す．図 3 では，元のコンポーネント FooBean を Undoable 拡張したものが UndoableFooBean である．復元/再設定の対象を状態値変化に限定するので，履歴クラスは状態値変化履歴クラス（PropertyChangeEdit）が共通に用いられる．Undoable 拡張クラスにおける再定義された設定操作（setData）には，状態（data）の変化前の値と変化後の値を状態値変化履歴オブジェクトに格納する処理が追加される．

イベント発火操作の起動は，直接的には復元/再設定の対象とならないが，受動的制御のきっかけを外部から把握するために，イベント発火操作の再定義によってイベント履歴（EventEdit）の通知機能を追加する．同様に，観測操作の起動は直接的には復元/再設定の対象とならないが，実行時の挙動の把握に重要であるので，観測操作の再定義によって観測履歴（PropertyReferEdit）の通知機能を追加する．履歴を状態値変化・状態値観測・イベント発火の 3 種に限定することで，Undoable 拡張時に操作ごとに再定義する際の必要なプログラム記述が定型的なものとなる．これにより，Undoable 拡張にかかる作業を自動化し，Undoable 拡張クラスを自動生成することが可能となる．3 種の履歴を原子履歴と呼ぶ.

ここで，Undoable 拡張によって対象コンポーネントが，復元/再設定可能となるには，以下の制限を満

たす必要がある．

（制限 1）状態の網羅性についての制限

ユーザ操作によって値が変化する属性はすべて，コンポーネントの設定可能かつ観測可能な状態として網羅されている必要がある．このとき，コンポーネントの内部でオブジェクトが生成された後に状態の値として設定して用いる場合には，null 代入などによって破棄が（暗に）明示されたとしても，履歴マネージャからの参照が維持されるので，その状態の値について，同じオブジェクトを用いた復元/再設定動作が可能となる．その場合において，状態の値が示すオブジェクトの生成の取消しは，null を設定することで実現する．

4.3 アプリケーション全体の Undo/Redo 動作

アプリケーションの本質的な挙動と状態の保持を，アプリケーションを構成する細粒度なコンポーネント群が網羅するとき，アプリケーション全体の振舞いは，構成する各コンポーネントの状態値変化の集合として表される．そこで，アプリケーションに対してあるユーザ操作が行われたときに，ユーザ操作によって引き起こされたすべてのコンポーネントの状態値変化を，コンポーネントごとに変化した順に復元/再設定することで，元のユーザ操作の取消し /再実行（Undo/Redo 動作）を実現可能となる．

ただし，ユーザ操作が複数回行われていくと，以前のユーザ操作がもたらした状態値変化が，その集合の中でどこからどこまでかが判別困難となる．そこで，ユーザ操作にともなうアプリケーションの振舞いが，イベントを起点とした受動的制御に基づくものであるとき，状態値変化の集合における範囲を特定するための区切りとして，イベント発火を用いることとする．

まず，アプリケーションを構成するコンポーネント群のうちで，ユーザ操作により影響をうけるすべてのコンポーネントを，Undoable 拡張して元のコンポーネントと置き換える．同一の実行環境上のすべての Undoable 拡張されたコンポーネントの状態値変化・状態値観測・イベント発火は，Singleton パターン[16]に従って実行環境上で実体が唯一である履歴マネージャによって原子履歴として捕捉され，時系列に沿って履歴マネージャが持つ履歴リストに保存される．この仕組みを**図 4** に示す．図 4 の例では，実行環境上に Undoable 拡張された 2 つのコンポーネントが存在し，それぞれが通知する履歴は時系列に沿って，1 つの履歴マネージャによって管理される．

続いて，Undoable 拡張されたコンポーネントから履歴マネージャに通知される原子履歴群より，それぞれ実行されたユーザ操作に対応する履歴のまとまりを

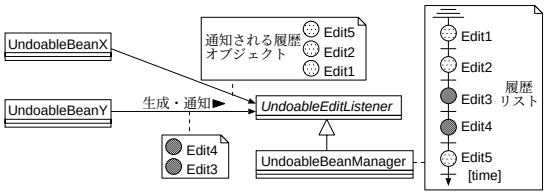

図 4 履歴リストの仕組み
Fig. 4 Mechanism of the History List.

特定するために，履歴マネージャにおいて原子履歴が通知されるつど，以下の基本階層化処理を行う．履歴リスト中で，イベント発火操作にともなうイベント履歴から次のイベント履歴までの間の原子履歴群は，イベントにともなう 1 つの振舞いを表すものとして 1 つの履歴に複合化することができる．この複合化は，イベント履歴に対して，次のイベント履歴までの間の原子履歴群を子として追加することで実現する．複合化の結果，イベント履歴を頂点とする階層化ツリーが得られる．この階層化処理を基本階層化と呼ぶ．履歴リスト中の原子履歴群の基本階層化により，イベントを起点とした 1 つの振舞いを，1 度の Undo/Redo 操作によって取消し /再実行可能となる．

ここで，アプリケーション全体が Undo/Redo 動作を実現するには，以下の制限を満たす必要がある．

（制限 2）アプリケーションの振舞いについての制限

ユーザ操作にともなってなされる将来 Undo/Redo 動作の対象とすべきアプリケーションのすべての振舞いは，コンポーネントのイベント発火をきっかけとして，コンポーネント群の設定操作の実行に基づく状態値の変化によって表現される必要がある．

（制限 3）並行性についての制限

基本階層化が正しく実行されない可能性があるので，アプリケーションを構成するコンポーネント群の状態値の変更操作は，ユーザ操作にともなう GUI イベントを扱う特別なスレッド（例：JavaBeans に対応した GUI フレームワークである Swing における実行系で唯一なイベントディスパッチスレッド[17]）を除いて，並行的に実行されない必要がある．

4.4 Undo/Redo 動作例

アプリケーションの Undo/Redo 動作の例として，文献 18) におけるサンプルに対していくつかのユーザ操作（サイズ変更・スタイル変更・フォント名変更）を Undo/Redo 操作の対象とする機能拡張を行った FontViewer を用いる．機能拡張された FontViewer は（制限 1)〜(制限 3）をすべて満たす．FontViewer を構成するコンポーネントのうちで Undo/Redo 操作の対象とすべきユーザ操作の影響を直接受けるコ

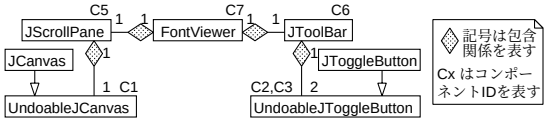

図 5 コンポーネント群の包含関係
Fig. 5 Components' relation of inclusion.

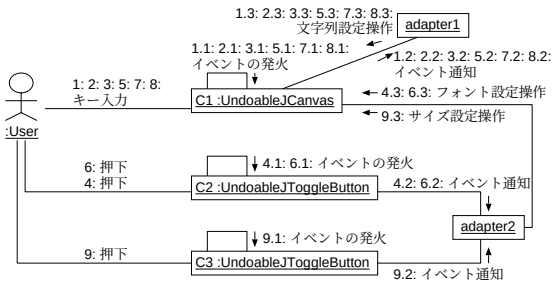

図 6 利用者の一連の操作
Fig. 6 Sequence of user's actions.

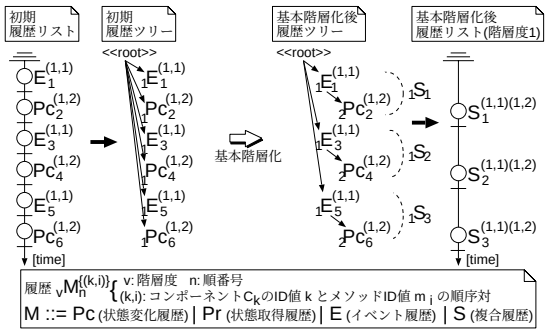

図 7 基本階層化適用例
Fig. 7 Example of basic hierarchization.

ンポーネント 2 種を特定し，そのコンポーネントを Undoable 拡張して Undo 機構を実装した．実装後のコンポーネント・クラス群の包含関係を**図 5** に示す．

図 6 に示す一連のユーザ操作のうちで，1：キー入力〜3：キー入力の一連の操作がユーザによって行われたときの基本階層化の仕組みを**図 7** に示す．我々の Undo 機構は，実行時にすべてのコンポーネントと，すべてのコンポーネントのメソッドにそれぞれ一意で固定的な ID 値を割り当てる．すべての原子履歴は，自身の生成元コンポーネントのコンポーネント ID 値と，自身の生成時に起動されたコンポーネントのメソッド ID 値の順序対を持つ．たとえば図 6 において，コンポーネント ID が C_1 である UndoableJCanvas は，持つメソッドのうちでイベント発火操作と文字列設定操作のメソッド ID がそれぞれ m_1，m_2 であり，各メソッドの起動時においてそれぞれ順序対 $(1,1)$，$(1,2)$ を持った履歴を履歴リストに通知している（図 7）．

履歴リスト中に原子履歴 E_1〜Pc_6 が存在する．次に，E_1，E_3，E_5 がイベント履歴なので，それらの間の原子履歴が直前のイベント履歴に追加され，その結果階層化された履歴ツリーが得られる．最後に，履歴ツリー上において階層度 1 を選択するとき，3 つの複合履歴 S_1，S_2，S_3 を持った履歴リストが得られる．得られた複合履歴を単位として Undo/Redo 動作を行うことで，自然なユーザ操作の取消し /再実行を実現可能となる．

5. 高度な履歴階層化

従来の Undo 機構実装手法では，複数のユーザ操作の連続によって 1 つの意味のあるアプリケーションの振舞いを実現するとき，ユーザにとっての利便性向上を目的として，複数のユーザ操作をまとめて取消し /再実行の対象とするように履歴クラスを定義し利用することができる．したがって，基本階層化の結果得られる複合履歴群は，従来手法における履歴群と比べて，取消し /再実行の対象とするユーザ操作の規模が小さく，履歴として原子的である可能性がある．

たとえば，文字をキーボードから入力可能なアプリケーションがあるとき，ユーザにとって最適な Undo/Redo 動作の対象として，1 文字の入力動作を対象とする状況と，連続した入力動作を一括して対象とする状況の 2 種類が考えられる．本論文ではどちらが利便性に優れるかということを議論しないが，アプリケーション開発者は対象アプリケーションについて Undo 機構を実装する際に，どちらの状況についても容易に対処できる必要がある．

そこで，我々は 2 つの高度な履歴階層化であるパターン階層化・コンテナ階層化を提案する．基本階層化後の複合履歴群に対して高度な履歴階層化を行うことによって，規模の大きい履歴を内部で機械的に形成し，基本階層化によって得られる 1 つの複合履歴が対応するユーザ操作を，複数回分 1 度にまとめて取消し /再実行可能となる．以降において，原子履歴を用いた Undo/Redo 操作を原子的 Undo/Redo 操作と呼ぶ．また，階層化の行われた履歴を用いた Undo/Redo 操作を階層化 Undo/Redo 操作と呼ぶ．

ただし，高度な履歴階層化を行うと，パターン階層化またはコンテナ階層化 Undo/Redo 操作のみを使用して，希望するアプリケーション状態へ復帰することは必ずしも保証されない．しかしながら，そのような“速い”Undo/Redo 操作と基本階層化 Undo/Redo 操作（“遅い”Undo/Redo 操作）を併用することで，希望するアプリケーション状態へ復帰できる．

5.1 パターン階層化

繰り返し生成される同種の履歴群は，ユーザにとって意味のある動作を表す適切な大きさのまとまりの候補である．そこで，対象アプリケーションの実行中に，繰り返される同種の履歴群を 1 つのまとまりとして認識することで，アプリケーション開発者が特別に指定することなく機械的に，ユーザにとって自然な履歴のまとまりを形成可能となり，後の Undo/Redo 操作において，パターンとして認識された履歴群を一括して Undo/Redo 動作の対象とすることができる．

基本階層化の結果得られる履歴リスト中で，複数の連続した複合履歴群について，コンポーネント ID 値とメソッド ID 値の順序対の連続が一致するため同種と判断される場合は，その一連の複合履歴群を頻出するパターンとして新たな 1 つの履歴に複合化することができる．この複合化は，新しい複合履歴を生成し，パターンとして認識された複数の連続する複合履歴群を子として新しい複合履歴に追加することで実現する．複合化の結果，新しい複合履歴を頂点とする階層化ツリーが得られる．この階層化処理をパターン階層化と呼ぶ．パターン階層化は，履歴マネージャにおいて，基本階層化処理が実行されて基本階層化後の複合履歴が形成されるつど，その新しい複合履歴と直前の複合履歴群の連続を仮のパターンとして，履歴リストの先頭から順次探索することで行う．

図 8 は，図 5 に示すコンポーネント構成において，図 6 に示した 1：キー入力〜7：キー入力の一連のユーザ操作が起きたときのパターン階層化の仕組みを示す．まず，履歴リスト中に基本階層化の結果として複合履歴 S_1〜S_7 が存在する．次に $S_3 \cdot S_4$ および $S_5 \cdot S_6$ の各連続は，コンポーネント ID 値とメソッド ID 値の対の連続が同種であることから履歴の連続として同一と認識される（それぞれ $(1,1)(1,2)(2,1)(1,3)$）．そこで，新たな複合履歴 S'_3，S'_6 を生成し，$S_3 \cdot S_4$ と $S_5 \cdot S_6$ の各連続がそれぞれ S'_3，S'_6 に子として追加され，その結果階層化された履歴ツリーが得られる．最後に，履歴ツリー上において階層度 1 を選択するとき，5 つの複合履歴を持った履歴リストが得られる．この結果，UndoableJToggleButton（C_2）を押して文字のフォントを変更し，続いて UndoableJCanvas（C_1）へ直接キーボードから文字を入力するという一連のユーザ操作を，1 度の Undo/Redo 操作によって取消し /再実行可能となる．

その後，続けて 8：キー入力のユーザ操作が行われると，基本階層化の結果として基本履歴 S''_6 が追加され，さらなるパターン階層化（B）が行われる．

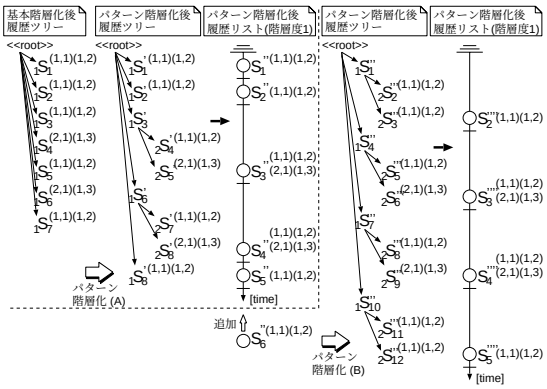

図 8 パターン階層化例
Fig. 8 Example of pattern hierarchization.

5.2 コンテナ階層化

同一のコンポーネントおよびそのコンポーネントが持つ包含関係上の子孫コンポーネント群の内部で閉じた履歴群は，ユーザにとって意味のある動作を表す適切な大きさのまとまりの候補となる．そこで，対象アプリケーションの実行中に，コンポーネントの物理的な包含関係に従って基本階層化後の履歴群が 1 つのコンポーネントおよび子孫コンポーネント群の内部で完結していたときに，それらの履歴群を 1 つのまとまりとして認識することで，アプリケーション開発者が特別に指定することなく機械的に，ユーザにとって自然な履歴のまとまりを形成可能となり，後の Undo/Redo 操作時において，それらの履歴群を一括して Undo/Redo 動作の対象とすることができる．

基本階層化の結果得られる履歴リスト中で，連続した複合履歴について，ある複合履歴の生成元コンポーネント群のコンポーネント包含構成上の親コンポーネントが，直前の複合履歴の生成元コンポーネント群の親コンポーネントに含まれているとき，または，2 つの親コンポーネントが同一であるときに，それら 2 つの複合履歴が表すユーザ操作の影響は，前の側の複合履歴の生成元コンポーネント群の親コンポーネントの中で閉じていると考えることができる．この親コンポーネントを複合履歴についてのコンテナコンポーネントと呼ぶ．同一のコンテナコンポーネントによって閉じられる連続する複合履歴群は，新たな 1 つの履歴として複合化することができる．この複合化は，複合履歴の生成元コンポーネント群の親コンポーネントの包含関係に従い，親コンポーネントが含まれる側の複合履歴を，含む側の複合履歴に子として追加することで実現する．複合化の結果，生成元コンポーネント群の親コンポーネントがコンテナコンポーネントである

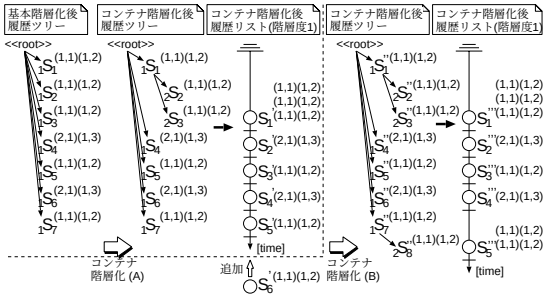

図 9 コンテナ階層化適用例
Fig. 9 Example of container hierarchization.

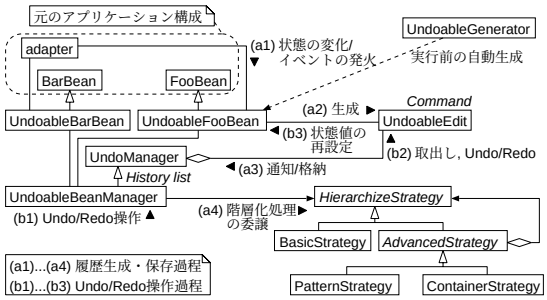

図 10 UndoableBean によるアプリケーション実装
Fig. 10 Developing application using UndoableBean.

複合履歴を頂点とする階層化ツリーが得られる．この階層化処理をコンテナ階層化と呼ぶ．コンテナ階層化は，履歴マネージャにおいて，基本階層化処理が実行されて基本階層化後の履歴が形成されるつど，その新しい複合履歴と直前の複合履歴群のそれぞれ生成元コンポーネントの親コンポーネント間の包含関係を検証して行う．

図 9 は，図 5 に示すコンポーネント構成において，図 6 に示した 1：キー入力〜7：キー入力の一連のユーザ操作が起きたときのコンテナ階層化の仕組みを示す．まず，履歴リスト中に基本階層化の結果として複合履歴 S_1〜S_7 が存在する．S_1 の生成元コンポーネント群は $\{C_1\}$ であり，図 5 に示すコンポーネント包含構成において，$\{C_1\}$ の親コンポーネントは C_5 である．S_1 の以降に連続する S_2 および S_3 の生成元コンポーネント群もまた $\{C_1\}$ であり，親コンポーネントは C_5 である．そこで，連続する S_1 と S_2，S_3 では親コンポーネントが同一であるので，S_2，S_3 は S_1 に子として追加される．一方，S_4 の生成元コンポーネント群は $\{C_2, C_1\}$ であり，その親コンポーネントは C_7 であるが，C_5 は C_7 を内包しないので，S_4 は S_1 に子として追加されない．この結果，コンポーネント C_5 および C_5 が子として持つコンポーネント群について閉じていた連続するキー入力のユーザ操作群を，1 度の Undo/Redo 操作によって取消し/再実行可能となる．

その後，8：キー入力のユーザ操作が行われると，基本階層化により複合履歴 S'_6 が追加され，コンテナ階層化（B）が行われる．

6. 実　　装

提案する Undoable 拡張系，および，実行時の履歴階層化機構を，Java 言語を用いて開発した．拡張系と履歴階層化機構を総称して “UndoableBean” と呼ぶ．コンポーネントシステムとして JavaBeans を対象としているが，我々が提案する手法は JavaBeans に依存するものではなく，細粒度コンポーネントの扱いを可能とする他のコンポーネントシステム（たとえば ActiveX）を対象とした実装も可能である．基礎的な Undo フレームワークとして undo パッケージを用いる．UndoableBean の主要構成要素間の関連を**図 10** に示す．

対象とするコンポーネントから Undoable 拡張クラスを自動生成するツール “UndoableGenerator” を準備した．UndoableGenerator を用いて，コンポーネントの状態設定・観測操作およびイベント発火操作について，ビジュアルな指定を行い，各メソッドを再定義した Undoable 拡張クラスを自動生成する．

6.1 Undo/Redo 動作の流れ

UndoableBean を用いて Undo 機構が実装されたアプリケーションの動作を図 10 を用いて以下に示す．

(1)　履歴生成・保存過程

Undoable 拡張クラス（図 10 の UndoableFooBean）の設定操作・観測操作およびイベント発火操作が起動されると（$a1$），Undoable 拡張クラスは起動された操作に対応した種類の履歴（UndoableEdit）を生成し（$a2$），UndoManager に通知する（$a3$）．UndoManager を継承した UndoableBeanManager は通知された履歴群に対して，HierarchizeStrategy オブジェクトを用いて階層化処理を行う（$a4$）．

(2)　Undo/Redo 操作・処理過程

利用者が Undo/Redo 操作を行うと（$b1$），UndoManager は自身が管理する履歴リスト中から直近の履歴を取り出し，履歴の Undo/Redo メソッドを起動する（$b2$）．履歴の種類が状態値変化履歴 PropertyChangeEdit であれば，履歴は対応するコンポーネントの状態の値を古い値/新しい値に戻す（$b3$）．

6.2 履歴階層化の実装

実行時に履歴群の階層化処理を行う枠組みとして，Strategy パターン[16]に従って処理アルゴリズムを持つ

階層化処理クラス HierarchizeStrategy を定義し，基本階層化・パターン階層化・コンテナ階層化の各アルゴリズムを持つサブクラス BasicStrategy・PatternStrategy・ContainerStrategy を準備した．各サブクラスは，Decorator パターン[16]の適用により，複数のインスタンスを組合せ可能である．また，HierarchizeStrategy クラスを継承したサブクラスを準備することで，新たな階層化アルゴリズムを追加できる．

7. 評　　価

UndoableBean を用いた本手法と，undo パッケージを用いた従来手法を，Undo 機構の実装コストおよび利便性・実行効率について比較評価する．評価サンプルとして，LunchApplet[2]（サンプル A），Test[19]（サンプル B），FontViewer[18]（サンプル C）を用いる．これらのサンプルは，undo パッケージを用いてすでに Undo 機構が実装済みである．さらに FontViewer について，より多くのユーザ操作（サイズ変更・スタイル変更・フォント名変更）を Undo/Redo 操作の対象とする拡張を行い，拡張後の FontViewer をサンプル D として用いる．全サンプルは（制限 1）〜（制限 3）を満たす．

本手法の評価にあたり，まずすべてのサンプルから，Undo 機構と関係のある部分をすべて取り除き，UndoableBean を用いて新たに Undo 機構を実装した．評価にあたり共通に用いた実験環境は，Sun Java2RE SE1.4.0，HotSpot ClientVM，Pentium4 1.4 GHz，Memory 512 MB，Windows2000 Professional である．

7.1 開発容易性

開発容易性の評価に全サンプルを用いる．本手法および従来手法で Undo 機構が実装された全サンプルについて，内部で Undo 機構と関連する部分のコード行数・クラス数の比較を**図 11**，**図 12** に示す．本手法において，コード行数およびクラス数には，開発者自身が書く必要のあるもの（手書き）と，自動生成ツールによって自動生成されるもの（自動生成）の 2 種類がある．

図 11 より，開発者が書く必要のあるコード行数は，すべてのサンプルについて本手法の結果が従来手法の結果よりも小さい値を示した．同コード行数の従来手法と本手法での全サンプルの平均は 125：27 であり，本手法では従来手法に比べて開発者の実質的な開発コストを 78.4%減少させている．この結果は，アプリケーションに対する Undo 機構の実装作業が，本手法ではアプリケーション中のすべてのコンポーネント

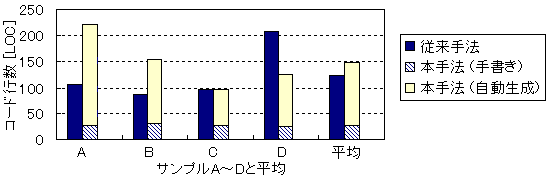

図 11 Undo 機構に関するコード数比較
Fig. 11 Code lines related to the undo facility.

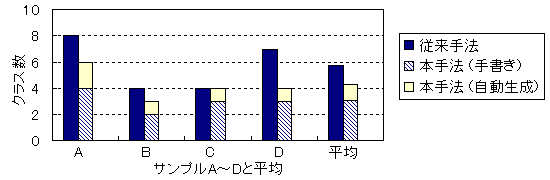

図 12 Undo 機構に関するクラス数比較
Fig. 12 Classes related to the undo facility.

表 1 必要修正コード行数（クラス数）
Table 1 Number of code lines (classes) necessary modified.

修正箇所	従来手法	本手法
アプリケーションロジック部分	6 (1)	6 (1)
Undo 機構との関連部分 — 手書き	14 (3)	0 (0)
Undo 機構との関連部分 — 自動生成	0 (0)	80 (1)

を自動生成ツールを用いて生成した Undoable 拡張クラスに置き換えることで大部分済むことに起因する．

図 12 より，Undo 機構実現のため開発の必要なクラス数について，本手法の結果はサンプル C で従来手法の結果と同じ値を示し，他のすべてのサンプルにおいて従来手法の結果よりも小さい値を示した．これらの結果は，主に開発者自身がユーザ操作に対応した履歴クラスを準備する必要があるかどうかに起因する．たとえばサンプル D において，従来手法ではサンプル C に加えて，追加された 3 つのユーザ操作に対応した履歴クラスを準備する必要があるが，本手法ではサンプル C とサンプル D で必要なクラス数は同じである．したがって，本手法は開発容易性について従来手法より優れていることが分かった．

7.2 保　守　性

保守性の評価にサンプル C を用いる．サンプルが用いるデータ型をクラス StringBuffer からクラス String に変更した．本手法と従来手法でそれぞれ Undo 機構が実装されたサンプルを，このアプリケーションロジックの仕様変更に応じて修正した．**表 1** は，本手法と従来手法で，仕様変更時にサンプル C 中のアプリケーションロジック部分と Undo 機構に関連する部分について，それぞれ修正の必要なコード行数とクラス数の

比較を示す．

表 1 より，従来手法では，Undo 機構に関連する部分において，修正の必要な箇所が 3 つのクラス（1 つの編集元クラスと 2 つの履歴クラス）に散在している．対して本手法では，Undo 機構に関連する部分について開発者自身はいっさい修正を行う必要がない．本手法において仕様変更にともなう修正作業は，自動生成ツールを用いてアプリケーションロジックの変更に対応した Undoable 拡張クラスを新しく生成し，古い Undoable 拡張クラスを新しい Undoable 拡張クラスに置き換えることでなされた．したがって本手法では，Undo 機構に関連する部分への，アプリケーションロジック部分の仕様変更の影響を，自動生成される Undoable 拡張クラス内で吸収しており，従来手法に比べて保守性について大きく優れることが分かった．

7.3 利 便 性

利便性の評価にサンプル D を用いる．従来手法で Undo 機構が実装されたサンプルは，undo パッケージに従う標準的な Undo/Redo 操作を提供する．本手法として，基本階層化のみ・パターン階層化＋基本階層化・コンテナ階層化＋基本階層化の 3 つの階層化処理の組合せを検証した．基本階層化のみのサンプルは，基本階層化 Undo/Redo 操作を提供する．パターン階層化＋基本階層化（コンテナ階層化＋基本階層化）の行われるサンプルは，階層度 1 におけるパターン階層化 Undo/Redo 操作（コンテナ階層化 Undo/Redo 操作）と基本階層化 Undo/Redo 操作を同時に提供する．

4 人の被験者が，従来手法と本手法（基本階層化・パターン階層化＋基本階層化・コンテナ階層化＋基本階層化）でそれぞれ実装された 4 つのサンプルについて，20 個のタスクを行った．すべてのタスクには目標文字列が設定され，別ウィンドウに表示される．編集中の文字列を目標文字列と同一とし OK ボタンを押すと，次の目標文字列が表示され，編集中の文字列が直前タスクの目標文字列の状態から，次の目標文字列を目指す．被験者は文字列編集操作として，サンプル独自のユーザ操作（キー入力・サイズ変更・スタイル変更・フォント名変更）を行うことができる．また被験者は行ったユーザ操作の Undo/Redo 操作を行うことができる．**図 13** にタスクごとの目標文字列（a）とサンプルの実行例（b）を示す．

すべてのタスクについて，タスク処理時間とサンプル独自のユーザ操作回数，および Undo/Redo 操作回数を測定した．**図 14** は，Undo/Redo 操作回数の全被験者についてのタスクごとの平均を示す．**図 15** は，サンプル独自のユーザ操作回数と全被験者についてのタスクごとの平均を示す．**表 2** は，タスク処理時間・サンプル独自のユーザ操作回数・Undo/Redo 操作回数・サンプ

タスク	目標文字列
1	**Sample**
2	*Sample Code*
3	Sample
4	Appl
5	Apple
6	A
…	…

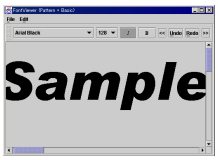

図 13 (a) 目標文字列（抜粋），(b) 実行例
Fig. 13 (a) Target strings (excerpt), (b) display example.

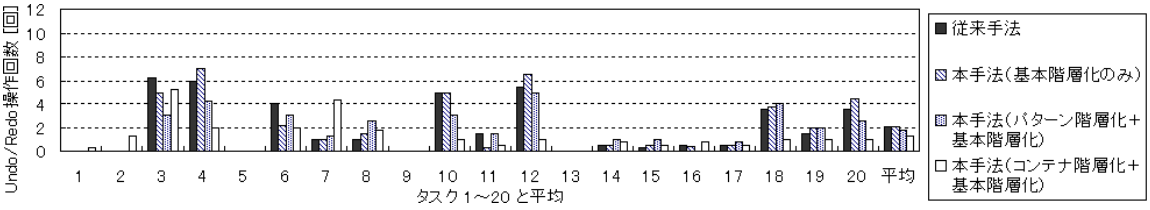

図 14 Undo/Redo 操作回数のタスクごとの比較
Fig. 14 Number of Undo/Redo operation times per task.

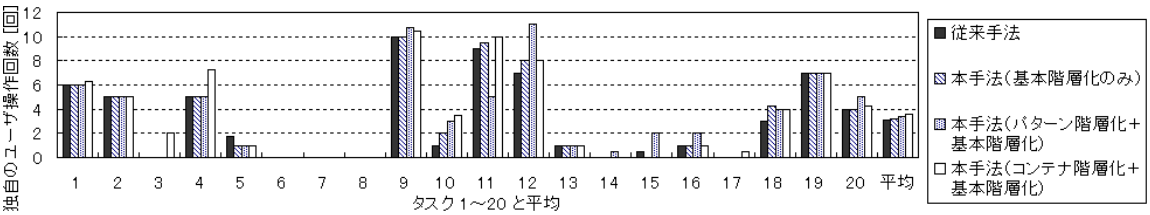

図 15 独自のユーザ操作回数のタスクごとの比較
Fig. 15 Number of original user's operation times per task.

表 2 1 タスクあたりの処理時間・操作回数・利便性
Table 2 Execution time, operation times, usability.

種類	Con	Basic	Patt	Ctan
処理時間 [msec]	3,648	3,773	4,377	4,259
Undo/Redo 回数	2.03	2.03	1.74	1.24
（速い Undo/Redo）	—	—	1.23	0.45
独自のユーザ操作回数	3.06	3.19	3.41	3.57
総ユーザ操作回数	5.09	5.21	5.15	4.80
利便性スコア [point]	—	97.6	98.8	105.9

Con：従来手法，Basic：本手法（基本階層化のみ）
Patt：本手法（パターン階層化＋基本階層化）
Ctan：本手法（コンテナ階層化＋基本階層化）

ル独自のユーザ操作回数と Undo/Redo 操作回数を合計した総ユーザ操作回数の，全被験者についての 1 タスクあたりの平均を示す．パターン階層化＋基本階層化とコンテナ階層化＋基本階層化については，高度な階層化 Undo/Redo 操作の回数もあわせて示す．表 2 はまた，総ユーザ操作回数についての相対利便性スコアの本手法における値を示す．相対利便性スコア（スコア (Non)〜スコア (Ctan)）は次のように計算される：スコア (x) = 総操作回数 (Con)/総操作回数 $(x) \times 100$．相対利便性スコアは値が大きいほど，利用者が行った操作回数が減少され，利便性が向上したことを示す．

ここで利便性とは，ユーザがアプリケーションを操作して目的とするタスクを迅速に実行可能であるかどうかの度合いを指す．サンプルは，アプリケーション独自の操作とは別に Undo/Redo 操作を提供するので，Undo/Redo 操作の使いやすさが利便性に大きく関与する．ユーザが Undo/Redo 操作を行うときに，ユーザが想定する Undo/Redo 動作に近い振舞いをサンプルが示すならば，結果として利便性は向上する．したがって，与えられたタスクをより少ない操作回数で達成できるほど，サンプルが提供する Undo/Redo 機能はユーザが想定する振舞いに近いと考えられる．

表 2 より，本手法の基本階層化では，Undo/Redo 操作回数と独自のユーザ操作回数ともに，従来手法とほぼ同一の結果が得られた．この結果は，基本階層化によって原子履歴群が従来手法における 1 つの履歴と同規模の 1 つの複合履歴に複合化され，ユーザが実行可能なすべての独自操作について，従来手法と同等の Undo/Redo 動作を実現できたことによる．

本手法のパターン階層化＋基本階層化とコンテナ階層化＋基本階層化では，Undo/Redo 操作回数は従来手法よりも平均について少ない．たとえば多くの被験者はタスク 6 を，従来手法では，タスク 4・5 の実現のために行ったキー入力操作を 4 回の Undo 操作によって取り消すことで達成した（図 14）．対して，コンテナ階層化＋基本階層化が行われる本手法では，タスク 6 について，多くの被験者は 1 回のコンテナ階層化 Undo 操作によりすべての文字を削除し，続いて 1 回の基本階層化 Redo 操作を行うことで文字「A」を追加して達成した（図 14）．これは，サンプルがコンテナコンポーネントを持ち，キー入力操作に関するコンポーネントがその中に内包されるので，基本階層化の結果としてキー入力操作に対応する履歴の連続がコンテナ階層化によって 1 つに複合化されたためである．また，パターン階層化＋基本階層化が行われる本手法では，タスク 6 について，1 回の基本階層化 Undo 操作と 1 回のパターン階層化 Undo 操作，および再度 1 回の基本階層化 Undo 操作により達成した（図 14）．これは，キー入力操作を 2 回連続して行う操作がパターンとして認識され，文字列「Apple」中の連続「Ap」と連続「pl」を実現する履歴群がそれぞれ 1 つの履歴に複合化されたためである．このように，パターン階層化およびコンテナ階層化を行うことで，規模の大きい履歴を内部で機械的に形成し “速い”Undo/Redo 操作を実現可能となった．

一方，表 2 より，独自のユーザ操作回数についてはパターン階層化＋基本階層化とコンテナ階層化＋基本階層化ともに，従来手法よりも多くなっている．これは，高度な階層化 Undo/Redo 操作によりユーザ操作を取り消しすぎたり，再実行しすぎたりした結果，希望のアプリケーション状態に移行するために従来手法よりも多い独自のユーザ操作が必要となったことによる．しかしながら，表 2 において本手法のコンテナ階層化＋基本階層化における相対利便性スコアは 100 ポイントを超えており，本手法による Undo 機構の利便性はコンテナ階層化によって，従来手法における利便性を上回っている．したがって，コンテナ階層化＋基本階層化が提供するコンテナ階層化 Undo/Redo 操作および基本階層化 Undo/Redo 操作は，タスクを達成するにあたって被験者がそれらの操作によって行われると想定する振舞いに近いものであったと考えられる．

表 2 において，本手法のパターン階層化＋基本階層化とコンテナ階層化＋基本階層化におけるタスク処理時間の平均は，従来手法よりも弱冠大きいが，これは主に被験者が “速い”Undo/Redo 操作に当初戸惑いを示したためであり，慣れることでタスク処理時間は従来手法に近づくと考えられる．

7.4 実行効率

本手法（基本階層化のみ・パターン階層化＋基本階層化・コンテナ階層化＋基本階層化）と従来手法の実行効率の比較評価を行った．評価サンプルとして各手

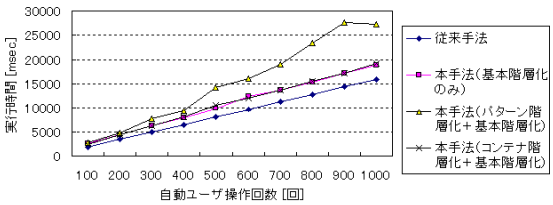

図 16 自動ユーザ操作時の実行時間
Fig. 16 Execution time for automatic user's operations.

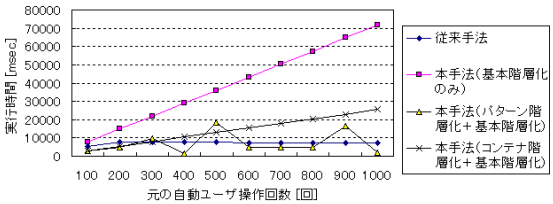

図 17 自動 Undo 操作時の実行時間
Fig. 17 Execution time for automatic undo operations.

法の Undo 機構が実装されたサンプル D を用い，可能なすべてのユーザ操作を自動的に一定回数繰り返し実行して，かかる時間を測定した．続いて，最初のアプリケーション状態に戻るために Undo 操作を自動的に繰り返し実行して，かかる時間を測定した．自動ユーザ操作時の計測結果と自動 Undo 操作時の計測結果をそれぞれ**図 16**，**図 17** に示す．

図 16 において，従来手法が最も実行効率に優れるが，本手法の基本階層化，コンテナ階層化＋基本階層化の遅延は最大で 3000 [msec] であり，実際にアプリケーションをユーザが対話的に利用する場合において実用上問題ないと考える．本手法のパターン階層化＋基本階層化の遅延は最大で 8500 [msec] と弱冠大きいが，これは，基本階層化後の履歴が追加されるたびにパターン階層化の処理が行われているためである．

図 17 において，本手法の基本階層化の結果は一定して従来手法の 9.8 倍ほど時間がかかっている．前節の利便性評価において従来手法と本手法の基本階層化では Undo/Redo 操作の基本単位の大きさがほぼ同一であることが分かったので，基本階層化は従来手法に比較して，1 回の Undo 操作にかかる時間が大きいことが分かる．これは，履歴マネージャ内部において基本階層化後の履歴は複数の原子履歴によって表されるので，従来手法に比べてつねに扱うオブジェクト数が大きいためである．本手法のパターン階層化＋基本階層化とコンテナ階層化＋基本階層化の結果は従来手法の結果に近いものとなっている．ただし，パターン階層化＋基本階層化では，実行時のパターンの形成状況によってアプリケーションの初期状態に戻るために必要な Undo 操作回数が大幅に変動する．

以上より実行効率の観点からは，本手法を適用するアプリケーションとして，想定するユーザ操作の回数が少なく，また，全体として短時間で処理が終了するものが望ましいと考えられる．ただし，評価で取り上げた 4 サンプルに関しては，本手法で Undo 機構を実装したものについて，実用上は問題なかった．

8. おわりに

3 つの制限を満たす細粒度コンポーネントおよび細粒度コンポーネントから構成される開発済みのアプリケーションについて，コンポーネントの状態値変化を利用して，開発容易性・保守性に優れる Undo 機構実装手法を実現し，Undo/Redo 動作対象の状態値変化への限定より生ずる履歴が原子的である問題を，履歴階層化により解消した．本手法の基本階層化を用いた結果は従来手法における Undo 機構とほぼ同一の適切な履歴の大きさを形成することが分かった．また，利便性評価の結果，高度な履歴階層化を併用することで，ユーザにとって利便性の高い Undo 機構を開発容易性に優れたままで提供可能となることを示した．本論文で示す UndoableBean は http://www.fuka.info.waseda.ac.jp/Project/CBSE/ より利用可能である．今後，本論文であげた制限の一部またはすべてを解消可能な方法を考え，本手法の適用範囲を広げる予定である．

参考文献

1) Olsen, R.: *Developing User Interfaces*, Morgan Kaufmann Publishers (1998).
2) Meshorer, T.: Add an Undo/redo Function to your Java Apps with Swing, *JavaWorld* (June 1998).
3) 西田知博，林　真志，辻野嘉宏，都倉信樹：ヒストリーグラフを用いたアンドゥ機構の提案と評価，情報処理学会情報メディア研究会報告，35-12 (1999).
4) Jeffrey, S.: A New Framework for Redoing, *IEEE Software*, Vol.1, No.4 (1984).
5) Berlage, T.: A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects, *ACM Trans. Computer Human Interaction*, Vol.1, No.3 (1994).
6) 増田尚則，今宮淳美：Undo 機能をもつグラフィカル履歴ブラウザ設計と視覚的探索分析，電子情報通信学会論文誌，Vol.J85-D-I, No.8 (2002).
7) Myers, B. and Kosbie, D.: Reusable Hierarchical Command Objects, *Conference on Human Factors in Computing Systems* (1996).
8) Backer, A.: JCommands: A Flexible Undo

Framework for Java, *JavaReport*, Vol.5, No.12 (2000).
9) Buschmann, F., et al.: *Pattern-Oriented Software Architecture*, Addison-Wesley (1998).
10) Hopkins, J.: Component Primer, *Comm. ACM*, Vol.43, No.10 (2000).
11) Vlissides, J.: Multicast, *C++ Report* (Sep. 1997).
12) Han, J.: A Comprehensive Interface Definition Framework for Software Components, *Asia-Pacific Software Engineering Conference* (1998).
13) Bachman, F., et al.: Technical Concepts of Component-Based Software Engineering, Technical Report, CMU/SEI-2000-TR-008 (2000).
14) Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley (1997).
15) Gao, J.: Component Testability and Component Testing Challenges, *International Workshop on Component-Based Software Engineering* (2000).
16) Gamma, E., et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley (1994).
17) Flanagan, D.: *Java Foundation Classes in a Nutshell*, O'Reilly & Associates (1999).
18) 松田慎一：Swing プログラミング，*Dr. Dobb's Journal Japan* (Sep. 1998).
19) Geary, D.: *Graphic Java2 Volume 2*, 3rd Edition, Prentice Hall (1999).

(平成 13 年 8 月 6 日受付)
(平成 14 年 10 月 7 日採録)

鷲崎 弘宜（正会員）

1976 年生．2001 年早稲田大学大学院理工学研究科修士課程修了．現在，同大学院理工学研究科博士課程に在学中．2002 年同大学理工学部助手．コンポーネント指向ソフトウェア開発，ソフトウェアパターン等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．

深澤 良彰（正会員）

1976 年早稲田大学理工学部電気工学科卒業．1983 年同大学院博士課程中退．同年相模工業大学工学部情報工学科専任講師．1987 年早稲田大学理工学部助教授．1992 年同教授．工学博士．ソフトウェア工学，コンピュータアーキテクチャ等の研究に従事．日本ソフトウェア科学会，IEEE，ACM 各会員．