筑波大学大学院修士課程

理工学研究科修士論文

アプレットのためのアニメーションヘルプ作成システム

三 浦 元 喜

平 成 11 年 2 月

筑波大学大学院修士課程

理工学研究科修士論文

アプレットのためのアニメーションヘルプ作成システム

著者氏名　　三浦 元喜

主任指導教官　電子・情報工学系 西川 博昭

目次

図一覧

表一覧

要旨

Java アプレットは，WWW ブラウザを用いることで誰にでも簡単に実行できる．しかし，その操作方法は文書で書かれることが多く，直感的に把握しにくい．本論文では，一般的な Java 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するのに，実際の操作を用いて簡単に作成するための方法について述べる．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アニメーションヘルプが生成されるため，編集作業が容易になり，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．また，ユーザはアニメーションヘルプを眺めることでシステムの概要を把握することができる．

第 1 章

序論

ネットワーク技術の向上，個人用計算機の普及などにより，インターネットの規模はこの数年で急激に拡大し，一般の人にも広く利用されている．インターネットを用いたサービスには，従来から電子メールやネットニュースなどが利用されていたが，最近特に利用されるようになってきたのが， WWW(World Wide Web) である．

WWW とは， Web ページと呼ばれる情報をハイパーリンクで連結したもので，ユーザは Web ブラウザを用いることで様々な情報を簡単に集めることができる．様々な情報としては文字，画像，動画，音楽，ソフトウェアなどがあるが，その中でも特に Java アプレット (applet) と呼ばれるプログラムは， Web ブラウザ上で動作するという特徴を持つ． Java アプレットは，特にユーザがダウンロード，インストールといった複雑な手順を踏む必要がなく，また様々な環境で動作するため， Web ページを閲覧するのと同様の手軽さで実行される．

Java アプレットの多くは， Web ページの一部として表示されるため， GUI (Graphical User Interface) を用いている．ユーザは，ボタンやメニュー，リストなどの画面上に表示された部品をマウスなどのポインティングデバイスを用いて操作することができる．また，より直感的な操作を可能とした「直接操作手法」などもアプレットの操作方法としてよく用いられる．

1.1 GUI の問題点

GUI は初心者のユーザにとっては親切なインターフェースであるといえる．なぜなら，メニューを見ればこのシステムが今何ができるのかをある程度理解することができ，それほど抵抗感を感じることなく操作できるからである．しかし，豊富な機能を持つアプリケーションにおいて GUI を用いると，以下のような弊害が生じる．

- 画面上のアイコンの数が多くなる．

- プルダウンメニューの項目が多くなる．

これらの結果として，必要な機能を探しだすことや，どんな機能が利用できるのかといったことを理解するのに時間がかかるという弊害が生じる．我々は，GUI を用いたアプリケーションにおいて，ヘルプ機能は必須であると考えている．

1.2 従来のヘルプの問題点

ここでは，GUI を用いたアプリケーションに適したヘルプについて考える．従来のヘルプは，文章 (テキスト) による記述が主流である．コマンド入力インターフェースを用いるアプリケーションでは文章によるヘルプでも問題はない．しかし，GUI を用いたアプリケーションの操作説明に文章を用いると，以下のような問題が発生する．

操作対象の認知的不和　GUI の操作説明において，操作対象は頻繁に指示される．例えば，

「編集」メニューの「オプション」を選択

という操作指示があったとする．ユーザはまずこの指示を読み，「編集」メニューがどこにあるかを探し，その後「オプション」を探して選ぶという手順を踏む．編集メニュー内にたくさん候補がある時は，探し出すのに苦労する．

さらに，アイコンボタンなど文字列によるラベル付けがされていない操作対象については指定すら困難である．このような場合は，例えば『はさみアイコン』というようにアイコンに描かれている「絵」を言葉で説明するしか方法はない．最近では，ハイパーテキストなどを利用し，ヘルプ文書内にアイコン画像を表示して説明する方法が一般的となっているが，結局ユーザはそのアイコンを見て，実際のアプリケーション画面内から同じものを探さなければならない．

操作の認知的不和　操作対象が特定できたとしても，それをどう操作するかを指示することは難しい．操作対象が「ボタン」であり，それは押すものと分かっていれば，マウスボタンを「押す」という行為は自然に行われる．しかし，この「メタファ」が理解されてない状態では，ユーザに正しく操作させることは難しい．特に直接操作手法を用いたアプリケーションでは，様々なメタファが用いられるので，マウスのドラッグ操作が多く用いられる．一見直感的に見える直接操作手法は，実は初心者ユーザに操作に慣れるまで試行錯誤を繰り返すことを強要しているともいえる．

第 2 章

アニメーションによるヘルプ

前章で述べた問題を解決するため，文章の代わりにアニメーションを用いてヘルプを提示する手法が提案されている [7, 26, 27, 2]．アニメーションによるヘルプでは，操作対象や操作をマウスカーソル (または，擬似的なポインタ) が動いて誰かが操作しているように見せることでユーザに操作方法を理解させる．ユーザは操作対象を探す必要がないため，短期間で理解することができる [22].

2.1 アニメーション手法

アニメーションを用いてヘルプを提示するには，提示する操作例をデータとして保存しなければならない．これらのデータを生成するにはアプリケーションにおける実際の操作を何らかの形式で記録するのが最も素直な手法である．実際の操作から生成するデータの記録・再生の手法として，以下の 2 つの手法が現実的なものとして考えられる．

動画像手法：アプリケーションの振舞いを動画像として記録　ムービーファイルなどの動画像として，アプリケーションの振舞いを記録する手法である．画像取り込みソフトを用いれば，比較的簡単に作成することができる．再現するのに動画像専用のビューアを用いる必要がある．アプリケーションのバージョン更新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．

イベント駆動手法：ユーザの操作をイベントとして記録　ユーザの操作をイベントとして記録しておき，再現するのにアプリケーションにそのイベントを送信することでアプリケーションの振舞いを見せる手法である．再現するのに特別なプログラムは必

要ない．そのかわり，イベントを送信する仕組みが必要となる．

我々は，ヘルプの編集のしやすさや再生時の柔軟性などの長所を考慮し，イベント駆動手法を採用する．この手法による付加的な長所としては，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当である．この手法を用いると， Thomas[8] らが提唱している Web を利用した視覚的でインタラクティブなアニメーション実行環境を簡単に実現できる．

2.2 イベント駆動手法における再生時の制限

イベント駆動手法では，アプリケーションにイベントを送信することで振舞いを再現するため，アプリケーションの状態がアニメーションに大きな影響を与える．任意の状態から記録したイベントを送信し始めると，その状態によって振舞いが異なるため，記録した操作の再現性は損われる．

再現性を満足するためには，再現をし始める前に，アプリケーションを記録を始める前の状態 (初期状態) に戻し，一貫性を保つ必要がある．少なくともヘルプシステムには「アプリケーションをいつでも初期状態に戻すことができる機能」と，「イベントを用いてアプリケーションに操作を再現させる機能」が要求される．

第 3 章

アプレットのためのアニメーションヘルプシステムの設計

イベントを用いてアニメーションを行うためにはシステムがユーザの操作を記録，再現する必要があることを前までに述べた．この章では，従来のアニメーションヘルプの問題点を提示した後，アプレットに特化しない汎用のアニメーションヘルプ環境の設計方針と実現方法について述べる．

3.1 従来の問題点

従来のアニメーションヘルプシステムが抱えている問題の１つとして，実装のための手間がかかるため，開発者がアニメーションヘルプを積極的に採用したがらないという現状があった．汎用のアニメーションヘルプシステムにするためにはシステムとヘルプ機能を完全に分離しなければならない．従来のシステムでアニメーションヘルプ機能を実装するには，少なくともシステムとアニメーションヘルプ機能モジュールとの結合を行うために，再コンパイルなどの作業が必要である．また，場合によってはシステム側からアニメーションヘルプ機能モジュールへの呼び出しを行うために，システムの変更を必要とする場合もある．このように，システムとヘルプ機能は分離されるべきであるが，密に連結されなければ機能しないという一見矛盾する性質を持ち合わせている．

我々もまた，アプレットに特化しない汎用性のため，アニメーションヘルプ機能を分離した．しかし，この機能をシステムと統合するときに発生する開発者の労力を最小に抑えるために，以下の設計上の工夫を行った．

3.2 設計方針

我々は，以下の 2 つの点を設計方針とした．

方針 1 ヘルプの対象となるアプレット (以下，対象アプレット) のソースコード，バイトコードには変更を加えない．また，再コンパイルも行わない．

方針 2 標準的な Java アプレットの実行環境 Java VM(Virtual Machine) には変更を加えない．

方針 1 を守ることで，開発者がアプレットにヘルプ機能を追加する際の手間を軽減することができると考えられる．**方針 2** を守ることで，現在標準的に普及している Web ブラウザ等の実行環境をそのまま利用できるため，ユーザがヘルプ機能を享受するのに特別な実行環境を必要としなくて済む．

我々は，以上の方針を考慮して，アプレットビューアモデルという新たな手法を考案した [17, 18]．従来，アプレットは Web ブラウザやアプレットビューアによって実行される．アプレットビューアモデルとは，ヘルプ機能を，アプレットビューアの機能を持つアプレットとして実現するモデルである．このモデルでは，ヘルプ機能を持つアプレット (以下ヘルプアプレット) は対象アプレットから見るとアプレットビューアとして認識され， Java VM から見た場合には一般的なアプレットとして認識される (**図 3.1参照**)．このヘルプアプレットは，対象アプレットを表示するだけでなく，対象アプレットの状態を調べたり，様々な操作を行うことができる．また，アプレットなので，一般に普及しているブラウザ上で動作する．

以下では，このアプレットビューアモデルを用いて，対象アプレットの操作を記録する方法について述べる．

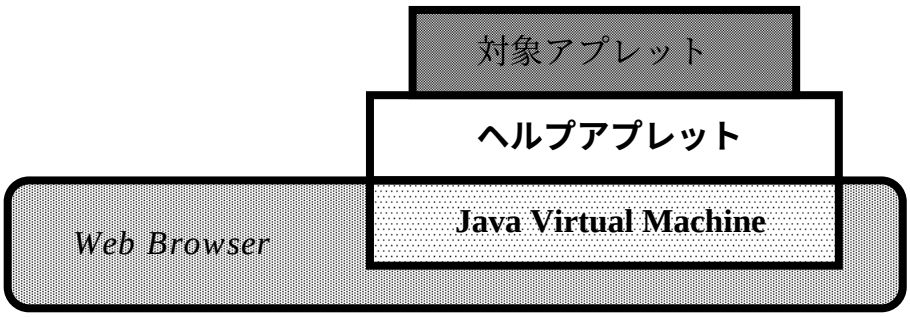

図 3.1: アプレットビューアモデル

3.3 対象アプレットの操作記録

3.3.1 部品

Java における部品とは，コンポーネントクラス (java.awt.Component) を継承するクラスを指す．部品はその大きさと，親の部品を持つ．親になることができる部品は，特にコンテナと呼ばれる．コンテナは，コンテナクラス (java.awt.Container) を継承している．コンテナは他の部品の大きさを調べ，それらを内部に配置することができる．

3.3.2 イベントリスナ

マウスやキーボードで行なわれた操作を記録するのに，上で述べた **方針 1** より，対象アプレットのソースコードを改変することはできない．そこで，我々は，イベントリスナ (Event Listener) と呼ばれる仕組みを用いて対象アプレットの操作を記録する．

イベントリスナとは， Java で用いられるイベント処理のためのオブジェクトである．イベントリスナを部品に登録しておくと，イベントリスナはその部品で発生したイベントを受けとって処理できる． 1 つの部品には複数のイベントリスナを登録しておくことができる．我々は，対象アプレットを構成する部品それぞれに，特別な記録専用のイベントリスナを登録することによって，マウスとキーボードの操作を記録する．この記録専用のイベントリスナは，対象となる部品で発生したマウスイベントやキーイベントを，ヘルプアプレットに通知するものである．以下にアクションイベントを記録する専用のイベントリスナのコードを示す．

```
public class ActionMessenger implements ActionListener{
  Recordable recorder;  // ヘルプアプレット
  public ActionMessenger(Recordable rec){  // コンストラクタ
    recorder = rec;
  }
  public void actionPerformed(ActionEvent e){  // イベントが発生すると呼ばれる
    recorder.recordEvent(e);  // 発生したイベントを記録する
  }
}
```

3.3.3 部品の階層構造

イベントリスナを部品に関連付けるには，対象アプレット (**図 3.2 参照**) を構成している部品のインスタンスを調べる必要がある．通常，部品のインスタンスは **図 3.3**

のように，階層構造を構成している．ただし，ここではインスタンスのクラス名を示している．ここで，[クラス名] のように矩形で囲んであるのはコンテナである．

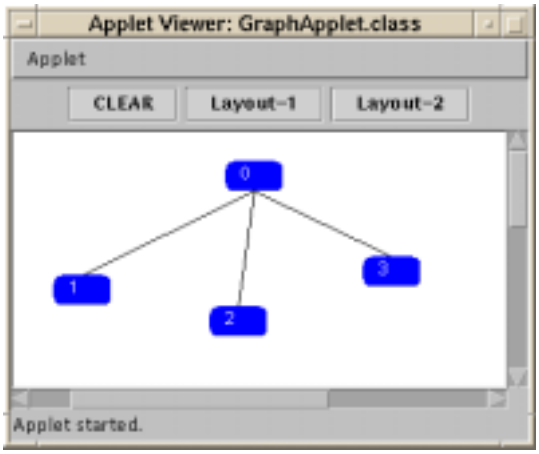

図 3.2: GraphApplet の画面

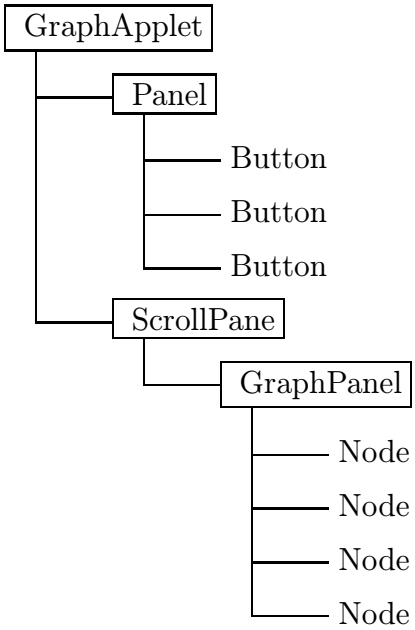

図 3.3: GraphApplet の部品階層構造

コンテナは，配置している部品を登録された順で記録している．通常，この登録順序の情報は画面の再描画の際に各部品を描画していく順番として用いられる．ヘルプアプレットは，対象アプレットから順にこの情報を調べていくことによって，部品の階層構造における位置を動的に取得することができる．取得できる情報 (部品管理情報) は，以下の 4 つである．

- 部品への参照
- 部品のクラス名
- 部品の階層構造における位置を示すパス情報 (**図 3.4** を参照)
- 部品の画面における座標位置と大きさ

これらの情報は，ヘルプアプレットが木構造を構成して管理する．「部品の階層構造における位置を示すパス情報」は，イベント実行時にオブジェクトを特定する情報として用いられる．

対象アプレットの操作を記録するには，以下の手順による準備を行う．

1. ヘルプアプレットは対象アプレットを初期化する (初期状態に戻す).

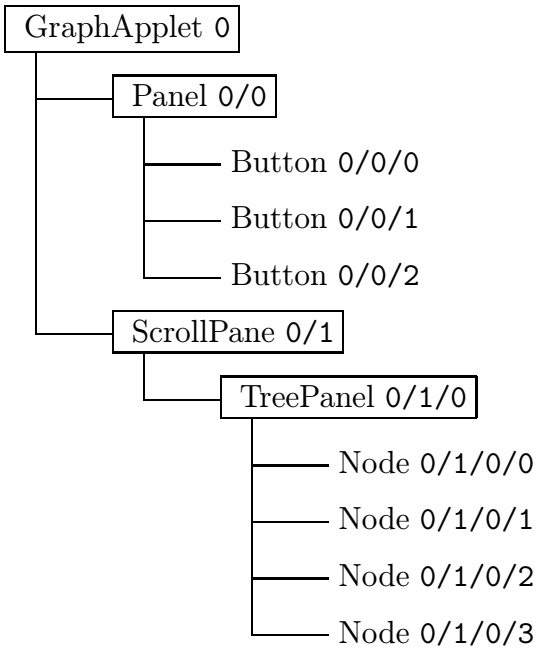

図 3.4: 部品の位置を示すパス情報

2. ヘルプアプレットは対象アプレットの部品管理情報を取得する．

3. 部品管理情報から，それぞれの部品のクラス名を取得し，登録できるイベントリスナを調べる．また，それぞれの部品への参照を用いて，そのイベントリスナを登録する．

この準備を行った後で，部品においてイベントが発生すると，その部品に登録されているイベントリスナに記述された処理が行われる．特別に追加したイベントリスナによって，イベントがヘルプアプレットに通知される．ヘルプアプレットはそれらのイベントを発生した順番で保存できる．

3.3.4 動的に登録される部品

対象アプレットによっては，部品が動的に作成されて登録 (追加) されることがある．例えば，**図 3.4** における Node などは，編集中に頻繁に作成される．それらの部品についても，発生したイベントを記録したいため，ヘルプアプレットは対象アプレットに部品が追加されたことを通知してほしい．また，部品が削除された場合も，ヘルプアプレットが管理している部品の階層構造を更新して一貫性を保つ必要があるため，同じく通知してほしい．

部品が追加，削除されたことをヘルプアプレットが知るために，我々はコンテナイベント (Container Event) を通知するイベントリスナ (以下コンテナイベントリスナ)

を用いる．コンテナイベントは，コンテナに新たな部品が追加されたときや，削除されたときに発行されるイベントで，

- 部品が追加されたか，削除されたかを示す情報
- 登録している部品に変更が起きたコンテナへの参照
- 追加または削除された部品への参照

を持つ．

記録を始める前に，ヘルプアプレットは対象アプレット内の部品のうち，すべてのコンテナにこのコンテナイベントリスナを登録する．そのコンテナに部品が追加，または削除されると，ヘルプアプレットにコンテナイベントが通知される．ヘルプアプレットはそのコンテナイベントを解釈して，以下の処理を行う．

1. 部品の階層関係を更新する．部品についての変更が起きたコンテについて，現在の部品構成を調べ，パス情報などの部品管理情報を更新する．
2. 部品の追加なら，イベントリスナを登録する．
3. 部品の削除なら，イベントリスナを解除する．

3.4 記録した操作の再現

3.3節で述べた方法により，対象アプレットで発生したイベントを取得することで操作を記録できた．本節では，これらのイベントを用いて，どのように操作を再現するか述べる．

3.4.1 イベントソースオブジェクト

すべての入力イベントは，「どの部品で発生したのか」を示す「イベントソースオブジェクト (Event-source object)」と呼ばれる情報を持つ．この情報は，そのイベントが発生した (生成された) 部品への参照である．

イベントを再現するには，イベントソースオブジェクトに対して

```
dispatchEvent(AWTEvent event)
```

というメソッドを呼ぶことで実現できる．ここで，`event` というのが再現したいイベントである．

イベントソースオブジェクトへの参照は，イベントを記録した時点では有効である．しかし，一旦対象アプレットを終了してしまうと，その参照は無効になる．なぜなら，

メモリ上の部品のアドレスは対象アプレットが読み込まれる毎に異なるからである．よって，開発者が記録したイベントのイベントソースオブジェクトへの参照はユーザがヘルプを見るときには無効となってしまい，正しく操作を再現できない．

この問題を解決するために，我々は先に述べた「パス情報」を用いる．イベントが発生したときに記録しておいたパス情報は，イベントを再現する際，対象アプレット内の部品から「イベントソースオブジェクト」を特定するために用いられる．

「パス情報」以外の方法としては，イベントが発生した位置 (座標) から，その位置にある部品を特定する「座標による方法」もある．しかし，「座標による方法」では，対象アプレット内の部品の座標がレイアウトなどで少し変更されただけでもイベントソースオブジェクトを正しく特定できないという問題がある．これに対して，「パス情報による方法」では，部品の位置が多少変更されても，プログラム中の部品の登録順が変更されない限り，イベントソースオブジェクトを正しく特定できる．また，たとえ対象アプレットの部品階層が変更されても，その変更部分のパス情報だけを書き替えることで以前に記録した操作を再現することが可能となり，柔軟性が高い．

第 4 章

アニメーションヘルプの作成

本章では，アニメーションヘルプのためのモデルであるコマンドと，その生成方法について述べる．

4.1 コマンド

これまでに述べたイベントを用いた手法で，アニメーションヘルプとしてのデモンストレーションを作成，実行することができる．

文章を推敲し，直しながらよりよいものにしていくように，アニメーションヘルプも編集できることが望ましい．記録中に誤った操作を消したり，順番を入れかえることは洗練されたアニメーションヘルプを作るために必要である．

しかし，イベントを単位としてアニメーションヘルプを編集するのは直感的ではない．なぜなら，アニメーションを構成するそれぞれのイベントは，「マウスボタンを押した」「キーボードの “M” を押した」など，比較的意味のない低レベルイベントがほとんどであり，それらを１つ１つ考慮して編集するのはそれほど意味がないからである．

そこで我々は，アニメーションヘルプ編集のために，「コマンド」という概念を導入する．「コマンド」は，イベントよりも上位の概念で，最低限の意味のある操作に対応する．例えば，「アイコンボタンを押す」「リストを選択する」「メニューを選択する」「フォームにメールアドレスを入力する」などがコマンドに該当する．

4.2 コマンドの生成

コマンドを生成するには，イベント列から意味のある部分を特定し，その意味を抽出する必要がある．

第 3章で述べたように，我々は，汎用性を高めるためヘルプシステム部 (ヘルプアプレット) と実際にヘルプを行うシステム (対象アプレット) とを完全に切り離して設計した．それゆえ，ヘルプアプレットは対象アプレットで発生したイベントを取得することはできても，そこで得られたイベント列が対象アプレットにおいてどんな作用を及ぼしたのか，その意味や効果を知ることはできない．例えば，同じドラッグイベントでも，対象アプレットによってその意味は異なる．また，同じ対象アプレットで発生したドラッグイベントでも，どの部品で行われたドラッグイベントかによって異なる可能性がある．ヘルプアプレットは対象アプレットからイベントの意味を逐次通知されれば可能だが，そのためにはヘルプアプレットへの参照を対象アプレットが持ち，なおかつ特別のインターフェースを備える必要がある．結果として対象アプレットのシステムに変更を加える必要があり，第 3章の方針に反する．

そこで我々は，対象アプレットで発生したイベント列が実際どのような意味を持つのかをヘルプアプレットに知識として与えることにした．その知識を用いることで，ヘルプアプレットはただ操作を記録するだけでなく，その意味を把握できる．

4.2.1 意味のあるイベント列の特定：セパレータ

対象アプレットから送られるイベント列には，意味を持つ部分列と意味を持たない部分列が存在する．意味を持たない部分列について，その意味を調べるのは効率が悪い．そこで，意味を持つ部分列と意味を持たない部分列を選り分けて，意味を持つ部分列のみについて調べることにする．

この選り分けのため，我々は「セパレータ」を用いる．セパレータは，２つの状態を持つ状態機械として記述できる．この２つの状態とは，「意味を持たないイベントを受け取っている状態」と，「意味を持つイベントを受け取っている状態」である．初期状態は前者である．入力されたイベントが「コマンド開始イベント」ならば，「意味を持つイベントを受け取っている状態」に遷移する．「意味を持つイベントを受け取っている状態」では，入力されたイベントをバッファに保管する．入力されたイベントが「コマンド終了イベント」ならば，「意味を持たないイベントを受け取っている状態」に遷移し，同時に今まで保管したバッファ内のイベント列について，意味を特定する．(意味の特定方法については 4.2.2節を参照．)

この２つの状態遷移は，入力イベントが「コマンド開始イベント」や「コマンド終了イベント」に含まれるかどうかの判定が引き金となる．この判定のために，それぞれ「コマンド開始イベント集合」「コマンド終了イベント集合」を用いる．セパレータは，この２つの集合によって定義される．

4.2.2　意味の特定：コマンドルール

セパレータによって選り分けられた，意味を持つ (または，その可能性のある) イベント列を「コマンド候補」と呼ぶことにする．コマンド候補の意味を特定するために，我々は「コマンドルール」を用いる．コマンドルールは，

- イベントパターン
- ラベル生成規則
- コマンド実行メソッド

の 3 つの要素を持つ．１つのコマンドルールが 1 種類のコマンドに対応する．

イベントパターンは，コマンド候補と照し合わせてこのコマンドルールの意味かどうかを判断するために用いる．もし，イベントパターンとコマンド候補が照合に失敗したら，別のコマンドルールとの照合を試みる．照合に成功したら，このコマンド候補が正式にコマンドとして保存され，ラベル生成規則によって意味 (ラベル) を文字列として付加する．また，コマンド実行メソッドによって対応するメソッドが付加される．この対応するメソッドは，ヘルプ再生の際に低レベルイベントを送信するかわりとして用いることができる．

第 5 章

実装システム

この章では，実際の例を用いて，アプレットのためのアニメーションヘルプ作成システム (以下，本システム) について説明する．

5.1 システム構成

本システムは， EventRecorder， EventAnalyzer， CommandRuleEditor， CommandGenerator， CommandPlayer という 5 つのシステムで構成されている．このうち， EventRecorder と CommandPlayer は Java アプレットで，その他は Java アプリケーションであり， JDK version 1.1 以上 + JFC Swing version 1.0.3 以上の環境で動作するよう実装されている．これらのシステムの間はイベントファイル，コマンドルールファイル，コマンドファイルの 3 つのファイル形式によってデータが受け渡される．**図 5.1** にそれらの流れを示す．

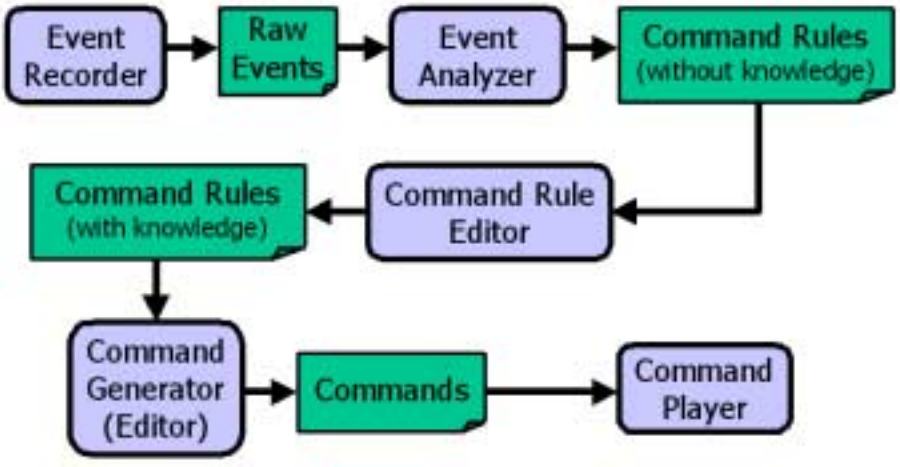

図 5.1: システム構成

5.2　対象アプレット

本システムを説明するために，まず対象アプレットについて説明する．**図 5.2** に，対象アプレットの例として用いたグラフ編集アプレット
(GraphApplet.class) の画面を示す．

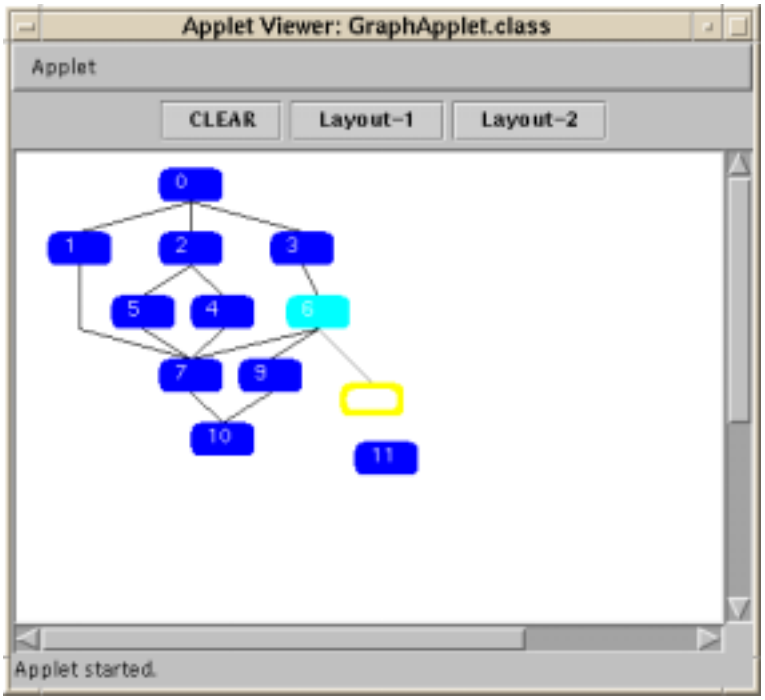

図 5.2: 対象アプレットとして用いたグラフ編集アプレット

編集はすべてマウスのクリックまたはドラッグによる直接操作を用いて行う．このグラフ編集アプレットの機能を以下に挙げる．

1. 新しいノードをつくる．キャンバス (画面上の白い部分) 上でクリックすると，どのノードにも連結していない新しいノードが作成される．

2. あるノードの子ノードをつくる．ある既存のノード上でクリックし，下方向にドラッグした後キャンバス上で離すと新しいノードが連結された子ノードとして作成される．**図 5.2** はこの操作におけるドラッグ中の画面である．**(図 5.3左も参照)**

3. ノードを移動する．移動したいノード上でクリックし，横方向または上方向にドラッグした後キャンバス上で離すと，ノードを移動できる． 2. と 3. はマウスカーソルがクリックしたノードから出たときの位置がノードの下部か，そ

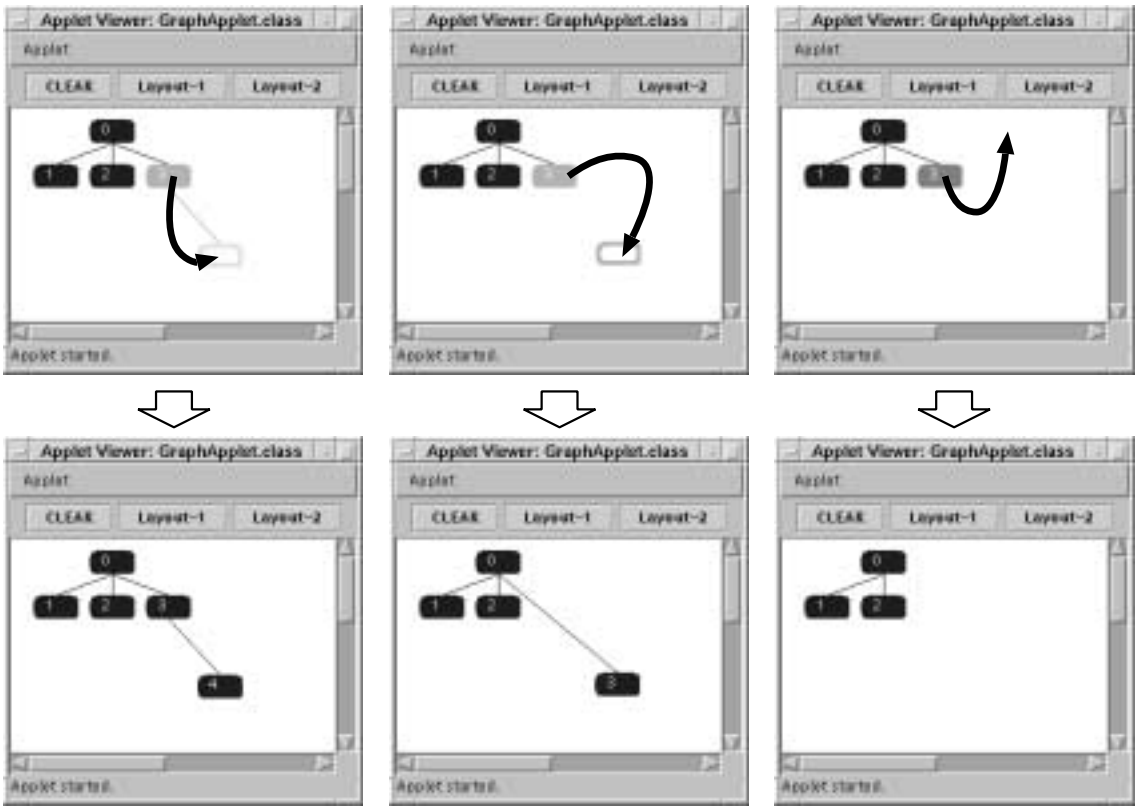

図 5.3: 対象アプレットの操作：ノードの作成 (2.)，移動 (3.)，削除 (4.)

うでないかの違いで決められるので，ノードを下に移動するときは一旦マウスを横方向に動かした後，下に動かすことで実現できる． **(図 5.3中を参照)**

4. ノードを削除する．消去したいノード上でクリックし，キャンバス上部で離すとそのノードと連結していたリンクが消える．ドラッグする方向は問わない．**(図 5.3右を参照)**

5. リンクを張る．親にあたるノード上でクリックし，そのままドラッグしながら子にあたるノード上で離す．ドラッグする方向は問わない．

6. リンクを消去する．子にあたるノード上でクリックし，そのままドラッグしながら親にあたるノード上で離す．ドラッグする方向は問わない．

7. レイアウトする． [Layout-1] または [Layout-2] ボタンをクリックする．レイアウト 1 と 2 はレイアウトアルゴリズムが異なるため，結果が異なる．

8. 初期状態に戻す． [CLEAR] ボタンをクリックすると初期状態に戻る．

それぞれのノードは部品として実装されているが，リンクはキャンバス上に描画しているので，部品としては扱われない．このアプレットの部品階層構造は 3.3.3 節 の **図 3.3** (p. 8) に示した通りである．**図 5.3**に対象アプレットのドラッグによる操作方法の一部を示す．このように，同じようなドラッグ操作でも異なる動作をするように割り当ててある．

5.3 EventRecorder

EventRecorder は，対象アプレットで発生したイベントを取得して，ファイルとして保存するアプレットである．以下に具体的な手順を示す．開発者が，ヘルプを作成しようとするアプレット (対象アプレット) の例として，ここではグラフ編集アプレット (GraphApplet) を用いて説明する．開発者は，この対象アプレットのクラス名を，アプレット EventRecorder のパラメータとして指定する．以下にその HTML 文書の一部を示す．

```
<applet code="EventRecorder.class">
  <param name="target" value="GraphApplet">
</applet>
```

上の HTML 文書を，ファイル名を指定して (ローカルの環境として) アプレットビューアで実行する． EventRecorder はパラメータで指定された GraphApplet クラスを読み込み，**図 5.4** のように画面内に表示する．ここで， “Record Start” ボタンを押すと，その時の部品構成を記録し，イベントリスナが各部品に追加され，記録を開始する．その後，開発者は対象アプレットが備える機能を一通り操作して，記録する． “Record Stop and Save as” ボタンを押すと，その時点までに記録したイベントをボタンの右にあるテキストフィールドに書かれた名前のファイルに保存する．ここでは， `sample.evf` という名前のファイルに保存している．

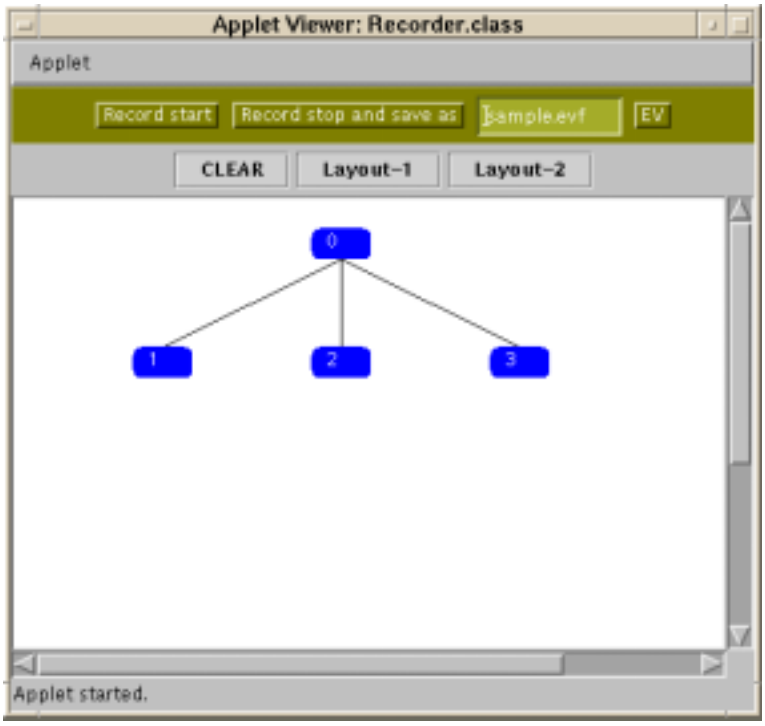

図 5.4: EventRecorder 上で実行されたグラフ編集アプレット

5.4 EventAnalyzer

EventAnalyzer は， EventRecorder で記録したイベントから，コマンドルールのイベントパターンを抽出する Java アプリケーションである．開発者は，実際に対象アプレットで発生したイベントを用いて，コマンドにあたる部分を指定する．

コマンドラインから，

```
% java Analyzer sample.evf
```

のように，イベントファイルを指定すると，**図 5.5** の画面が現れる．この画面は，記録されたイベント列をアイコン化して表示したものである．**表 5.1** にアイコンの一覧を示す．初期画面では，イベント列は区切られていないため， 1 列の長い「行」として表示されている．

イベントの種類	発生する条件	アイコン
MousePressed	マウスボタンが押されたとき	↘
MouseDragged	マウスがドラッグされたとき	_
MouseReleased	マウスボタンが離されたとき	↗
MouseClicked	押して離すまでの時間が十分短かいとき	▼
MouseEntered	マウスカーソルが部品に入ったとき	in
MouseExited	マウスカーソルが部品から出たとき	out
MouseMoved	マウスが動いたとき	·
ActionPerformed	ボタンが押されたとき	A
ComponentAdded	部品が追加されたとき	C+
ComponentRemoved	部品が削除されたとき	C−

表 5.1: イベントの種類と対応するアイコン

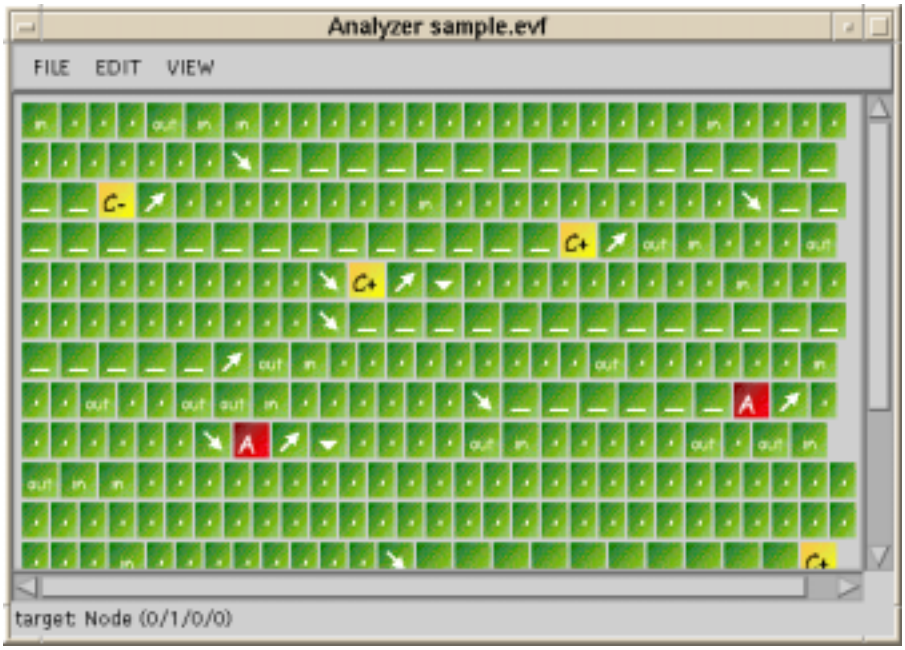

図 5.5: EventAnalyzer の初期画面

開発者はまず，セパレータを定義する．ここでは，「コマンド終了イベント集合」のみを定義し，それ以外のイベント要素を「コマンド開始イベント集合」とする，簡易定義を行っている．開発者は，**図 5.6** のように，セパレータの「コマンド終了イベント集合」となる範囲の始点と終点を指定して，[EDIT] メニューから [as a separator] を選択すると，範囲指定された部分に含まれるイベントを「コマンド終了イベント集合」として登録し，同時に記録されたイベント列に対してセパレータを適用する

ことができる．

セパレータを適用すると，**図 5.7** のように，セパレータ行とイベントアイコン行が交互に現れる．

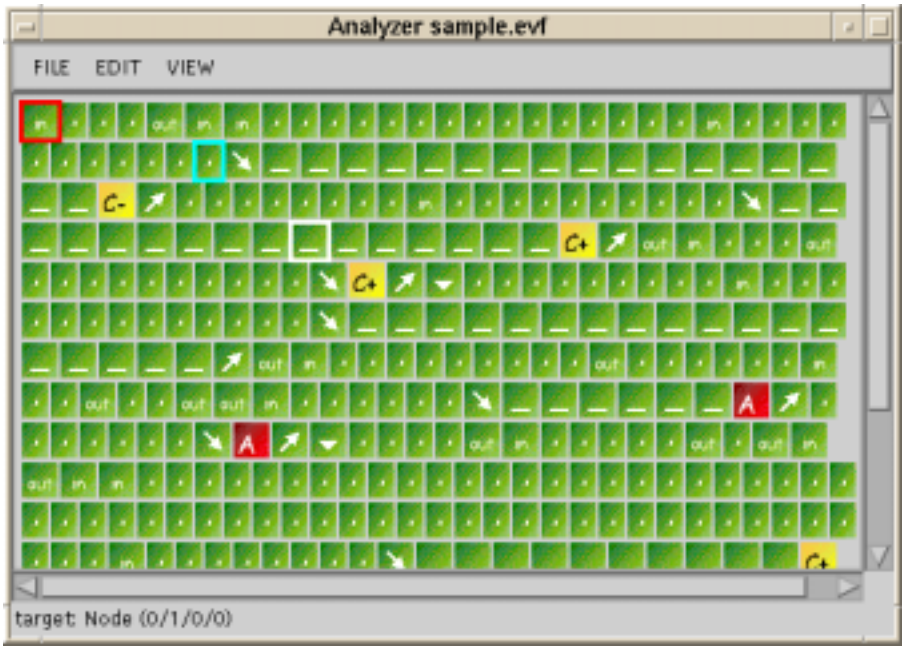

図 5.6: セパレータとしてマウス移動部分のイベントを指定

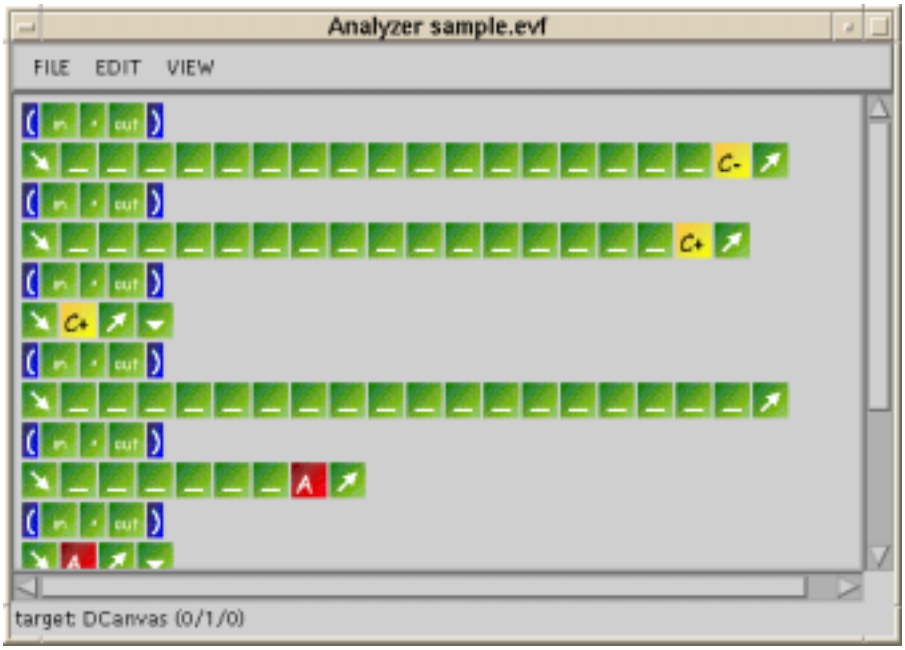

図 5.7: セパレータの適用後

この対象アプレットでは，イベントアイコン行中に現れるマウスドラッグイベントの個数によって意味は変わらない．そのため，開発者はこのイベントの個数を「1 個以上のイベント」として定義したい．**図 5.8** のように，あるドラッグイベント列を指定して，[EDIT] メニューの [as a command] を選択すると，**図 5.9** の画面になる．

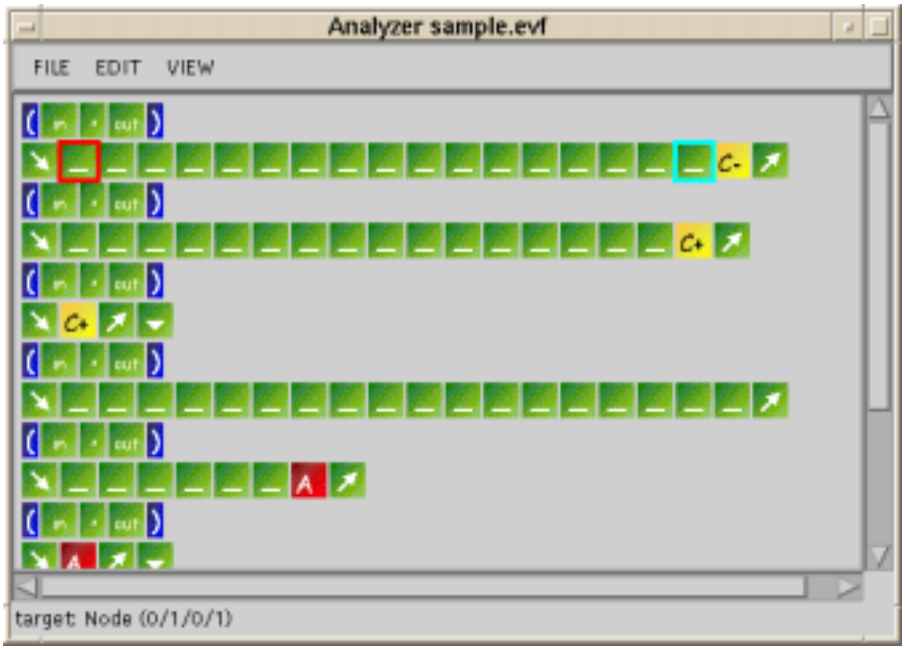

図 5.8: ドラッグイベントを指定

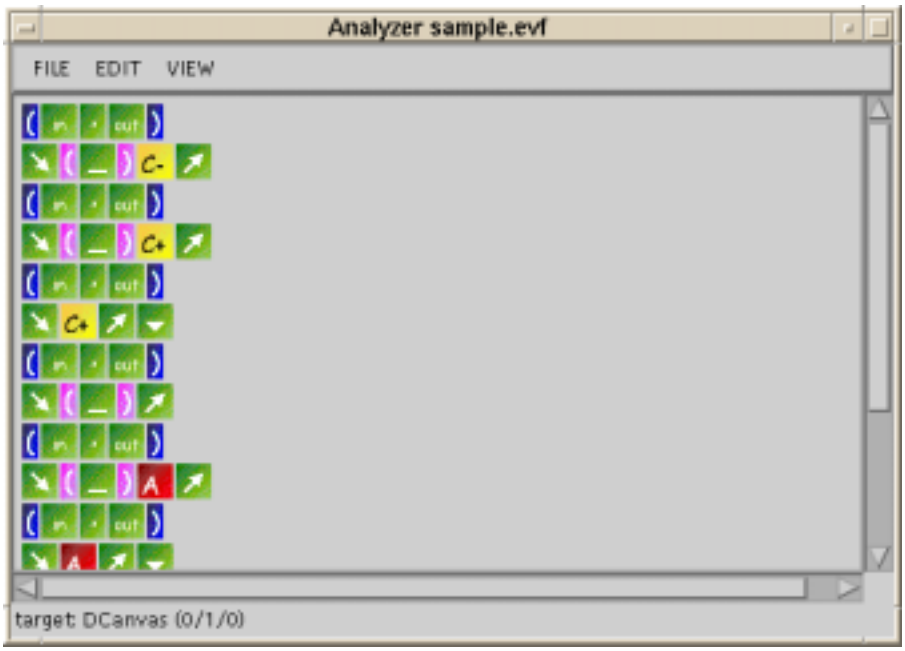

図 5.9: ドラッグイベントの 1 つ以上の繰り返しとして指定した画面

ここで，同じイベントアイコンの並びからなる行は 1 つのコマンドとして同じ意味を持つ．[VIEW] メニューから [commands] を選択すると，現在画面上にあるイベント行のうち，同じものを 1 つずつ集め，コマンドルールのイベントパターンとする．集められたイベントパターンはコマンドルールの要素として変換され，**図 5.10** のように CommandRuleEditor によって読み込まれる．

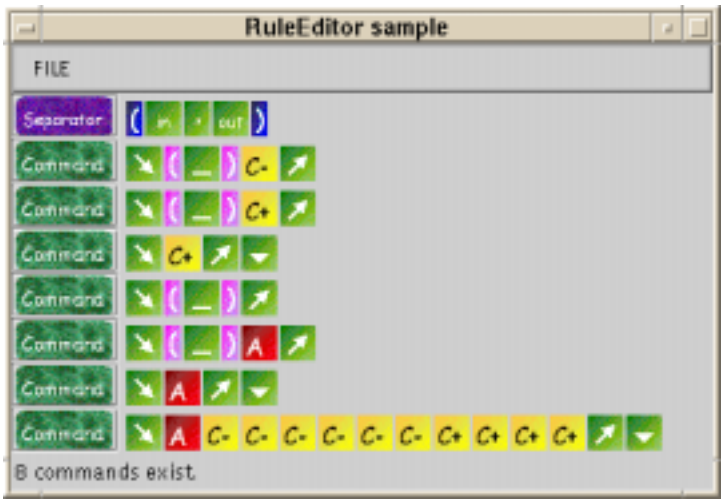

図 5.10: CommandRuleEditor 初期画面

5.5 CommandRuleEditor

CommandRuleEditor は，コマンドルールのイベントパターン，ラベル生成規則，コマンド実行メソッドを編集するアプリケーションである．開発者が，[Command] アイコンを押すと **図 5.11** のようにラベル生成規則とコマンド実行メソッドを入力することができる．

図 5.11: CommandRule を編集するウィンドウ

ラベル生成規則には，@ で始まる文字列を用いてメソッドを指定することで，動的

な情報を取得，挿入できる．**図 5.11** の例における，`@remove` の部分は，コンテナイベントで取り除かれた部品への参照を示し，`getLabel()` の部分によって，このメソッドを呼び出して，その返り値に置き替えることを示している．ただし，ここで指定するメソッドの返り値は整数値型か，文字列型でなければならない．@ で始まるオブジェクト指定 (動的なオブジェクト指定子) には，**表 5.2** のようなものがある．

指定子	オブジェクトの意味	参照に必要となるイベント
`@press`	マウスボタンを押したオブジェクト	MouseEvent ↘
`@release`	マウスボタンを離したオブジェクト	MouseEvent ↗
`@add`	追加したオブジェクト	ContainerEvent C+
`@remove`	削除したオブジェクト	ContainerEvent C−
`@added`	追加された親のオブジェクト (コンテナ)	ContainerEvent C+
`@removed`	削除された親のオブジェクト (コンテナ)	ContainerEvent C−
`@action`	アクションイベントが発生したオブジェクト	ActionEvent A

表 5.2: 動的なオブジェクト指定子

このメソッド指定の方法を用いることで，コマンド実行メソッドの部分も同様に指定できる．コマンドルールを編集した後の画面を **図 5.12** に示す．編集したコマンドルールはコマンドルールファイルとして保存する．

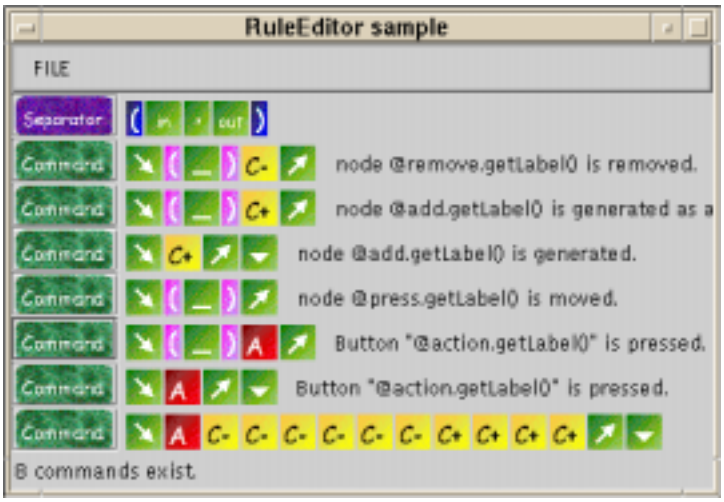

図 5.12: CommandRuleEditor ルール編集後の画面

5.6 CommandGenerator

実際のアニメーションヘルプを生成するアプリケーションである．コマンドルールファイルを指定すると，そのコマンドルールに関連付けられた対象アプレットが起動する．開発者が，対象アプレットで操作を行なうと，それによって発生したイベントが CommandGenerator に送られる． CommandGenerator は，送られたイベント列を指定されたコマンドルール内のセパレータによって区切り，抽出されたイベント列をコマンドルール内のイベントパターンと 1 つずつ照合していく．照合に成功した場合は，そのコマンドルールによってイベント列にラベルが付けられ，コマンドが生成される (**図 5.13** を参照)．すべてのコマンドルールとの照合に失敗したときは，そのイベント列は破棄され，コマンドは生成されない．

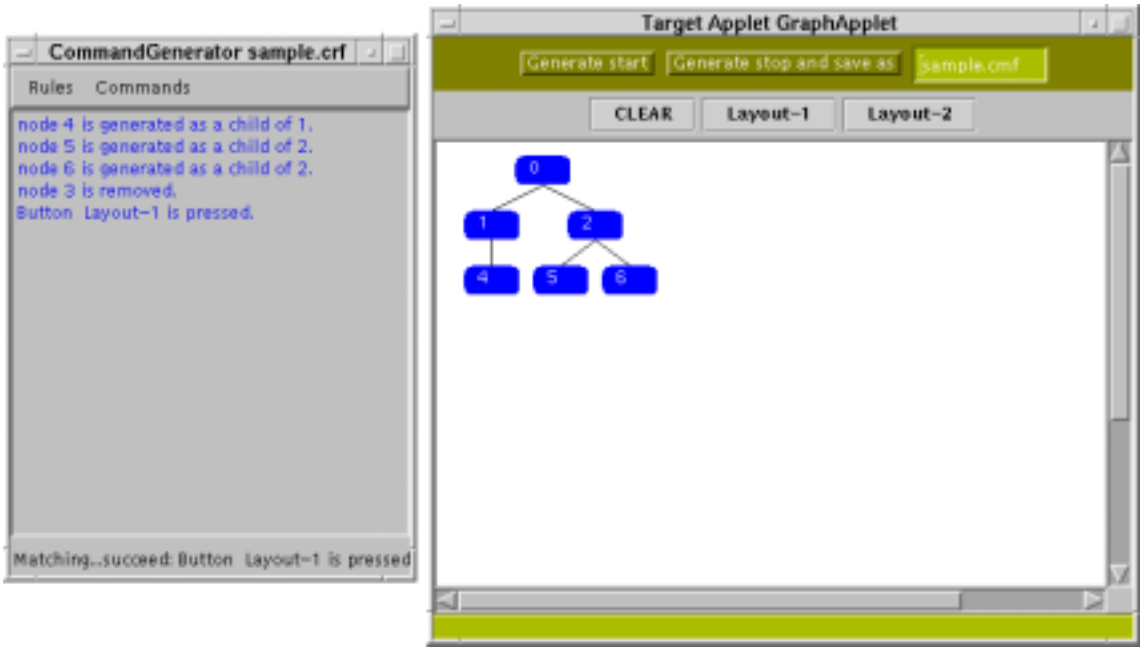

図 5.13: CommandGenerator の画面

5.7 CommandPlayer

CommandPlayer は，ヘルプを利用するユーザが用いるシステムであり，ヘルプアプレットに相当する．開発者は， EventRecorder の時と同様にアプレットタグ中のパラメータとして対象アプレットのクラス名と，ヘルプのためのコマンドファイルを指定する．指定した HTML ファイルの一部を以下に示す．

```
<applet code="CommandPlayer.class">
```

```
  <param name="target" value="GraphApplet">
  <param name="helpfile" value="sample.jdm">
</applet>
```

図 5.14 に，コマンドを実行している画面を示す．コマンドのラベルが一覧表示され，概要を把握できる．ユーザは，再生ボタンを押すだけで開発者の用意した操作を眺めることができる．ラベルの一覧から知りたい項目をダブルクリックすることによって，その項目から再生することもできる．また，擬似マウスカーソルやポップアップウィンドウによるメッセージによって，誰かが対象アプレットを操作しているかのような感覚を与える．イベントを送信する時間の間隔を変えることで，早送りなどの操作も可能である．

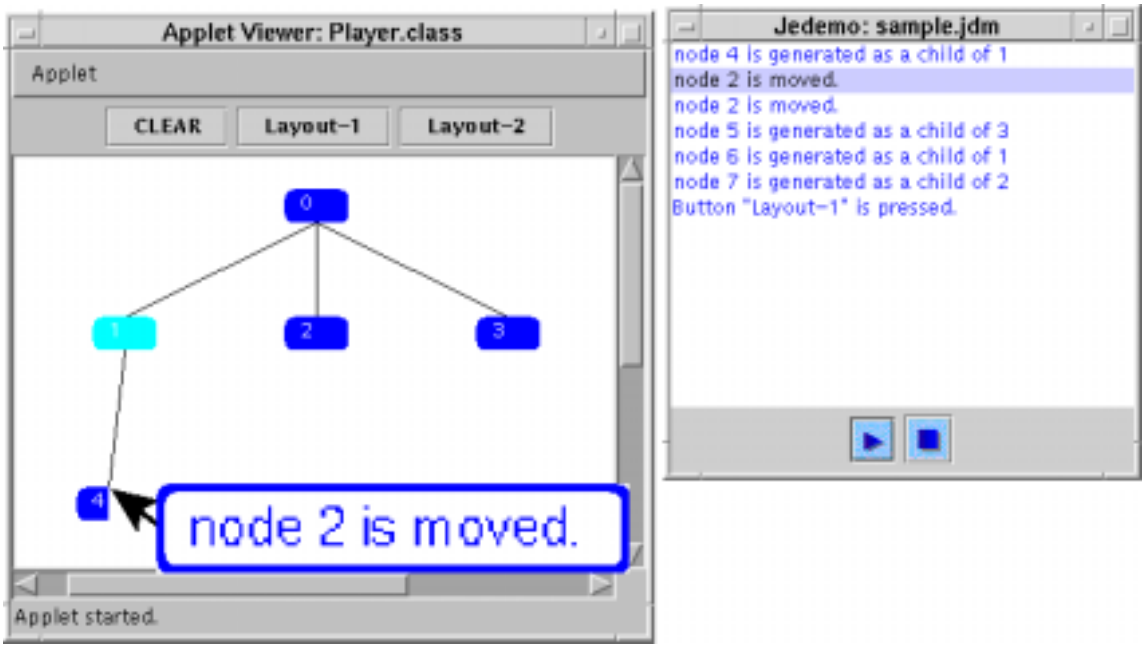

図 5.14: CommandPlayer の画面

第 6 章

議論

6.1 コマンドルールと対象アプレット実装との関係

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，必ずしも実装の方法と等価ではない．例として用いたグラフ編集アプレットでは，ノードの上でドラッグ操作を開始してキャンバス上で離すという操作に，**図 5.3**で示した 3 つの異なる動作 (ノードの作成，消去，移動) が割り当てられている．これらの実装にあたって，ノードの上下左右のうち，どの部分を通過してノード外部にドラッグしながら最初に出ていったかを記録している．しかし，実際のコマンドルールでは，

ノードの作成コマンドルール ＝ ↘ (_) C+ ↗

ノードの消去コマンドルール ＝ ↘ (_) C- ↗

ノードの移動コマンドルール ＝ ↘ (_) ↗

のように，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，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，それともコンポーネントイベントが「発生しない」かによって区別している．このように，定義を工夫すれば，複雑な実装を簡単な定義に置き換えて記述できる．

6.2 本システムが認識できるイベントの種類

本システムが認識できるイベントの種類 (区別できるイベントの数) は，以下の要因によって決定する．

- イベントクラス数 — ヘルプアプレットが対象アプレットに追加するイベントリスナの数によって決まる．
- 各イベントの状態の数 — 各イベントクラスに割り当てられている ID の数によって決まる．

また，イベントが持つソースオブジェクトのクラス名，パス情報によっても区別できる．ソースオブジェクトのクラス数は，対象アプレットに含まれるクラスの数によって決まる．ソースオブジェクトへのパス情報は，対象アプレットのインスタンス数によって決まる．これらの要素を考慮した照合は，必要なときに用いることができる．

ちなみに，本システムでは，上で述べたイベントを終端記号とする正規言語 (regular language) をコマンドとして認識できる．

6.3 問題点

本システムは，基本的にどのような対象アプレットであっても，イベントを記録し，コマンドルールを用いてコマンドを生成し，実行することができる汎用性を備えている．しかし，対象アプレットが以下のような事例にあてはまる場合には，うまく動作しないという問題がある．

記録できない事例 (1)　対象アプレットが新しいウィンドウを生成するとき，既存のウィンドウを親ウィンドウとしない場合，新しいウィンドウが表示されてもウィンドウイベントが発生しないため，ヘルプアプレットは感知できない．よって新しいウィンドウ内で発生したイベントは記録できない．

記録できない事例 (2)　対象アプレットが独自のイベントを定義し，それを用いて実装されているとき．独自のイベントを受けとるリスナを用意していないため，ヘルプアプレットがそのイベントを受けとることはできない．ただし，共通で用いられる低レベルイベントは受けとることができるので，それほど問題はない．

また，独自のイベントのためのイベントリスナが登録できる場合 (このような方法で実装するよう推奨されている)，ヘルプアプレットに追加するイベントのクラスを通知すれば記録できる．

コマンドルールがうまく定義できない事例　対象アプレットが単一の部品で構成されている場合 (**図 6.1右を参照**)．本システムでは，イベントに含まれるイベントが発生した部品のクラス名やパス情報を用いてコマンドを特定する．単一の部品内で「イメー

ジ」として扱われている画面オブジェクトを区別することができないので，コマンドルールがうまく定義できない．この点については，今後 JavaBeans などのコンポーネント化が進むにつれて，多くの部品を組み合わせて実装 (**図 6.1左**) する傾向にあるので，現実的な対象アプレットについてほぼ対応できると思われる．

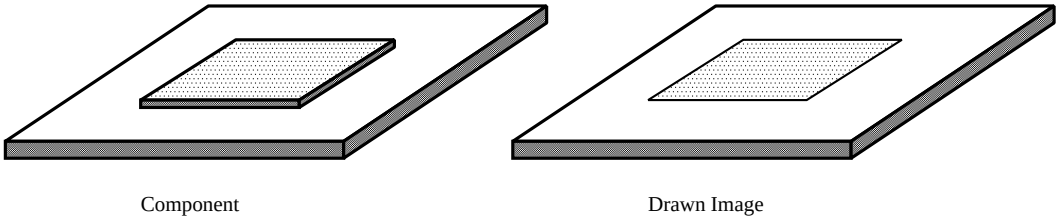

図 6.1: 部品とイメージ

うまく再現できない事例 (1)　部品が独自のイベント以外は受けとらない仕様になっている場合．独自のイベントを記録できないので，再現することができない．(このようなときは，代替メソッドによって再現できることもある．)

うまく再現できない事例 (2)　偶然性や時間経過によって状態変化が発生する場合．正しく再生するには記録した時と同じ初期状態からイベントを送信していく必要があるが，この初期状態が乱数によって毎回異なる場合や，スレッドなどを用いて変化させている場合には，記録した動作を正しく再現することができない．

第 7 章

関連研究

7.1 操作の履歴と再利用

ユーザの労力を軽減する目的で，頻繁に行われる操作を記録して再利用する方法 (マクロ) は， GNU Emacs[9] などのテキストエディタを対象に古くから行われている．また，表計算ソフト [16] など特定のアプリケーションにもマクロ機能を持ち，スクリプトとして実行できるものがある．これらのマクロ機能やスクリプトはアプリケーション内で閉じているため，複数のアプリケーションにまたがるような操作は記述できない． AppleScript [1] は複数の Macintosh アプリケーション (対応しているものに限る) を扱うマクロが記述できる．また， X Window System 上で動作するアプリケーションでは [3] があるが，実行中はマウスやキーボードの制御が占有されてしまうためユーザの操作ができないという欠点がある．そのため，最近ではツールキットとして操作履歴を管理するものが提案されている．アリゾナ大の Artkit[10] はマウスの動きからジェスチャを認識する仕組みを持つツールキットである．カーネギーメロン大ではアンドゥ機能を持つ独自のツールキット Amulet[21] を開発している． Tcl/Tk ツールキットにおける操作を記録，再生するシステムとして TkReplay[4] がある．

7.2 例示プログラミング

ユーザがアプリケーションの機能を操作例を示すことによって拡張する方法を一般に「例示プログラミング (Programming by Demonstration, Programming by Example)[6]」と呼ぶ．マクロを記録するときには，通常記録の開始と終了をシステムに指示し，ユーザは繰り返しなどを考慮して操作を行わなければならない．例示プログラミングでは，そのような指示なしでマクロを生成するなど，ユーザの負担を減らす工夫がなされて

いるものが多い．Eager[5] は HyperCard 上でのユーザの操作を監視し，同じ操作の繰り返しを検知するとユーザに操作の代行を提案する．図形エディタを対象としたものとして Metamouse[15] や Chimera[11]，Mondrian[12, 14] がある．

上に挙げたシステムは，特定のアプリケーションを対象としている．Triggers[24] は，画面上のピクセルデータを用いてマクロを生成する Macintosh 上のシステムであり，アプリケーションに特化しない．AIDE project [23] の AIDE WORKBENCH 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．

操作履歴を用いてユーザの負担を減らすのでなく，プログラマの負担を減らすことを目的としたシステムも少なくない．Tinker[13] は，Lisp のプログラミング初心者のためのシステムで，例示を用いてプログラムを記述できる．Peridot[19] や，その後継である Garnet[20] は，例示アクションを用いてユーザインターフェースを生成するシステムである．

Kosbie[25] はコマンドをさらに階層化することを提案している．コマンドの階層化については本研究にも必要であると思われるが，階層を深くしすぎるとユーザに混乱を招く恐れがあるため注意が必要である．

7.3 アニメーションヘルプ

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[28] が試験的に仕様公開されている．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 を用意しているが，アニメーションヘルプ機能はない．

Cartoonist[27] は，ユーザインターフェースの知識とアプリケーションの仕様を用いて状況に応じたアニメーションヘルプを動的に生成する仕組みを備えている．様々なアニメーション手法を用いているが，ヘルプ中にテキストを生成，表示する仕組みはない．

第 8 章

まとめ

アプレットのためのアニメーションヘルプ作成システムを設計，実装した．対象アプレットから取得したイベント列を記録，再生するための枠組みについて述べた．また，記録したイベント列に意味付けするための手法について考察した．

このシステムを用いることで，開発者にとってアプレットの効果的な操作説明を短時間で簡単に作成できる．コマンドルールを用いることで開発者が記録した操作を編集する手間を減らすことができ，バージョンの更新などによるシステムの変更にともなうヘルプの再構成も容易にできる．またユーザにとっても操作を再現することで直感的にアプレットの概要を把握できる．これらの技術はアプレットに限らず，アプリケーションにおいてもそのまま応用できるため，有用なものである．

謝辞

本研究を進めるにあたり，指導教官の西川博昭教授には細かい点まで配慮いただいた．また，田中二郎教授には研究内容についてのコメントを多数いただいた．この場を借りて感謝の意を表したい．

参考文献

[1] Apple Computer,Inc. Introduction to the Macintosh Family — Second Edition.

[2] Krishna Bharat and Piyawadee“Noi”Sukaviriya. Animating User Interfaces Using Animation Servers. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 69–79, 1993.

[3] Krishna Bharat, Piyawadee “Noi” Sukaviriya, and Scott Hudson. Synthesized Interaction on the X Window System. Technical Report 95-07, Graphics and Usability Center, Georgia Tech, USA, 1995.

[4] Charles Crowley. TkReplay: Record and Replay for Tk. In *USENIX Tcl/Tk Workshop Toronto*, pages 131–140, July 1996. http://www.cs.unm.edu/ ˜crowley/papers/replay.tk95.html.

[5] Allen Cypher. Eager : Programming Repetitive Tasks by Example. In *Proceedings of CHI*, pages 33–39, May 1991.

[6] Allen Cypher, editor. *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.

[7] Cullingford R. E., Krueger M. W., Selfridge M., and Bienkowski M. A. Automated Explanations as a Component of a Computer-Aided Design System. *IEEE Transactions on systems Man and Cybernetics*, 12(2):168–181, 1982.

[8] Thomas Naps et al. Using the WWW as the delivery mechanism for interactive, visualization-based instructional modules. In *ITiCSE'97 Working Group Reports and Supplemental Proceeding*, pages 13–26, 1997.

[9] Free Software Foundation. *GNU Emacs Lisp Manual — Version 18 2nd DRAFT —*. 株式会社ビレッジセンター出版局, 1992.

[10] Tyson R. Henry, Scott E. Hudson, and Gary L. Newell. Integrating gesture and snapping into a user interface toolkit. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 112–121, 1990.

[11] David Kurlander and Steven Feinter. A History-Based Macro By Example System. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 99–106, 1992.

[12] Henry Lieberman. Mondrian: A Teachable Graphical Editor. In [6], chapter 16, pages 341–358. The MIT Press, 1993.

[13] Henry Lieberman. Tinker: A Programming by Demonstration System for Beginning Programmers. In [6], chapter 2, pages 48–64. The MIT Press, 1993.

[14] Henry Lieberman. A demonstrational interface for recording technical procedures by annotation of videotaped examples. *International Journal of Human-Computer Studies*, 43:383–417, 1995.

[15] David L. Maulsby, Ian H. Witten, and Kenneth A. Kittlitz. Metamouse: Specifying Graphical Procedures by Example. In *Proceedings SIGGRAPH '89*, pages 127–136, 1989.

[16] Microsoft Inc. Microsoft office for windows95 standard edition.

[17] 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.

[18] Motoki Miura and Jiro Tanaka. Jedemo: The Environment of Event-driven Demonstration for Java Toolkit. In *International Symposium on Future Software Technology (ISFST)*, pages 215–218, October 1998.

[19] Brad A. Myers. Creating Dynamic Interaction Techniques by Demonstration. In *Proceedings CHI + GI '87*, pages 271–278, 1987.

[20] Brad A. Myers, Dario A. Giuse, Andrew Mickish, and David S. Kosbie. Making Structured Graphics and Constraints Practical for Large-Scale Applications. Technical Report CMU-CS-94-150, School of Computer Science, Carnegie Mellon University, May 1994.

[21] Brad A. Myers, Rich McDaniel, Rob Miller, Alan Ferrency, Patrick Doane, Andrew Faulring, Ellen Borison, Andy Mickish, and Alex Klimovitski. The Amulet Environment: New Models for Effective User Interface Software Development. Technical Report CMU-CS-96-189, School of Computer Science, Carnegie Mellon University, November 1996.

[22] Susan Palmiter and Jay Elkerton. An Evaluation of Animated Demonstrations for Learning Computer-based Tasks. In *Proceedings of CHI*, pages 257–263, May 1991.

[23] Philippe P. Piernot and Marc P. Yvon. The AIDE Project: An Application-Independent Demonstrational Environment. In [6], chapter 18, pages 383–401. The MIT Press, 1993.

[24] Richard Potter. Triggers: Guiding Automation with Pixels to Achieve Data Access. In [6], chapter 17, pages 361–380. The MIT Press, 1993.

[25] David S.Kosbie and Brad A. Myers. Extending Programming By Demonstration With Hierarchical Event Histories. In *Proceeding of East-West International Conference on Human-Computer Interaction EWCHI '94*, 1994.

[26] Piyawadee Sukaviriya. Dynamic Construction of Animated Help from Application Context. In *Proceedings of the ACM SIGGRAPH User Interface Software Symposium, Banff, Canada*, pages 190–202, 1988.

[27] Piyawadee "Noi" Sukaviriya and James D. Foley. Coupling A UI Framework with Automatic Generation of Context-Sensitive Animated Help. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 152–166, 1990.

[28] Sun Microsystems Inc. JavaHelp Homepage. http://java.sun.com/products/javahelp/.

付録 A

システムのソースリスト

本システムのソースを付録として添付する．本システムを構成するクラスの属するパッケージは以下の通りである．

- Package jedemo
  - Recordable (interface)
  - CompoModel
  - EventModel
  - Command
  - Content
  - CommandRule
  - ActionMessenger
  - ContainerMessenger
  - MouseMessenger
  - MouseMotionMessenger
  - WindowMessenger
- Package jedemo.record
  - Recorder
  - EventRecorder
- Package jedemo.analyze
  - Analyzer
  - EventAnalyzer
  - ViewPanel
  - RuleEditor
  - CommandRuleView
  - CommandRuleProperty
  - Generator
  - CommandGenerator
  - Filter
  - Sieve
  - Reductor
  - IconParade
- Package jedemo.play
  - Player
  - CommandPlayer
  - PlayController
  - MouseCursor
  - PopupWindow

package jedemo

interface jedemo.Recordable

```
package jedemo;

import java.awt.*;

/** イベントを受け取るオブジェクトが実装すべきインターフェース */
public interface Recordable {

/** Messenger が呼び出すメソッド
 * @param e 通知されたイベント
 */
  public void addEvent(AWTEvent e);
}
```

class jedemo.CompoModel

```
/* CompoModel
 * @target    JDK1.1+Swing1.0.3
 * @version   Sep 14, 1998
 * @author    Motoki Miura
 */
package jedemo;

import java.awt.*;
import java.io.Serializable;

/** コンポーネントのラッパークラス */
public class CompoModel implements Serializable {

  /** (transient なのでファイルに保存されない) Wrapping されたコンポーネント */
  transient Component compo;  // wrapped target object (component)
  /** コンポーネントへのパス */
  String comID;               // component ID
  /** コンポーネントのクラス名 */
  String target;              // target component longname

  public CompoModel(Component c,String path){
    compo = c;
    comID = path;
    target = c.getClass().getName();
  }
  public Component getCompo(){ return compo; }
  public String getComID(){ return comID; }
  public String getTarget(){ return target; }
  public void setComID(String comid){ comID = comid; }
}
```

class jedemo.EventModel

```
/* EventModel
 * @target    JDK1.1+Swing1.0.3
 * @version   Sep 14, 1998
 * @author    Motoki Miura
 */
package jedemo;

import java.awt.event.*;
import java.awt.AWTEvent;
import java.io.Serializable;
import com.sun.java.swing.tree.*;
import java.util.*;
import jedemo.CompoModel;

/** AWT イベントに足りない情報を補間するラッパークラス。 */
public class EventModel implements Serializable {

  /** Wrap するイベント */
  AWTEvent event; // one event object
  /** Component ID。 "/0/1/3" のように、イベントが発生したターゲットコンポーネントへのパス
を表す。 */
  String comID;  // component ID
  /** ターゲットコンポーネントのクラス名。 "java.awt.Component" のように、パッケージ名を含
む、全部の名前。 */
  String target; // target component longname
  /** Analyzer で表示するアイコンを特定する文字列。 現状では、
   * mm,md,mi,mo,mp,mr,mc,c+,c-,ac のどれかで、このクラスの
   * Private setIconID() で決められる。 setIconID() メソッドは
   * コンストラクタから呼ばれる。 */
  String iconID; // ID for Icon

  public EventModel(AWTEvent ev){
    event = ev;
    target = event.getSource().getClass().getName();
    setIconID();
  }

  public AWTEvent getEvent(){ return event; }
  public String getComID(){ return comID; }
  public String getTarget(){ return target; }
  public String getIconID(){ return iconID; }

  /** jedemo.record.EventRecorder から、イベント生成時に呼ばれる。 */
  public void setComID(Enumeration e){
    while(e.hasMoreElements()){
      DefaultMutableTreeNode t = (DefaultMutableTreeNode) e.nextElement();
      CompoModel c = (CompoModel) t.getUserObject();

      if (c.getCompo()==event.getSource()){
comID = c.getComID();
return;
      }
    }
    comID = "0/1/0/1";
  }

  /** コンストラクタから呼ばれる。自動的にアイコンを示す文字列
   * (mm,md,mi,mo,mp,mr,mc,c+,c-,ac のうちの 1 つ) がつけられる */
  private void setIconID(){
    int id = event.getID();
    if (event instanceof MouseEvent){
```

```
      switch(id){
      case MouseEvent.MOUSE_MOVED : iconID = "mm" ; break;
      case MouseEvent.MOUSE_DRAGGED : iconID = "md" ; break;
      case MouseEvent.MOUSE_ENTERED : iconID = "mi" ; break;
      case MouseEvent.MOUSE_EXITED : iconID = "mo" ; break;
      case MouseEvent.MOUSE_PRESSED : iconID = "mp" ; break;
      case MouseEvent.MOUSE_RELEASED : iconID = "mr" ; break;
      case MouseEvent.MOUSE_CLICKED : iconID = "mc" ; break;
      }
      return;
    }
    if (event instanceof ContainerEvent){
      switch(id){
      case ContainerEvent.COMPONENT_ADDED : iconID = "c+" ; break;
      case ContainerEvent.COMPONENT_REMOVED : iconID = "c-" ; break;
      }
      return;
    }
    if (event instanceof ActionEvent){
      iconID = "ac";
    }
  }
}
```

---

class jedemo.Command

```
/** Command.java
 *  @version Oct 16, 1998
 *  @author  Motoki Miura
 */

package jedemo;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import jedemo.*;
import jedemo.analyze.*;
import java.io.*;

/** コマンド。最低限意味のあるイベント列、その意味 (ラベル)、代替メソッドを持つ */
public class Command implements Serializable {

  /** イベントモデルの集合 */
  Vector eventmodels;
  /** ラベル */
  String label;
  /** ターゲットオブジェクトのクラス名 */
  String target;
  /** ターゲットオブジェクトへのパス */
  String comID;
  /** 実行再現メソッド */
  String method;

  /** コマンドを作成する。通常、コマンドを作成するには、このコンストラ
      クタを呼んだあと、 setLabel(), setMethod() メソッドを呼んで意味を
      つける。 */
```

```
  public Command(){
  }

  public void setEventmodels(Vector ems){ eventmodels = ems; }
  public void setLabel(String l){ label = l;  }
  public void setTarget(String classname){ target = classname; }
  public void setComID(String c){ comID = c; }
  public void setMethod(String m){ method = m; }
  public Vector getEventmodels(){ return eventmodels; }
  public String getLabel(){ return label;  }
  public String getTarget(){ return target; }
  public String getMethod(){ return method; }
  public String getComID(){ return comID;  }
  public String toString(){ return label; }
}
```

---

class jedemo.Content

```
/** Content.java
 *  @version Nov 6, 1998
 *  @author  Motoki Miura
 */

package jedemo;

import java.util.*;
import java.io.*;

/** コンテンツ。一連のデモンストレーション。 */
public class Content implements Serializable{

  /** タイトル */
  protected String title;
  /** 対象アプレットのクラス名 */
  protected String target;
  /** コマンドの集合 (列) */
  protected Vector commands;
  /** 実行前の初期化メソッド名 */
  protected String initMethod;

  public Content(String t, String tg, Vector coms, String im){
    title = t;
    target = tg;
    commands = coms;
    initMethod = im;
  }
  public String getTitle(){ return title; }
  public String getTargete(){ return target; }
  public Vector getCommands(){ return commands; }
  public String getInitMethod(){ return initMethod; }
}
```

---

class jedemo.CommandRule

```
/** CommandRule.java
 *  @version Oct 1, 1998
 *  @author  Motoki Miura
 */

package jedemo;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;
import jedemo.analyze.*;
import java.io.*;

/** コマンドルール。イベントのパターンマッチを行います。 */
public class CommandRule extends Filter implements Serializable {

/** イベントパターン */
  Vector rule;  // includes only "String," not EventModel
/** ラベル生成ルール */
  String label;
  String target;
/** メソッド */
  String method;
/** sep ならセパレータ、 cmd ならコマンドルール */
  String type;       // "sep" or "cmd"

  transient CommandRuleView crv;
  transient private boolean ismatch;
  transient private StringBuffer strbuf;  // for dynamic replacement
  transient Hashtable dlltable;
  transient private Point recentP; // for identify @release object precisely.

  public CommandRule(Vector v){
    rule = v;
    Object firstE = v.elementAt(0);
    type = "cmd";                       // determine type (sep or cmd)
    if (firstE instanceof String){
      String s = (String) firstE;
      if (s.equals("lb")) type = "sep";
    }
    crv = new CommandRuleView(this);
  }

  public void setView(){
    crv = new CommandRuleView(this);
  }
  public CommandRuleView getView(){
    return crv;
  }
  public void setLabel(String l){ label = l;  }
  public void setTarget(String classname){ target = classname; }
  public void setMethod(String m){ method = m; }
  public String getLabel(){ return label;  }
  public String getTarget(){ return target; }
  public String getMethod(){ return method; }
  public String getType(){ return type;  }
  public Vector getRule(){ return rule;  }
```

```
/** イベント列 (中身はすべて EM) がルールにマッチしたら、ラベルとメソッ
 * ドを入れたコマンドを生成して返す。マッチしなかったら null を返す
 * @param events イベント列 */
public Command generateCommand(Vector events){
  // match?
  if (!isMatch(rule,events)) return null;
  // add Label
  Command c = new Command();
  c.setLabel(createDynamicLabel(label,events));
  // add method
  c.setMethod(method);
  return c;
}

/** ルールとイベント列がルールとして一致するかどうか調べる。
 * @param r ルール
 * @param e イベント列
 */
private boolean isMatch(Vector r, Vector e){
  ismatch = false;
  Enumeration re = r.elements();
  Enumeration ee = e.elements();
  Object ro = re.nextElement();
  Object eo = ee.nextElement();
  if (isMeta(ro)) meta_match(re,ee,ro,eo);
  else normal_match(re,ee,ro,eo);
  return ismatch;
}

/** 通常のマッチングを行う */
private void normal_match(Enumeration re, Enumeration ee,
  Object ro     , Object eo){
  //    System.out.println("normal ro= "+getIconID(ro)); // test
  //    System.out.println("       eo= "+getIconID(eo)); // test

  if (!areEqualElements(ro,eo)) return;
  if (re.hasMoreElements()) {
    ro = re.nextElement();
    if (isMeta(ro)) meta_match(re,ee,ro,ee.nextElement());
    else normal_match(re,ee,ro,ee.nextElement());
  } else {
    if (ee.hasMoreElements()) return;
    else ismatch = true;
  }
}

/** 繰り返しなど、通常でない場合のマッチングを行う */
private void meta_match(Enumeration re, Enumeration ee,
Object ro,      Object eo){
  //    System.out.println("meta ro= "+getIconID(ro)); // test

  Vector collection = new Vector();
  String end = "r"+getIconID(ro).substring(1);
  while(true){  // collect elements in ( )
    ro = re.nextElement();

    if (getIconID(ro).equals(end)) break;
    else collection.addElement(ro);
    //      System.out.println("  collect= "+getIconID(ro)); // test
  }
  while(true){ // proceed events while exists
```

```
      //      System.out.println("       eo= "+getIconID(eo)); // test
      if (!isInclude(eo,collection)) {
// System.out.println("       break.");
break;
      }
      if (!ee.hasMoreElements())
if (!re.hasMoreElements()) {
  ismatch = true; return;
} else return;
      eo = ee.nextElement();
    }
    if (re.hasMoreElements()) {
      ro = re.nextElement();
      if (isMeta(ro)) meta_match(re,ee,ro,eo);
      else normal_match(re,ee,ro,eo);
    }
  }

  /** オブジェクトがメタ文字かどうか調べる */
  protected boolean isMeta(Object o){
    String type = getElementType(o);
    if (type.indexOf("E")==0) return false;
    if (type.equals("lb")) return true;
    if (type.equals("rb")) return true;
    if (type.equals("lB")) return true;
    if (type.equals("rB")) return true;
    if (type.equals("lp")) return true;
    if (type.equals("rp")) return true;
    if (type.equals("lP")) return true;
    if (type.equals("rP")) return true;
    return false;
  }

  /**
   * ラベルを返す。動的なものを含めたものを返す。
   * @param source コマンドルールのラベル生成ルール。
   * @param events 対応するイベント列
   */
  protected String createDynamicLabel(String source,
      Vector events){
    // retrieve dynamic object links (@press, @release, and so on)
    dlltable = getDynamicLinkTable(events);
    return getMethodToken(source+" ");
  }

  /** トークンに分解し、再帰呼び出しで動的情報を取得する */
  protected String getMethodToken(String source){
    // 開始点と終了点を明確にする
    // 開始点以前はバッファに入れてよい。終了点以後は持っている。
    // その中に他のメソッドがなければ、実行
    // あれば、再帰で呼ぶ。
    int s = source.indexOf("@");
    if (s == -1) return source;
    int e = source.indexOf(")",s);
    if (e == -1) return source;
    String stoken = source.substring(0,s);      // create link
    String mtoken = source.substring(s+1,e-1);  // press.getName()
    String etoken = source.substring(e+1);      // (rest)

    System.out.println("|"+stoken+"|"+mtoken+"|"+etoken+"|");
```

```
    if (mtoken.indexOf("@")!=-1) mtoken = getMethodToken(mtoken);
    else mtoken = callMethod(mtoken);
    if (etoken.indexOf("@")!=-1) etoken = getMethodToken(etoken);

    return stoken+mtoken+etoken;
  }

  /** トークンに書かれたメソッドを呼び、返り値を String で返す。 */
  protected String callMethod(String token){
    System.out.println("token: "+token);
    StringTokenizer tokenizer = new StringTokenizer(token,".()",false);
    String target = null, method = null;
    if (tokenizer.hasMoreElements()) target = tokenizer.nextToken();
    else return "error target";
    if (tokenizer.hasMoreElements()) method = tokenizer.nextToken();
    else return "error method";
    Object obj = null;
    try{
      Object tarobj = dlltable.get(target);
      Class[] args = {};
      Object[] objargs = {};
      Method mez = tarobj.getClass().getMethod(method,args);
      obj = mez.invoke(tarobj,objargs);
    }catch(Exception ex){
      System.out.println(ex.toString());
    }
    return String.valueOf(obj);
  }

  /** 動的な情報のためのタグ (@press,@action etc.) を得る
   * @param ems イベント列
   */
  protected Hashtable getDynamicLinkTable(Vector ems){
    Hashtable table = new Hashtable();
    for (Enumeration e = ems.elements();e.hasMoreElements();){
      EventModel em = (EventModel) e.nextElement();
      String type = em.getIconID();
      /*      if (type.equals("md")) {
MouseEvent me = (MouseEvent) em.getEvent();
recentP = new Point(me.getX(),me.getY());
Component c = (Component) me.getSource();
Point pressP = c.getLocation();
Container ct = (Container) c.getParent();
recentP = new Point(recentP.x + pressP.x, recentP.y + pressP.y);
System.out.println(recentP.toString() + me.getSource().toString() + pressP.toString());
      }*/
      if (type.equals("mp")) table.put("press",em.getEvent().getSource());
      if (type.equals("mr")) table.put("release",em.getEvent().getSource());
      // ((Container)em.getEvent().getSource()).getParent().getComponentAt(recentP));
      if (type.equals("c+")) {
table.put("addbase",em.getEvent().getSource());
table.put("add",((ContainerEvent) em.getEvent()).getChild());
      }
      if (type.equals("c-")) {
table.put("removebase",em.getEvent().getSource());
table.put("remove",((ContainerEvent) em.getEvent()).getChild());
      }
      if (type.equals("ac")) table.put("action",em.getEvent().getSource());
    }
    return table;
  }
```

```
}
```

class jedemo.ActionMessenger

```
package jedemo;

import java.awt.event.*;

/** アクションイベントを通知する特殊なイベントリスナ */
public class ActionMessenger implements ActionListener{

/** 通知先 */
  Recordable erec;

/** @param erec 通知先 */
  public ActionMessenger(Recordable erec){
    this.erec = erec;
  }

/** アクションイベントが発生すると実行される */
  public void actionPerformed(ActionEvent e){
    erec.addEvent(e);
  }
}
```

class jedemo.ContainerMessenger

```
package jedemo;

import java.awt.event.*;

/** コンテナイベントを通知する特殊なイベントリスナ */
public class ContainerMessenger implements ContainerListener{

/** 通知先 */
  Recordable erec;

/** @param erec 通知先 */
  public ContainerMessenger(Recordable erec){
    this.erec = erec;
  };

/** 部品が追加されると実行される */
  public void componentAdded(ContainerEvent e){
    erec.addEvent(e);
  }

/** 部品が削除されると実行される */
  public void componentRemoved(ContainerEvent e){
    erec.addEvent(e);
  }
}
```

class jedemo.MouseMessenger

```
package jedemo;

import java.awt.event.*;

/** マウスイベントを通知する特殊なイベントリスナ */
public class MouseMessenger extends MouseAdapter {

/** 通知先 */
  Recordable erec;

/** @param erec 通知先 */
  public MouseMessenger(Recordable erec){
    this.erec = erec;
  }

/** マウスクリックイベントが発生すると実行される */
  public void mouseClicked(MouseEvent e){
    OutputLog(e,"V");
  }
/** マウス押下イベントが発生すると実行される */
  public void mousePressed(MouseEvent e){
    OutputLog(e,"\\");
  }
/** マウスボタンが離されると実行される */
  public void mouseReleased(MouseEvent e){
    OutputLog(e,"/");
  }
/** マウスカーソルが部品に入ると実行される */
  public void mouseEntered(MouseEvent e){
    OutputLog(e,"i");
  }
/** マウスカーソルが部品から出ると実行される */
  public void mouseExited(MouseEvent e){
    OutputLog(e,"o");
  }
/** 通知する */
  void OutputLog(MouseEvent e,String s){
    erec.addEvent(e);
  }
}
```

---

class jedemo.MouseMotionMessenger

```
package jedemo;

import java.awt.event.*;

/** マウスイベントを通知する特殊なイベントリスナ */
public class MouseMotionMessenger extends MouseMotionAdapter {

/** 通知先 */
  Recordable erec;

/** @param erec 通知先 */
  public MouseMotionMessenger(Recordable erec){
    this.erec = erec;
  }

/** マウスドラッグイベントが発生すると実行される */
```

```
  public void mouseDragged(MouseEvent e){
    OutputLog(e,"_");
  }
/** マウスが動くと実行される */
  public void mouseMoved(MouseEvent e){
    OutputLog(e,".");
  }
/** 通知する */
  void OutputLog(MouseEvent e,String s){
    erec.addEvent(e);
  }
}
```

class jedemo.WindowMessenger

```
package jedemo;

import java.awt.event.*;

/** ウィンドウイベントを通知する特殊なイベントリスナ */
public class WindowMessenger implements WindowListener{

/** 通知先 */
  Recordable erec;

/** @param erec 通知先 */
  public WindowMessenger(Recordable erec){
    this.erec = erec;
  }

/** ウィンドウがアクティブになったとき実行される */
  public void windowActivated(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウがクローズされたとき実行される */
  public void windowClosed(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウがクローズ中のとき実行される */
  public void windowClosing(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウがアクティブではなくなったとき実行される */
  public void windowDeactivated(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウが非アイコン化されたとき実行される */
  public void windowDeiconified(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウがアイコン化されたとき実行される */
  public void windowIconified(WindowEvent e){
    erec.addEvent(e);
  }
/** ウィンドウがオープンされたとき実行される */
  public void windowOpened(WindowEvent e){
    erec.addEvent(e);
  }
}
```

package jedemo.record

class jedemo.record.Recorder

```
/* Recorder
 * Author: Motoki MIURA
 * Date: Sep 9, 1998
 */

package jedemo.record;

/** イベントを記録するベースアプレット
 *
 * usage: appletviewer example.html
 *
 * [part of example.html]
 * <applet codebase="./jedemo" code="Recorder.class" width=500 height=400>
 * <param name="target" value="GraphApplet">
 * <param name="output" value="sample.evf">
 * </applet>
*/

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import jedemo.record.*;

public class Recorder extends Applet {

  Applet app;
  EventRecorder erec;
  Button start,stop,view;
  TextField name;

  public void init(){
    String target = getParameter("target");
    String output = getParameter("output");
    if (output == null) output = "sample.evf";

    setLayout(new BorderLayout());

    try{
      app = (Applet) Class.forName(target).newInstance();
    }catch(ClassNotFoundException e){
      System.out.println("No such class: "+target);
    }catch(IllegalAccessException e){
      System.out.println("IllegalAccess: "+target);
    }catch(InstantiationException e){
      System.out.println("Cannot instantiate: "+target);
    }

    add(app,BorderLayout.CENTER);
    //    System.out.println("target: "+target);

    Panel controlp = new Panel();
    controlp.add(start = new Button("Record start"));
    controlp.add(stop = new Button("Record stop and save as"));
    controlp.add(name = new TextField(output,20));
```

```
    controlp.add(view = new Button("EV"));
    controlp.setBackground(new Color(120,120,20));
    controlp.setForeground(Color.white);

    add(controlp,BorderLayout.NORTH);

    app.init();

    erec = new EventRecorder();

    start.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
erec.registerTarget(app);
erec.goInteractiveMode(name.getText());
      }
    });
    stop.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
erec.saveEventModels(name.getText());
erec.stopRecording();
      }
    });
    view.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
erec.goInteractiveMode(name.getText());
      }
    });

  }
}
```

---

class jedemo.record.EventRecorder

```
/* EventRecorder.java
 * Author: Motoki MIURA
 * Date: Oct 1, 1998
 * Last: Nov 3, 1998
*/
package jedemo.record;

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.tree.*;
import com.sun.java.swing.AbstractButton;
import jedemo.*;
import jedemo.analyze.EventAnalyzer;

/** Recorder から起動される機能クラス。 GUI は持たない。 */
public class EventRecorder implements Recordable {

  /** Target applet for help */
  static Applet target;
  /** コンテナイベントを通知するエージェント */
  private ContainerMessenger cm;
```

```
  /** アクションイベントを通知するエージェント */
  private ActionMessenger am;
  /** マウスイベントを通知するエージェント */
  private MouseMessenger mm;
  /** マウスモーションイベントを通知するエージェント */
  private MouseMotionMessenger mmm;

  /** Stores events as EventModel instances */
  Vector eventList = new Vector();      // includes EventModel

  /** コンポーネント構造木のルート */
  protected DefaultMutableTreeNode root = new DefaultMutableTreeNode();
  /** It enables to reject MouseEnter/Exit while mouse-dragging.  It
      is used for keep compativility between WIN and UNIX */
  protected boolean skipIO;
  /** Whether records interactive or non-interactive(batch-record) */
  private boolean isInteractiveMode = false;
  /** It is used at interactive mode */
  protected EventAnalyzer eana;

  public EventRecorder(){
    cm = new ContainerMessenger(this);
    am = new ActionMessenger(this);
    mm = new MouseMessenger(this);
    mmm = new MouseMotionMessenger(this);
  }

  /** Switchover to interactive mode from batch-mode.  Specify
      event-file name */
  public void goInteractiveMode(String evfname){
    eana = new EventAnalyzer(eventList, root,
     target.getClass().getName(), evfname);
    isInteractiveMode = true;
    eana.show();
  }

  /** Each EventMessenger calls this method. */
  public synchronized void addEvent(AWTEvent e){
    // for compativility WIN and UNIX (avoid in/out event during dragging)
    if (e.getID() == MouseEvent.MOUSE_PRESSED) skipIO = true;
    if (e.getID() == MouseEvent.MOUSE_RELEASED) skipIO = false;
    if (skipIO)
      if (e.getID() == MouseEvent.MOUSE_ENTERED ||
  e.getID() == MouseEvent.MOUSE_EXITED) return;

    EventModel em = new EventModel(e);
    em.setComID(root.depthFirstEnumeration());
    // batch recording
    eventList.addElement(em);
    if (e instanceof ContainerEvent){
      ContainerEvent ce = (ContainerEvent) e;
      if (e.getID() == ContainerEvent.COMPONENT_ADDED)
addDynamicComponent(ce);
      if (e.getID() == ContainerEvent.COMPONENT_REMOVED)
removeDynamicComponent(ce);
    }
    if (isInteractiveMode) {
      eana.validate(); eana.repaint(); }
    else System.out.println(e.toString());
  }
```

```
/** This method is called by addEvent method when the component is added. */
protected void addDynamicComponent(ContainerEvent ce){
  addListeners(ce.getChild());               // 新部品にリスナを追加
  Container parent = ce.getContainer();
  DefaultMutableTreeNode t = getTreeNode(parent);  // 親ノードのモデル検索
  int index = t.getChildCount() + 1;
  String tid = ((CompoModel) t.getUserObject()).getComID();
  t.add(new DefaultMutableTreeNode
(new CompoModel(ce.getChild(),tid+"/"+String.valueOf(index))));
}

/** This method is called by addEvent method when the component is removed. */
protected synchronized void removeDynamicComponent(ContainerEvent ce){
  Container parent = ce.getContainer();
  DefaultMutableTreeNode t = getTreeNode(parent);
  t.removeAllChildren();
  getComponentTreeNoAddListener(parent,t);
  System.out.println(((CompoModel) t.getUserObject()).getComID());
  Enumeration enu = t.children();
  while (enu.hasMoreElements()){
    DefaultMutableTreeNode tre = (DefaultMutableTreeNode) enu.nextElement();
    System.out.println(((CompoModel) tre.getUserObject()).getComID());
  }
}

/** This method is called by addDynamicComponent method. */
DefaultMutableTreeNode getTreeNode(Component c){//lookfor treenode from comp
  Enumeration e = root.depthFirstEnumeration();
  while(e.hasMoreElements()){
    DefaultMutableTreeNode t = (DefaultMutableTreeNode) e.nextElement();
    CompoModel cm = (CompoModel) t.getUserObject();
    if (cm.getCompo() == c) return t;
  }
  return null;
}

/** Stop recording.  This method is invoked by Recorder's button. */
public void stopRecording(){
  Enumeration e = root.breadthFirstEnumeration();
  while(e.hasMoreElements()){
    DefaultMutableTreeNode n = (DefaultMutableTreeNode)e.nextElement();
    CompoModel cm = (CompoModel) n.getUserObject();
    Component comp = (Component) cm.getCompo();
    removeListeners(comp);
  }
  if (!isInteractiveMode) {
    eventList.removeAllElements();
    root.removeAllChildren();
  }
}

private void removeListeners(Component o){
  o.removeMouseListener((MouseListener)mm);
  o.removeMouseMotionListener((MouseMotionListener)mmm);
  if (o instanceof Container)
    ((Container)o).removeContainerListener((ContainerListener)cm);
  if (o instanceof AbstractButton)
    ((AbstractButton)o).removeActionListener((ActionListener)am);
  if (o instanceof Button)
    ((Button)o).removeActionListener((ActionListener)am);
}
```

```
  public void registerTarget(Applet app){
    target = app;

    root.setUserObject(new CompoModel(target,"0"));
    getComponentTree(target,root);

    printComponentTree();
    System.out.println("target: "+target.getClass().getName());
  }

  void getComponentTree(Container ct,DefaultMutableTreeNode node){
    DefaultMutableTreeNode tempnode;
    String nodeComID = ((CompoModel) node.getUserObject()).getComID();
    Component[] c = ct.getComponents();
    for(int i=0;i<c.length;i++){
      tempnode = new DefaultMutableTreeNode
(new CompoModel(c[i],nodeComID+"/"+String.valueOf(i)));
      node.add(tempnode);
      addListeners(c[i]);
      if (c[i] instanceof Container){
        getComponentTree((Container)c[i],tempnode);
      }
    }
  }

  synchronized void getComponentTreeNoAddListener(Container ct,DefaultMutableTreeNode node){
    DefaultMutableTreeNode tempnode;
    String nodeComID = ((CompoModel) node.getUserObject()).getComID();
    Component[] c = ct.getComponents();
    for(int i=0;i<c.length;i++){
      tempnode = new DefaultMutableTreeNode
(new CompoModel(c[i],nodeComID+"/"+String.valueOf(i)));
      node.add(tempnode);
      if (c[i] instanceof Container){
        getComponentTree((Container)c[i],tempnode);
      }
    }
  }

  public void printComponentTree(){
    Enumeration e = root.breadthFirstEnumeration();
    while(e.hasMoreElements()){
      DefaultMutableTreeNode n = (DefaultMutableTreeNode)e.nextElement();
      CompoModel cm = (CompoModel) n.getUserObject();
      System.out.println(cm.getComID() + " "+cm.getCompo().toString());
    }
  }

  public void addListeners(Component o){
    o.addMouseListener((MouseListener)mm);
    o.addMouseMotionListener((MouseMotionListener)mmm);
    if (o instanceof Container)
      ((Container)o).addContainerListener((ContainerListener)cm);
    if (o instanceof AbstractButton)
      ((AbstractButton)o).addActionListener((ActionListener)am);
    if (o instanceof Button)
      ((Button)o).addActionListener((ActionListener)am);
  }

  public void saveEventModels(String filename){
```

```
    FileOutputStream fout;
    try{
      fout = new FileOutputStream(filename);
      ObjectOutputStream oos = new ObjectOutputStream(fout);
      oos.writeObject(eventList);
      oos.writeObject(root);
      oos.writeObject(target.getClass().getName());

      oos.flush();
      oos.close();
      fout.close();
    }catch(IOException e){
      System.err.println("Cannot save file.");
    }
    System.out.println("eventList (size:"+
       eventList.size()+") is saved in "+filename);
  }
}

  /*  public synchronized void targetInitialize(){
    Jedemo tgt = (Jedemo)target;
    tgt.initJedemo();
  }*/
```

---

package jedemo.analyze

class jedemo.analyze.Analyzer

```
/* Analyzer
 * Author: Motoki MIURA
 * Date: Sep 9, 1998
*/

package jedemo.analyze;

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import jedemo.analyze.*;
import jedemo.*;
import com.sun.java.swing.tree.*;

/**
 * イベントをアイコン表示するシステムのベース
 * (アプリケーションだが、アプレットとしても動作する)
 */
public class Analyzer extends Applet {

  static EventAnalyzer eana;
  static String evf;

  public void init(){
    evf = getParameter("evf");
    showAnalyzer();
  }
```

```
  static void showAnalyzer(){
    if (evf != null) {
      eana = new EventAnalyzer(evf);
      eana.show();
      eana.validate();
    } else {
      System.out.println("Please specify event file.");
      System.exit(0);
    }
  }

  public static void main(String args[]){
    evf = args[0];
    showAnalyzer();
  }
}
```

---

class jedemo.analyze.EventAnalyzer

```
/** EventAnalyzer.java
 *  @version Sep 19, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;
import jedemo.*;
import jedemo.analyze.*;

/** イベントをアイコンで表示して、コマンドの候補を作成する。 */
public class EventAnalyzer extends JFrame{

  /** EventModels */
  Vector eventList;                    // includes EventModel
  /** CompoModels */
  DefaultMutableTreeNode root;         // includes CompoModel
  /** コマンド候補とセパレータの集まり。編集モデル */
  Vector rules = new Vector();
  /** ファイル名のボディ */
  String bodyname;
  /** 対象アプレット名 */
  String target;

  /** 編集モデルのビュー */
  ViewPanel vp;
  static Label label = new Label("ready.");

  /**  Constructor for Isolated mode */
  public EventAnalyzer(String eventfile){
    super("EventMonitor "+eventfile);
```

```
    int i = eventfile.lastIndexOf(".evf");
    bodyname = eventfile.substring(0,i);
    load(eventfile);
    setting();
  }

  /**  Constructor for Interactive mode */
  public EventAnalyzer(Vector eventL, DefaultMutableTreeNode r,
       String tgt, String evfname){
    super("Interactive EventMonitor "+evfname);
    int i = evfname.lastIndexOf(".evf");
    bodyname = evfname.substring(0,i);
    eventList = eventL;
    root = r;
    target = tgt;
    setting();
  }

  /**  initialization */
  private void setting(){
    setSize(550,550);
    getContentPane().setLayout(new BorderLayout());

    vp = new ViewPanel();
    vp.addMouseListeners();              // Mouse Listener
    vp.setLayout(new BorderLayout());

    ScrollPane sc = new ScrollPane(ScrollPane.SCROLLBARS_AS_NEEDED);
    sc.add(vp);
    getContentPane().add(sc,BorderLayout.CENTER);

    // MENU Setting
    MenuBar mb = new MenuBar();
    Menu file = new Menu("FILE");
    MenuItem reload = new MenuItem("Reload");
    reload.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
vp.setIconV(eventList);
repaint();
      }
    });
    file.add(reload);
    file.add(new MenuItem("Load rules"));
    file.add(new MenuItem("Save rules"));
    file.add(new MenuItem("Load commands"));
    file.add(new MenuItem("Save commands"));
    Menu edit = new Menu("EDIT");
    MenuItem AsSep = new MenuItem("as a separator");
    edit.add(AsSep);
    AsSep.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
vp.asSeparator();
validate();
repaint();
      }
    });
    MenuItem AsCom = new MenuItem("as a command");
    edit.add(AsCom);
    AsCom.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
vp.asCommand();
```

```
validate();
repaint();
      }
    });
    Menu view = new Menu("VIEW");
    MenuItem commandV = new MenuItem("commands");
    view.add(commandV);
    commandV.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
commandViewSetting();     // definition is below ↓
      }
    });
    mb.add(file);
    mb.add(edit);
    mb.add(view);

    setMenuBar(mb);

    getContentPane().add(label,BorderLayout.SOUTH);

    vp.setIconV(eventList);
    vp.repaint();
  }

  /**  Load event file (.evf) */
  public void load(String filename){
    FileInputStream fin;
    try{
      fin = new FileInputStream(filename);
      ObjectInputStream ois = new ObjectInputStream(fin);
      eventList = (Vector) ois.readObject();
      root = (DefaultMutableTreeNode) ois.readObject();
      target = (String) ois.readObject();

      ois.close();
      fin.close();
    }catch(IOException e){
      System.err.println("Cannot load file "+filename);
    }catch(ClassNotFoundException cnfe){
      System.err.println("Cannot find class"+cnfe.toString());
    }
  }

  /**  編集が終わったあと、コマンドルールを呼びだすときに使う */
  void commandViewSetting(){
    Sieve sieve = new Sieve(vp.iconV);
    Vector ruleV = sieve.getRules();
    RuleEditor re = new RuleEditor(ruleV);
    re.setBodyname(bodyname);
    re.setTarget(target);
    re.show();
  }

}
```

---

class jedemo.analyze.ViewPanel

```
/** ViewPanel.java
```

```
 *  @version Sep 19, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;
import jedemo.*;
import jedemo.analyze.*;

/** ViewPanel: イベントに対応するアイコンをエディタのように表示する部品。 */
public class ViewPanel extends Panel implements Runnable {

  Vector iconV = new Vector();  // this contains Components and Strings
  int xlimit = 500;
  Vector mapV = new Vector();
  Vector locationV = new Vector();
  Rectangle cp, sp, ep;
  int ci, start, end;

  Image offscreen, finalscreen;
  Graphics offgraphics, finalgraphics;
  boolean refresh;

  Reductor r;
  Thread thread;

  ViewPanel(Vector v){
    this();
    setIconV(v);
  }
  ViewPanel(){
    // Resize causes Re-layouting
    addComponentListener(new ComponentAdapter(){
      public void componentResized(ComponentEvent e){
Dimension d = getSize();
xlimit = d.width - 20;
if (d.width > 0 && d.height > 0) {
  offscreen = createImage(d.width,d.height);
  offgraphics = offscreen.getGraphics();
  finalscreen = createImage(d.width,d.height);
  finalgraphics = finalscreen.getGraphics();
  refresh = true;
  repaint();
}
      }
    });
  }

  void cpclean(){ cp = null; repaint(); }
  void cpspepclean(){
    cp = sp = ep = null;
    Toolkit.getDefaultToolkit().beep();
    repaint();
  }
```

```
  public Dimension getMinimumSize(){
    return new Dimension(200,30);
  }

  /** [as a separator] を選んだときの実装。(設計としては問題あり...) */
  public void asSeparator(){
    r = new Reductor();
    r.makeCompositV(iconV,start,end);
    r.setSeparateMode(true);
    thread = new Thread(this);
    thread.start();
  }
  /** [as a separator] を選んだときの実装。(設計としては問題あり... */
  public void asCommand(){
    r = new Reductor();
    r.makeCompositV(iconV,start,end);
    r.setSeparateMode(false);
    thread = new Thread(this);
    thread.start();
  }

  public void run(){
    while(r.canReduct(iconV)){
      iconV = r.reduct(iconV);
      refresh = true;
      cpspepclean();
      try{ thread.sleep(700);   }catch(InterruptedException ex) {
System.err.println("error in sleep"+ex.toString());
      }
    }
    //    thread.stop();
  }

  public void setIconV(Vector v){
    iconV = v;
  }

  public void update(Graphics g){
    paint(g);
  }

  public void paint(Graphics g){
    if (refresh) {
      offgraphics.clearRect(0,0,getSize().width,getSize().height);
      if (sp != null) sp = (Rectangle) locationV.elementAt(start);
      if (ep != null) ep = (Rectangle) locationV.elementAt(end);
      paint_icons(offgraphics); refresh = false ;
    }
    finalgraphics.drawImage(offscreen,0,0,this);

    finalgraphics.setColor(Color.white);
    if (cp != null) drawCursor(cp,Color.white,finalgraphics);
    if (sp != null) drawCursor(sp,Color.red,finalgraphics);
    if (ep != null) drawCursor(ep,Color.cyan,finalgraphics);
    g.drawImage(finalscreen,0,0,this);
  }

  void drawCursor(Rectangle r, Color c, Graphics g){
    g.setColor(c);
    g.drawRect(r.x-2,r.y-2,r.width+4,26);
    g.drawRect(r.x-1,r.y-1,r.width+2,24);
```

```
    g.drawRect(r.x,r.y,r.width,22);
  }

  public void paint_icons(Graphics g){
    int sx = 5;
    int sy = 5;
    int padx = 2;
    int pady = 3;
    int x = sx, y = sy;
    int cy = 22;
    int xmax = 0;
    mapV.removeAllElements();        // location to em
    locationV.removeAllElements();        // location to em
    ImageIcon ii;

    EventModel[] hmap = new EventModel[xlimit];
    for(Enumeration e = iconV.elements(); e.hasMoreElements();){
      Object o = e.nextElement();
      if (o instanceof EventModel){
EventModel m = (EventModel) o;
ii = (ImageIcon)IconParade.hash.get(m.getIconID());
int cx = ii.getIconWidth();
if (x + cx > xlimit) {
  x = sx; y = y + cy + pady;
  mapV.addElement(hmap);
  hmap = new EventModel[xlimit];
}
ii.paintIcon(this,g,x,y);
for(int i=x;i<x+cx;i++){ hmap[i] = m; } // location to em
locationV.addElement(new Rectangle(x,y,cx,22));

x += cx + padx;
if (x > xmax) xmax = x;
      }
      if (o instanceof String){
String s = (String) o;
if (s.equals("cr")){
  x = sx; y = y + cy + pady;
  mapV.addElement(hmap);
  locationV.addElement(new Rectangle(x,y,0,22));
  hmap = new EventModel[xlimit];
} else {
  int cx = 0;
  ii = (ImageIcon)IconParade.hash.get(s);
  if (ii == null) g.drawString(s,x+10,y+16);
  else {
    cx = ii.getIconWidth();
    if (x + cx > xlimit) {
      x = sx; y = y + cy + pady;
      mapV.addElement(hmap);
      hmap = new EventModel[xlimit];
    }
    ii.paintIcon(this,g,x,y);
    locationV.addElement(new Rectangle(x,y,cx,22));
  }
  x += cx + padx;
  if (x > xmax) xmax = x;
}
      }
    }
    mapV.addElement(hmap);
```

```
    setSize(getSize().width,y+cy+pady);
    super.paint(g);
  }

  void addMouseListeners(){
    addMouseMotionListener(new MouseMotionAdapter(){
      public void mouseMoved(MouseEvent e){
int x = e.getX();
if (x >= xlimit) { cpclean(); return; }
int y = e.getY();
int ln = (y - 5)/25;
if ((y - 5) % 25 > 22) { cpclean(); return; }

EventModel[] emv = (EventModel[]) mapV.elementAt(ln);

if (emv[x] != null){
  EventAnalyzer.label.setText("target: "+emv[x].getTarget()+
      " ("+emv[x].getComID()+")");
  ci = iconV.indexOf(emv[x]);
  cp = (Rectangle) locationV.elementAt(ci);
} else {
  refresh = true; cp = null ;
}
repaint();
      }
    });

    addMouseListener(new MouseAdapter(){
      public void mouseClicked(MouseEvent e){
if (cp == null) {
  cpspepclean() ; return;
}
if (sp != null) {
  ep = cp; repaint();
  if (start == ci) {   // cancel start and end
    cpspepclean() ; return;
  }
  end = ci;
} else {
  if (cp != null) {
    sp = cp; repaint();
    start = ci;
  }
}
      }
    });
  }
}
```

---

class jedemo.analyze.RuleEditor

```
/** RuleEditor.java
 *  @version Oct 23, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;
import jedemo.*;
import jedemo.analyze.*;

/** RuleEditor */
public class RuleEditor extends JFrame{

  Vector rules = null;

  private JPanel p;
  private Label lb = new Label();
  private String bodyname;
  private String target = null;
  private int rulenum = 0;

  // ファイルを使って初期化
  public RuleEditor(String rulefile){
    rules = new Vector();
    rulenum = rules.size();
    p = new JPanel();
    loadRules(rulefile);
    view_update();
    menu_setting();
  }

  // ルールを使って初期化
  public RuleEditor(Vector rules){
    this.rules = rules;
    rulenum = rules.size();
    p = new JPanel();
    view_update();
    menu_setting();
  }

  /** initial setting */
  void menu_setting(){
    // MENU Setting
    MenuBar mb = new MenuBar();
    Menu file = new Menu("FILE");
    MenuItem saveRule = new MenuItem("Save Rules");
    MenuItem loadRule = new MenuItem("Load Rules");
    MenuItem generator = new MenuItem("invoke Generator");
    MenuItem close = new MenuItem("close");
    saveRule.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
saveRules();
      }
    });
    loadRule.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
loadRules();
      }
    });
    generator.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
invokeGenerator(bodyname+".crf");
      }
```

```
    });
    close.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
dispose();
      }
    });

    file.add(saveRule);
    file.add(loadRule);
    file.add(generator);
    file.add(close);

    mb.add(file);
    setMenuBar(mb);

    getContentPane().add(p,BorderLayout.CENTER);
    getContentPane().add(lb,BorderLayout.SOUTH);
  }

  /** generator invoking method */
  public void invokeGenerator(String filename){
    java.applet.Applet app = null;
    final CommandGenerator comgen;
    Button start,stop;
    final TextField name;
    String target;

    comgen = new CommandGenerator(filename);
    target = comgen.getTargetClassName();

    Frame genf = new Frame("Target Applet "+target );
    genf.setLayout(new BorderLayout());

    try{
      app = (java.applet.Applet) Class.forName(target).newInstance();
    }catch(ClassNotFoundException e){
      System.out.println("No such class: "+target);
    }catch(IllegalAccessException e){
      System.out.println("IllegalAccess: "+target);
    }catch(InstantiationException e){
      System.out.println("Cannot instantiate: "+target);
    }

    if (app == null) System.exit(0);
    genf.add(app,BorderLayout.CENTER);

    Panel controlp = new Panel();
    controlp.add(start = new Button("Generate start"));
    controlp.add(stop = new Button("Generate stop and save as"));
    controlp.add(name = new TextField("sample.jdm"));
    controlp.setBackground(new Color(120,120,20));
    controlp.setForeground(Color.white);

    genf.add(controlp,BorderLayout.NORTH);

    app.init();

    genf.setSize(550,550);
    genf.show();

    comgen.registerTarget(app);
```

```
    stop.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
comgen.stopRecording();
comgen.saveContent(name.getText());
      }
    });
  }

  public void setBodyname(String bn){
    bodyname = bn;
    setTitle("RuleEditor "+bodyname);
  }
  public void setTarget(String tg){
    target = tg;
  }

  /** ルールを保存する */
  void saveRules(){
    FileOutputStream fout;
    try{
      fout = new FileOutputStream(bodyname+".crf");
      ObjectOutputStream oos = new ObjectOutputStream(fout);

      oos.writeObject(rules);
      oos.writeObject(target);

      oos.flush();
      oos.close();
      fout.close();
    }catch(IOException e){
      System.err.println("Cannot save rule file.");
    }
    System.out.println("Rules are saved in "+bodyname+".crf");
  }

  /** ルール群を読みこむ */
  void loadRules(){
    FileDialog fdialog = new FileDialog(this,"Load Rule");
    fdialog.show();
    loadRules(fdialog.getFile());
  }

  /** ルール群を読みこむ */
  void loadRules(String filename){
    Vector exrules = null;
    String extarget = null;
    FileInputStream fin;
    try{
      fin = new FileInputStream(filename);
      ObjectInputStream ois = new ObjectInputStream(fin);

      exrules = (Vector) ois.readObject();
      extarget = (String) ois.readObject();

      System.out.println("extarget: "+extarget);

      ois.close();
      fin.close();
    }catch(IOException e){
      System.err.println("Cannot load rule file "+filename);
```

```
    }catch(ClassNotFoundException cnfe){
      System.err.println("Cannot find class"+cnfe.toString());
    }
    // append rules
    if (rulenum == 0) {
      int i = filename.lastIndexOf(".crf");
      setBodyname(filename.substring(0,i));
      target = extarget;
    }
    if (rulenum != 0 && !extarget.equals(target)) {
      System.out.println("Cannot merge: target is different.");
      return;
    }
    for (Enumeration e = exrules.elements();e.hasMoreElements();){
      CommandRule cr = (CommandRule) e.nextElement();
      rules.addElement(cr);
    }
    rulenum = rules.size();
    view_update();
  }

  /** ビューを更新する */
  void view_update(){
    p.removeAll();
    p.setLayout(new GridLayout(rulenum,1));
    for(Enumeration e = rules.elements();e.hasMoreElements();){
      CommandRule cr = (CommandRule) e.nextElement();
      cr.setView();
      // ラベル文字列を ViewPanel に表示 (ルールのモデルには変更なし)
      Vector vv = (Vector) cr.getRule().clone();
      vv.addElement(cr.getLabel());
      cr.getView().getViewPanel().setIconV(vv);

      cr.getView().setRuleEditor(this);
      p.add(cr.getView());
    }
    setSize(400,rulenum*25+110);
    lb.setText(String.valueOf(rulenum)+" commands exist.");
    validate();
    repaint();
  }
  /** ルールを 1 つ消去 */
  public void delete(CommandRule cr){
    rules.removeElement(cr);
    view_update();
  }

  public static void main(String argv[]){
    RuleEditor re = new RuleEditor(argv[0]);
    re.show();
  }
}
```

---

class jedemo.analyze.CommandRuleView

```
/** CommandRuleView.java
 *  @version Oct 1, 1998
 *  @author  Motoki Miura
```

```
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;
import jedemo.*;
import jedemo.analyze.*;


/** 1つのコマンドルールの表示 */
public class CommandRuleView extends Panel {

  /** モデル */
  CommandRule comR;
  /** 編集するときに押すアイコンボタン */
  JButton editB;
  /** ルールをアイコン表示 */
  ViewPanel vp;

  static transient Insets insets = new Insets(0,0,0,0);
  //  TextField labelF,methodF;
  /** ルールエディタ(複数のルールを管理するフレーム) */
  RuleEditor re;

  public CommandRuleView(CommandRule r){
    comR = r;
    setLayout(new BorderLayout());
    editB = new JButton((ImageIcon)IconParade.hash.get(r.getType()));
    editB.setMargin(insets);
    vp = new ViewPanel(r.getRule());
    add(editB,BorderLayout.WEST);
    add(vp,BorderLayout.CENTER);
    validate();

    editB.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
JFrame f = new CommandRuleProperty(comR);
f.show();
      }
    });

    vp.addMouseListener(new MouseAdapter(){
      public void mouseClicked(MouseEvent e){
if (e.getModifiers()==e.BUTTON2_MASK)
  re.delete(comR);
      }
    });
  }
  public void setRuleEditor(RuleEditor r){  re = r;  }
  public ViewPanel getViewPanel(){
    return vp;
  }
}
```

class jedemo.analyze.CommandRuleProperty

```
/** CommandRuleProperty.java
 *  @version Oct 1, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;
import jedemo.*;
import jedemo.analyze.*;


/** コマンドルールを定義・書き替えることができるコマンドルールエディタ */
public class CommandRuleProperty extends JFrame{

  CommandRule comR;
  CommandRuleView crv;
  TextField label, method;

  public CommandRuleProperty(CommandRule r){
    super("CommandRuleProperty");
    setSize(400,200);
    getContentPane().setLayout(new BorderLayout());

    comR = r;
    label = new TextField();
    label.setText(r.getLabel());
    method = new TextField();
    method.setText(r.getMethod());
    JPanel jp = new JPanel();
    jp.setLayout(new GridLayout(5,1));
    crv = new CommandRuleView(comR);
    jp.add(crv);
    jp.add(new Label("label:"));
    jp.add(label);
    jp.add(new Label("method:"));
    jp.add(method);
    JButton okB = new JButton("OK");
    JButton cancelB = new JButton("cancel");
    JPanel jpack = new JPanel();
    jpack.add(okB);
    jpack.add(cancelB);
    getContentPane().add(jpack,BorderLayout.SOUTH);
    getContentPane().add(jp,BorderLayout.CENTER);


    //    MenuBar mb = new MenuBar();     mb.add(new Menu("EDIT"));     setMenuBar(mb);
    //getContentPane().add(mb,BorderLayout.NORTH);

    okB.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
comR.setLabel(label.getText());
comR.setMethod(method.getText());
Vector vv = (Vector) comR.getRule().clone();
```

```
vv.addElement(label.getText());
comR.getView().getViewPanel().setIconV(vv);
comR.getView().validate();
comR.getView().repaint();
dispose();
      }
    });
    cancelB.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
dispose();
      }
    });
  }
}
```

---

class jedemo.analyze.Generator

```
/* Generator
 * Author: Motoki MIURA
 * Date: Oct 21, 1998
*/

package jedemo.analyze;

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import jedemo.*;
import jedemo.analyze.*;

/** デモファイル (Content file, --.jdm) を生成するベースアプリケーション
 *
 */

public class Generator extends Frame {

  Applet app;
  CommandGenerator comgen;
  Button start,stop;
  TextField name;
  String target;

  public Generator(String rule){
    super();
    comgen = new CommandGenerator(rule);
    target = comgen.getTargetClassName();  // get target class name

    setTitle("Target Applet "+target);
    setLayout(new BorderLayout());

    try{
      app = (Applet) Class.forName(target).newInstance();
    }catch(ClassNotFoundException e){
      System.out.println("No such class: "+target);
    }catch(IllegalAccessException e){
```

```
      System.out.println("IllegalAccess: "+target);
    }catch(InstantiationException e){
      System.out.println("Cannot instantiate: "+target);
    }

    add(app,BorderLayout.CENTER);

    Panel controlp = new Panel();
    controlp.add(start = new Button("Generate start"));
    controlp.add(stop = new Button("Generate stop and save as"));
    controlp.add(name = new TextField("sample.jdm"));
    controlp.setBackground(new Color(120,120,20));
    controlp.setForeground(Color.white);

    add(controlp,BorderLayout.NORTH);

    app.init();

    start.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
comgen.registerTarget(app);
      }
    });
    stop.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
comgen.stopRecording();
comgen.saveContent(name.getText());
      }
    });
  }

  public static void main(String args[]){
    Generator gen;
    if (args.length == 1)  gen = new Generator(args[0]);
    else  gen = new Generator("null");
    gen.setSize(550,550);
    gen.show();
  }

}
```

---

class jedemo.analyze.CommandGenerator

```
/* CommandGenerator
 * Author: Motoki MIURA
 * Date: Oct 14, 1998 -- Oct 17, 1998
*/

package jedemo.analyze;

import java.applet.Applet;
import java.awt.*;
import com.sun.java.swing.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.Enumeration;
import java.io.*;
import jedemo.*;
```

```
import jedemo.record.*;
import jedemo.analyze.*;

/** コマンドを生成するシステム */
public class CommandGenerator extends EventRecorder {

  /** Separator + CommandRules */
  Vector origrules;               // Separator + CommandRules
  /** CommandRules only */
  Vector rules;                   // CommandRules
  String target;
  static Label label = new Label("ready.");
  static Filter filter = new Filter();

  List comList;
  protected Vector separators = new Vector();
  protected boolean isCommandState = false;
  /** 照合対象となるイベントをためておくバッファ */
  protected Vector emBuffer = new Vector();
  /** 状態表示メッセージ */
  protected StringBuffer state = new StringBuffer();
  /** 生成されたコマンドはここにためられる */
  protected Vector commands = new Vector();
  String rulefile = null;

  public CommandGenerator(String rulef){
    super();
    setting(rulef);
  }

  public String getTargetClassName(){
    return target;
  }

  /** ルールファイルを読み込む。
  * @param rulef ルールファイル名。null ならファイル選択ダイアログが出る
  */
  public void setting(String rulef){
    if (rulef == "null"){
      Frame f = new Frame();
      FileDialog fdialog = new FileDialog(f,"Load Rule");
      fdialog.show();
      rulef = fdialog.getFile();
      loadRules(rulef);
    } else loadRules(rulef);

    commandGeneratorViewSetting(rulef);
    rulefile = rulef;

    // divide separators from (command)rules;
    rules = (Vector) origrules.clone();
    for (Enumeration e = rules.elements(); e.hasMoreElements();){
      CommandRule cr = (CommandRule) e.nextElement();
      if (cr.getType().equals("sep")) {
for (Enumeration se = cr.getRule().elements(); se.hasMoreElements();)
  separators.addElement(se.nextElement());
rules.removeElement(cr);
      }
    }
  }
```

```
  /** 通知されたイベントの処理をする */
  public synchronized void addEvent(AWTEvent e){
    // for compativility WIN and UNIX
    if (e.getID() == MouseEvent.MOUSE_PRESSED) skipIO = true;
    if (e.getID() == MouseEvent.MOUSE_RELEASED) skipIO = false;
    if (skipIO)
      if (e.getID() == MouseEvent.MOUSE_ENTERED ||
  e.getID() == MouseEvent.MOUSE_EXITED) return;

    EventModel em = new EventModel(e);
    em.setComID(root.depthFirstEnumeration());
    // interactive recording
    if (filter.isCollection(em,separators))
      if (isCommandState) match(); else ;
    else bufferEventModel(em);

    if (e instanceof ContainerEvent) {
      ContainerEvent ce = (ContainerEvent) e;
      if (e.getID() == ContainerEvent.COMPONENT_ADDED)
addDynamicComponent(ce);
      if (e.getID() == ContainerEvent.COMPONENT_REMOVED)
removeDynamicComponent(ce);
    }
  }

  /** 照合を行う */
  void match(){
    state = new StringBuffer();
    label.setText("Matching...");

    Command c = null;
    // 照合 matching for each rule
    Enumeration e = rules.elements();
    int inc = 1;
    while(e.hasMoreElements()){
      CommandRule cr = (CommandRule) e.nextElement();
      System.out.println("rule "+inc+" "+cr.getLabel());
      c = cr.generateCommand(emBuffer);
      if (c != null) break;
      inc++;
    }
    // clear buffer
    isCommandState = false;
    if (c != null) {
      c.setEventmodels(emBuffer); // keep corresponding eventmodels
      commands.addElement(c);

      comList.addItem(c.getLabel());
      comList.select(comList.getItemCount());
      comList.makeVisible(comList.getItemCount());
      comList.validate();
      comList.repaint();
      System.out.println("------");
      label.setText("Matching...succeed: "+c.getLabel());
    } else {
      System.out.println("No match any rules.\n------");
      label.setText("Matching...failed.");
    }
    emBuffer = new Vector();
  }
```

```
  /** ステータスメッセージ更新 (重要度低) */
  void bufferEventModel(EventModel em){
    isCommandState = true;
    emBuffer.addElement(em);
    state.append(".");
    label.setText("Stacking"+state.toString());
  }

  /** 初期化で行われる */
  void commandGeneratorViewSetting(String rulef){
    final Frame f = new Frame("CommandGenerator "+rulef);
    f.setLayout(new BorderLayout());

    // MENU Setting
    MenuBar mb = new MenuBar();
    Menu rulemenu = new Menu("Rules");
    MenuItem loadRule = new MenuItem("Load Rules");
    loadRule.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
FileDialog fdialog = new FileDialog(f,"Load Rule");
fdialog.show();
loadRules(fdialog.getFile());
      }
    });
    MenuItem invokeRE = new MenuItem("invoke RuleEditor");
    invokeRE.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
if (rules != null){
  RuleEditor re = new RuleEditor(origrules);
  re.setTitle(rulefile);
  re.show();
}
      }
    });
    rulemenu.add(loadRule);
    rulemenu.add(invokeRE);

    Menu commmenu = new Menu("Commands");
    MenuItem saveContent = new MenuItem("Save Commands");
    saveContent.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
int crf = rulefile.lastIndexOf(".crf");
saveContent(rulefile.substring(0,crf)+".jdm");
      }
    });
    commmenu.add(saveContent);
    mb.add(rulemenu);
    mb.add(commmenu);
    f.setMenuBar(mb);

    // LIST Setting
    comList = new List(10,false);
    comList.setForeground(Color.blue);
    f.add(comList,BorderLayout.CENTER);
    f.add(label,BorderLayout.SOUTH);
    f.setSize(300,400);
    f.validate();
    f.show();
  }

  /** 生成されたコマンドをまとめて Content としてファイルに保存 */
```

```
  public void saveContent(String filename){
    FileOutputStream fout;
    try{
      fout = new FileOutputStream(filename);
      ObjectOutputStream oos = new ObjectOutputStream(fout);

      oos.writeObject(new Content(filename, target, commands, "initJedemo"));

      oos.flush();
      oos.close();
      fout.close();
    }catch(IOException e){
      System.err.println("Cannot save content file.");
    }
    System.out.println("Content saved in "+filename);
  }

  /** ルールを読みこむ */
  protected void loadRules(String filename){
    FileInputStream fin;
    try{
      fin = new FileInputStream(filename);
      ObjectInputStream ois = new ObjectInputStream(fin);

      origrules = (Vector) ois.readObject();
      target = (String) ois.readObject();

      ois.close();
      fin.close();
    }catch(IOException e){
      System.err.println("Cannot load rule file "+filename);
    }catch(ClassNotFoundException cnfe){
      System.err.println("Cannot find class"+cnfe.toString());
    }
  }
}
```

---

class jedemo.analyze.Filter

```
/** Filter.java
 *  @version Sep 28, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.util.*;
import jedemo.*;
import jedemo.analyze.*;

/** イベントモデルを扱うのに用いる共通機能を集めたクラス */
public class Filter {

  public Filter(){
  }

  /**  (完全に) 同じ要素なら true を返す。
   * EventModel(Emm) とアイコン ID(mm) は区別する */
```

```
  protected boolean areEqualVectors(Vector iv, Vector cv){
    if (iv.size() != cv.size()) return false;
    for(int i=0;i<iv.size();i++){
      Object o1 = iv.elementAt(i);
      Object o2 = cv.elementAt(i);
      if (!areEqualElements(o1,o2)) return false;
    }
    return true;
  }

  /**  (完全に) 同じ要素なら true を返す。
   * EventModel(Emm) とアイコン ID(mm) は区別する */
  protected boolean areEqualElements(Object o1, Object o2){
    String s1 = getElementType(o1);
    String s2 = getElementType(o2);
    return s1.equals(s2);
  }

  /**  イベントモデルなら true を返す */
  protected boolean isEventModel(Object o){
    return (o instanceof EventModel);
  }

  /**  識別子を返す。 EventModel なら E をつける。アイコン ID ならそのまま。 */
  protected String getElementType(Object o){
    // differentiate String and EventModel
    String ret = null;
    if (o instanceof String){
      return (String) o;         // if String, return itself.
    }
    if (o instanceof EventModel){
      EventModel em = (EventModel) o;
      ret = "E"+ em.getIconID(); // if EventModel, return "E"+itself.
    }
    return ret;
  }
  /**  アイコン ID を返す。 EventModel でも E をつけない。 */
  protected String getIconID(Object o){// return iconID
    if (o instanceof String){          // no difference between String
      return (String) o;                              // and EventModel
    }
    if (o instanceof EventModel){
      EventModel em = (EventModel) o;
      return em.getIconID();
    }
    return null;
  }

  /**  "cr" で区切られたトークンを返す。 */
  protected Vector nextToken(Enumeration pe){
    Vector token = new Vector();

    while(pe.hasMoreElements()){
      Object o = pe.nextElement();
      token.addElement(o);
      if (o instanceof String){
String s = (String) o;
if (s.equals("cr")) {
  return token;
}
      }
```

```
    }
    return null;
  }

  /**  ベクトルの内容を表示する。テスト用。 */
  protected void showContents(Vector v){
    for(int i=0;i<v.size();i++){
      System.out.print(getElementType(v.elementAt(i)));
    }
    System.out.println("");
  }

  /**  範囲選択されたイベント列の中にあるイベント要素を
   * 1つずつ集めたものを返す。
   * @param v 対象イベント
   * @param s 開始インデックス
   * @param e 終了インデックス
   */
  protected Vector getElementsV(Vector v, int s, int e){
    Vector newV = new Vector();
    for(int i=s;i<=e;i++){
      Object o = v.elementAt(i);
      if (!isCollection(o,newV)){
EventModel em = (EventModel) o;
String c = em.getIconID();
newV.addElement(c);
      }
    }
    return newV;
  }

  /** 要素が集合に含まれるかどうか調べる。
   * (注) アイコンIDの一致を調べている。参照の一致を調べるわけではない
   * @param o 調べる要素 (EventModelのみ、でなければfalseを返す)
   * @param cV 集合
   */
  public boolean isCollection(Object o, Vector cV){
    if (o instanceof EventModel){
      EventModel e = (EventModel) o;
      for(Enumeration en = cV.elements();en.hasMoreElements();){
String s = (String) en.nextElement();
if (e.getIconID().equals(s)) return true;
      }
    }
    return false;
  }

  /** 要素が集合に含まれるかどうか調べる。
  * (注) isIncludeは、 o がEventModelでも受け付けるのに対し、
  * isCollectionは、受け付けない。
  * @param o 調べる要素 (なんでも可)
  * @param cV 集合
  */
  public boolean isInclude(Object o, Vector cV){  // contain check
    String type = getIconID(o);
    for(Enumeration e = cV.elements();e.hasMoreElements();){
      Object eo = e.nextElement();
      if (type.equals(getIconID(eo))) return true;
    }
    return false;
  }
```

```
}
```

---

class jedemo.analyze.Sieve

```
/** Sieve.java
 *  @version Sep 21, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import jedemo.*;
import jedemo.analyze.*;

/** Sieve : ViewPanel から、コマンドルールを 1 つずつ抽出する */
public class Sieve extends Filter {

  Vector iconV;
  Vector commandV;  // collection of Command instance
  private Enumeration e;
  int comNum;       // Numbers of Commands

  /** 手順 1。まずイベント列ベクトルを指定する。 */
  public Sieve(Vector v){
    iconV = v;
    comNum = 0;
    commandV = new Vector();
  }

  /** 手順 2。ルールの抽出を行う */
  public Vector getRules(){
    boolean isExist;
    e = iconV.elements();
    Vector comV = new Vector();

    while(e.hasMoreElements()){
      Vector iv = nextToken(e);

      Enumeration ie = iv.elements(); // null error
      Enumeration ce = comV.elements();

      if(comV.size()==0){
comNum++;
while(ie.hasMoreElements()) comV.addElement(ie.nextElement());
continue;
      } else {
isExist = false;
while(true){
  Vector cv = nextToken(ce);
  if (cv == null) break;
  //   showContents(iv); showContents(cv); TEST
  if (areEqualVectors(iv,cv)){
    isExist = true;
    break;
```

```
  }
}
if (!isExist){  // add iv to commands
  comNum++;
  Enumeration av = iv.elements();
  while(av.hasMoreElements()) comV.addElement(av.nextElement());
}
      }
    }
    //  then, here makes comV (mm,mc,cr,...) to commandV (CommandRule)

    Enumeration ce = comV.elements();  // re-store
    while(true){
      Vector cv = nextToken(ce);
      if (cv == null) break;
      int lastindex = cv.size() - 1;
      cv.removeElementAt(lastindex);  // remove last cr
      commandV.addElement(new CommandRule(cv));
    }
    return commandV;
  }

  /** 未使用 */
  public Vector command_scan(Vector targetV){
    Vector retV = new Vector();
    Enumeration enu = targetV.elements();
    String preID;
    boolean areSame;

    Object oo = enu.nextElement();
    preID = getElementType(oo);
    while(enu.hasMoreElements()){
      Object o = enu.nextElement();
      if (preID.equals(getElementType(o))){
areSame = true;
      }
    }
    return retV;
  }
}
```

---

class jedemo.analyze.Reductor

```
/** Reductor.java
 *  @version Sep 20, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import jedemo.*;
import jedemo.analyze.*;

/** Reductor: ViewPanel の編集で、 [as a Separator][as a command] を選ん
 * だときにイベントモデル列に対して操作するのに用いられる。 */
```

```
public class Reductor extends Filter{

  /** 簡約イベント集合 */
  Vector compV; // includes "mi","mo","mm" and so on.
  /** 簡約化した後のイベント列 */
  Vector newV;
  private Enumeration enu;
  /** [as a separator] が選ばれたかどうか。
    true だと簡約後に改行 ("cr") を挿入する */
  private boolean isSeparate = false;

  public Reductor(){
    compV = new Vector();
  }

  /** 簡約イベント集合を作る */
  protected void makeCompositV(Vector v, int s, int e){
    compV = getElementsV(v,s,e);
  }

  /** [as a separator] が選ばれたかどうかを設定する */
  public void setSeparateMode(boolean b){
    isSeparate = b;
  }

  /** 簡約化できるかどうか試す
  * @param targetV 元のイベント列
  */
  public boolean canReduct(Vector targetV){
    newV = new Vector();
    enu = targetV.elements();
    Object o = enu.nextElement();
    if (isCollection(o,compV)) inCollection(o);
    else outofCollection(o);
    return (newV.size() != targetV.size());
  }

  /** 簡約イベント集合に基づいて簡約化する
  * @param targetV 元のイベント列
  */
  public Vector reduct(Vector targetV){
    newV = new Vector();
    enu = targetV.elements();
    Object o = enu.nextElement();
    if(isCollection(o,compV)) inCollection(o);
    else outofCollection(o);
    return newV;
  }

  /** 簡約イベント集合に含まれている間の処理 */
  private void inCollection(Object o){
    if (isCollection(o,compV)){
      if (enu.hasMoreElements()) inCollection(enu.nextElement());
      else {
      if (isSeparate) newV.addElement("lb"); else newV.addElement("lp");// (
      for(Enumeration en = compV.elements();en.hasMoreElements();)
newV.addElement(en.nextElement());
      if (isSeparate) newV.addElement("rb"); else newV.addElement("rp");// )
      if (isSeparate) newV.addElement("cr");
      //  miuramo0128    if (enu.hasMoreElements()) outofCollection(enu.nextElement());
      while(enu.hasMoreElements()) newV.addElement(enu.nextElement());
```

```
      return;
      }
    } else {
      if (isSeparate) newV.addElement("lb"); else newV.addElement("lp");// (
      for(Enumeration en = compV.elements();en.hasMoreElements();)
newV.addElement(en.nextElement());
      if (isSeparate) newV.addElement("rb"); else newV.addElement("rp");// )
      if (isSeparate) newV.addElement("cr");
      newV.addElement(o);
      //  miuramo0128    if (enu.hasMoreElements()) outofCollection(enu.nextElement());
      while(enu.hasMoreElements()) newV.addElement(enu.nextElement());
      return;
    }
  }

  /** 簡約イベント集合に含まれていない間の処理 */
  private void outofCollection(Object o){
    if (isCollection(o,compV)){
      if (isSeparate) newV.addElement("cr");
      inCollection(o);
      return;
    }
    newV.addElement(o);
    //    while(enu.hasMoreElements()) newV.addElement(enu.nextElement());
    if (enu.hasMoreElements()) outofCollection(enu.nextElement());
    else return;
  }

}
```

---

class jedemo.analyze.IconParade

```
/** IconParade.java
 *  @version Sep 19, 1998
 *  @author  Motoki Miura
 */

package jedemo.analyze;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import jedemo.*;
import jedemo.analyze.*;
import com.sun.java.swing.*;

/** アイコン管理クラス */
public class IconParade {

  static String[] iconfns = {"action","contadd","contrem",
     "Mpress","Mdrag","Mrelease","Mclick",
     "Mmove","Min","Mout",
     "lb","rb","lB","rB",
     "lp","rp","lP","rP",
     "sep","cmd","play","stop"};
  static String[] iconIDs = {"ac","c+","c-",
     "mp","md","mr","mc","mm","mi","mo",
     "lb","rb","lB","rB",
```

```
    "lp","rp","lP","rP",
    "sep","cmd","play","stop"};
  static Hashtable hash = new Hashtable(20);

  static{
    for (int i=0;i<iconfns.length;i++)
      hash.put(iconIDs[i],new ImageIcon("jedemo/analyze/gifs/"+
      iconfns[i]+".gif"));
  }

  IconParade(){
  }

  /** 指定されたグラフィックの指定された位置にイベントアイコンを表示する
   * @param m イベントモデル
   */
  public void iconPaint(EventModel m,Component c,Graphics g,
int x, int y){
    ImageIcon ii = (ImageIcon)hash.get(m.getIconID());
    ii.paintIcon(c,g,x,y);
  }
  /** 指定されたイベントのアイコンを返す
   * @param m イベントモデル
   */
  public ImageIcon getImageIcon(EventModel m){
    return (ImageIcon)hash.get(m.getIconID());
  }
  /** 指定されたイベント ID のアイコンを返す
   * @param id イベント ID
   */
  public ImageIcon getImageIcon(String id){
    return (ImageIcon)hash.get(id);
  }
}
```

---

package jedemo.play

class jedemo.play.Player

---

class jedemo.play.CommandPlayer

---

class jedemo.play.PlayController

---

class jedemo.play.MouseCursor

---

class jedemo.play.PopupWindow