Javaアプレットのためのアニメーションヘルプシステム

三浦元喜[†] 田中二郎[††]

Javaアプレットは，Webブラウザ上で動作するため誰にでも簡単に実行できる．その操作方法はテキストで書かれることが多いが，ほとんどのアプレットのインタフェースは直接操作を用いているため，ユーザは直感的に把握しにくい．本論文では，アプレットの操作を説明するアニメーションヘルプを作成，編集，再生するシステムについて述べる．ここでいうアニメーションヘルプとは，擬似的なマウスカーソルが動いて具体的な操作手順を連続的に提示するものである．

我々のシステムを用いてアニメーションヘルプを生成するには，実際のアプレット上で操作を行うだけでよい．発生したイベントは自動的に記録され，それを基にアニメーションヘルプを生成する．ただ，アニメーションヘルプを編集する段階においては，イベント列より抽象度を高めたコマンドを単位とするほうが効率がいい．コマンドを生成するために，イベント列とコマンドの対応をルールとして与える．このルールを用いて，本システムはイベント列から意味のある部分を抽出し，その意味を表す文字列を付加したコマンドを生成する．この文字列はアニメーションヘルプ再生中に表示するポップアップメッセージや，ヘルプの概要として用いられる．

本システムを用いることで，アニメーションヘルプを生成・再生する機能を，アプレットに変更を加えることなく追加することができる．そのため，開発者がヘルプ記述にかかる労力を軽減することができ，ユーザに典型的な動作を手軽に見てもらうことができる．

An Animated Help System for Java Applets

Motoki Miura[†] and Jiro Tanaka[††]

Java applets are executed on Web browsers. Most applets are publicized with textual explanations. However, there is a gap between textual instructions and graphical interfaces which makes it hard for users to properly manipulate the applets.

We have developed a system which enables the applet-developers to prepare an animated help for their applets. The animated help means to perform demonstration of the applet's behavior with a pseudo mouse cursor.

Our system generates animated help from captured event-objects which occur due to user actions. To edit the animated help more efficiently, we have introduced the concept of "command" instead of event-object stream. Each command should be identified by a label which reflects its meaning. For generating commands, the system loads "command production rules" which map a set of event-objects to a command and give a label to it. The label can be used not only for editing but also for showing the abstract list of the demonstration for the user.

We have implemented the system as a plug-in for the target applet. The developers can add animated help functions without any changes in the target applet's source/class files. For both users and developers animated help is intuitive and effective.

1. はじめに

Javaアプレットは，HTML文書から指定することによってWebブラウザ上で動作するため，誰にでも簡単に実行できる．

多くのアプレットで用いられているGUI（Graphical User Interface）は一般にコマンドベースのインターフェースよりも初心者にとって優れているとされている．しかし，システムが多機能になるにつれ，ボタンやリストの項目が増えるため，ユーザにとってどんな操作が可能なのかを理解しづらくなるという問題がある．そのため，操作方法（ヘルプ）をユーザに提供することがアプレット開発者に求められている．

一般に，操作方法はテキストで記述されることが多い．テキストには，ユーザがあるタスクを遂行するのに行うべき操作が順を追って記述される．そこでは「操作

† 筑波大学大学院 博士課程 工学研究科
Doctoral Program in Engineering,
University of Tsukuba
†† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics,
University of Tsukuba

の対象」となるメニューやアイコンボタンは文章やアイコン画像によって提示されるが，ユーザはそれらの「操作の対象」を実際のアプレット内から探し出す必要がある．これは，純粋に操作方法や操作の概要を知りたいユーザにとって負担となる．

こうした問題を解決するため，文章を主体としたヘルプの代わりに一連の操作をデモンストレーションで見せる方法（アニメーションヘルプ手法）が提案されている[1),2)]．擬似的なマウスカーソルがアイコンボタンやメニューなどの「操作の対象」を明示するので，ユーザは操作方法の理解に集中することができる．しかし，従来そのようなアニメーションヘルプ機能を提供するにはシステムに始めから操作を説明する機能を組み込む必要があり，開発者にとって労力がかかっていた．

本論文では，操作を説明する機能を持たない一般的なJavaアプレットに対してアニメーションヘルプ機能を付加し，実際にユーザに提示するヘルプを作成するシステムと，その実装について述べる．本システムにおいては，低次元のイベント列から，操作の概要を表す文字列を生成するためのルールをまず準備する．それらのルールを用いることで，操作を行うだけで意味付けされた操作が記録できる．そのため，従来よりも編集作業が容易になる．また開発者がヘルプ記述にかかる労力を軽減することができる．ユーザにとっても，直感的にわかりやすいヘルプを利用することでシステムの機能を把握しやすい．

2. デモンストレーションの記録形態

実際の操作から生成するデータの記録形態として，動画像として記録するものと，システムが解釈できるイベントとして記録するものがある．ムービーファイルなどの動画像として記録した場合，再現するのに動画像専用のビューアを用いれば簡単に参照でき，かつ問題も少ない．しかし，システムのバージョンやメニュー構成の変化などに対応できない．また，ユーザのレベルに合わせて説明の粒度を調節することも難しい．

我々は，ヘルプの編集のしやすさ，再生時の柔軟性などのメリットを考慮し，ヘルプとしてのアニメーションをイベントとして記録する（イベント駆動）方式を採用した．この手法による付加的なメリットとしては，一般にイベントとして記録されたものは動画像よりもデータサイズを小さく抑えることができるので，ネットワークから取得するJavaアプレットのヘルプとしては適しているという点が挙げられる．ヘルプを提示する際には，基本的には記録しておいたイベントを再現すればよい．

3. デモヘルプシステム “Jedemo”

3.1 設計方針

イベント駆動方式のアニメーションを記録，再生するためには，通常，個々のシステムが以下の3つの機能を持つことが必要になる．

(1) システムをある状態（例えば，初期状態）にする機能 （initialize）
(2) システムで発生したイベントを記録する機能（recording）
(3) システムに記録したイベントを実行させる機能（playing）

しかし，これらの機能をそれぞれのシステムについて実装するためには，ソースコードに変更を加えたり，再コンパイルするなどの作業が必要となり，開発者にとって余分な作業を強いることになる．もし，これらの機能を個々のアプリケーションに実装しないならば，アプリケーションの実行系（Javaアプレットの場合では Java Virtual Machine） がこれらの機能を有していればよいことになる．しかし，これらの実行系を変更してしまうと，アニメーションヘルプを参照するために特殊な環境を必要とすることになる．アニメーションヘルプを作成・編集する時はともかく，再生するのは一般のユーザであるので，標準となっている実行系で動くことが望ましい．そのため， Java Virtual Machine には手を加えずに実現する方針をとることにした．

3.2 アプレットビューアモデル

我々は，上に述べたような制約を満たしつつ，記録と再生の機能を一般的なアプレットに持たせる「アプレットビューアモデル」という枠組みを提案した[3)]．この枠組みでは，ヘルプ機能を持たせようとするアプレット（以下，「対象アプレット」と呼ぶ）を，イベント管理機構を持つアプレット（以下，「マネージャアプレット」と呼ぶ）に貼り付ける．こうすることで記録と再生の機能を追加し，対象アプレットの機能を拡張することができる．**図1** に，このアプレットビューアモデルの機構を示す．ユーザがアニメーションヘルプを参照できるようにするための開発者がすべき作業としては，マネージャアプレットを呼び出す際にパラメータとして対象アプレットのクラス名とヘルプのファイル名を指定するようにHTMLファイルを書き替えるだけである．具体的には，

```
<applet code="TargetApplet.class">
</applet>
```

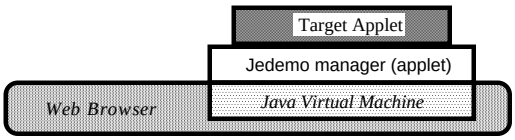


図 1 アプレットビューアモデルのイメージ
Fig. 1 A graphical image of "AppletViewer Model."

となっている部分を,

```
<applet code="ManagerApplet.class">
  <param name="target" value="TargetApplet">
  <param name="helpfile" valie="sample.jdm">
</applet>
```

のように変更することで，対象アプレット TargetApplet.class にヘルプ機能が追加されるようになる．もちろんこの書き替えは，機械的に行うことが可能である．

図 1 では， Jedemo manager が Target Applet を emulation によって実行しているように見えるかもしれないが，実際は単に Target Applet を部品として貼っているだけなので，アプレットの動作速度は通常の Target Applet のみを実行した場合とそれほど変わらない．

3.3 イベントオブジェクトの記録と再生

マネージャアプレットは，特殊なイベントリスナを対象アプレットの部品に登録することでイベントオブジェクトを取得できる．イベントリスナとは， Java で標準的に用いられているイベントを処理するオブジェクトである．通常，プログラマが Java のアプリケーションを構築するときには，操作が行われたときの処理をこのイベントリスナに記述する．操作が行われる部品（イベントが発生する部品）は，このイベントリスナを複数登録できる．登録したイベントリスナに関連したイベントが発生した場合に，そのイベントリスナのメソッドを呼び出す仕組みである．

我々はあらかじめ典型的なイベントを取得するための特殊なイベントリスナを用意した．これらの特殊なイベントリスナは，通常では処理された後捨てられてしまうイベントを保存するため，独自の Recordable インターフェースを備えるオブジェクトに転送する機能がある．この Recordable なオブジェクト（実際はマネージャアプレット）では，イベントオブジェクトを発生順に保存する機能 `addEvent()` を実装している．我々が用意したイベントリスナは `MouseListener`, `MouseMotionListener`, `ContainerListener`, `ActionListener` の 4 つに対応するものであるが， `java.lang.reflect.Proxy` クラスを用いることでユーザ定義のイベントにも対応できる．

以下に `ActionListener` に対応する `ActionMessenger` クラスの定義を示す．

```
public class ActionMessenger
                  implements ActionListener{
 Recordable rec;             // 転送先
 public ActionMessenger(Recordable rec)
   {   this.rec = rec;   }
 public void actionPerformed(ActionEvent e){
   {   rec.addEvent(e);  }  // rec へ転送する
}
```

マネージャアプレットは，対象アプレット内に含まれているすべての部品を調べ，可能な限りこの特殊なイベントリスナを追加する．もし，新たな部品が動的に追加された時には，その部品にも特殊なイベントリスナを追加することで対象アプレット内に存在するすべての部品から，発生するイベントオブジェクトを常に監視し，記録できる．

記録したイベントオブジェクトを使って操作を再現するためには， `Component.dispatchEvent()` メソッドを用いる．イベントオブジェクトが発生した元の部品にこのメソッドの引数としてイベントオブジェクトを指定することで部品がイベントオブジェクトを処理し，動作を再現する．

保存したイベントオブジェクトをどの部品に送り返すべきかは，それぞれのイベントオブジェクト自身が参照として持ってはいるが，実際に再現する場面ではそれらの参照までは保存されていない．そのためイベントオブジェクトを取得した時に部品固有の情報（部品の階層構造における位置や，部品名など）を記録しておき，その情報を元に実際に再現する場面において，該当する部品を対象アプレット中から探し出すことになる．

4. 操作の意味付け

4.1 イベントオブジェクトとコマンド

Java アプリケーションにおいては，一般に操作を行うことで大量のイベントオブジェクトが発行される．これらのイベントオブジェクトの多くは「マウスを動かした」などの低レベルな意味を持つものである．アニメーションヘルプを編集するのに低レベルなイベント 1 つ 1 つを単位として行うのは効率が悪い．また低レベルイベント列の意味を理解していないと編集できないという問題もある．

我々はこれらの問題を解決するため，イベント列をまとめた**コマンド**という概念を導入する． 1 つのコマンド

表 1 アイコンとイベントの対応
Table 1 Event Icons

アイコン	Event の種類	発生する条件
	MousePressed	マウスボタンが押された時
	MouseDragged	マウスがドラッグされた時
	MouseReleased	マウスボタンが離された時
	MouseClicked	押して離すまでの時間が短かい時
	MouseEntered	マウスカーソルが部品に入った時
	MouseExited	マウスカーソルが部品から出た時
	MouseMoved	マウスが移動した時
	ComponentAdded	部品が追加された時
	ComponentRemoved	部品が削除された時
	ActionPerformed	ボタンが押されるなど，アクションが発生した時

は最低限の意味を持つイベント列である．アニメーションヘルプを編集する時には，このコマンドを単位として削除や順番の入れ替えを行えるようにすることで，低レベルな情報を隠蔽する効果が得られる．

コマンドには，意味を表す文字情報を**ラベル**として付加する．ラベルは，編集時のインデックスとして用いられるだけでなく，再生時にユーザに示す操作の概要として利用できる．

4.2 コマンドルール

コマンドとしてイベント列をまとめて管理するためには，どのようなイベント列がコマンドになり得るのかといった情報が必要になる．しかし一般には，対象アプリケーションの実装の違いによって，イベントとそれによる振舞いとの対応は異なってくる．

ここで，対象アプレットに手を加えることができるならば，それらの実装がどのようになっているかを，外部のシステムであるヘルプシステム（マネージャアプレット）が知ることができる．しかし，対象アプレットに変更を加えることは我々の方針に反するので適切ではない．そこで，我々は特定の対象アプレットにおいて，どのような操作がどのような意味を持つのかという対応を**コマンドルール**を用いて定義することで意味付けを行う．それぞれのコマンドルールは，

- イベント列のマッチングパターン
- ラベル生成ルール

を持っている．もし，あるイベント列がマッチングパターンとの照合に成功した場合には，そのイベント列がコマンドとして対応づけられ，ラベル生成ルールによってラベル付けされる．

5. コマンド生成の手順

この章では，実際に低レベルイベント列から，どのようにコマンドルールを照合して，コマンドを抽出するかについて述べる．

対象アプレットで発生した低レベルのイベントオブジェクトを集めたものは，発生時間順に並べられた一列の長いイベント列になっている．このイベント列は，操作として意味のある部分列と，意味のない部分列の組み合わせから成り立っていると考えることができる．よって，この意味のない部分列をうまく処理することで，コマンドルールとの照合に成功する可能性のある部分のみを抽出できるため，コマンド照合にかかる時間を短縮できる．

そこで，照合の前処理として，イベント列から意味のある部分列（コマンド候補）を切り出す処理を行う．この処理のために，我々はセパレータというものを定義する．セパレータとは，対象アプレットにとって意味のないイベントオブジェクトの集合である．

実際の前処理は以下のようにして行われる．イベント列の先頭からイベントオブジェクトを１つずつ読み込んでいき，もしそれがセパレータに含まれるならば，それはイベント候補ではないので捨てる．セパレータに含まれないのであれば，そのイベントオブジェクトから次にセパレータに含まれるイベントオブジェクトの１つ前までを，１つのコマンド候補とする．この作業を繰り返し行うことで，複数のコマンド候補をイベント列から切り出すことができ，コマンドルールとの照合が効率良く行える．

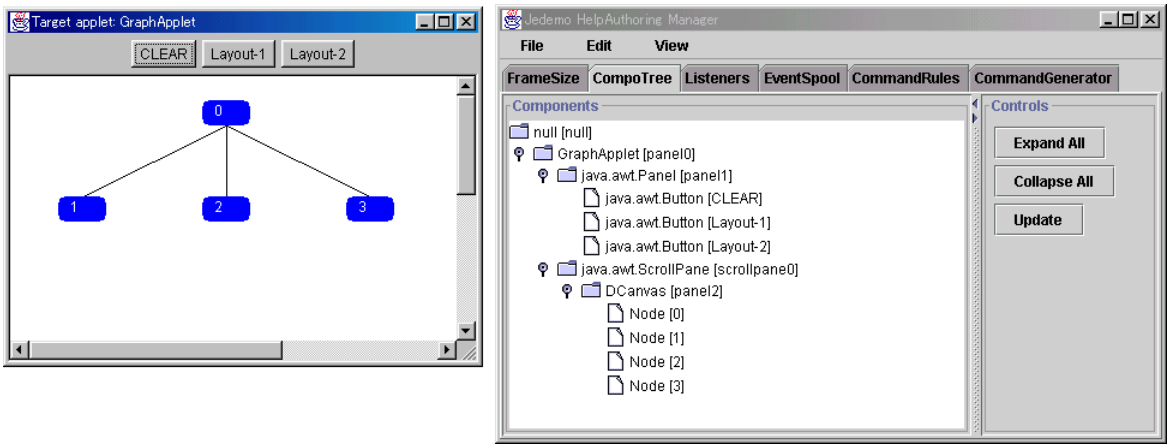


図 2 JedemoAuthor：対象アプレットと含まれる部品の階層構造
Fig. 2 JedemoAuthor: The target applet and its component structure.

6. 実装システム

これらの技術を用いてヘルプ編集システムを実装した．ここでは具体的にヘルプを作成し，編集して公開するまでの手順を順を追って説明する．

主に開発者が用いるヘルプ編集システム（JedemoAuthor）は，Java アプリケーションとして実装されている．ここで，開発者が行う作業は以下の 4 つである．

(1) 対象アプレットであるクラスファイル，もしくはそれを呼び出す HTML ファイルを指定し，読み込む．
(2) 対象アプレットで典型的な操作を行う．
(3) コマンドルールを定義する．
(4) 生成されたコマンドを編集し，保存する．

6.1 対象アプレットの指定

JedemoAuthor を Java アプリケーションとして起動し，対象アプレットを呼び出している HTML ファイルを指定する．ここで指定する HTML ファイルは，アニメーションヘルプ機能を追加する前に Web 上に公開していたものと同じでよい．JedemoAuthor はアプレットタグの情報から対象アプレットを起動し，そこに含まれている部品の階層構造を示す（**図 2**）．部品に関して JedemoAuthor が得られる情報は，クラス名，インスタンス名，位置，大きさ等である．ここでは，例としてグラフ編集アプレットを起動する HTML ファイル（GraphApplet）を指定した．JedemoAuthor は 6 つのサブシステムをタブで切り替えながら順番に操作していくことで最終的にはアニメーションヘルプが準備できる．

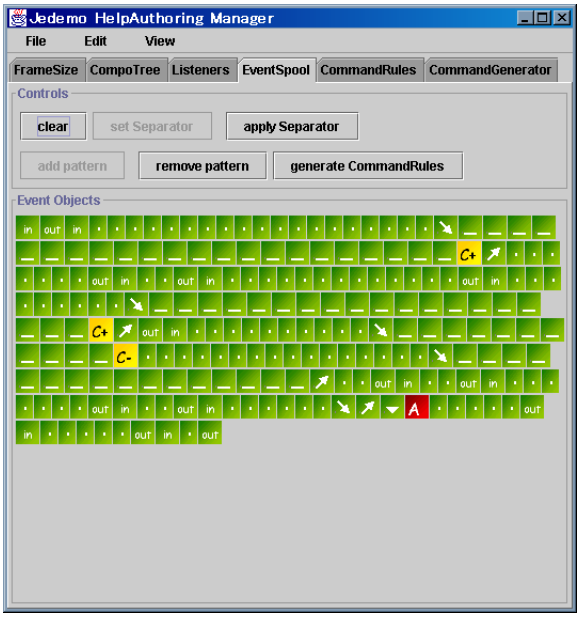


図 3 JedemoAuthor：スプールに保存されたイベントオブジェクト
Fig. 3 JedemoAuthor: Recorded event objects.

6.2 操作の記録

[Listeners] タブにある [Add Listeners] ボタンを押すと，JedemoAuthor は階層内に表示されている部品に特殊なイベントリスナを追加する．この時点から，対象アプレットにおける操作で発生したイベントオブジェクトは，JedemoAuthor に送られる．

[EventSpool] タブが選択されている状態で，送られたイベントオブジェクトはスプールに保存される．保存されたイベントオブジェクトはアイコンで表示される．

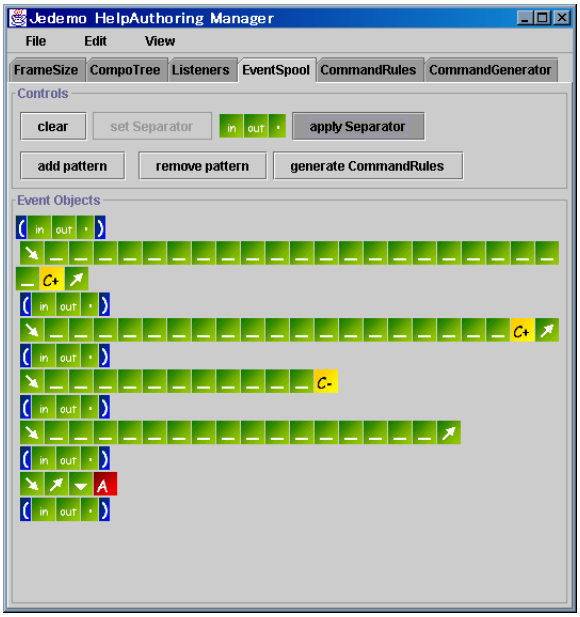


図 4 セパレータを適用した画面
Fig. 4 Applied separator to event objects.

図 5 ドラッグイベントの簡約化を行った画面
Fig. 5 Applied abbreviation to event objects.

イベントオブジェクトとアイコンとの関係を**表 1** に示す．開発者は，典型的な操作を行うことでスプールにイベントオブジェクトをためる（**図 3**）．

6.3 コマンドルールの定義

ある程度操作を行った後で，開発者は，イベントオブジェクトの中で意味を持たない部分（セパレータ）を指定する．スプール内のアイコンをドラッグすると範囲を指定できる．[set Separator] ボタンを押すと，範囲に含まれるイベントオブジェクトの要素が 1 つずつセットされ，表示される．ここで，[apply Separator] を押すことで，実際にセットされたセパレータによってイベント列が区切られ，**図 4** のように，セパレータとコマンド候補が交互に現れる．また，コマンド候補中に現れるドラッグイベントの数によってコマンドを区別したくないため，ドラッグイベント列を指定して，[add pattern] を押すと，ドラッグイベントが簡約表示されて**図 5** の画面になる．

ここで，[generate CommandRules] ボタンを押すと，図 5 中にあるイベントパターンのうち，同一のものを 1 つずつ残してコマンドルールのイベントマッチングパターンとしたコマンドルールが [CommandRules] タブに表示される（**図 6**）．図 6 では，1 つのセパレータと，4 つのコマンドルールが表示されている．このあと，開発者はそれぞれのコマンドルールにおけるラベル生成ルールをテキストを用いて定義し，完成させ（**図 7**），必要に応じて保存する．

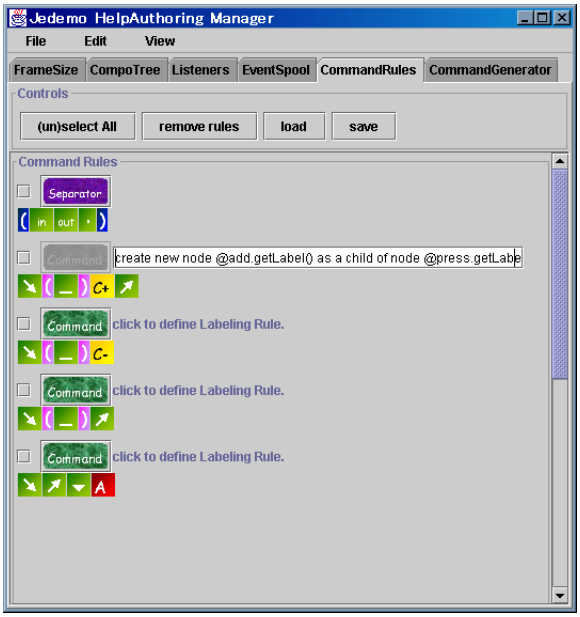


図 6 コマンドルールのラベル生成ルールの定義
Fig. 6 Editing a label production rule.

6.4 コマンドの生成，編集

[CommandGenerator] タブが選択されている状態では，対象アプレットにおける操作は [CommandRules] タブで定義されているコマンドルールによって照合が行われ，成功すればコマンドが生成される．この時，コマンドルールのラベル生成ルールによってラベルが付加さ

図 7 完成したコマンドルール
Fig. 7 Defined command rules.

図 8 自動生成されたコマンドのラベル
Fig. 8 Command labels generated automatically.

表 2 オブジェクト指定子
Table 2 Object Identifier

オブジェクト指定子	対応するオブジェクト
`@press`	マウスボタンが押された部品
`@release`	マウスボタンが離された部品
`@add`	新たに追加した部品
`@addbase`	追加した部品の親
`@remove`	取り除かれた部品
`@removebase`	取り除かれた部品の親
`@action`	アクションイベントの起きた部品

れる．このラベルには，静的な文字列だけでなく，動的な情報を挿入することができる．ラベル生成ルールにおいて，`@press.getLabel()` と指定した部分は，マウスボタンが押された部品の `getLabel()` メソッドの返り値に（コマンド生成時に）置き換えられる．ラベル生成ルール内においてオブジェクトを指定する特別な文字列として，現在**表 2** にあるものが利用できる．このようにして生成したコマンドを必要に応じて編集し，ファイルに保存する．このときのファイル名を使って，JedemoAuthor はオリジナルの HTML ファイル内のアプレットタグのみを書き替えたファイルを出力する．開発者はこのファイルをアニメーションヘルプ機能付きアプレットとして Web 上で公開すればよい．

6.5 JedemoAnimator

ユーザは，開発者によってアニメーションヘルプ機能が提供されているアプレットにおいて，マネージャアプレットによって提供される [start DEMONSTRATION] ボタンを押すことで，開発者によって記録されたコマンドを実行し，連続的な動作として見ることができる．**図 9** に，実際に生成されたコマンドを実行している画面を示す．擬似マウスカーソルが画面内を動くことで，ユーザの注目すべき場所を指示する．アニメーション制御ウィンドウにコマンドのラベルを一覧表示することで，現在の位置を示したり実行の制御をする．デモンストレーションの実行スピードやポップアップメッセージの有無などの設定はユーザが自由に変更できる．

7. 議　　論

開発者が記録されたイベントを用いて記述したコマンドルールは，必ずしも実装に用いたイベント処理の方法と等価である必要はない．例として用いたグラフ編集アプレットでは，ノードの上でのドラッグは 3 つの異なる動作に使われている．1 つは，ノードの消去である．この操作はドラッグした後，ボタンを離した位置が画面外であれば，ドラッグを開始したノードを消去する．2 つめは，子ノードの生成である．ボタンを押した後，ノード底部を通ってノード外にドラッグしていった場合（以下，ドラッグアウトと表現する），新たなノードを生成して，それをボタンを押したノードの子として登録する．3 つめは，ノードの移動である．ノード底部以外からドラッグアウトした場合，ボタンを押したノードを離した位置へ移動する．これらの知識を厳密にコマンドルール内で定義するには，ノードのどの部分を通過して

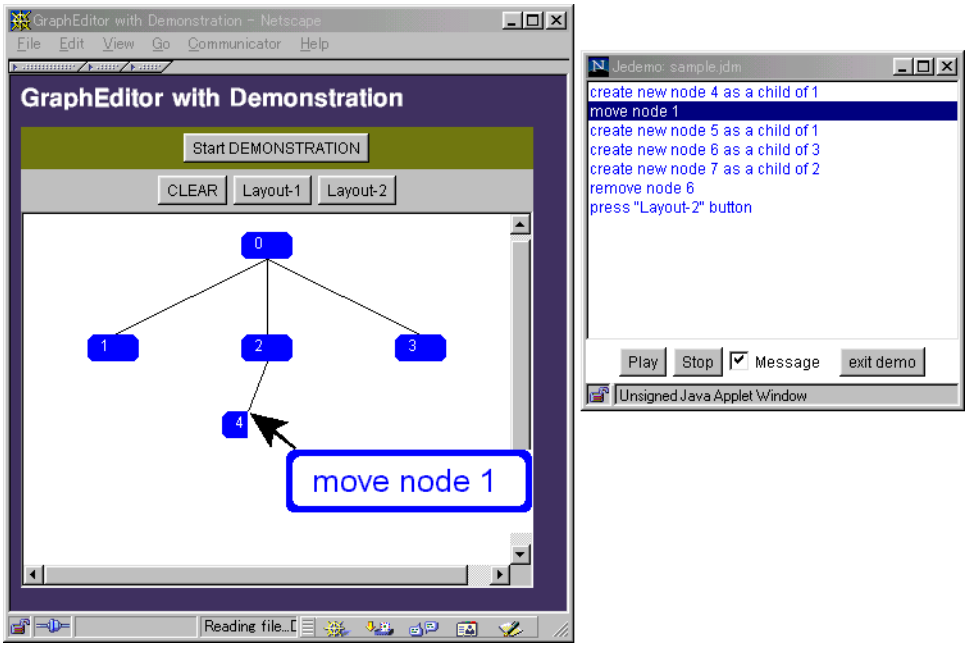


図 9 JedemoAnimator: ユーザによるアニメーションの実行
Fig. 9 JedemoAnimator: Playing animation help.

外部にドラッグアウトされたかを記録してやる必要がある．しかし，例では，ノードが消去されたときに発生するコンポーネントイベント C− があるか，新しいノードが作られたときに発生するコンポーネントイベント C+ があるか，それともコンポーネントイベントがないかによって区別している．このように，対象アプリケーションの振舞い（追加された部品の種類を取得し判別すること）を利用すれば，実装とは異なる方法でコマンドを特定することもできる．このことは複雑なジェスチャインタフェースを用いたシステムであってもコマンドの判別，生成は比較的容易に行なる可能性があることを意味している．

8. 関 連 研 究

操作を記録して，再利用しようとする試みは様々な Operating System やツールキット，またアプリケーション上で実装されている． Macintosh 上では操作をスクリプト化して再利用するものとして AppleScript[4] がある．ツールキットのレベルでは X Window System からイベントを送る方法[5] や， Tcl/Tk での実装[6] がある．

アプリケーションに特化しないデモンストレーション環境として， AIDE project[7] の AIDEWORKBENCH がある．これは， SmallTalk で作られたアプリケーションについてマクロ機能やアンドゥ機能を追加できる．

Java アプリケーションやアプレットのためのヘルプシステムとしては， JavaHelp[8] がある． JavaHelp では，全文検索などが可能なプラットフォームやブラウザに依存しないヘルプビューアや，文脈依存ヘルプのための API も用意しているが，アニメーションヘルプのような機能はない．

9. ま と め

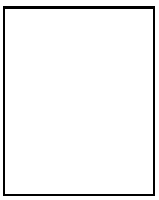
開発者が簡単にアニメーションヘルプを作成し，ユーザに見せるための枠組みとそのシステムについて述べた．アプレットビューアモデルを用いることでアプレットの機能を比較的簡単に拡張できるので，アニメーションヘルプをユーザに提示することが容易になる．また，開発者が対象アプレットの実装知識をコマンドルールとして保存し，再利用することで，アニメーションヘルプを簡単に生成できる．

参 考 文 献

1) Piyawadee Sukaviriya and James D. Foley. Coupling A UI Framework with Automatic Generation of Context-Sensitive Animated Help. In *Proceedings UIST '90*, pages 152–166, 1990.
2) Krishna Bharat and Piyawadee Sukaviriya. Animating User Interfaces Using Animation

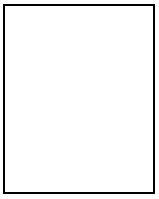
Servers. In *Proceedings UIST '93*, pages 69–79, 1993.
3) Motoki Miura and Jiro Tanaka. A Framework for Event-driven Demonstration based on the Java Toolkit. In *Asia Pacific Computer Human Interaction (APCHI-98)*, pages 331–336, July 1998.
4) Apple Computer,Inc. Introduction to the Macintosh Family — Second Edition.
5) Krishna Bharat, Piyawadee Sukaviriya, and Scott Hudson. Synthesized Interaction on the X Window System. Technical report, Graphics and Usability Center, Georgia Tech, USA, 1995.
6) Charles Crowley. TkReplay: Record and Replay for Tk. In *USENIX Tcl/Tk Workshop Toronto*, pages 131–140, http://www.cs.unm.edu/~crowley/papers/replay.tk95.html, July 1996.
7) Philippe P. Piernot and Marc P. Yvon. *The AIDE Project: An Application-Independent Demonstrational Environment*, chapter 18, pages 383–401. In "Watch What I Do : Programming by Demonstration," The MIT Press, 1993.
8) Sun Microsystems Inc. JavaHelp Homepage, http://java.sun.com/products/javahelp/.

(平成11年10月8日受付)
(平成11年10月8日採録)



三浦　元喜（学生会員）

1974 年生．1997 年筑波大学第三学群情報学類卒．1999 年筑波大学修士課程理工学研究科修了．修士（工学）．同年筑波大学博士課程工学研究科入学．グラフレイアウト，視覚化，ヘルプシステムに興味を持つ．情報処理学会，日本ソフトウェア科学会， ACM 各会員．



田中　二郎（正会員）

1951 年生．1975 年東京大学理学部卒．1977 年同大学院理学系研究科修士課程修了．1978 年から米国ユタ大学計算機科学科博士課程に留学．Ph.D. in Computer Science. 1993 年より筑波大学に勤務．現在，電子・情報工学系教授．

プログラミング言語やヒューマンインタフェースに興味を持つ．最近は，ビジュアルプログラミングや 3 次元インタフェースに関心を深めている．また，オブジェクト指向に基づく仕様からのアニメーションの作成や自動プログラムコード生成にも興味を持つ． ACM， IEEE Computer Society，日本ソフトウェア科学会，電子情報通信学会，人工知能学会，計測自動制御学会各会員．