

筑波大学大学院博士課程

工学研究科博士論文

**ネットワークアプリケーションのための
アニメーションヘルプと
構造化文書の閲覧法に関する研究**

著者氏名 三浦 元喜

主任指導教官 電子・情報工学系 田中 二郎

ネットワークアプリケーションのための
アニメーションヘルプと
構造化文書の閲覧法に関する研究

工学研究科

筑波大学

2001 年 7 月

三浦 元喜

目次

第 1 章	導入	1
1.1	WWW	1
1.2	ネットワークアプリケーション	1
1.3	本論文の構成	2
第 2 章	ネットワークアプリケーションのためのアニメーションヘルプ	3
2.1	緒言	3
2.1.1	テキストによる GUI のヘルプが抱える問題	4
2.1.2	アニメーションによるヘルプ	5
2.1.3	研究の目的	5
2.2	アニメーション手法	6
2.2.1	フィードバックレベル	6
2.2.2	実装レベル	7
2.2.3	イベント駆動手法における再生時の制限	9
2.3	アプレットののためのアニメーションヘルプ環境の設計	9
2.3.1	アニメーション機能の追加にまつわる問題	10
2.3.2	アプレットビューアモデル	10
2.3.3	イベントオブジェクトの記録	12
	Java における部品	12
	イベントリスナ	12
	部品の階層構造	14
	イベントオブジェクトの記録	15
	動的に登録される部品	16
2.3.4	記録した操作の再現	17
	イベントソース部品	17

2.3.5	操作の意味付け	18
	イベントオブジェクトとコマンド	18
	コマンドルール	18
	コマンド生成の手順	19
2.4	実装システム	19
2.4.1	対象アプレット	20
2.4.2	対象アプレットの指定と初期設定	22
	対象アプレットウィンドウの調整 — [FrameSize]	24
	対象アプレットの部品構造 — [CompoTree]	24
2.4.3	操作の記録	24
	イベント記録用リスナ (Messenger) の調整 — [Listeners]	24
	記録したイベントの一覧表示 — [EventSpool]	25
2.4.4	コマンドルールの定義	25
	セパレータの指定と適用 — [EventSpool]	26
	コマンドルール候補の一般化 — [EventSpool]	26
	コマンドルールの意味定義 — [CommandRules]	28
	コマンドの生成と保存 — [CommandGenerator]	30
2.4.5	アプレットタグの変更	32
2.4.6	Jedemo Animator	32
2.5	議論	33
2.5.1	コマンドルールの記述	33
2.5.2	Jedemo の限界	34
	コマンドルールが定義できない事例	34
	再現できない事例	35
2.5.3	Jedemo によるオーバーヘッド	35
	実験の設定	35
	アニメーション機能の追加にかかるオーバーヘッド	36
	アニメーション実行時のメモリ使用量	38
2.6	関連研究	39
2.6.1	操作履歴の保存とその再利用	39
2.6.2	例示プログラミング	40
2.6.3	ヘルプシステム	40

2.7	結言	41
第3章	挿入機能を用いた構造化文書の閲覧法	42
3.1	緒言	42
3.2	従来のブラウザにおけるリンク機能とその問題点	42
3.2.1	ナビゲーション行為	43
3.2.2	ナビゲーション負荷の軽減	44
3.3	“ inlineLink ” の概念	45
3.3.1	文書内挿	45
3.3.2	機能アンカー	46
3.3.3	閲覧者の行動	47
3.3.4	内挿表示方式: 全文内挿と部分内挿	48
3.4	既存のブラウザを用いた inlineLink の実現	49
3.4.1	方針	49
3.4.2	挿入するための仕組み	50
	リンク先文書データの取得	50
	画面に表示された文書の手換え	51
3.4.3	方法	52
3.4.4	閲覧者が可能な操作	53
3.5	議論	54
3.5.1	inlineLink の特徴	54
3.5.2	inlineLink の問題点	55
3.5.3	関連する Web ページ	56
3.6	実験	56
3.6.1	実験の目的, 概要と設定	56
3.6.2	実験結果	59
3.6.3	考察	59
3.6.4	被験者からのコメント	62
3.7	関連研究	63
3.7.1	リンク先情報の提示	63
3.7.2	閲覧動作の簡略化	64
3.8	結言	65

第 4 章	発展例：社会的インタラクションを導入した協調型ヘルプ環境	66
4.1	緒言	66
4.2	オンラインヘルプ検索時の問題点	66
4.3	社交的ヘルプ：社会的インタラクションを導入した協調型ヘルプ	67
4.4	インタラクションシナリオ	68
4.5	社交的ヘルプシステム環境 “Social-Help” の実装	71
4.6	関連研究	73
4.7	結言	74
第 5 章	結論	75
	謝辞	77
	参考文献	78
	著者論文リスト	85
付録 A	inlineLink 実験詳細	87
A.1	実験に用いた文書の詳細	87
A.2	t 検定の結果	87
A.3	被験者に与えた問題 (日本語版)	90
A.4	被験者に与えた問題 (英語版)	92

図一覧

2.1	アプレットビューアモデル	11
2.2	HTML ファイルの書き換えによるアニメーションヘルプ機能の適用 . .	11
2.3	GraphApplet の画面	14
2.4	GraphApplet の部品階層構造	14
2.5	部品の位置を示すパス情報	15
2.6	対象アプレットとして用いたグラフ編集アプレット	20
2.7	対象アプレットの操作：ノードの作成 (2.), 移動 (3.), 削除 (4.)	21
2.8	対象アプレットの指定	23
2.9	[FrameSize] 機能パネル	23
2.10	[CompoTree] 機能パネル	23
2.11	[Listeners] 機能パネル	24
2.12	[EventSpool] 機能パネルに保管されたイベントオブジェクト	25
2.13	セパレータを適用した画面	27
2.14	ドラッグイベントの個数について一般化を行った画面	28
2.15	コマンドルールのラベル生成ルールの定義	29
2.16	1つのコマンドルール	29
2.17	完成したコマンドルール	30
2.18	自動生成されたコマンドのラベル	31
2.19	Jedemo Animator: ユーザによるアニメーションの実行	33
2.20	部品とイメージ	35
2.21	メモリ使用量の比較	37
2.22	アニメーション実行時のメモリ使用量	37
3.1	inlineLink の内挿：(左) 既存のブラウザによる手法 (右) inlineLink 手法	45

3.2	inlineLink の機能アンカー	46
3.3	閲覧者の典型的な行動	47
3.4	階層的な内挿	48
3.5	全文内挿 (Full insertion)	49
3.6	部分内挿 (Partial insertion)	49
3.7	アンカーが書き替えられる場所, タイミングとそれぞれの内容	52
3.8	CSS によって変更した内挿文書の装飾	55
3.9	inlineLink 実験に用いたブラウザと文書	57
3.10	実験結果 (クリック数, 移動量, 作業時間の平均) と標準誤差	59
3.11	実験 (1)(2) と (3)(4) の比較: クリック数, 移動量, 作業時間の平均) . . .	60
3.12	実験 (3)(4) の結果 (クリック数, 移動量, 作業時間の平均)	60
4.1	社交的ヘルプにおける処理の流れ	69
4.2	(左) キーワードによるヘルプ検索画面 (右) キーワードを公開して反 応を得るチャットシステム Open Search Square	70
4.3	(左) システム情報やユーザ設定を自動的に付加する機能をもつ質問作 成ウィンドウ (右) 質問とその回答を表示する掲示板 Answering Board	71
4.4	チャットクライアント OpenSearchSquare	72
4.5	inlineLink を利用した Web 掲示板	72

表一覧

2.1	アイコンとイベントオブジェクトの対応	26
2.2	オブジェクト指定子	32
2.3	関連クラス数とサイズ	36
3.1	inlineLink 実験の流れ	59
3.2	実験結果 (クリック数, 移動量, 作業時間の平均)	61
A.1	inlineLink 実験に用いた文書 (日本語翻訳版)	88
A.2	inlineLink 実験に用いた文書 (英語版)	88
A.3	t 検定: 一対の標本による平均の検定ツール: クリック数	89
A.4	t 検定: 一対の標本による平均の検定ツール: 移動量	89
A.5	t 検定: 一対の標本による平均の検定ツール: 作業時間	89

第 1 章

導入

1.1 WWW

ネットワーク技術の向上，個人用計算機や携帯端末の普及により，インターネット利用者の数はこの数年で急激に拡大し，ネットワーク上のサービスは一般の人に広く使われるようになった．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，1990 年代後半から特に利用されるようになってきたのが，World Wide Web (WWW) である．WWW は，文書や画像，音声などの情報をネットワークの利用者に広く提示するためのシステムである [1]．利用者は，WWW ブラウザを用いることによりこれらの情報を取得できる．

1.2 ネットワークアプリケーション

WWW ブラウザや FTP クライアントのような，ネットワークに接続して各種サービスを利用するアプリケーション全体を指してネットワークアプリケーションと呼ぶことがある．本論文では狭義の定義として，ネットワーク上で流通するアプリケーション，特に必要な時にダウンロードされるアプリケーションをネットワークアプリケーションと定義する．

WWW ブラウザを用いると，Web ページと呼ばれる情報を閲覧する他に，プログラムなども実行できる．その中でも Java アプレットは，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，様々な環境で動作するという特徴がある．また，Java アプレットはこれらの特徴により，Web ページを閲覧するのと同様の手軽さで実行される．よって，ネットワークを介して流通する Java アプレットはネットワークアプリケーションの 1 つであると言える．

1.3 本論文の構成

本論文の構成は以下のようになる。

第 2 章：「ネットワークアプリケーションのためのアニメーションヘルプ」では、操作を説明する機能を持たない一般的な Java アプレットに対してアニメーションヘルプ機能を付加する方法について述べる。また、実際にユーザに提示するアニメーションを作成する際に、操作の内容を表す注釈 (文字列) を自動的に生成する技術について述べる。第 3 章：「挿入機能を用いた構造化文書の閲覧法」では、閲覧者がハイパーテキストを読む際に発生するナビゲーション負荷を抑えるための、構造化文書の閲覧手法とその実現について述べる。第 4 章では、第 2 章と第 3 章で述べた技術を利用した上で、従来のオンラインヘルプ検索時における問題を解決するための社会的インタラクションの導入について述べる。

第 2 章

ネットワークアプリケーションのためのアニメーションヘルプ

2.1 緒言

近年，計算機の普及と共に流通するアプリケーションの数は急激に増加している．また，それぞれのアプリケーションのバージョンが増す毎にそれらが提供する機能の数も増加する傾向にある．そのため，ユーザはそれぞれのアプリケーションを使うために，より多くの知識を要求される．

GUI (Graphical User Interface) は近年，多くのアプリケーションで採用されている．コマンド入力を主体とする CUI (Character User Interface) に比べて，GUI を用いたアプリケーションではメニューやアイコンを用いて機能をユーザに提示することができる．ユーザはマウスなどのポインティングデバイスを用いてメニューを選択したり，また画面内に表示された操作対象を直接編集する操作 (直接操作) を行うことができる．そのため，GUI は一般に CUI よりも短期間で操作できるようになるため，初心者にとって優れているとされている．

しかし，アプリケーションが多機能になるにつれ，メニュー項目やアイコンが増えるため，ユーザにとってどんな操作が可能なのかを理解しづらくなるという問題がある．ユーザにアプリケーションが持つ大量の機能を理解させるため，ヘルプやチュートリアルといったガイド機能をユーザに提供することがアプリケーション開発者にとって求められている．

2.1.1 テキストによる GUI のヘルプが抱える問題

一般に、ヘルプやチュートリアルはテキストで記述されることが多い。その理由として、テキストはアプリケーション特有の概念などを表現するのに有効で強力な手段であることが挙げられる。しかし、操作方法を記述するのにテキストが常に適切であるとは限らない。コマンド入力が主体となる CUI では、コマンドを提示することが操作方法を提示することになるため、コマンドをテキストで記述することに特に問題はない。しかし、画面内の操作対象を選択する作業が主体となる GUI の操作をテキストで表現する場合、以下の 2 つの問題が発生する。

- 操作対象の認知的不和
- 操作の認知的不和

それぞれについて、以下で詳しく説明する。

操作対象の認知的不和 GUI の操作説明において、操作対象は操作手順中頻繁に指示される。例えば、

「編集」メニューの「オプション」を選択
という操作指示が与えられたとする。ユーザはまずこの指示を読み、「編集」メニューを画面内から探し出して選択し、その後「オプション」メニューアイテムを探し出して選択するという手順を踏む。メニュー内にたくさん候補がある時には、それぞれ探し出すのが困難になる。

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テキストによる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」を言葉で説明する方法もあるが、直感的でない。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し出す必要がある。

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るならば、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」が理解されていない状態では、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、マウスのドラッグ操作が多く用いられる傾向にあ

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。

2.1.2 アニメーションによるヘルプ

上で述べた「操作対象の認知的不和」と「操作の認知的不和」の問題を解決するため、テキストによる説明の代わりに一連の操作をアニメーションを用いて提示する方法(アニメーションヘルプ手法)が提案されている [2, 3, 4, 5]。アニメーションヘルプでは、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により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 [6]。

2.1.3 研究の目的

一般的に、アニメーションによるヘルプは GUI を持つアプリケーションにとって有効であるが、我々は特に Java アプレットなどのネットワークアプリケーションに着目した。その理由として、Java アプレットなどのネットワークアプリケーションが対象とするユーザは、一般のアプリケーションのユーザとは性質が異なると考えられるからである。

Java アプレットは、一般のアプリケーションのように、ユーザがアプリケーションを明示的にインストールする必要がない。Java アプレットは WWW ページ内に記述された HTML(Hyper Text Markup Language) のタグ(<applet> もしくは <object> タグ)に従って、WWW ブラウザ上で自動的にロードされ実行される。アプリケーションを使う目的で訪問したわけではない場合、ユーザがアプリケーションを使うモチベーションは低い。このようなユーザに、アプリケーションの典型的な使用例を見せることによりユーザの興味を誘引する効果がある。

また、テキストによるヘルプをアプレットを置く WWW ページ内に記述する場合、説明を読みながらアプレットを使用することが画面領域の制約によって困難な場合がある。このような場合、ユーザはページをスクロールしながら説明の理解と操作を繰り返す必要があり、ユーザの負担は増す。

以上の理由により、我々は、Java アプレットにアニメーションヘルプを適用することに意味があると考える。

2.2 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、ユーザの知識レベルに合わせて説明内容の粒度を変更することも一般には困難である。

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によりアプリケーションの振舞いを見せる手法である。動画像の再生時におけるビューアのような特別なプログラムは必要ない。そのかわり、アプリケーションが振舞いを再現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、ヘルプとしてのアニメーションをイベントとして記録・再生する後者の手法を採用する。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。

2.2.1 フィードバックレベル

イベント駆動によって、実際のアプリケーションの振舞いをアニメーションで再現する場合に、何を記録・再現すればよいのかという点(フィードバックレベル)について考える必要がある。

Tk を用いたアプリケーションのイベントを記録、再生するシステム TkReplay を構築した Crowley は、GUI におけるユーザの入力に対するシステムのフィードバック

を以下の3つのレベルに分類し、その必要性について論じている [7].

1. 語彙 (lexical) レベル
2. 統語 (syntactic) レベル
3. 意味 (semantic) レベル

「語彙レベル」のフィードバックとは、ウィンドウマネージャが生成するフィードバックである。マウスの動きによるマウスポインタの移動や、キーボードから入力した文字のエコーバックなどが含まれる。「統語レベル」とは、画面内の部品 (ウィジェット) が行うフィードバックレベルである。オブジェクトを選択による強調や、ウィジェットのボタンを押したときの反転、スクロールバーのスライドやメニューの展開などが含まれる。「意味レベル」とは、アプリケーションが行うフィードバックである。文字のフォントを太くする、図形を歪ませるなどの視覚的なフィードバックだけではなく、メールの送付、ファイルの削除といった実際のアクションも含まれる。

Clowley は、システムのアニメーションで意味レベルを含むすべてのレベルのフィードバックを再現する必要はないと述べている。なぜなら、意味レベルで再現するためにはアプリケーション側が完全に再現する機構を備える必要があり、それはアプリケーションの実装を困難にするからである。統語レベルについても必要ではないが、ユーザがアニメーションの内容を理解するためにはあったほうがよい。語彙レベルのフィードバックについても厳密には必要ではないが、マウスポインタの移動による演出効果は高いと述べている。

我々は、Java アプレットのアニメーションについて、語彙レベルと統語レベルをサポートすることにした。意味レベルでのフィードバックを再現するには、アプリケーションの内部状態を完全に再現する機構が必要になり、場合によってはアプリケーションの設計から見直して再実装する必要があるため、現実的ではない。

2.2.2 実装レベル

語彙レベルと統語レベルのフィードバックを実現するための実装レベルについて考察する。

Bharat らは、X Window System 環境における再現のメカニズムを決定する上で、以下の7つの規範を掲げた [8].

- (a) 可搬性 (Portability)：一般的な環境ならばどこでも動作すること
- (b) アプリケーション境界サポート (Application End Support)：アイコンの位置などが変わっても動作することや、実行したことが確認できること
- (c) 頑強性 (Robustness)：他の要因に左右されずに高確率で再現できること
- (d) 表現性 (Expressiveness)：マウスの移動など、プリミティブなアクションを視覚化すること
- (e) プログラミング負荷 (Programming Overhead)：アプリケーションのプログラマやツールキット作成者にかかる負担を減らすこと
- (f) 侵入性 (Intrusiveness)：普通のアプリケーション操作などの邪魔をしないこと
- (g) セキュリティ (Security)：権限の与えられているイベントだけが実行できるように制限すること

その上で、Bharat らは Crowley と同様に、どのレベルで擬似イベントを生成するべきかを、以下のそれぞれのレベルで論じている。

1. デバイスドライバレベル
2. ウィンドウシステムレベル
3. アプリケーションレベル

デバイスドライバレベルでは、プラットフォームやサーバに依存してしまうため、(a) 可搬性 に欠ける。イベントを生成するレベルが低いとサーバの状態によって再現できない場合があり (c) 頑強性 も十分でない。また、個々のウィンドウによってイベントを規制することができないので (g) セキュリティ についても問題があるとしている。アプリケーションレベルでは、(d) 表現性 に問題があるとしている。その理由として、Crowley のいう語彙レベルのフィードバックが達成できないことを挙げている。また、(a) 可搬性 (c) 頑強性 (e) プログラミング負荷 についても問題があると述べており、結局、Bharat らはウィンドウシステムレベルを採用し、Xlib の拡張によって実現している。

Crowley は、Tk ツールキットウィジェットのレベルを採用している。その理由として、X Window System のレベルで記録するよりも頑強でアプリケーションの状態

を取得し易いことを挙げている。Bharat らは Artkit[9] というツールキットに拡張 Xlib を適用するための改良方法についても述べているが、アプリケーションとの距離が遠くなることは避けられない。

Java アプレットに関して言えば、特に WWW ブラウザが起動する Java Virtual Machine で動作するため、デバイスドライバやウィンドウシステムからのインタラクションは困難になる。アニメーションを実行している間にユーザがアプレットを表示している WWW ページをスクロールした場合、絶対座標でのアプレットの位置が移動したことをイベント送信側が認識する必要があるが、現実的に実現するのが困難である。よって、アプリケーションレベルもしくはツールキットレベル (Java Virtual Machine のレベル) で実現することが望ましい。

2.2.3 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により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。少なくともアニメーション再現システムには「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。

さらに、再現するアニメーション動作を簡単に記録するためは、「アプリケーションで発生したイベントを記録する機能」が不可欠である。この機能が欠けていると、アニメーションを制作するのに特殊なスクリプト言語などを用いて記述する必要がある。

2.3 アプレットのためのアニメーションヘルプ環境の設計

イベントを用いたアニメーションを効果的に行うためには、なるべくアプリケーションに近いレベルで記録、再現する必要があることを述べた。この章では、Java アプレットに適した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。

2.3.1 アニメーション機能の追加にまつわる問題

イベント駆動方式のアニメーションを記録、再生するためには、個々のシステムが以下の3つの機能

1. [initializing] アプレットをある状態 (例えば、初期状態) にする機能
2. [recording] アプレットで発生したイベントを記録する機能
3. [playing] アプレットに記録したイベントを実行させる機能

を実装することが必要であることを述べた。(以下では、これらの3つの機能を「アニメーション機能」と呼ぶ)

しかし、これらのアニメーション機能をそれぞれのアプレットについて実装するには、ソースコードの変更や再コンパイルといった作業が必要となり、開発者にとって余分な作業を強いることになる。このことは、(e) プログラミング負荷を増大させる。

アニメーション機能を個々のアプレットに実装せずに、アプレットを実行するシステム (Java Virtual Machine) に実装する方法もある。しかし、Java Virtual Machine を変更してアニメーション機能を実装することは得策ではない。アニメーションを再現するために特殊な環境を必要とするため (a) 可搬性が失われるからである。アニメーションヘルプを作成・編集する時はともかく、再生するのは一般ユーザであるため、標準の Virtual Machine 上で動作することが望ましい。

2.3.2 アプレットビューアモデル

我々は、上に述べた条件を満たしつつ、アニメーション機能を一般的なアプレットに持たせる「アプレットビューアモデル」という枠組みを提案した [10]。アニメーション機能を持たせようとするアプレット (以下「対象アプレット」と呼ぶ) を、イベント管理機構を持つアプレット (以下「マネージャアプレット」と呼ぶ) に貼り付けることにより記録と再生の機能を追加し、対象アプレットの機能を拡張することができる。

図 2.1 に、このアプレットビューアモデルの機構を示す。アニメーション機能を追加するために開発者が行う作業は、マネージャアプレットを呼び出す際にパラメータとして対象アプレットのクラス名とアニメーションのファイル名を指定するように HTML ファイルを書き替えるだけで済む。具体的には、図 2.2 [Normal Case] から [With Animation Help Functions] に示すよう変更することにより対象アプレット TargetApplet.class にアニメーション機能が追加できる。

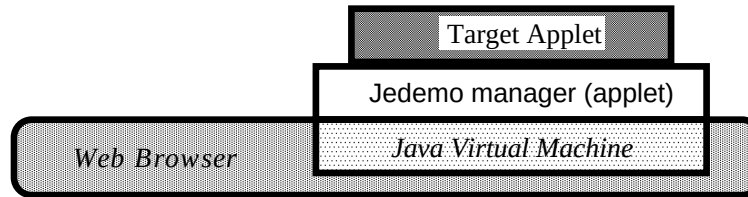


図 2.1: アプレットビューアモデル

Normal Case :

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

With Animation Help Functions :

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

図 2.2: HTML ファイルの書き換えによるアニメーションヘルプ機能の適用

アプレットビューアモデルは、特定のアプレットに限定しない機構を備えた上でツールキットレベルとアプリケーションレベルの中間にあたる実装レベルを用いているので、Bharat らの掲げた 7 つの方針を全て満足している。その理由を以下に示す。

- (a) 可搬性 (Portability) : 一般的な Java Virtual Machine があれば動作する
- (b) アプリケーション境界サポート (Application End Support) : 対象アプレットの公開している内部情報にアクセスでき、ウィジェットの位置なども取得できる
- (c) 頑強性 (Robustness) : アプレットと同等の頑強性で実行できる
- (d) 表現性 (Expressiveness) : プリミティブなアクションは擬似的に視覚化することができる
- (e) プログラミング負荷 (Programming Overhead) : アプレットのクラスに変更を加えずにアニメーション機能を追加できる

(f) 侵入性 (Intrusiveness) : Java Virtual Machine の中で閉じているので、その他のアプリケーションには影響を与えない

(g) セキュリティ (Security) : 対象アプレットとイベントは開発者が明示的に指定するので、通常のアプレットと同等のセキュリティレベルを保つ

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・再生する方法について述べる。

2.3.3 イベントオブジェクトの記録

以下では、対象アプレットの操作を記録する方法について詳しく述べる。

Java における部品

Java では、AWT (Abstract Window Toolkit) や JFC (Java Foundation Classes) Swing などのツールキットが提供するウィジェット (部品) を利用することができる。

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、親の部品への参照を持つ。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。

イベントリスナ

イベントリスナとは、部品で発生するイベントの実際の処理を行うオブジェクトであり、Java では標準的に用いられる。通常、プログラマが Java のアプリケーションを構築するときには、操作が行われたときの処理をこのイベントリスナに記述する。操作が行われる部品 (イベントを発行する部品) は、このイベントリスナを複数登録できる。登録したイベントリスナに関連したイベントが発生した場合に、部品がイベントオブジェクトを生成し、対応するイベントリスナのメソッドを呼び出す仕組みである。通常、イベントオブジェクトは処理された後捨てられる。

我々は処理された後のイベントオブジェクトを保存するため、特殊なイベントリスナ (Messenger) と、Recordable インタフェースを用意した。Messenger は、受け取ったイベントオブジェクトを Recordable インタフェースを備えるオブジェクトに転送す

る機能を持つ。マネージャアプレットが `Recordable` インタフェースを実装し、
`recordEvent(AWTEvent e)`
という名前と引数のメソッドにイベントオブジェクトを呼出し順に保存する機能を実装することにより実現する。

我々は以下の4種類のイベントリスナ

- `MouseListener`
- `MouseMotionListener`
- `ContainerListener`
- `ActionListener`

にそれぞれ対応する、

- `MouseMessenger`
- `MouseMotionMessenger`
- `ContainerMessenger`
- `ActionMessenger`

というクラスを用意した。以下に `ActionMessenger` クラスの定義を示す。

```
public class ActionMessenger implements ActionListener{
    Recordable recorder; // マネージャアプレット
    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
    }
}
```

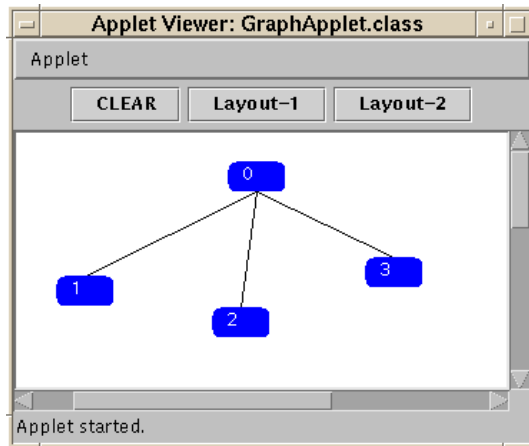


図 2.3: GraphApplet の画面

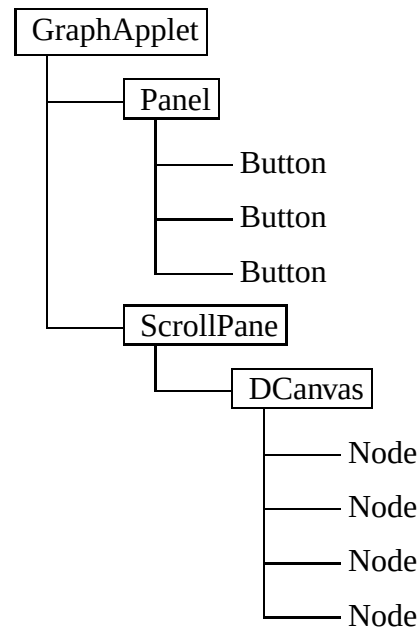


図 2.4: GraphApplet の部品階層構造

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 2.3 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 2.4 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部品はコンテナを表す。

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。マネージャアプレットは、この情報を調べていくことによって、対象アプレットを構成する部品の階層構造における位置を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 2.5 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、マネージャ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、イベント再現時にオブジェクトを特定する情報として用いられる。

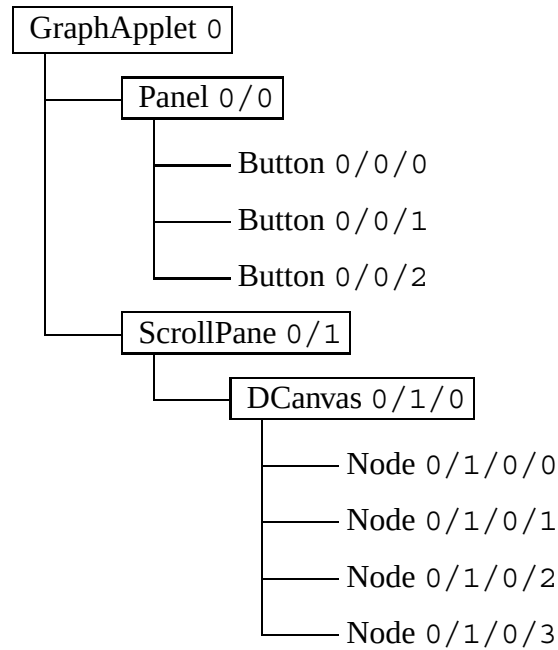


図 2.5: 部品の位置を示すパス情報

イベントオブジェクトの記録

対象アプレットの操作を記録するには、以下の手順によって準備を行う。

1. マネージャ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)。
2. マネージャ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、イベントがマネージャアプレットに通知される。マネージャアプレットはそれらのイベントが発生した順番で保存する。

記録可能なイベントの種類は、Recordable インタフェースが規定するメソッド `recordEvent()` の引数の設計によって決まる。例えば、`java.awt.AWTEvent` を引数とした場合、`AWTEvent` のサブクラスであるイベントが記録可能である。実際に記録するイベントの種類は、各部品に追加する Messenger イベントリスナによって決定する。

動的に登録される部品

対象アプレットにおいては、実行中に部品が作成されて登録 (追加) されることがある。例えば、図 2.5 における Node は、初期状態では 4 つであるが、新しい Node が作成される毎に増加する。これらの部品についても、発生したイベントを記録する必要がある。また、部品が削除された場合には、マネージャアプレットが管理している部品の階層構造 (モデル) を更新して一貫性を保つ必要がある。

対象アプレットにおける部品の追加・削除をマネージャアプレットに通知するため、我々はコンテナイベント (Container Event) を通知する「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・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発生し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。

イベントの記録を始める前に、マネージャアプレットは対象アプレット内の部品のうち、コンテナ部品全てにこのコンテナイベントリスナを登録する。あるコンテナに部品が追加・削除されると、マネージャアプレットにコンテナイベントが通知される。マネージャ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発生したコンテナについて、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。

2.3.4 記録した操作の再現

2.3.3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できた。本節では、これらのイベントを用いて、操作を再現する方法について述べる。

イベントソース部品

ComponentEvent クラスのサブクラスとして定義されているイベントのインスタンスは、発生時に「イベントを生成した部品への参照 (イベントソース部品)」を持つ。この情報は、ComponentEvent.getComponent() メソッドで調べることができる。

イベントが発生したときの動作を再現するには、イベントソース部品の dispatchEvent(AWTEvent event) を呼ぶことにより実現できる。ここで、event というのが再現するイベントである。

イベントソース部品への参照は、イベントを記録した時点では有効だが、イベントをファイルに保存し、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、部品のインスタンスが読み込まれるメモリ上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可能性が高いからである。よってユーザがアニメーションを参照する時点で、開発者が記録したイベントのイベントソース部品への参照をそのまま利用しようとする、正しく操作を再現できない。

この問題を解決し、失われたイベントソース部品への参照を回復するために、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「イベントソース部品」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) を記録しておく方法も考えられる。しかし、この「座標による方法」では、レイアウトの変更などによって対象アプレット内の部品の存在する位置が移動するとイベントソースオブジェクトを正しく特定できなくなる。これに対して、「パス情報による方法」では、プログラム中で指定する部品の登録順が変更されない限り、イベントソース部品を特定できる。そのため、レイアウトの変更に対して頑強である。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により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。

その他の方法として、部品に登録されたラベルやを比較することによって正しいイベントソース部品かどうかを検査することができる。これらの方法はパス情報による参照の誤りを検知し、開発者に注意を促すために利用している。

2.3.5 操作の意味付け

イベントオブジェクトとコマンド

Java アプレットにおいては、一般に操作を行うことによって大量のイベントオブジェクトが発行される。これらのイベントオブジェクトの多くは「マウスを動かした」などの低レベルなもので、単体では意味を持たない。アニメーションを編集するのに低レベルなイベントを単位として扱うのは効率が悪い。また低レベルイベント列の意味を理解していないと編集できないという問題もある。

我々はこれらの問題を解決するため、イベント列をまとめた**コマンド**という概念を導入した [11]。1つのコマンドは最低限の意味を持つイベント列である。アニメーションを編集する時には、このコマンドを単位として削除や順番の入れ替えを行うことにより低レベルな情報を隠蔽する効果が得られる。

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

コマンドルール

コマンドとしてイベント列をまとめて管理するためには、どのようなイベント列がコマンドになり得るのかといった情報が必要である。一般に、対象アプレットのにおけるイベントとそれに対応する振舞いは、実装によって異なる。

対象アプレットが処理内容に関する詳細なメッセージをマネージャアプレットへ送る仕組みを備えていれば、マネージャアプレットがイベント列をコマンドとして認識することは可能である。しかし現実的には、それぞれの対象アプレットを変更しない限り実現することは困難である。対象アプレットを変更することは汎用性を失わせ、プログラマの負担を増大させる。

我々は対象アプレットにおける「イベントと、対応する振舞い」という知識をマネージャアプレットに与えることによって対象アプレットの処理内容をマネージャアプレットに推測させ、イベント列に対して意味付けを行うことにした。この知識を**コマンドルール**と呼ぶ。

それぞれのコマンドルールは、

- イベント列のパターン
- ラベル生成ルール

を持つ。もし、あるイベント列がコマンドルール中のパターンとの照合に成功した場合は、そのイベント列がコマンドとして対応付けられ、ラベル生成ルールが意味付けを行う。

コマンド生成の手順

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

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

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

実際の前処理について述べる。イベント列の先頭からイベントオブジェクトを1つ読み込む。もしそのイベントがセパレータに含まれるならば、それはイベント候補になり得ないので捨てる。セパレータに含まれないのであれば、そのイベントオブジェクトから次にセパレータに含まれるイベントオブジェクトの1つ前までを、1つのコマンド候補とする。この作業を繰り返し行うことにより複数のコマンド候補をイベント列から切り出すことができ、コマンドルールとの照合が効率良く行える。

2.4 実装システム

上に述べた技術を用いて Java アプレットのためのアニメーション編集・再現システム Jedemo¹を実装した [12, 13]。本節では具体的にアニメーションを作成し、編集して公開するまでを順を追って説明する。

主に開発者が用いるアニメーション編集システム (Jedemo Author) は、Java アプリケーションとして実装した。ここで、開発者が行う作業は以下の4つである。

¹Jedemo = Java Event-driven Demonstration の略

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

2.4.1 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 2.6 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．

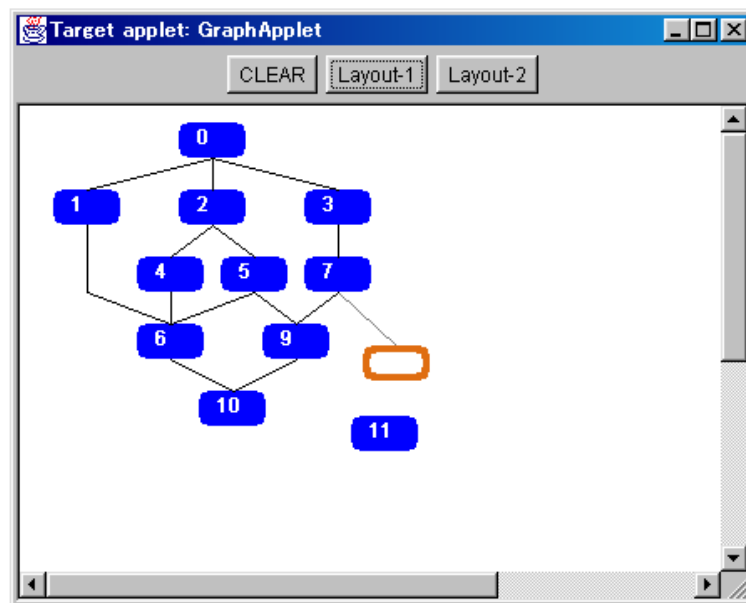


図 2.6: 対象アプレットとして用いたグラフ編集アプレット

グラフ編集アプレットでは，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. 新しいノードを作成する．キャンバス (画面上の白い部分) 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．

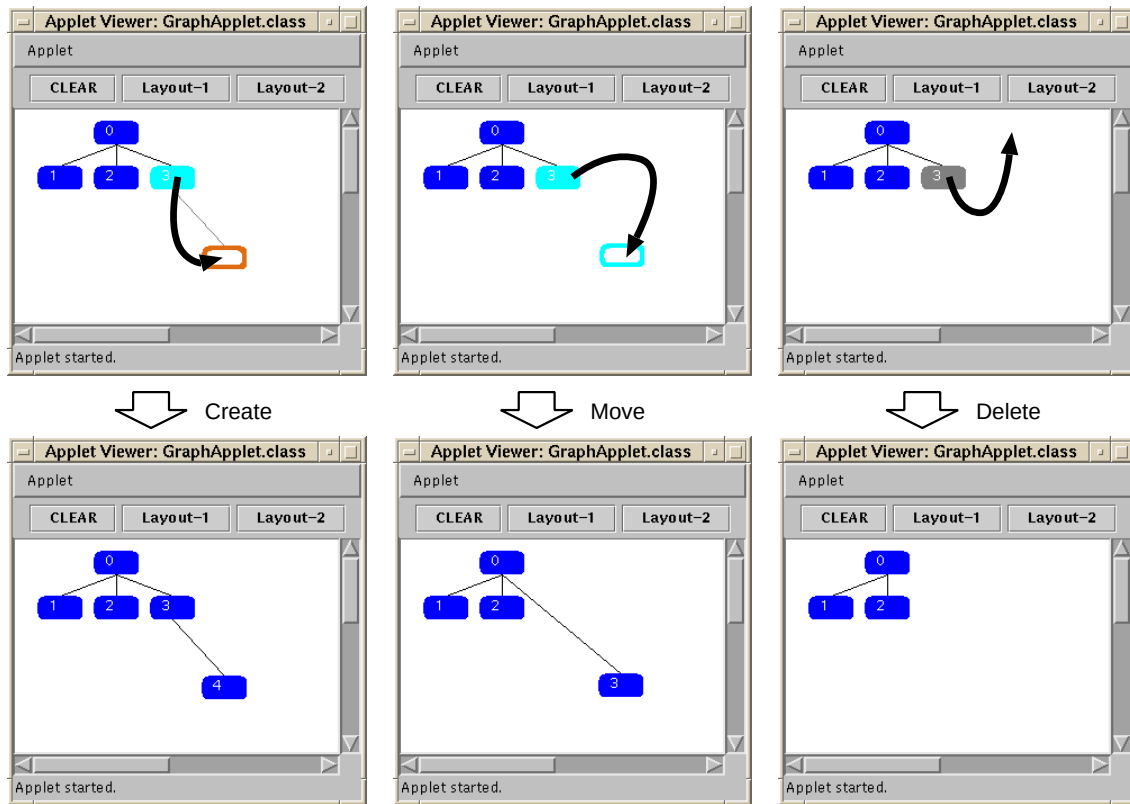


図 2.7: 対象アプレットの操作：ノードの作成 (2.), 移動 (3.), 削除 (4.)

2. あるノードの子ノードを作成する．ある既存のノードでクリックし，下方向にドラッグした後キャンバスでリリースする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 2.6** はこの操作におけるドラッグ中の画面である．(**図 2.7** 左も参照)
3. ノードを移動する．移動したいノードでクリックし，横方向または上方向にドラッグした後キャンバスでリリースすると，ノードを移動できる．2. と 3. の機能の切り替えは，マウスカursor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，それ以外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横方向にドラッグした後，下にドラッグすることにより実現できる．(**図 2.7** 中央を参照)
4. ノードを削除する．消去したいノードをクリックし，キャンバス上部でリリースする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 2.7** 右を参照)

5. リンクを張る. 親にあたるノードでクリックし, そのままドラッグしながら子にあたるノードでリリースする. 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でリリースする. 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 2.3.3 の **図 2.4** (14ページ) に示した通りである. **図 2.7** 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, ドラッグ操作のジェスチャによって異なる動作を割り当ててある.

2.4.2 対象アプレットの指定と初期設定

Jedemo Author を Java アプリケーションとして起動し, 対象アプレットのクラスファイルまたは対象アプレットを呼び出している HTML ファイルを指定する (**図 2.8**). ここで指定する HTML ファイルは, アニメーション機能を追加する前に Web で公開していたものと同じファイルでよい. ここでは, 例としてグラフ編集アプレットを起動する HTML ファイル (GraphApplet) を指定した. Jedemo Author は指定された HTML ファイルに含まれるアプレットタグの情報から対象アプレットを起動する (**図 2.9**). Jedemo Author は 6 つの機能パネル (FrameSize, CompoTree, Listeners, EventSpool, CommandRules, CommandGenerator) をタブで切り替えて左から順番に操作していくことによりアニメーションが準備できる.

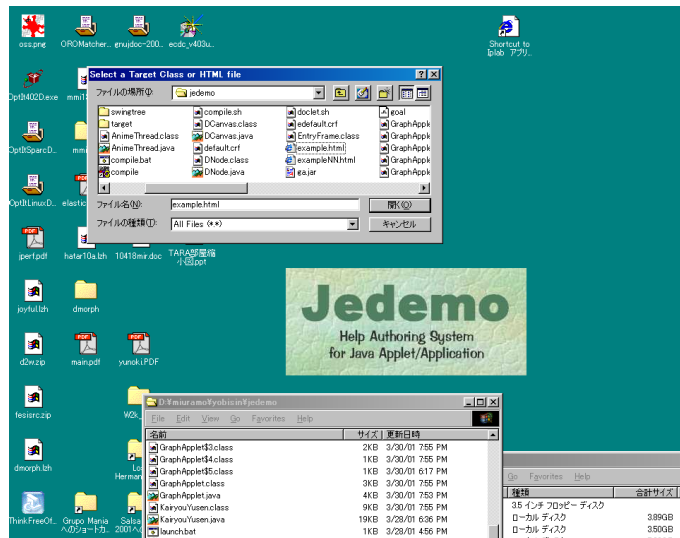


図 2.8: 対象アプレットの指定

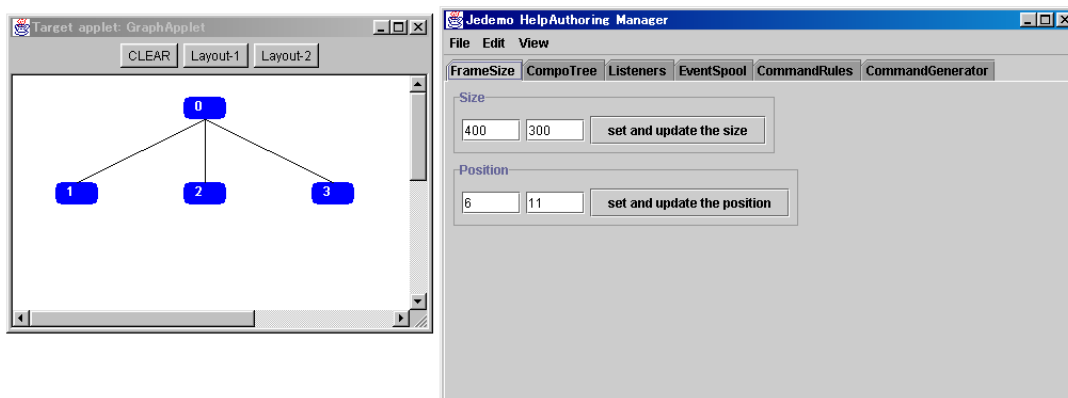


図 2.9: [FrameSize] 機能パネル

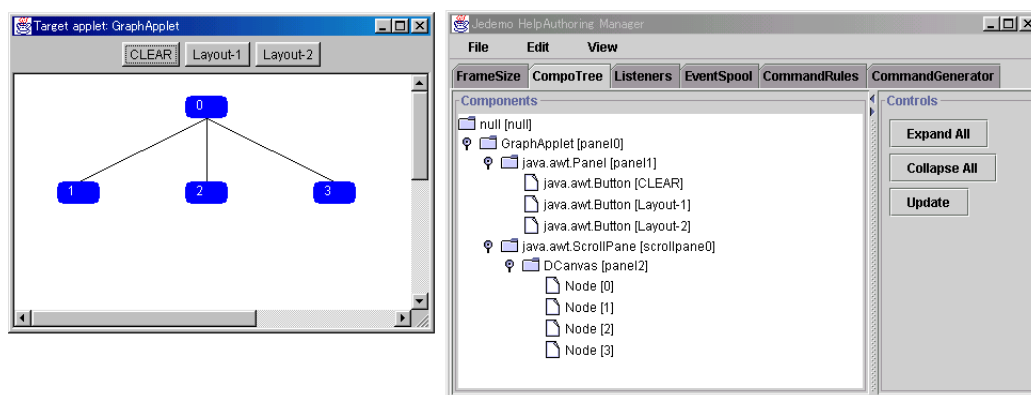


図 2.10: [CompoTree] 機能パネル

対象アプレットウィンドウの調整 — [FrameSize]

[FrameSize] 機能パネルでは、対象アプレットウィンドウのサイズと位置を表示している。対象アプレットのサイズ変更や移動を行うと Jedemo Author のプロパティに反映される。また、Jedemo Author のプロパティにサイズや位置を入力し、[set and update the size] (位置の場合は [set and update the position]) ボタンを押すことにより対象アプレットウィンドウのサイズや位置を変更できる (図 2.9)。

対象アプレットの部品構造 — [CompoTree]

[CompoTree] 機能パネルでは、対象アプレットに含まれている部品構造を木構造 (ツリー) で表示する (図 2.10)。また、部品に関して Jedemo Author が得た情報 (クラス名とラベル) を表示する。[Expand All] ボタンを押すことにより部品階層ツリーを全て展開表示する。[Collapse All] ボタンを押すと部品階層ツリーを全て閉じた状態にする。[Update] ボタンを押すと現在の対象アプレットの部品構造から部品階層ツリーを再構成し表示を更新する。

2.4.3 操作の記録

イベント記録用リスナ (Messenger) の調整 — [Listeners]

[Listeners] 機能パネルにある [add Listeners] ボタンを押すと、Jedemo Author は [CompoTree] 機能パネルで表示されていた部品にイベント記録用リスナ (Messenger) を追加する (図 2.11)。ここで追加される Messenger は、[Installed Extra EventListeners] 欄に表示されている。

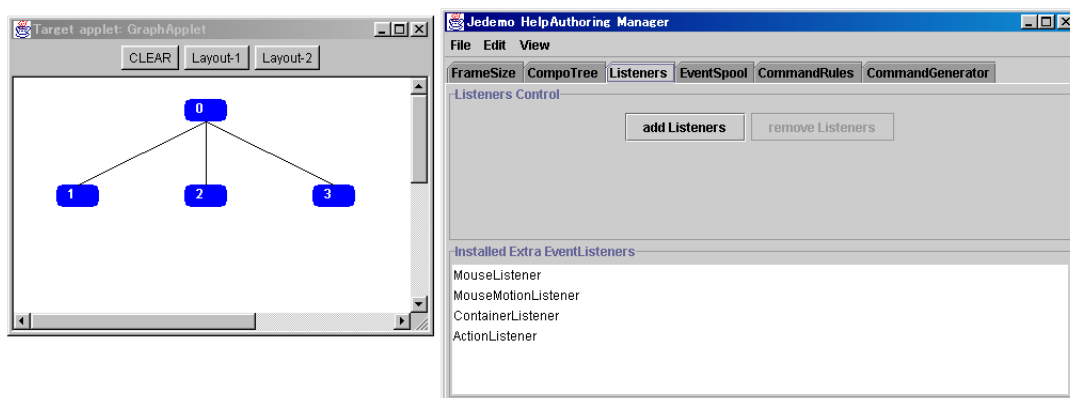


図 2.11: [Listeners] 機能パネル

記録したイベントの一覧表示 — [EventSpool]

Messenger が追加され、かつ [EventSpool] 機能パネルが選択されている状態では、イベントオブジェクトは Jedemo Author に送られる。ここで送られたイベントオブジェクトは [EventSpool] 機能パネルが持つスプールに保管される。保管されたイベントオブジェクトはアイコンで表示される。イベントオブジェクトとアイコンとの関係を 表 2.1 に示す。ここで開発者は、典型的な操作を行いスプールにイベントオブジェクトを保管する (図 2.12)。

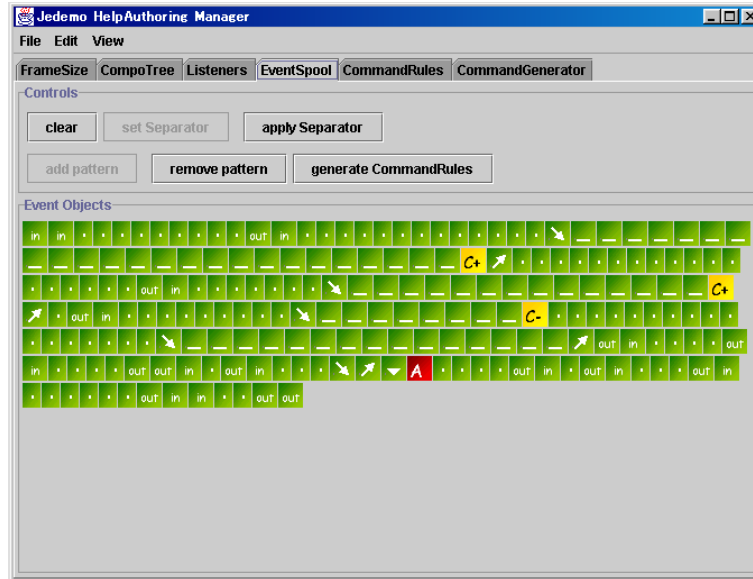


図 2.12: [EventSpool] 機能パネルに保管されたイベントオブジェクト

2.4.4 コマンドルールの定義

典型的な操作を行った後で、開発者は、コマンドルールを定義することにより生成するイベント列に対する注釈を自動生成することができる。コマンドルールの定義は、

1. セパレータの指定と適用
2. コマンドルール候補の一般化
3. コマンドルールの意味定義

という手順で行う。

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

表 2.1: アイコンとイベントオブジェクトの対応

セパレータの指定と適用 — [EventSpool]

イベントオブジェクトの中で意味を持たない部分を「セパレータ」として指定する。セパレータを指定するには、

1. [EventSpool] 機能パネルに表示されたアイコンのうち、セパレータ部分をドラッグし、範囲選択する
2. [set Separator] ボタンを押す

という手順で行う。すると、選択された範囲に含まれるイベントオブジェクト要素が1つずつ選択され、画面上部に表示される。ここで [apply Separator] を押すことにより実際に指定されたセパレータによってイベント列が分割され、コマンドルール候補が抽出される。例として、[図 2.12](#) におけるイベントオブジェクトのうち、GraphApplet において意味を持たないマウスの移動イベントオブジェクト    をセパレータとして選択し、適用した。セパレータを適用することによりセパレータとコマンドルール候補が[図 2.13](#) のように交互に現れる。

コマンドルール候補の一般化 — [EventSpool]

コマンドルール候補からコマンドルールを作成する前に、コマンドルール候補の一般化を行う。これは、コマンドルールのパターンを一般化することに相当する。

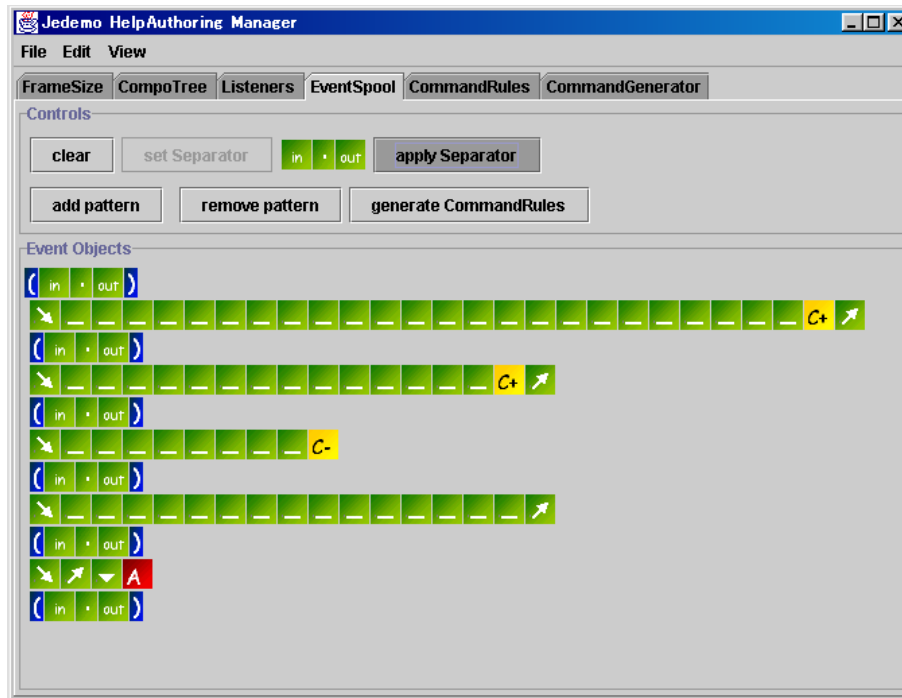


図 2.13: セパレータを適用した画面

図 2.13 において、スプール内に表示されている上 2 つのコマンドルール候補は、どちらも「新しいノードを作成」するときに発生したイベント列をパターンとして持つ。これらのコマンドルール候補の相異点は、マウスドラッグイベント  の個数のみである。GraphApplet では、マウスドラッグイベントの個数によって動作を区別しない。そのため、これらのコマンドルール候補は同一のコマンドルールとして扱ったほうがコマンド生成時の比較が効率良く行える。また、コマンドルールの意味定義における手間も軽減できる。よって、コマンドルール候補のパターンにおけるドラッグイベントの個数を一般化する。

ドラッグイベントの個数について一般化するには、

1. [EventSpool] 機能パネルに表示されたアイコンのうち、ドラッグイベント部分をドラッグし、範囲選択する
2. [add pattern] ボタンを押す

という手順で行う。もし、適用した一般化を取りやめる場合は [remove pattern] ボタンを押す。

図 2.13 のドラッグイベントの個数について一般化した結果を図 2.14 に示す。

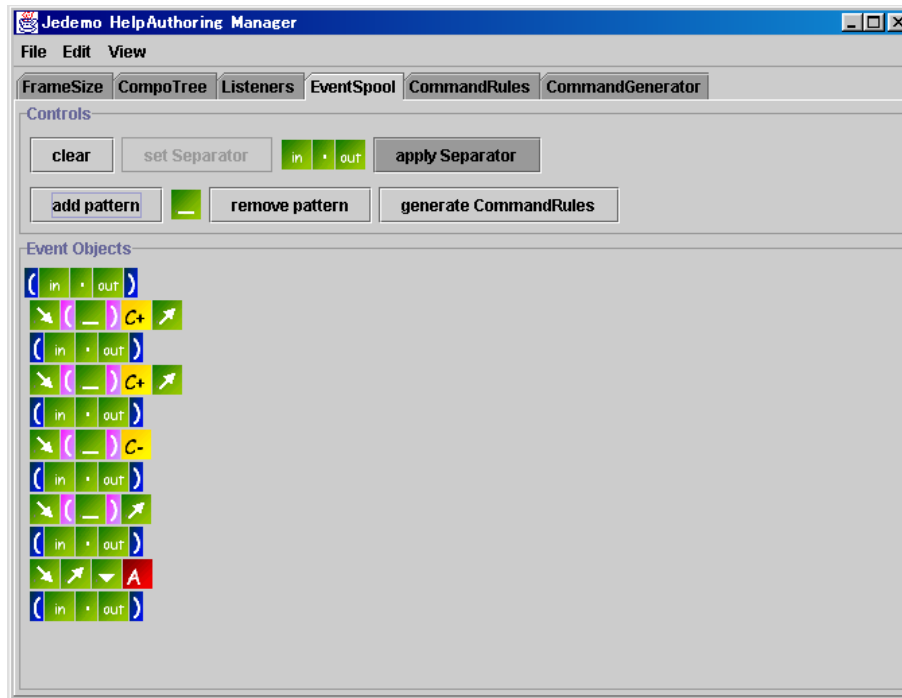


図 2.14: ドラッグイベントの個数について一般化を行った画面

コマンドルールの意味定義 — [CommandRules]

適切なコマンドルール候補の一般化を行った後で、実際のコマンドルールを定義し、作成する。コマンドルールの定義は、コマンドルールによって照合に成功した場合に付加されるイベント列の意味を表すラベルを生成するためのルールを記述することによって行う。

図 2.14 の画面において、[generate CommandRules] ボタンを押すと、コマンドルール候補のうち、異なるものが1つずつ選択され、自動的にコマンドルールが生成される。生成されたコマンドルールは[CommandRules] 機能パネルに表示される (図 2.15)。図 2.15 では、1つのセパレータと、4つのコマンドルールが表示されている。1つのコマンドルールは、図 2.16 に示すラベル生成ルール (Labeling Rule) と、イベントパターン (Event Pattern) から構成されている。ラベル生成ルールをクリックすると、内容を編集することができる。編集が終わったら Return キーを押すか [ok] を押す。編集が終了したコマンドルールを図 2.17 に示す。

その他、[CommandRules] 機能パネルには、チェックボックスで選択したコマンドルールを削除する機能、ファイルに保存したコマンドルールを読み込む機能、コマンドルールをファイルに保存する機能を持つ。

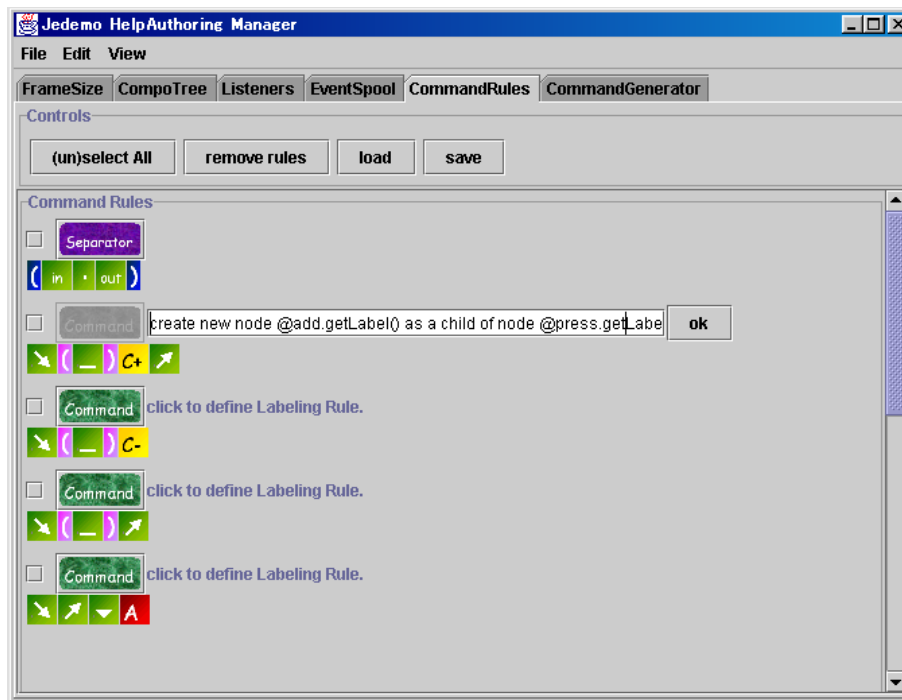


図 2.15: コマンドルールのラベル生成ルールの定義



図 2.16: 1 つのコマンドルール

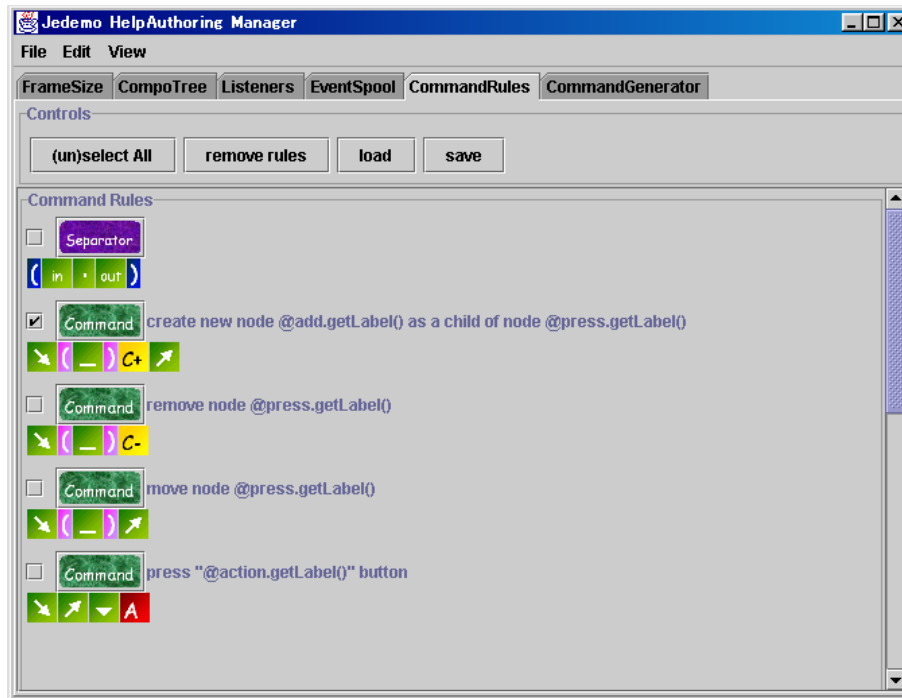


図 2.17: 完成したコマンドルール

コマンドの生成と保存 — [CommandGenerator]

定義したコマンドルールを用いると、イベント列から自動的にコマンドを生成することができる。

Messenger が各部品に追加されていて、さらに [CommandGenerator] 機能パネルがタブによって選択されている状態の時、イベントオブジェクトは Jedemo Author に送られる。ここで送られたイベントオブジェクトは [CommandGenerator] 機能パネルが持つプールに保管される。ただし、[CommandGenerator] 機能パネルは [EventSpool] 機能パネルのように 1 つ 1 つのイベントを表示しない。代わりに、イベント列をコマンドルールによって照合、生成されたコマンドのラベルを表示する。ここで照合されるコマンドルールは、[CommandRules] 機能パネルに読み込まれているものが使用される。

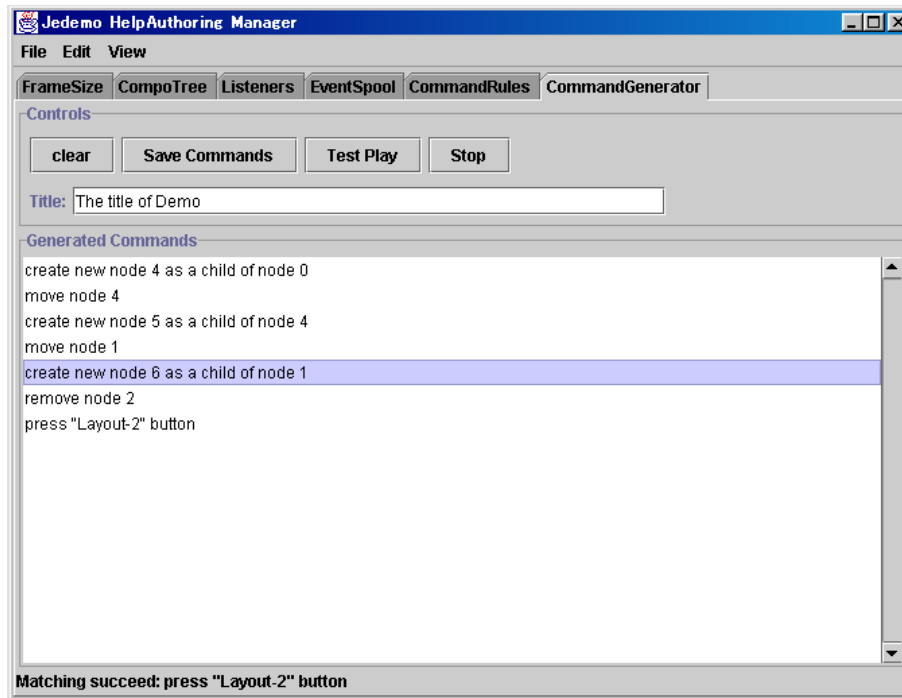


図 2.18: 自動生成されたコマンドのラベル

コマンドルールのラベル生成ルールによって付加されるラベルには、静的な文字列だけでなく、動的な情報を挿入することができる。ラベル生成ルールにおいて

“@press.getLabel()”

と指定した部分は、マウスボタンが押された部品 `getLabel()` メソッドの返り値に置き換えられる。このメソッド呼び出しによる置き換えは、コマンドルールの照合時に行われる。ラベル生成ルール内においてオブジェクトを指定する特別な文字列 (オブジェクト指定子) として、現在 **表 2.2** にあるものが利用できる。

開発者は、対象アプレットにおける典型的な操作を行うだけで、コマンドルールによるラベル付けを行うことができる。そのため、それぞれのイベント列に対して操作の内容を表す文字列を入力する必要がなくなり、アニメーション生成の手間が軽減できる。GraphApplet において記録したコマンドの例を **図 2.18** に示す。

このようにして生成したコマンドと、その元となるイベント列が、アニメーション生成時に用いられるデータとなる。このデータをファイルに保存するには [Save Commands] ボタンを押してファイル名を入力する。

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

表 2.2: オブジェクト指定子

2.4.5 アプレットタグの変更

開発者がアプレットを指定する HTML ファイルのアプレットタグを書き替えて、Jedemo Animator を指定すると、対象アプレットのアニメーションをユーザが再現できるようになる。GraphApplet において記録したアニメーションのデータファイルを sample.jdm というファイル名で保存した場合、以下のようにして対象アプレットとアニメーションのデータファイルをパラメータとして指定する。

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

2.4.6 Jedemo Animator

開発者によってアニメーション機能が提供されているアプレットをユーザが実行すると、まず Jedemo Animator が開始する。Jedemo Animator は関連付けられている対象アプレットを読み込み、アプレット領域に貼り付ける。

ユーザが Jedemo Animator の [start DEMONSTRATION] ボタンを押すと、アニメーションファイルが読み込まれる。コントロールウィンドウが開き、その中にアニメーションに含まれるコマンドのラベルが一覧として表示される。

ユーザがコントロールウィンドウにある [Play] ボタンを押すことによりアニメーションが開始され、開発者によって記録された操作を連続的な動作として見るができる。図 2.19 に、実際に生成されたコマンドを実行している画面を示す。擬似マウ

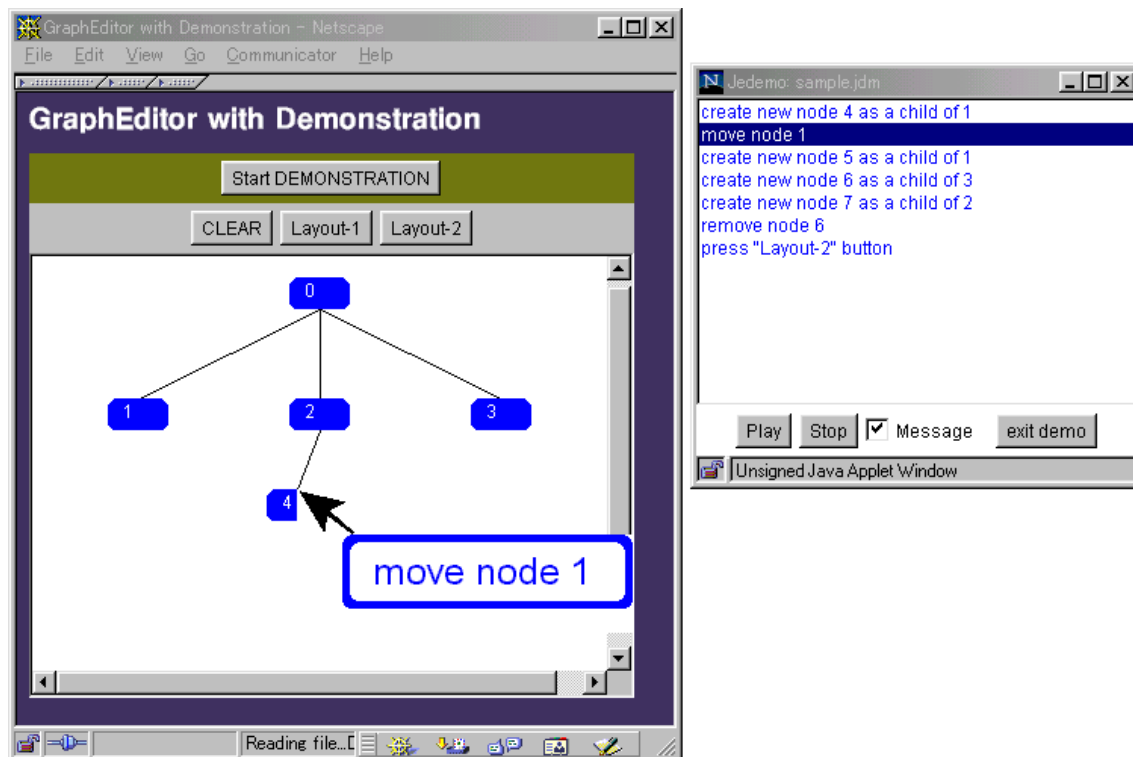


図 2.19: Jedemo Animator: ユーザによるアニメーションの実行

マウスカーソルが画面内を動くことによりユーザの注目すべき場所を指示する。アニメーション制御ウィンドウにコマンドのラベルを一覧表示することにより現在の位置を示したり実行の制御をする。アニメーションの実行スピードやポップアップメッセージの有無などの設定はユーザが自由に変更できる。擬似マウスカーソルやポップアップメッセージは `java.awt.Window` クラスのサブクラスとして実装したため、画面の自由な位置に配置・表示できる。

2.5 議論

2.5.1 コマンドルールの記述

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、必ずしも対象アプレットの実装におけるイベント処理の方法と等価である必要はない。

例として用いた `GraphApplet` では、ノードの上で開始するドラッグイベントは3つの異なる動作に使われている。1つは、ノードの消去である。この操作はドラッグした後、ボタンを離れた位置が画面上部または画面外であれば、ドラッグを開始したノー

ドを消去する。2つめは、子ノードの生成である。ボタンを押した後、ノード底部を通してノード外にドラッグしていった場合(以下、ドラッグアウトと表現する)、新たなノードを生成して、それをボタンを押したノードの子として登録する。3つめは、ノードの移動である。ノード底部以外からドラッグアウトした場合、ボタンを押したノードを離れた位置へ移動する。

これらの知識を厳密にコマンドルール内で定義するには、ノードのどの部分を通して外部にドラッグアウトされたかを記録する必要がある。しかし、例では、ノードが消去されたときに発生するコンポーネントイベント **C-** を含むか、新しいノードが作られたときに発生するコンポーネントイベント **C+** を含むか、それともコンポーネントイベントを含まないかによって区別している。このように、対象アプリケーションの振舞いや、追加された部品の種類(クラスなど)を取得すれば、実装とは異なる方法でコマンドを特定することができる。このことは複雑なジェスチャインタフェースを用いたシステムであってもコマンドルールの定義は比較的簡潔に行なえる可能性があることを意味している。

2.5.2 Jedemo の限界

Jedemo は基本的に、対象アプレットの実装に関らずイベントを記録、コマンドルールを用いたコマンドの生成、実行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条件にあてはまる場合には期待される動作が行われれないという問題がある。

コマンドルールが定義できない事例

Jedemo 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。対象アプレットが単一の部品で構成されている場合には、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 (図 2.20右) を区別することができないので、コマンドルールが簡単に定義できない。しかし、多くのアプレットやアプリケーションはコンポーネント技術を用いて複数の部品を組み合わせる (図 2.20左) 実装を行う傾向にあり、現実的なアプレットの多くは対応可能であると考えている。

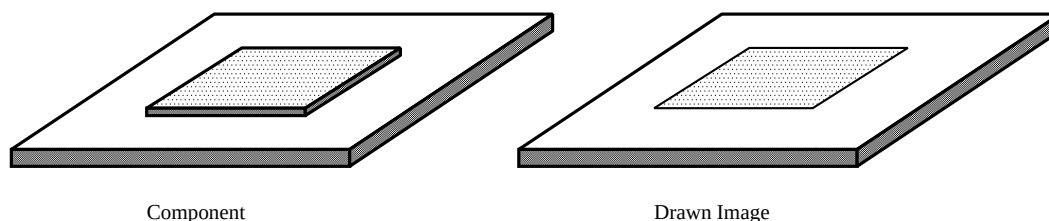


図 2.20: 部品とイメージ

再現できない事例

偶然性や時間経過によって状態変化が発生する場合、再現に失敗する可能性が高い。正しく再現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。

2.5.3 Jedemo によるオーバーヘッド

Jedemo によってアニメーション機能を追加した場合、アプリケーションのパフォーマンスが低下することが考えられる。そこで、Jedemo のアニメーション機能を組み込むことによるオーバーヘッドの計測を試みた。また、アニメーション実行中のオーバーヘッドについても計測を行った。

実験の設定

対象アプレットとして GraphApplet を用いた。通常の GraphApplet と、Jedemo Animator から指定することによりアニメーション機能を組み込んだ GraphApplet それぞれについて、実行時のメモリ使用量を計測した。

アプレット実行環境について述べる。PC (Pentium III 500MHz, 256MB memory) にインストールされた Windows 98 Second Edition 上の Netscape Communicator 4.75 を使用した。Java Virtual Machine (Java VM) は、この Web ブラウザに実装されていた Netscape Communications Corporation – Java 1.1.5 である。また、この Java VM には Symantec Java! ByteCode Compiler Version 210.065 が利用されている。

メモリ使用量を計測するために Java Console を用いた。Java Console にて [g] キーを押すことによりガベージコレクション (GC) が行われ、GC 後のメモリ使用量が以下のように表示される。我々は、Java が確保したメモリ領域 (memory) から、解放領

	GraphApplet	Jedemo Animator
クラス数	14	47
合計サイズ (Byte)	44,165	78,814
jar 圧縮後の合計サイズ (Byte)	26,970	50,483

表 2.3: 関連クラス数とサイズ

域 (free) を引いた値をメモリの使用量として用いた。この値は、純粋にアプレットによる使用量ではなく、Java Virtual Machine の使用量を含んでいる。

```
# Performing a garbage collection...
# GC complete: memory: 557080 free: 162952 (29%)
```

アプレット動作中には多数のスレッドが動作しているため、Java Console が表示するメモリ使用量は GC を行う回数によって多少ばらつきが出る。今回は GC を数回連続して行った後の安定した値をメモリ使用量として記録した。なお、アプレットのサイズは (400, 400) に統一した。

GraphApplet と Jedemo Animator のバイトコンパイルは、Java Development Kit 1.1.3 に附属する javac を用いて行った。関連クラスの数とサイズを表 2.3 に示す。

アニメーション機能の追加にかかるオーバーヘッド

GraphApplet と、Jedemo Animator から呼び出した GraphApplet の 2 つについて、以下のタスクを順に実行した後のメモリ使用量を計測した。

1. 初期状態で [Clear] ボタンを押す。 (4nodes)
2. 10 個のノードを作成する。 (10nodes)
3. 20 個のノードを作成する。 (20nodes)
4. 30 個のノードを作成する。 (30nodes)
5. 30 個のノードを作成した状態で、[Layout-1] ボタンを押す。 (30nodes layout1)
6. さらに、[Layout-2] ボタンを押す。 (30nodes layout2)

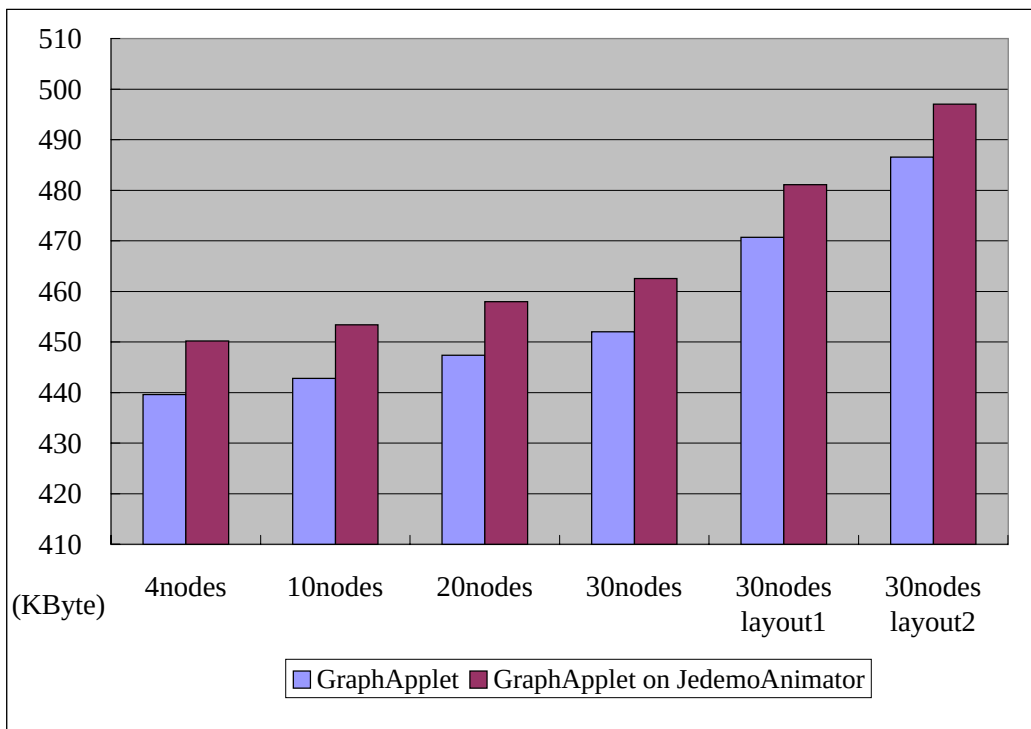


図 2.21: メモリ使用量の比較

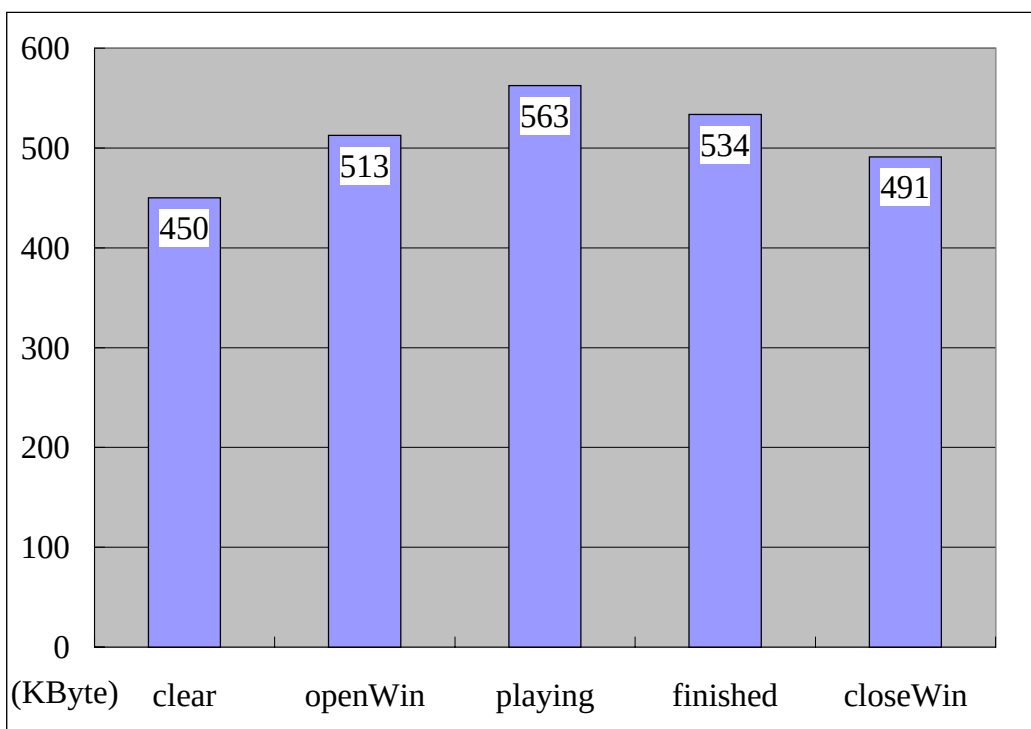


図 2.22: アニメーション実行時のメモリ使用量

図 2.21 に、計測結果を示す。縦軸がメモリ使用量 (KByte)、横軸が各状態を表す。全ての状態において、約 10KByte の差が生じている。これは JedemoPlayer から GraphApplet を呼び出した際、関連クラスのインスタンス生成によるオーバーヘッドと考えることができる。アニメーション機能を追加した初期状態のメモリ使用量は、GraphApplet を操作してレイアウト機能呼び出した時の使用量よりも少ない。よって、アニメーション機能を追加したことによるオーバーヘッドはアプレットの使用量に比べたら高々知れている。

アニメーション実行時のメモリ使用量

実際にアニメーションを実行している場面でのオーバーヘッドを調べるために、アニメーション実行時のメモリ使用量について計測した。Jedemo Animator によってアニメーション機能が追加された GraphApplet において以下のタスクを順に実行した後のメモリ使用量を計測した。

1. 初期状態で [Clear] ボタンを押す。 (clear)
2. [start Demonstration] ボタンを押してアニメーションファイルをロードし、コントロールパネルを表示する。 (openWin)
3. [play] ボタンを押してアニメーションを再生する。 (playing)
4. アニメーション再生を終了する。 (finished)
5. [exit Demo] ボタンを押してコントロールパネルを消す。 (closeWin)

図 2.22 に、計測結果を示す。縦軸にメモリ使用量 (KByte)、横軸に計測時の状態を表す。状態 (openWin) において、アニメーション実行のためのクラスとアニメーションファイル (18,829Byte) のロードが行われるためメモリ使用量が増加する。状態 (playing) において、アニメーション実行に用いられるスレッドやポップアップメッセージ、擬似マウスカーソルなどの表示が行われるため、さらにメモリ使用量が増加する。ちなみに、ここで記録したメモリ使用量は、再生中の記録したデータのうち最大のものを示している。状態 (finished) においては、アニメーション再生のためのスレッドが利用していた分が減少し、状態 (closeWin) においては、コントロールパネルウィンドウが消費していた分が解放されている。最もメモリを消費するアニメーション実行時

であっても、高々 113KByte 程度のオーバーヘッドで済むことから、Jedemo Animator が動作する計算機を限定することはないといえる。ちなみに、Java のトップページ²を開いたときに実行される 5 つのアプレットのメモリ使用量は 586Kbyte であった。

2.6 関連研究

2.6.1 操作履歴の保存とその再利用

ユーザの操作にかかる労力を軽減する目的で、ユーザが繰り返し行う操作を記録して再利用する技術は、古くから様々なアプリケーションやツールキット、Window System や Operating System で用いられてきた。アプリケーションとして操作を登録し再現する機能 (マクロ機能) は、GNU Emacs[14] などのテキストエディタ、Excel[15] などの表計算アプリケーションでは一般的である。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。

MacOS で動作するスクリプト言語 AppleScript[16] は、AppleEvent という機構を利用して、複数のアプリケーション (Open Scripting Architecture に準拠しているもの) を連携して操作することができる。X Window System 上で動作するアプリケーションに Synthesized Interaction を導入した研究として [8] がある。Xlib を拡張することにより X 環境で動作するアプリケーションにほとんど変更を加えずにアニメーション機能を追加できる。

TkReplay[7] は、Tk ツールキットの操作をスクリプトで記録し、再生するシステムである。Jedemo と同様、ヘルプやチュートリアル、マーケティングプレゼンテーションなどに応用が可能である。Jedemo にはない特徴として、テキストファイルでスクリプトを記録するので、テキストエディタを用いればイベントの編集が可能である。ただし、生成されるスクリプトは低レベルなイベントなので効果的に編集するのは普通のエディタでは難しい。また、イベント列から注釈を自動生成する機能は持っていない。

²<http://java.sun.com/>

2.6.2 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)[17]」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[18]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[19]やChimera[20]、Mondrian[21, 22]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[23]は、画面上のピクセルデータを認識することによりマクロを生成する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 [24]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はなく、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[25]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[26]や、その後継であるGarnet[27]は、例示アクションを用いてユーザインタフェースを生成するシステムである。カーネギーメロン大学で開発されているアンドゥ機能を持つツールキットAmulet[28]は、Garnetの後継となるシステムである。

2.6.3 ヘルプシステム

Humanoid Hyper Help[29]は、ハイパーテキストによる状況依存ヘルプを自動生成するシステムである。開発者はリンクやメッセージを付加でき、カスタマイズやメンテナンスも可能である。ユーザはリンクを辿ることで簡単に関連情報を得ることができる。Javaアプリケーションやアプレットのためのヘルプシステムとしては、Sun Microsystemsが提供しているJavaHelp[30]がある。JavaHelpでは、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、状況依存ヘルプ、全文検索機能などの枠組みを提供している。これらのシステムはアニメーション機能は備えていない。

Cartoonist[4]は、UIDE(User Interface Development Environment)で用いられるインタフェース設計仕様(モデル)を利用してアニメーションヘルプを自動生成するシステムである。知識として、3種類のモデル(Application Action, Interface Action, Inter-

action Techniques) を用いることにより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。UIDE を用いて設計することを前提としているので、既に実装したアプリケーションについては UIDE の知識を作成する必要がある。最近ではアプリケーションを実装する際にツールキットや JavaBeans などのコンポーネント技術を用いる傾向にあるため、UIDE やその設計仕様の存在価値は低い。

2.7 結言

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

第 3 章

挿入機能を用いた構造化文書の閲覧法

3.1 緒言

構造化文書の 1 つである HTML(Hyper Text Markup Language) で書かれた文書 (ハイパーテキスト) を閲覧する行為は、計算機を利用した情報獲得の手段として一般に用いられるようになった。最近では、ハイパーテキストによって準備された教材を利用する機会が増加している。また、検索エンジンを利用した調べ学習など、教育分野においても活用されている。そのため、より使いやすい Web ブラウザが求められている。

しかし、これまで利用者が使いやすいように追加・工夫・改良されてきた Web ブラウザの機能は主にブックマークや履歴、オフライン閲覧などの周辺的な機能であった。ハイパーテキストの表示・閲覧機能は Web ブラウザにおける中心的な機能であるにもかかわらず、Nexus[31] や NCSA Mosaic[32] などに代表されるグラフィカルなインタフェースを備えた初期の Web ブラウザ以来ほとんど改善されていない。

3.2 従来のブラウザにおけるリンク機能とその問題点

アンカーをクリックすることでリンク先文書を表示する機能 (リンク機能) は、ハイパーテキストにおける主要な機能であり、ハイパーテキストを閲覧する際に頻繁に用いられる。そのため、現在用いられている主要なブラウザは、リンク機能を実現する操作を複数用意している。しかし、それらは少なからず閲覧者に余計な負担を強いるという問題を含んでいる。本節では、これらの操作とその問題点を挙げる。

現在使われている主なブラウザに実装されているリンクをたどる操作としては、以下の 3 つがある。

操作 1 アンカー上でマウスボタンをクリックする。

操作 2 新しいウィンドウで表示する。アンカー上でメニューを開き，“Open Link in New Window”を選択する。

操作 3 アンカー位置でポインティングし、そのままドラッグして別のウィンドウにドロップする。

「操作 1」は、最も一般的に行われている操作である。この操作でリンク先文書を表示すると、リンク元の文書は画面から失われる。閲覧者がリンク元文書を再度参照するには、通常 Web ブラウザの“Back”ボタンを用いる。“Back”ボタンを押すには文章から目を離さなければならないので、閲覧者の作業が妨げられる。

また「操作 1」でリンク元文書が失われることの弊害として、閲覧者のメンタルマップが破壊されることが挙げられる。これは、閲覧者と操作者が異なる場合に特に顕著に現れる。例えば、計算機リテラシなどの講義でハイパーテキスト形式で記述された教材を教師がプロジェクタで投影しながら説明する場面を考える。学生は教師の説明を聞きながら投影画面を見るだけでなく、自分の計算機を操作するため手元の画面を見る必要がある。学生が投影画面から目を離している間に教師があるリンクを開くと、学生は教師がどのリンクを開いたのか分からなくなる。

「操作 2」は、リンク元の文書を残しつつ、リンク先の文書を表示したい場面で有効である。制作者によって意図的に新しいウィンドウを開くように設計されている文書やリンク集も存在する。しかし、新しいウィンドウが元のウィンドウと重なってしまい、閲覧作業を妨げる場面がしばしば生じる。このような場合閲覧者がウィンドウの移動やリサイズ操作を行う必要があるが、これらの操作は本来の文書を読む作業を中断させるので、効率が悪い。

「操作 3」は、リンク先の文書を表示するウィンドウを明示的に指定できる。既に複数のウィンドウが重ならないように配置されていれば、閲覧者はウィンドウの移動やリサイズ操作を行う必要はない。しかし、複数のリンク先文書を個々のウィンドウに表示した場合、どのリンクをどのウィンドウに表示したかという情報は画面に残らないので、閲覧者が対応関係を記憶しておく必要がある。

3.2.1 ナビゲーション行為

上に挙げた問題を解決し、効果的なハイパーテキストの閲覧を行うためには、ハイパーテキストの閲覧時に発生する「ナビゲーション行為」を考える必要がある。ナビ

ゲーショ ン行為には、

- ナビゲーショ ン操作
- 視点の移動

の 2 つが含まれると考えられる。「ナビゲーショ ン操作」は、ハイパーリンクを「辿る」「戻る」といった操作や、閲覧者が希望するページを適切な状態で開くのに必要な操作である。「視点の移動」は、ナビゲーショ ン操作で開いた複数のリンク先ページ間を渡り歩く場面や、文章とナビゲーショ ン操作の操作対象(「戻る」ボタンや、ウィンドウサイズを変更するための枠、ウィンドウの重なり順や位置を変更するためのタイトルバーなど)との間で発生する。

通常のテキストを閲覧する場合には、閲覧者は「文章を読む行為」を行うだけでよいが、ハイパーテキストを閲覧する場合には、「文章を読む行為」に加えて「ナビゲーショ ン行為」が必要になる。我々は「ナビゲーショ ン行為」にかかる負荷(ナビゲーショ ン負荷)を軽減することによって、閲覧者がハイパーテキストを「読む行為」に集中することができると考えている。

3.2.2 ナビゲーショ ン負荷の軽減

ナビゲーショ ン負荷を軽減し、閲覧者がハイパーテキストに集中するために、我々は以下のアプローチをとる。

- リンクを「辿る」「戻る」機能を、1 つの操作オブジェクトに持たせる
- リンク先の文書をアンカーの近傍に表示する

1 つの操作オブジェクトに対してリンクを「辿る」「戻る」機能を付加することで、3.2.1 で述べたナビゲーショ ン操作の負荷を抑える効果がある。また、リンク先の文書をアンカーの近傍に表示することにより文書間を渡り歩くときの視点の移動量を抑える効果が期待できる。

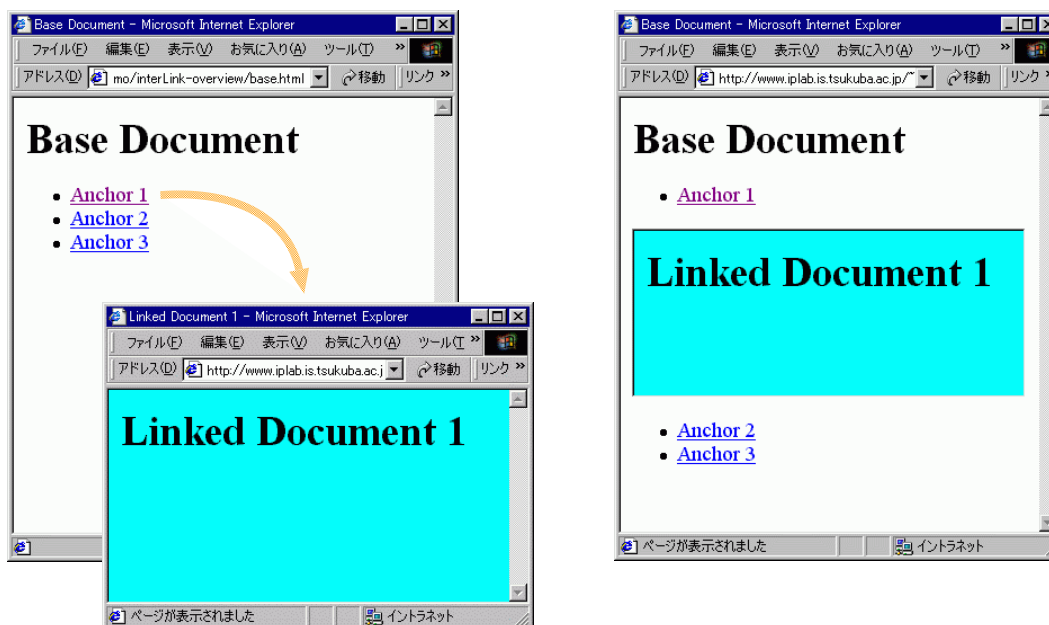


図 3.1: inlineLink の内挿：(左) 既存のブラウザによる手法 (右) inlineLink 手法

3.3 “inlineLink” の概念

我々は、ハイパーテキスト閲覧時に特徴的なナビゲーション負荷を軽減するためのリンク手法 “inlineLink” を開発した [33, 34, 35].

“inlineLink” では、

- リンク先文書を内挿・展開して表示する
- 内挿・展開を調整する機能をアンカーとして提供する

という 2 つの機能によって効果的な閲覧を可能にしている.

3.3.1 文書内挿

inlineLink は、閲覧者がアンカーをクリックした際に通常 1 つのウィンドウ全体に表示されるリンク先文書をリンク元文書の内部、アンカー位置の後に埋め込んで表示する. 我々はこれを「文書内挿機能」と呼ぶ. 図 3.1 は、“Base Document” と “Anchor 1” によってリンクされる “Linked Document 1” を同時に表示する場合において、従来のリンク方式と inlineLink によるリンク方式を比較した模式図である. 図 3.1 (左) は、従来のリンク方式「操作 2」を用いて別ウィンドウで表示した “Linked Document

1” が元のウィンドウの一部を隠してしまう例を表している。 **inlineLink** の文書内挿機能によって “Linked Document 1” を表示すると、図 3.1 (右) のように “Base Document” 中の “Anchor 1” の下にリンク先文書が内挿・展開される。

文書内挿による利点を述べる。アンカー位置にリンク先文書を埋め込むため閲覧者がリンク先文書にアクセスするための視点移動量は少なくて済む。また、アンカー以前にあったリンク元文書はそのまま残っているため、文脈を保存したまま閲覧が続けられる。

3.3.2 機能アンカー

inlineLink における機能アンカーとは、内挿・展開した文書を表示調整する機能を閲覧者に提供するための仕組みである。機能アンカーのうち、主に用いられるものは、リンク先文書を開くためのアンカー (open anchor) と、開いた文書を閉じるためのアンカー (close anchor) の 2 種類である。閲覧者が open anchor を選択すると、トグルスイッチのようにその open anchor は close anchor に変化する。この close anchor を始点 close anchor と呼ぶ。 **inlineLink** は展開したリンク先文書の終点にも、close anchor を配置する。これを終点 close anchor と呼ぶ。図 3.2 に、open anchor と close anchor (始点、終点) を示す。

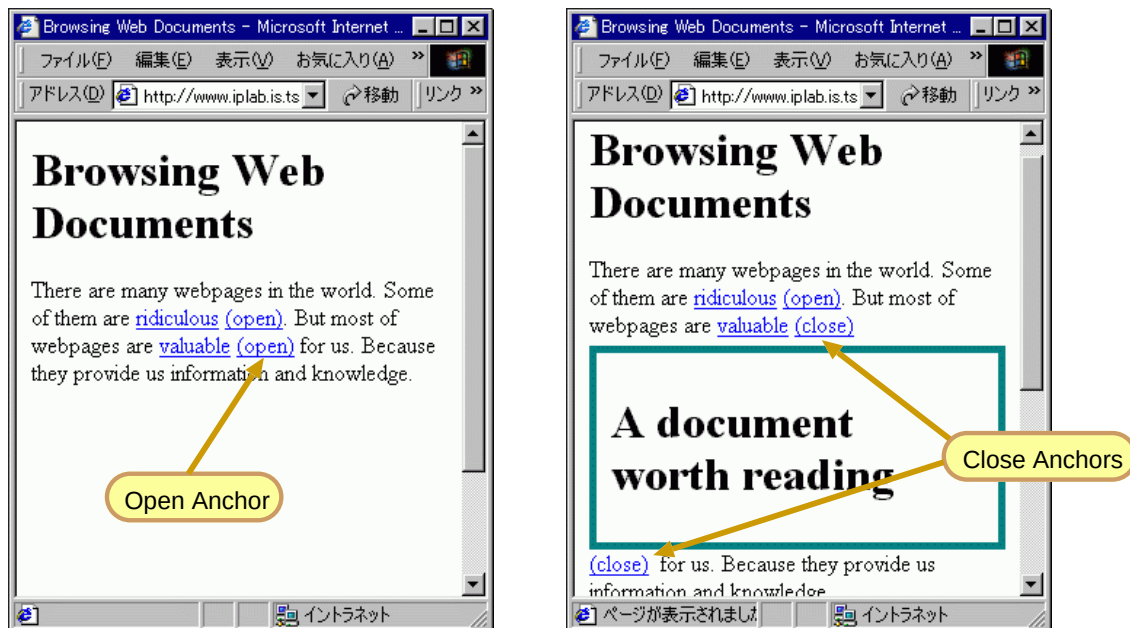


図 3.2: **inlineLink** の機能アンカー



図 3.3: 閲覧者の典型的な行動

3.3.3 閲覧者の行動

inlineLink を用いた場合の文書閲覧例を示す。閲覧者は、興味のあるリンクがあった場合、通常のリンク先文書を開く操作と同じように **open anchor** をクリックする(図 3.3 [a])。すると、**inlineLink** によって、**open anchor** は始点 **close anchor** に書き換えられ、リンク先文書と終点 **close anchor** が内挿・展開される(図 3.3 [b])。閲覧者は、リンク先文書を眺めて、その文書を読むかどうか判断できる。もし、その文書を読むのをやめて、元の文書のみを表示した画面にしたい際には始点 **close anchor** をクリックする(図 3.3 [c])。始点 **close anchor** は“Back”ボタン押下に相当する操作を閲覧者が全くポインタを動かさずに実行できるという点で効果的である。

もし、リンク先文書を読む場合(図 3.3 [d])は、展開した文書をそのまま読み進める(図 3.3 [e])。必要に応じてスクロールする(図 3.3 [f])。リンク先文書を読み終えると、その下には元の文書が続いているのでそのまま読み進めることができる。また、リンク先文書を閉じるときには終点 **close anchor** をクリックする(図 3.3 [g])。

閲覧者が **inlineLink** を用いて連続的かつ再帰的に内挿を繰り返した例を図 3.4に示

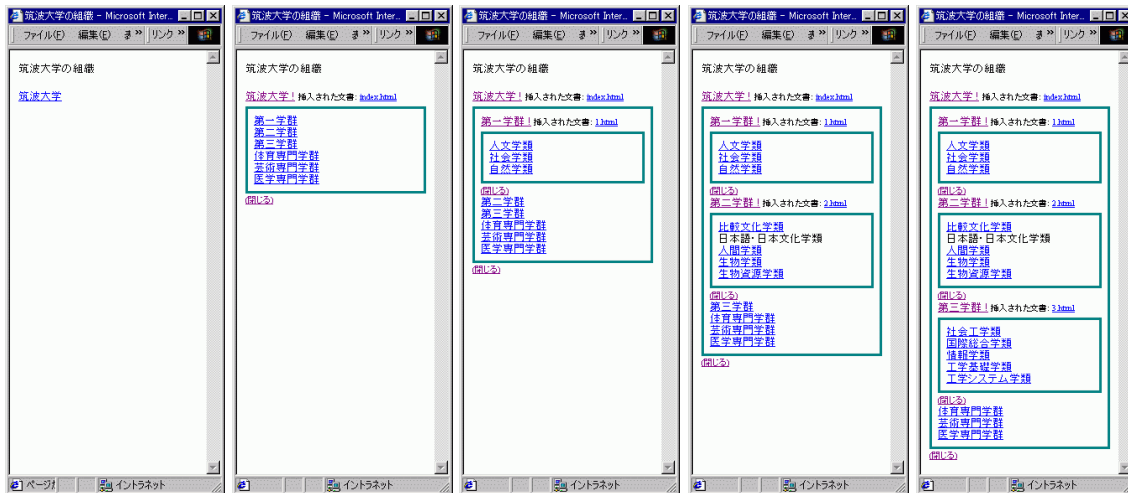


図 3.4: 階層的な内挿

す。図 3.2 では、従来のアンカーとは別に“(open)”という機能アンカーを付加しているのに対し、図 3.4 では機能アンカーのみを提供している。“(open)”などの余分な文字列を表示しないことにより文章自体の読みやすさを重視している。なお、この例では従来のアンカーは展開後に表示される「挿入された文書：」という文字列の後に付加している。

3.3.4 内挿表示方式: 全文内挿と部分内挿

inlineLink は、リンク先文書を内挿する方法として 2 種類の表示形態を提供する。ここでは、それぞれの表示形態における長所と短所について述べ、さらに閲覧者がこれらの方式をどのように切り替えて利用すべきか述べる。

全文内挿 (Full insertion) リンク先文書全体をアンカー位置に挿入する方法である(図 3.5)。リンク先文書が長い場合、アンカー位置より後方にある文書が移動するため画面から失われるという欠点があるが、通常のスクロール操作のみで閲覧作業を継続できるという利点がある。

部分内挿 (Partial insertion) リンク先文書の一部だけを表示する方式である。アンカー位置に、文書を表示する部品 (Document Pane) を挿入し、その中にリンク先文書を表示する(図 3.6)。Document Pane の表示サイズを適切に設定することにより、リンク先文書を普通に表示すると多くの画面領域を占有する場合でも、アンカーより後方にある文書を画面に残すことができる。また、Document Pane の内部スクロールバーを用いて表示する箇所を選択できる。

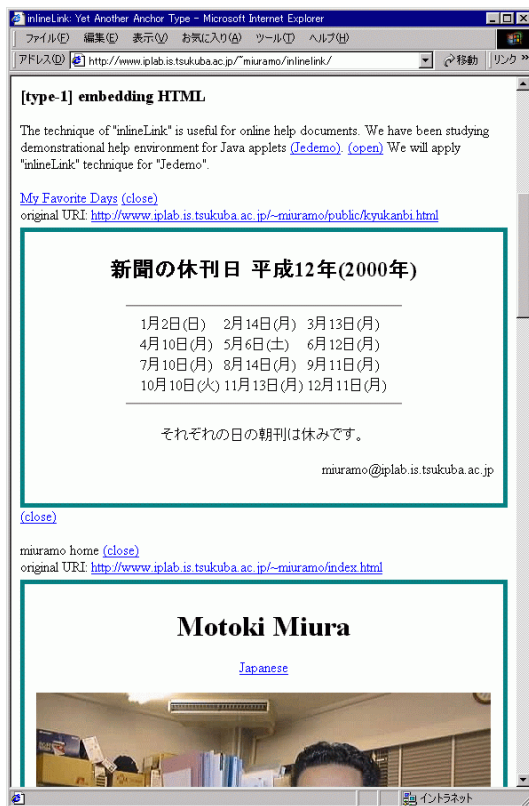


図 3.5: 全文内挿 (Full insertion)



図 3.6: 部分内挿 (Partial insertion)

閲覧者は、通常は「全文内挿」を用いることにより通常のスクロール操作のみで閲覧作業が続けられる。挿入された文書が長い場合や大きな画像を含むような場合、また同時に参照したい限られた個所のみを表示したい場合には、「部分内挿」を用いてリンク先文書の一部分を参照すればよい。

3.4 既存のブラウザを用いた inlineLink の実現

一般に使われている Web ブラウザを改変することなく **inlineLink** 機能を実現する方法について述べる。

3.4.1 方針

我々は、Web ブラウザ上で **inlineLink** を実現するために、Dynamic HTML もしくは DHTML と呼ばれている動的なページ変更技術を利用する。Dynamic HTML とは、HTML で記述された文書要素をオブジェクトとみなし、言語によってオブジェ

クトの表示・加工など操作を行う仕組みである。HTML 文書は、World Wide Web Consortium (W3C)[36] で開発された Document Object Model (DOM)[37] によって階層構造を持つオブジェクトとして扱うことができる。Dynamic HTML を用いた理由として、一般に広く利用されている Web ブラウザ上で機能し、動作することが挙げられる。このことは、**inlineLink** の機能をブラウザを拡張して実装する場合に比べ、閲覧者の環境やブラウザを制限しないという点で優れている。Dynamic HTML で HTML オブジェクトを操作する言語には、VBScript や JavaScript[38] などがある。我々は、プラットフォームに依存しないという点で JavaScript を採用した。

3.4.2 挿入するための仕組み

inlineLink におけるリンク先文書や機能アンカーの内挿を実現するには、open anchor がクリックされたときに

- (1) リンク先文書データを取得する
- (2) 画面に表示された文書を書き替える

という 2 つの処理を行うことで実現できる。close anchor がクリックされたときは文書を取得する必要がないため (2) の処理のみを行う。

リンク先文書データの取得

リンク先文書を取得するには、Inline Frame を用いる。Inline Frame は、文書中にオブジェクトとして内挿可能なフレームで、HTML4.0 Transitional DTD[39] で iframe 要素として定義されている。文書中で、

```
<iframe src=URI></iframe>
```

と記述すると、任意の URI (Uniform Resource Identifier) で指定されたページを Inline Frame が読み込み、フレームとして画面に表示する。また、読み込んだデータを加工したり、必要に応じて Inline Frame のを画面に表示しないことも可能である。「部分内挿」では、Inline Frame を直接画面に表示することで実現している。「全文内挿」は、非表示にした Inline Frame に一旦文書データを読み込んでおき、それを改めて画面に流し込むことで実現している。

画面に表示された文書の手換え

機能アンカーをクリックした際に、画面に表示された文書データや、Inline Frame に読み込んだ文書データを書き替えるのには、JavaScript を用いている。ここでは、**図 3.2**のように通常のアンカーとは別に open anchor を追加する例について実際の文書データ書き換えがどのように行われるかを説明する。まず、ハイパーテキストに含まれる通常のアンカーを表すタグ (a 要素) の後に、インライン要素である span 要素を挿入する。この span 要素には、後に JavaScript の関数が参照し書き替えることができるように一意な ID を id 属性によって設定しておく。

最初にページが画面に表示される時点では、この span 要素の内容 (タグで囲まれた部分) に open anchor を定義しておく (**図 3.7**[open anchor] を参照)。閲覧者が open anchor をクリックすると、JavaScript の関数 insert_page () が実行される。insert_page () は、閲覧者が**図 3.3.4**節で示した「部分内挿」を指定している場合、この span 要素の内容を**図 3.7** の [close anchors and inlined document] のように書き替える。すると、Inline Frame が画面に挿入され、src 属性で定義されている URI のページが表示される。もし、閲覧者が「全文内挿」を選択している場合、既に非表示としてある Inline Frame にリンク先の文書データを読み込み、読み込みが完了した時点でそのデータと close anchor を span 要素内に挿入する。閲覧者が close anchor をクリックすると、remove_page () 関数が [close anchors and inlined document] から [open anchor] に書き替えることで挿入されていた文書を閉じる。

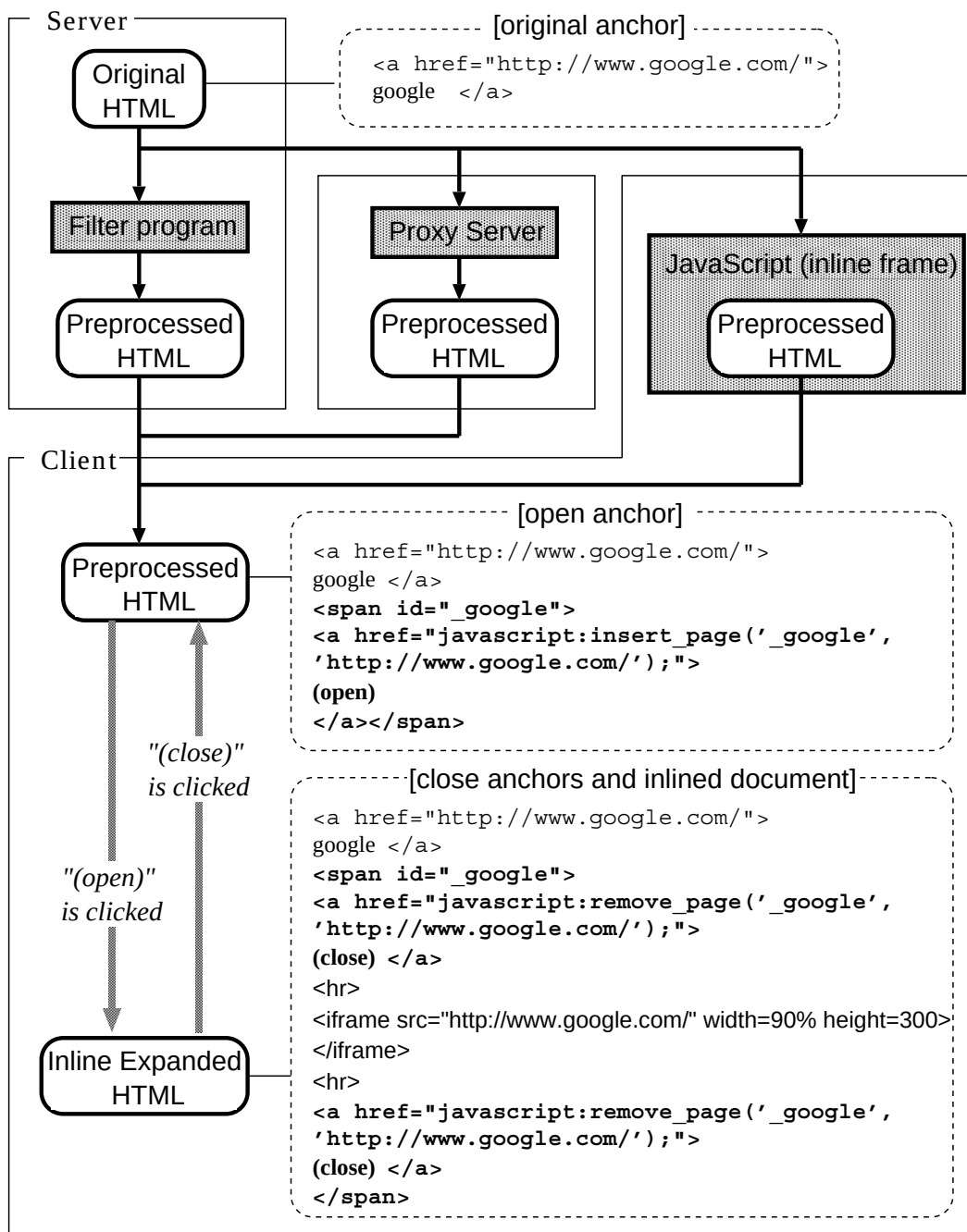


図 3.7: アンカーが書き替えられる場所、タイミングとそれぞれの内容

3.4.3 方法

閲覧者は、閲覧するハイパーテキスト (図 3.7 における “Original HTML”) が、以下の 2 つの要件

1. **inlineLink** が使用する `insert_page()`, `remove_page()` などの JavaScript 関数定義をロードしていること
2. 対象とするアンカーが JavaScript 関数を実行する `open anchor` (図 3.7 の [open anchor]) に書き替えられていること

を満たしていれば, **inlineLink** を利用できる. そのための方法として, 我々は以下の 3 つの方法を用意した.

コンテンツ制作者があらかじめサーバにあるページを書き替えておく方式 Perl で実装したフィルタプログラムを用いて, 文書データを書き替えておくことによりそのページのリンクを辿る場合のみ **inlineLink** 方式でリンク先文書が埋め込まれる.

プロキシサーバを用いて書き替える方式 コンテンツ制作者が対応していないページについて **inlineLink** を適用したい場合がある. その場合は, 閲覧者が明示的にプロキシサーバを指定することで文書データを書き替えることができる.

JavaScript で書き替える方式 上の 2 つの方法はそれぞれコンテンツ制作者, 閲覧者に負担を強いることがあり, あまり適切ではない. この方式では, [Style-1]において一旦 Inline Frame に読み込んだ文書データを **inlineLink** の JavaScript 関数で書き替える方法である. この方式を利用する場合, コンテンツ制作者はトップページだけを **inlineLink** 対応にしておく. そのページからリンクしている文書が対応していなくても埋め込む前に自動的に書き替えるので, コンテンツ制作者はすべてのページを変換しておく必要はない. そのため latex2html で自動生成されたページなど, 階層が深い場合についても簡単に対応できる. ちなみに, 図 3.4 の例では, 通常のインデックスページ (`index.html`) へのリンクを持つトップページ (`top.html`) だけを準備し, 本方式を利用して変換した.

3.4.4 閲覧者が可能な操作

inlineLink の初期状態では「全文内挿」モードが選択されているため, `open anchor` をクリックした場合全体内挿が行われる. 閲覧者が画面をダブルクリックすると「全文内挿」から「部分内挿」へとモードが切り替わり, 今後展開するリンクの表示方式を替えることができる. 図 3.6 に表されるように「部分内挿」モードでは, “(resize)” をドラッグすることで挿入している Inline Frame のサイズを調節できる. 同様に,

“(40)”や“(-40)”という機能アンカーをクリックすることで調節することができる。挿入された文書の URI は挿入されている文書の上部にアンカーとして表示しており、これを選択すると通常通りウィンドウ全体にリンク先文書を表示する。閲覧者はもう一度画面をダブルクリックすることで「全文内挿」モードに戻る。

3.5 議論

3.5.1 inlineLink の特徴

inlineLink の特徴として、下方向にスクロールするだけで元の文書の続きを読む作業を継続できるので、“Back”ボタンを用いることなくリンク元文書をシームレスに参照できる。Tauscher らによる Web 閲覧者の行動調査 [40] では、Web ページを開く行為のうちの約 30% 以上が“Back”ボタンによる再参照であり、これはリンクを辿る行為(約 70%)の次に頻繁に行われる行為であると述べている。よって、**inlineLink** 手法を用いることによって今まで頻繁に用いられていた“Back”ボタンを押す操作が減り、閲覧作業に集中できると言える。

スクロール操作は、閲覧者に負担を強いる操作であるが、通常の文書閲覧時に行われている操作であり、近年普及しているホイール付きマウスを用いることにより、ポインタや視点を移動する必要がないため文章を読む作業への影響は軽減される。また、アンカーを含むリンク元文書が画面に残るので、文脈(コンテキスト)が保存される。さらに、新しいウィンドウを開くことによる余計な操作が発生しないので、文書に集中できる。**inlineLink** ではアンカーとリンク先文書の関係が明示されているので、閲覧者は容易に対応関係を把握できる。新しいウィンドウを用いて文書を表示した場合、閲覧者の視点はウィンドウ間を移動するが、**inlineLink** ではリンク先の文書をアンカーの近くに表示するので、文書を閲覧するのに必要な視点移動距離は少なくなる。この特徴は、これらの文書間を交互に参照する場面で特に有効である。

また、付加的な利点としては、閲覧者にリンク元文書を常に意識させることにより知らないうちに本来の作業と関係ないページを閲覧してしまうことを防止する効果が挙げられる。また、展開すると 1 つのページに記述した状態が得られるため、関連リンク先の文書の印刷やキーワードによる検索を一括して行える。従来では個別に印刷していたページも、**inlineLink** を用いることでアンカーとリンク先の文書との対応を保存して印刷できる。

我々が提案する **inlineLink** は、閲覧者が興味のある部分だけを詳細に表示するイン

タフェースを備えるという意味では、ハイパーテキスト閲覧に Semantic Zooming 機能 [41, 42] を提供していると考えることができる。

3.5.2 inlineLink の問題点

リンク先文書を大量に埋め込んでいくと、ウィンドウ内の文書全体の量が多くなり、かさばるという問題がある。このような場合では、リンクインデックスを用いるなどスクロール以外の文書移動手法が必要だと考えている。また、リンク先文書に含まれるリンクをたどって多段階の入れ子状にした場合、階層を深くしすぎるとかえって使いづらい。ただし、「全文内挿」の場合、挿入する文書まわりの装飾の種類や色、インデント量などは CSS (Cascading Style Sheet) で定義しており簡単に変更できる。また、画面の制約以外の理由による入れ子数に制限はない。CSS によって変更した内挿文書の装飾を図 3.8 に示す。

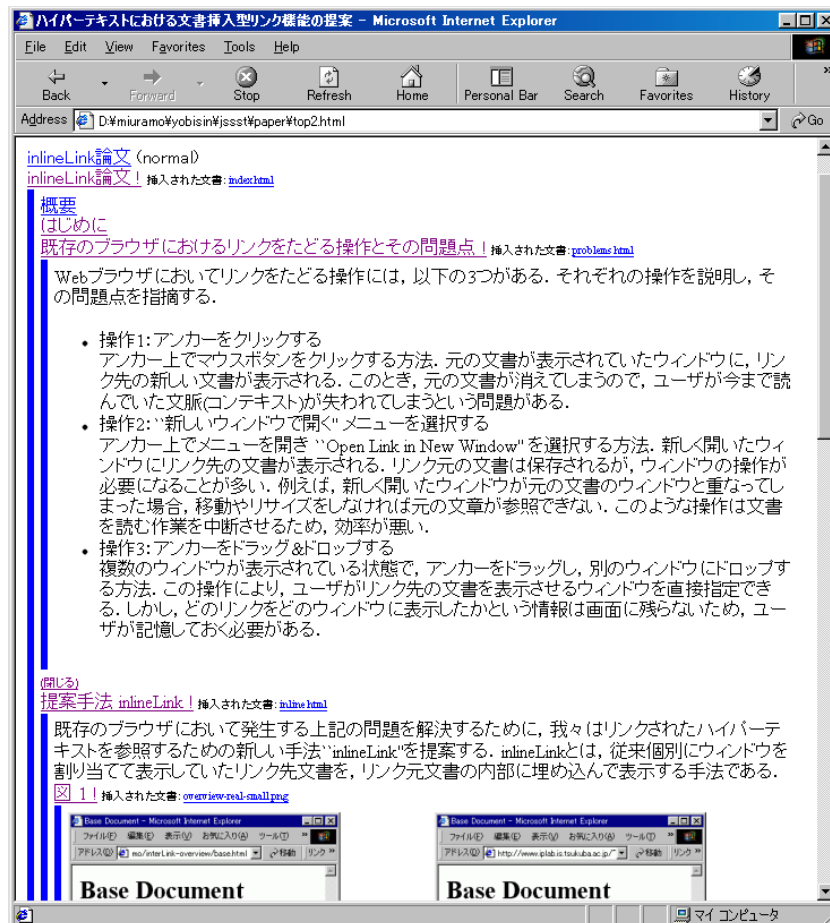


図 3.8: CSS によって変更した内挿文書の装飾

ハイパーテキストの場合、アンカーが文章中に出現する場合も多い。そのような場合にリンク先文書を埋め込むと文章が途切れてしまい、1つの文として理解しづらくなる。対策としては、文書を内挿する位置を文と文の間、もしくは段落と段落の間にする可以考虑が、アンカーと文書の位置が離れてしまう可能性がある。

初期の **inlineLink** では、終点 close anchor を用いて高さの長い内挿文書を閉じた場合、文書の長さが縮小するため閲覧者の注目したい位置が保存されず、上方向に移動してしまう問題があった。現在の **inlineLink** では、終点 close anchor をクリックした際、閉じる内挿文書の高さ分を下方向にスクロールすることにより文書削除後のユーザのポインティング位置を維持するよう工夫している。しかし、ページ上部にスクロールする余地がない場合には対応できない。対策として、ページ上部に余白を挿入することや、アニメーションを用いて段階的に内挿文書を縮小することを考えている。

3.5.3 関連する Web ページ

inlineLink に似た挿入インタフェースを備える Web ページは Microsoft Windows Update[43] を含め既に複数存在している [44, 45]。これらのページも Dynamic HTML 技術を用いて実現されているが、ほとんどのページにおいて挿入する内容は同一ページに書かれており、動的にリンク先を読み込んで表示する方法は取られていない。ただ、[44] は文章を挿入するよりも見出しの階層構造のみを内挿機能で提供し、実際の文章は別ウィンドウを用いて表示されている点で、内挿の利点と欠点を上手に使い分けているといえる。

3.6 実験

inlineLink によって、閲覧者のナビゲーション負荷をどの程度軽減することができるかを測るために、実験を行った。

3.6.1 実験の目的、概要と設定

実験の目的、概要と設定について述べる。

実験の目的 構造化文書を閲覧する際に、**inlineLink** を使ったらどの程度効率がよくなるかを調べる。

実験の概要 Emacs の dired (Directory Editor) モードのマニュアル (HTML) 版を閲覧させる。閲覧しながら、順次質問に答えるというタスクを与えた。マウスのクリック回数、マウスの移動量と、タスク遂行にかかった時間を計測した。

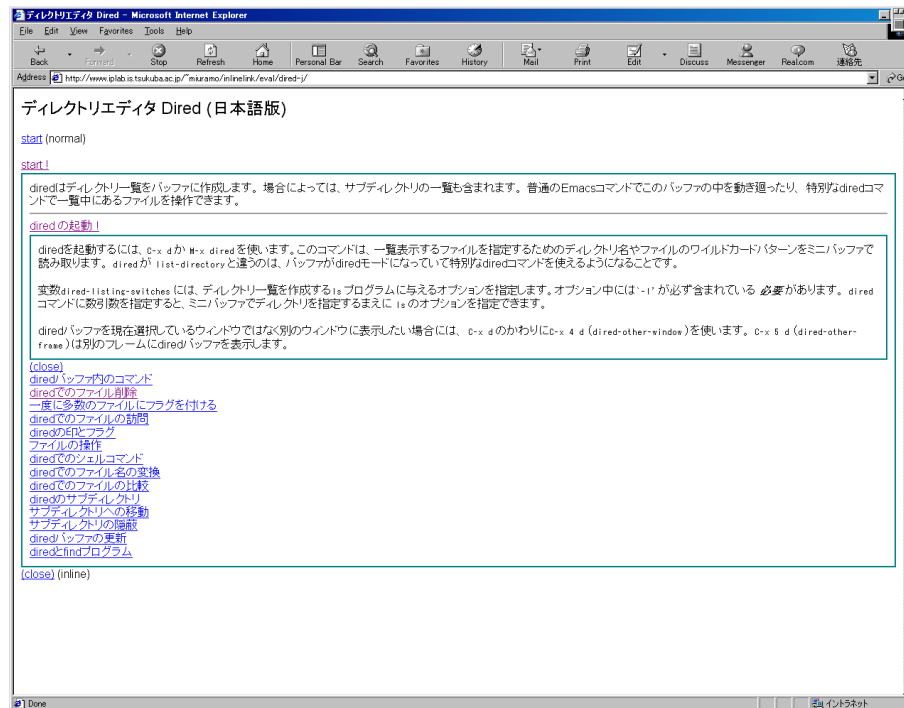


図 3.9: inlineLink 実験に用いたブラウザと文書

実験の設定 Windows 98 Second Edition を PC (Pentium III 500MHz, 256MB memory) にインストールしたマシンを用いて実験を行った。日本語 106 キーボードとホイール付きマウス (Logitech M-S48) を接続した。

閲覧用のブラウザとして Microsoft Internet Explorer (version 6.00.2462.0000) を用いた。ブラウザの“Back”ボタンはアイコンと文字列両方を表示しておく。また、URL 表示部とステータスバーも表示しておく。フォントの大きさは「中 (Medium)」に設定して実験を行った。ちなみに、この状態で 1 画面に納まる (スクロールバーが現れない) 状態での最大行数はおよそ 44 行である。ディスプレイには、解像度 1280 × 1024 で設定した 15.4 インチの TFT 液晶ディスプレイを用いた。ブラウザは最大化で表示した状態で実験を行った (全画面表示ではないので、タイトルバーやウィンドウは表示されている)。図 3.9 に、実験に用いたブラウザの実際の画面を示す。

実験に用いた文書は、Emacs version 20.6 のマニュアルのうち、dired モードに関する記述部分だけを選択して実験用文書とした。この文書は、texi2html version 1.52

¹ を用いて Texinfo ファイルから HTML ファイルを生成した。生成するときに各ノード毎にページを作成するオプション (-split_node) を追加した。実験用文書の元となる Texinfo ファイルは、英語版²とその日本語翻訳版³ から取得した。

dired モードに関する記述部分は、15 セクション (15 ページ) と 1 つのインデックスページの合計 16 個のファイルで構成されている。それぞれのページは文字とアンカーのみで構成されており、画像は含まない。参考のため、**付録 A.1** に実験に用いた文書の情報を示す。

実験の詳細 実験には外国人留学生 3 名を含む工学系の学部生および大学院生 15 名が参加した。実験は一人の被験者につき約 30 分にわたって行われた。

我々は、実験用文書 (15 ページ) に記載されている dired モード特有のコマンドから 15 個を抽出し問題を作成した。これらのコマンドは 10 ページにわたって記載されていた。15 個の問題を実験用文書の出現順に並べ替え、前後半 2 つに分けた。前半の問題は 8 つ、後半の問題は 7 つである。問題は、コマンドの部分を空欄にし、そのコマンドの効果を表す文を実験用文書から複写して A4 の紙に前後半別に印刷した。前半の問題を「問題 1」、後半の問題を「問題 2」とした (実際に被験者に与えた問題を **付録 A.3, A.4** に示した)。

我々は被験者を (A),(B) の 2 つのグループに分け、**表 3.1** に示す順序 (1)–(4) にて実験を行った。最初に [normal] を用いるグループと [inlineLink] を用いるグループに分けることによって、一方が有利となる効果は相殺される。全ての実験で、被験者は空欄に合てはまるコマンドを効果を表す文から推測し実験用文書内から探し出す。[normal] では、被験者は通常の HTML を用い、[inlineLink] では、**inlineLink** 手法を用いて閲覧した。実験 (1)(2) では、被験者はコマンドのみを実験遂行者に口頭で伝える。実験 (3)(4) では、被験者はコマンドを問題用紙の空欄に記述する。これらの実験においてそれぞれの手順を完了するまでのマウスのクリック数と移動量、経過した秒数を記録した。

各被験者に対して実験前に **inlineLink** の使い方について 1 分程度確認・練習させた。実験ではスクロール操作にホイールを利用するよう口頭で説明した。

実験	タスク	被験者 (A)	被験者 (B)	セッション
(1)	問題 1 口述	normal	inlineLink	1st
(2)	問題 2 口述	inlineLink	normal	1st
(3)	問題 1 記述	inlineLink	normal	2nd
(4)	問題 2 記述	normal	inlineLink	2nd

表 3.1: inlineLink 実験の流れ

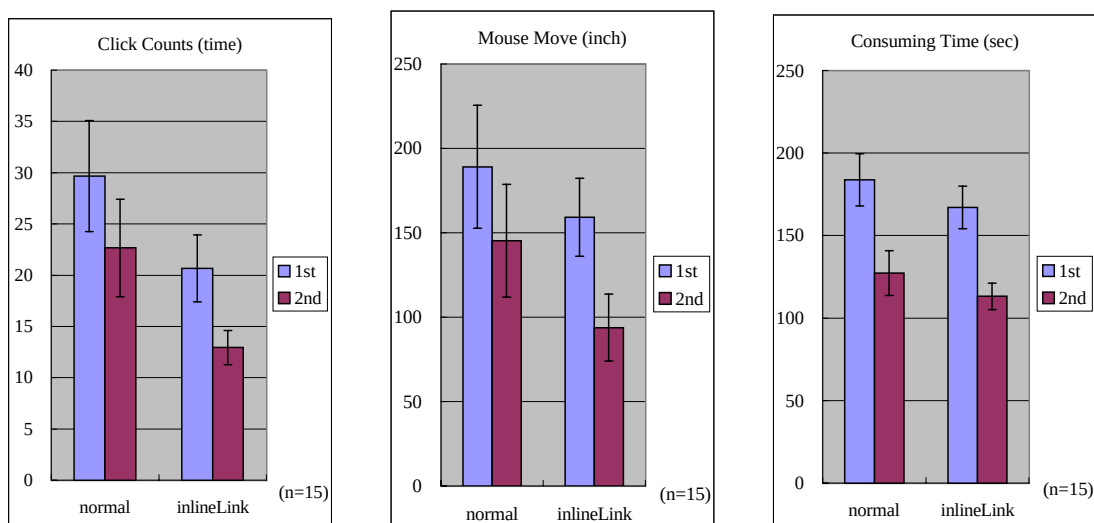


図 3.10: 実験結果 (クリック数, 移動量, 作業時間の平均) と標準誤差

3.6.2 実験結果

実験結果 (クリック数, 移動量, 作業時間の平均を, セッション毎に表示したグラフ) を図 3.10, 表 3.2 に示す. また, 実験 (3)(4) の結果を図 3.12 に示す. 図 3.10 に示した範囲は標準誤差を表す.

3.6.3 考察

被験者のほとんどは実験用文書に不慣れだったため, 実験 (1)(2) においては, ページ内を彷徨うことによる時間と操作のノイズが入っている. 実験 (3)(4) においては同一の問題を 2 回行うことによる学習効果 (図 3.11) が顕著に見られる. しかし 2 回目の実

¹<http://wwwinfo.cern.ch/dis/texti2html/>

²GNU Emacs Manual - Table of Contents (<http://www.gnu.org/manual/emacs/index.html>)

³GNUjdoc (<http://openlab.ring.gr.jp/gnujdoc/>)

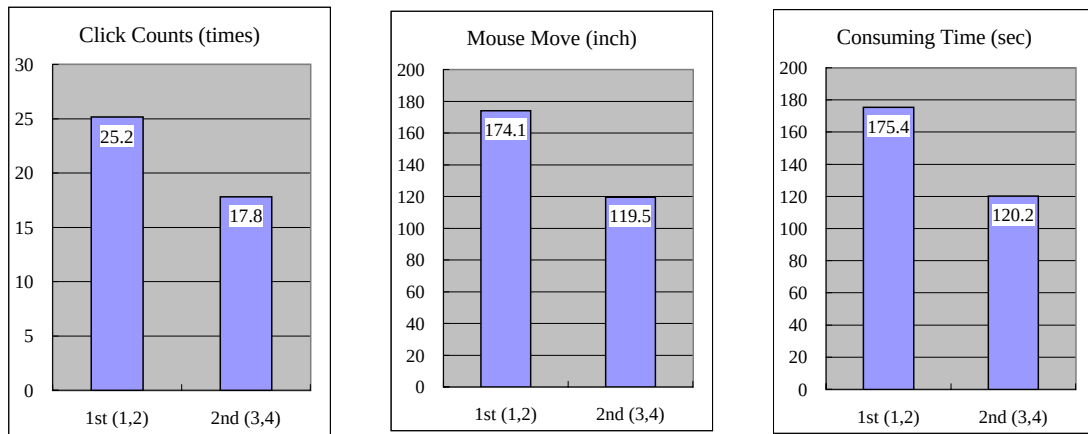


図 3.11: 実験 (1)(2) と (3)(4) の比較：クリック数，移動量，作業時間の平均)

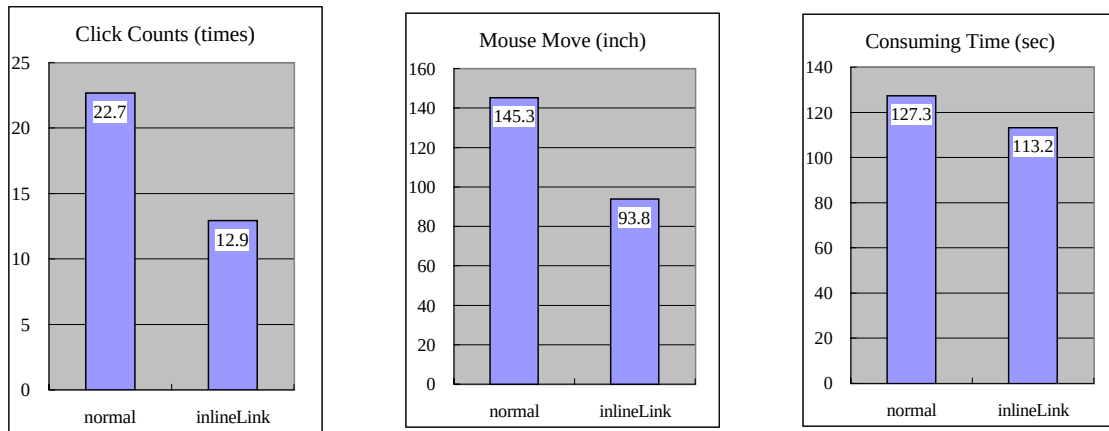


図 3.12: 実験 (3)(4) の結果 (クリック数，移動量，作業時間の平均)

験 (3),(4) においても，記述された場所を完全に覚えていない被験者による操作ミスが若干確認された。

作業時間の平均については [normal] と [inlineLink] ではそれほど差は見られないが，マウスのクリック数と移動量の平均は [inlineLink] によって減少していることが図 3.10 より読みとれる。

inlineLink の効果を統計的に示す．対応がある 2 条件の平均値に有意な差が存在するかどうかを判別する t 検定を有意水準 5% にて行った．その結果を付録 A.2 に示す．t 分布表における自由度 29，片側確率 5% の臨界値 (1.70) で比較した結果，作業時間 ($t = 1.04 < 1.70$) においては **inlineLink** の優位性を示すことはできなかったが，マウスのクリック数 ($t = 2.78 > 1.70$) と移動量 ($t = 2.24 > 1.70$) について帰無仮説「2 群

実験	タスク	normal			inlineLink		
		click	move (inch)	time (sec)	click	move (inch)	time (sec)
(1)	問題 1 口述	30.3	192.7	184.1	13.6	123.3	156.4
(2)	問題 2 口述	29.1	185.9	183.4	28.7	200.3	179.3
(3)	問題 1 記述	16.5	87.8	105.9	15.9	121.6	121.7
(4)	問題 2 記述	29.7	211.0	151.7	10.4	69.5	105.8

表 3.2: 実験結果 (クリック数, 移動量, 作業時間の平均)

の母平均値に差がない」を棄却した。よって信頼係数 95% でマウスのクリック数と移動量について **inlineLink** の有効性が示された。

マウスのクリック回数は“Back” ボタンを押さずに元の文書が閲覧できる **inlineLink** の利点が影響していると考えられる。マウスの移動量も, “Back” ボタン操作の軽減による影響が大きい。ただ, マウスの移動量の平均差はクリック数ほど顕著に現れなかった。その原因として, **inlineLink** で要求されるスクロール操作にホイールを使用せず, スクロールバーを直接操作した被験者が見られたことが挙げられる。キーボードによるショートカットを使用する被験者はいなかったが, **inlineLink** 利用中に “Back” ボタンを押してしまう被験者も若干見られた。また, [normal] において “Back” ボタンを使用せずマウスの右ボタンを用いてコンテキストメニューを表示し, “Back” メニューアイテムを選択する被験者も見られた (マウスのクリック数に右クリック数は含まれていない)。実験後メニューを用いて “Back” を選択した被験者に理由を聞くと, マウスを “Back” ボタンまで持っていく作業を避けるためにメニューによる操作を普段から使用しているという答えが帰ってきた。また, ハイパーテキスト閲覧作業中にマウスでなぞり読みする, しないといった個人差や癖による影響も大きいと考えられる。

作業時間において優位性が示せなかった原因としては, 被験者に与えられたタスクが複数のページを同時に見ることを要求するものではなかったことが挙げられる。特に, どのページに記述されているか分からず探したり迷ったりすることによるデータのばらつきや, 口述や記述にかかる時間が無視できない。また記述作業はマウスを一旦離して筆記する必要があるため純粋なブラウジング時間を得ることができなかったと思われる。

上記の結果から, 閲覧者が **inlineLink** の特性について理解しており, かつ文書の階層構造が直感的または既知である場合には, **inlineLink** は効果的に作用すると考えら

れる。

3.6.4 被験者からのコメント

実験後、被験者から記述または口述で寄せられたコメントを紹介する。(いくつかは 3.5.1 で述べた特徴と重複している)

- 一回毎に戻らなくてよいし、近くにあるリンク先は同時に見られるのが便利
- マウス一本で操作できるのがよい。
- 文書を挿入しすぎて長くなってしまうとスクロールが面倒。
- inlineLink での “Back”, “Forward” の機能がないのは不満。
- Close anchor までマウスを持っていくのが負担なので、どこでクリックしても文書を閉じれるようにしてほしい。
- ダブルクリック操作を割り合っていないことは評価できる。
- inlineLink に慣れるまで多少だが時間がかかる。
- 思わず慣れている “Back” ボタンを押してしまう。タスクで交互にやらせるのは混乱してよくない。
- inlineLink でも毎回開く / 閉じるをやっていたらクリック数は変わらないのでは。
- 中身の展開状態が保存されないのが困る。
- 挿入、削除をアニメーションでなめらかにしてほしい。
- 通常、マニュアルから必要な部分を探す場合、ページ内検索機能を使っているため、タスクが苦痛だった。階層文書を全部展開する機能があると、一括して検索するのに便利である。
- 通常、タブで切り替えることができるブラウザを使っている。
- 通常、“Back” ボタンを押す代わりに、“BackSpace” キーや “ALT+Left” などのショートカットキー、マウスの横に付いているボタンを使っている。

- 挿入される文書が長い場合、一度に全部挿入してしまうのではなく、先頭の数行だけを表示するようにして、さらに読み進めたい場合に全部挿入できるようにしたかどうか。
- 実験のタスク内容が、2つのページを同時に開くことを要求するものではないので、inlineLink のメリットをあまり引き出せていないのではないか。
- Directory Editor マニュアルの構成が悪く、見出しだけで内容を推測できないものが多く、無駄なリンクを辿ってしまった。

数名の被験者は普段からタブブラウザや特殊なボタンを持つマウスを使用したり、ショートカットを使って inlineLink が対象とする “Back” ボタンを押す問題を回避していた。これらの被験者に共通していたのは、比較的 Web 文書閲覧を行う時間が多いということだった。このことから普段から特に Web 文書閲覧を行うユーザは、潜在的に文書閲覧に対する不満を持っていたといえる。よって必要に応じて inlineLink を利用することができる環境を提供し、選択子を増やすことには意味があると考ええる。

全体一括挿入機能は、ぜひ今後検討したい機能である。段階的な挿入や、アニメーションにも興味があるが、これらの挿入方法に対する好みには個人差があると思われる。また、アニメーションに必要な時間も適切に調節する必要があると考えている。

3.7 関連研究

ハイパーテキストを閲覧する際に発生するナビゲーション負荷を軽減する研究は、大きく2つに分類できる。1つはアンカーに関連付けられたリンク先ページに関する情報を少ないコストで参照できるようにすることにより閲覧者がそのリンクを辿るべきか否かを判断する材料を事前に提供する研究である。もう1つはハイパーテキストを閲覧する際の操作を簡略化する機構を提供する研究である。

3.7.1 リンク先情報の提示

VisualLinkPreview[46] は、リンク先ページの情報を先読みし、縮小画像 (サムネイル) を生成して表示する。HyperScout Linktool[47] は、リンク先の情報をテキストとアイコンによる表現を用いてユーザに提示する。提示される情報としては、タイトル、著者、言語、前回の訪問日時、サーバの位置、サーバの反応時間、リンク先文書が存在するか否か、などである。どちらもアンカーにマウスポインタを合わせたときにポッ

プアップウィンドウを用いて元のアンカーの近くに表示するため、閲覧者の視点の移動量を減らすよう考慮されている。これらの情報は閲覧者がリンクを辿る行動を取る際の判断材料としては有効であるが、リンク先の文書の内容を確実に理解するためにはそれぞれリンクを辿る必要がある。**inlineLink** では、複数のリンク先文書を同時に可読な状態で提示することを可能にしている。

Fluid Link[48, 49] では、注釈などの付加情報 (gloss と呼ばれる) を、元の文章のアンカーの近くに配置するためのアニメーションや半透明表示などを用いた高度な表現方法をいくつか提案している。Fluid Link は専用のドキュメントブラウザで実現されているため、**inlineLink** よりも表現能力は高い。**inlineLink** は可搬性を重視し、Fluid Link に似た文章閲覧インタフェースを従来の Web ブラウザ上で実現した。

3.7.2 閲覧動作の簡略化

Elastic Windows[50, 51] は、指定した複数のアンカーを一度に開いて閲覧することができる Web ブラウザである。元のページを表示したままリンク先ページを開くという点では、第 3.2 節 で述べた「操作 2」における問題点を解消したものといえる。リンク先文書はそれぞれリンク元の文書の右側に配置され、閲覧者は文書構造を把握しながら閲覧することができる。また、ウィンドウ開閉と移動操作を複数まとめて実行でき、階層を持つページの一覧性と操作性の向上に貢献している。ただし、アンカーとリンク先文書との位置の対応はタイトルバーの文字列によって照合するか、閲覧者が記憶していなければならない。我々の手法を用いた場合では、アンカー位置の直後にリンク先文書が配置されるので、閲覧者が位置を記憶する必要はない。

SmallBrowse[52] は、Web の閲覧履歴を用いて閲覧者が次に辿るリンクを推測し提示する PBE システムである。PDA などの比較的小さな画面において、スクロールしながらリンクを探す手間を抑えることを目的としている。SmallBrowse では、提示したアンカーリンクに関する情報を TipHelp というポップアップウィンドウで表示する。閲覧動作における負荷を軽減するという点では **inlineLink** と共通している。SmallBrowse は定期的にページを訪問し閲覧する際に効果を発揮する。**inlineLink** は構造を持つページ群に対して閲覧者が表示する情報の粒度を調節できる。

3.8 結言

リンク先の文書を元の文書に埋め込むことにより閲覧者のナビゲーション負荷を軽減するリンク手法 **inlineLink** を提案した．本手法は構造化文書を閲覧する際に頻繁に発生する“Back” ボタン操作を不要とする．また視点の移動を抑えることにより効率よいハイパーテキスト閲覧が可能である．既存の Web ブラウザを利用して **inlineLink** を実現した．また実験によって既存のリンク手法に比べマウスのクリック回数と移動量を抑える効果があることを確認した．

第 4 章

発展例：社会的インタラクションを導入した協調型ヘルプ環境

4.1 緒言

第 2 章 では Java アプレットを対象としたアニメーションを簡単に準備する方法について述べた。第 3 章では構造化文書を閲覧するための文書挿入方式について述べた。これらの技術を利用することによりヘルプコンテンツである Java アプレットを対象としたアニメーションと関連する構造化文書を効率よく提供できる。しかし、これらの枠組みをネットワークアプリケーションのヘルプとして利用者に提供するには検索機能が不可欠である。しかし、従来のヘルプ検索機能においては、利用者が必ずしも効率よく目的の情報を探し出すことができなかった。我々は、従来の検索機能が持つ問題点を掲げ、それらを解決する方法を提案する。

4.2 オンラインヘルプ検索時の問題点

オンラインヘルプは、有限の情報の中から知りたい情報をキーワード検索手法などを用いて通常一人で調べるものである。そのため、利用者がヘルプを検索するためのキーワードや関連知識がないと、ヘルプをうまく検索できない。知りたい情報を検索しても見つからない場合は、以下の 2 つの問題のうち少なくとも一方に問題がある。

- 検索方法の問題 (キーワードの選択が不適切、条件指定の誤りなど)
- ヘルプの問題 (情報を含んでいない)

この状況において、どちらに原因があるのか特定することは一般に困難である。その理由として、ヘルプに情報が含まれていないことを短時間で確認するのは困難であることが挙げられる。ヘルプは通常キーワードのゆれを吸収するための辞書を持っていることはあっても、含まれないトピックに関する辞書は持っていない。そのため、問題を特定する代わりに、それぞれ

- 異なるキーワードを用いて検索を繰り返す
- 必要な情報が含まれていないことを調べるために関連する項目全てを調べる

という2つの作業を同時または逐次に行うことが多い。

これらの作業を行っても情報が得られなかった場合、またヘルプに知りたい情報が含まれていない場合、ユーザはその問題について詳しい(と思われる)人に連絡を試みる。例えば、近くにいる人を呼んだり、メールや電話、FAX等を用いて友人に連絡したりする。それでも問題が解決しない場合、ニュースやWeb掲示板などに質問し、不特定多数の他人からのアドバイスを受けることになる。しかし、このような場面において、質問に不備があるため、熟練者が質問に答えたくても答えられない場合がしばしば発生する。この場合、質問者と熟練者は背景知識を共有し、問題を特定するまでに労力を費してしまうことが多く、実際の問題解決になかなか取りかかれない。

4.3 社交的ヘルプ：社会的インタラクションを導入した協調型ヘルプ

上に挙げた問題を解決するために、我々はヘルプ機能に社会的インタラクションを導入した「社交的ヘルプ」を提案した[53]。社交的ヘルプでは、社交を利用することにより前に挙げた問題点を解決する。

「社交」とは、人と人とのつきあい、社会での交際、世間のつきあいの意であり、「社交性」とは、人とのつきあいを上手に行う性質を表す語である。計算機とネットワークの発達によって、我々は電子メールやチャットなどの新たな社交を利用できるようになったが、「社交性」については、前に挙げた問題により必ずしも簡単に達成されるものではない。社交的なヘルプでは、基本的に助け合いの精神を助長するための仕組みを用意し、質問をする際の敷居を低くすることにより活発なインタラクションが行われることを目標とする。

我々は、ユーザがヘルプを必要とする場面の心理状況によって、「ヘルプ検索フェーズ」と「質問フェーズ」の2つに分類できると考える。「ヘルプ検索フェーズ」

ズ」とは、ユーザが他人に頼らず、基本的に自分だけで問題解決を図ろうとする状態で、例えばローカルにあるヘルプを検索する場合である。それに対して、「質問フェーズ」では、ユーザが自分だけで問題解決が困難であると判断したときに、他人からの情報を期待する状態である。社交的なヘルプでは、それぞれの場合について、アプリケーションとヘルプ間の連携を図ることにより初心者でも必要最小限の努力で問題解決ができることを目指す。

具体的には、以下の機能を備えることによって問題点を解決する。

検索キーワードの共有、訂正機能 (ヘルプ検索フェーズ) オンラインヘルプを検索するのに用いたキーワードや質問文を公開する機能である。質問者が自分の検索キーワードに自信がない場合に用いる。熟練者からのコメントや、より効果的なキーワード、または直接その問題の解決方法へのポイントを提示してもらうことが考えられる。この機能により、初心者が不適切なキーワードを入力しても、希望する質問と回答を得る可能性が高まる。

質問に必要な情報の補間機能 (質問フェーズ) これは、アプリケーションの設定や状態などを、自動的に質問に追加する機能である。アプリケーションとヘルプが統合されている環境で、ヘルプ側からアプリケーションに対して問い合わせを行うことで実現する。この機能により、質問者が現在の状況を説明する手間が省ける。また、熟練者にとっても必要な情報が得られるので、回答しやすい。

ここで、ヘルプ検索フェーズにおけるキーワードなどの情報は、リアルタイム性が要求されるので、反応を素早く手軽に返せるチャットを用いて公開する。質問フェーズにおける質問は、返答に時間がかかることが予想されるため、Web 掲示板によって公開する。図 4.1 に、社交的ヘルプを使う場合における処理の流れを示す。

4.4 インタラクションシナリオ

具体的な手順を示しながら、社交的ヘルプについて説明する。

ヘルプ検索フェーズにおけるシナリオ A 君は、表計算ソフトを使って表を作成中、表の左上に置く斜めの線を入力しようとした。その方法が分からなかった A 君は、ヘルプ検索ウィンドウを開いて調べようとした。

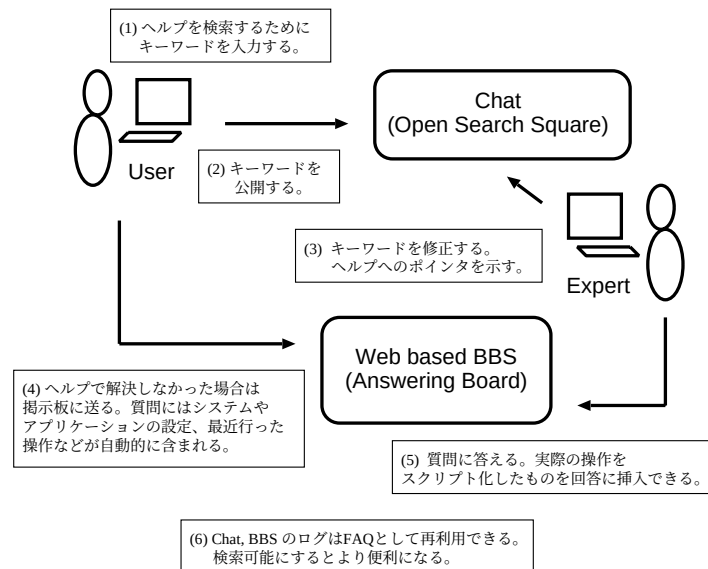


図 4.1: 社会的ヘルプにおける処理の流れ

ヘルプ検索ウィンドウ (図 4.2 左) では、キーワードによって検索できる。また、**add** ボタンを使うことでキーワード一覧から検索に使うキーワードを選ぶこともできる。ある言葉の意味が分からない場合、その言葉を入力しておいてから **glossary** ボタンを押すことで用語集を検索できる。A 君は「斜め線 表 左上 罫線」と入力し、**search** ボタンで検索したが、該当する検索結果が得られなかったため、**send keywords** ボタンを押して、検索キーワードを専用チャットルーム (Open Search Square) の発言として公開した。また、A 君はフレーズによる検索を試みたが、同じく目的に結果は得られなかった。

B さんは、Open Search Square ウィンドウ (図 4.2 右) で A 君が入力した「斜め線 表 左上 罫線」というキーワード、「表の左上に斜めの罫線を引きたい」というフレーズと、それぞれの検索結果が 0 件であるという表示を見て、アドバイスをしようと考えた。まず、A 君の発言をクリックし「斜め線 表 左上 罫線」という文字列を「セルの書式設定 罫線」に変更し、**test search** ボタンを押して、自分のキーワードで検索に成功することを確認したのち、**reply** ボタンで発言した。ちなみに **show pointer** ボタンを押すと、現在参照しているヘルプへの URI を示すことができる。A 君は B さんの「セルの書式設定 罫線」で検索した結果から、目的の情報を得ることができた。

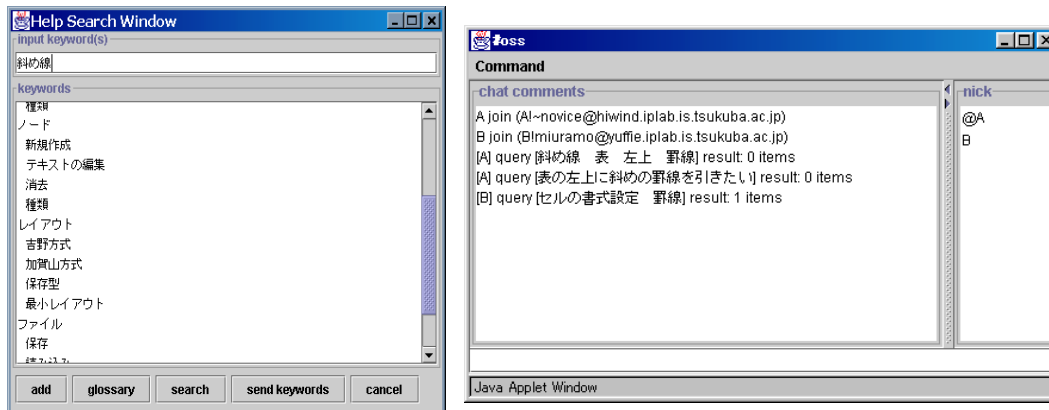


図 4.2: (左) キーワードによるヘルプ検索画面 (右) キーワードを公開して反応を得るチャットシステム Open Search Square

質問フェーズにおけるシナリオ C 君は、最近メールクライアント (Mail User Agent: MUA) を変えた。新しい MUA はメールの分類を読み終った直後に自動的に行う機能を持っている。しかし、C 君はどうやって分類するための設定を行えばいいのかわからなかった。他の MUA は使ったことがありキーワードについては自信があった。キーワードを替えて何度かヘルプを検索したが該当する項目が見当たらなかったため、質問を掲示板 (Answering Board) に送ることにした。

質問作成ウィンドウ (図 4.3 左) を開くと、既にこの MUA の設定情報、OS に関する情報、最近行われた操作などが追加されているので「メールを振り分けたいのですが、具体的な設定方法を教えてください」と質問の内容だけを書いて、**send** ボタンを押して送信した。

C 君の質問は Web ブラウザで閲覧可能な電子掲示板に掲載される。C 君の質問を見て教えてあげようと考えた D 氏は、MUA を起動し、C 君の質問 ID(数字) を入力することで返答用の投稿ウィンドウが開く。**log start** ボタンを押して、実際に設定操作を行うと、カーソル位置に

```
select [Preferences]->[Filters] % [設定] メニューの [フィルタ] を選択
press [Apply] % [更新] ボタンを押す
```

のようにスクリプト化された操作が挿入される。操作が終わったら **log end** ボタンを押す。**send** ボタンで送信する。

C 君は D 氏の回答 (図 4.3 右) をヘルプシステムに読み込ませて、スクリプト化された操作を解釈させると、画面内の操作すべき場所が逐次ポインタで指し示される。C 君は D 氏の具体的な操作を追うことで設定方法を理解することができた。

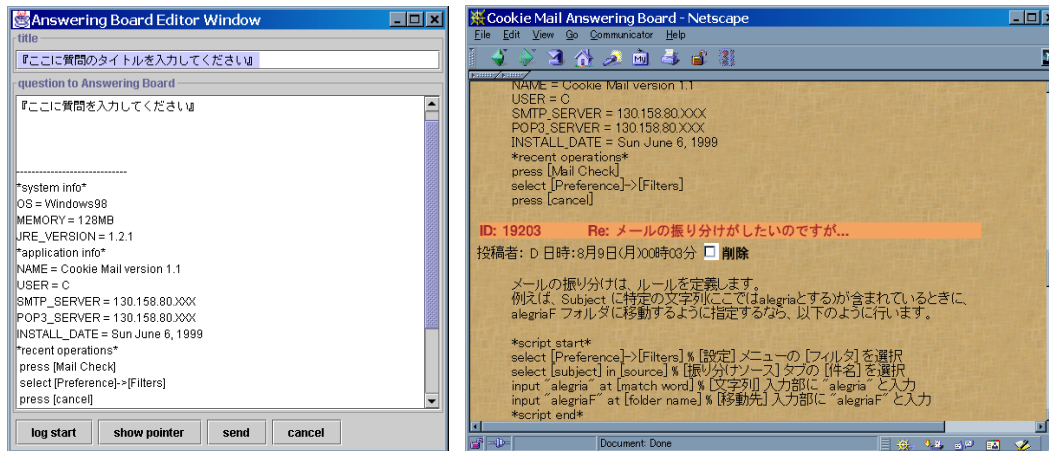


図 4.3: (左) システム情報やユーザ設定を自動的に付加する機能をもつ質問作成ウィンドウ (右) 質問とその回答を表示する掲示板 Answering Board

これらの質問と回答の例が蓄積されたものを検索可能にしておくことによりユーザーにとって有効な情報源として利用されることが期待できる。

4.5 社会的ヘルプシステム環境 “Social-Help” の実装

我々は、ヘルプ検索フェーズにおけるチャットクライアント OpenSearchSquare を Java アプレットとして実装した (図 4.4). 実装言語に Java を用いた理由として、第 2 章 で述べたアニメーションヘルプシステム Jedemo との連携が容易であることが挙げられる。Jedemo では、ヘルプの対象となるアプリケーションを変更せずに、操作履歴を記録し保存することができる。また、記録した操作を使ってユーザーに操作方法を示すことができる。この操作履歴の取得機能を使うことにより質問や回答におけるスクリプト挿入機能が実現できる。

また、質問フェーズで蓄積された質問の一覧を閲覧する際には、第 3 章 で述べた文書挿入型リンク機能 **inlineLink** を用いることにより階層構造を持つ多量のヘルプ情報の閲覧が効果的に行われると考えられる。我々は **inlineLink** を利用した掲示板 Answering Board を実装した (図 4.5). 実装言語には PHP3 (PHP:Hypertext Preprocessor) を利用した。利用者は興味のある項目のみを選択して内挿することができる。掲示板に **inlineLink** を利用することによって各項目の内容を全て挿入する必要がないため、サーバに与える負荷も少ない。

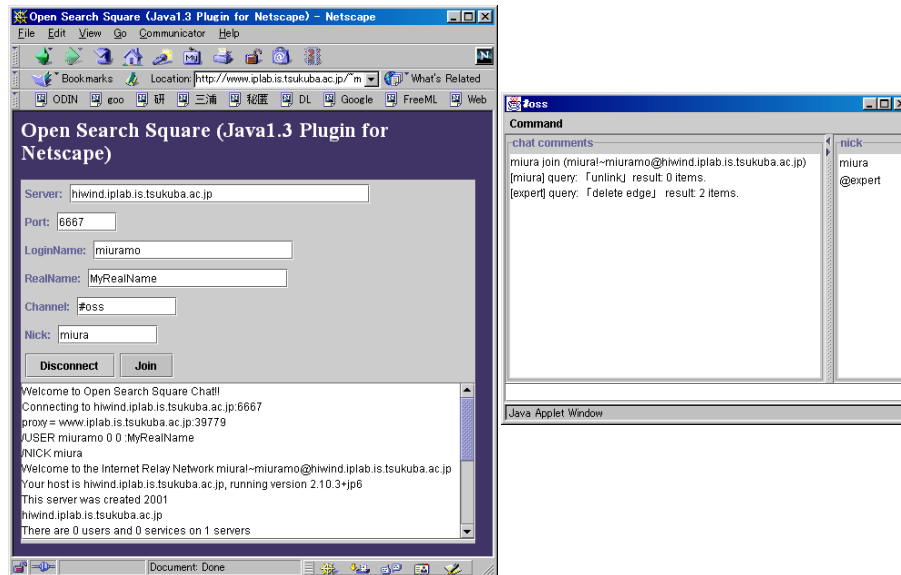


図 4.4: チャットクライアント OpenSearchSquare

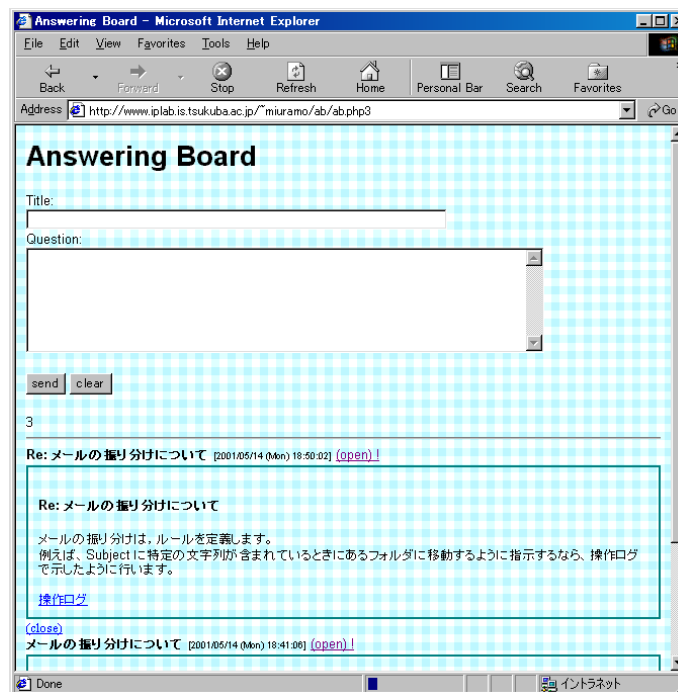


図 4.5: inlineLink を利用した Web 掲示板

4.6 関連研究

Microsoft Office では、Microsoft 社のページにある FAQ を参照するためのボタンが用意されている。しかしヘルプ検索タスクを公開したり、アプリケーションの状態を質問に含めるような仕組みは用意されていない。

検索行為に対して社会的インタラク션을導入することについては岩山が IRC (Internet Relay Chat) を用いたチャットサーチ (Chat-and-Search) 環境を構築し実験を行っている [54, 55]。チャットサーチ環境では、ユーザはチャットを利用して検索行為の開示が行える。参加者がチャットに Web 検索キーワードを明示的に発言するとチャットロボットが認識する。チャットロボットはキーワードから、特定のサーチエンジンが認識可能な URI 文字列を生成し、チャットに発言する。検索を要求した参加者のチャットクライアントは URI 文字列をブラウザに送り検索結果をブラウザに表示する。その他の参加者は、アンカーに書き替えられた URI 部分をクリックすれば検索結果を知ることができる。我々はアプリケーションのヘルプ機能にこの枠組みを応用し、検索行為を開示することによって、質問者の負担を軽減することを考えている。

ExpertFinder[56] は、質問の答えを見つける代わりに質問を答えてくれる専門家を見つけるエージェントである。ExpertFinder は Java プログラミングに関する領域知識を持っており、ユーザが作った Java のプログラムを解析して使用しているクラスやパッケージの情報からユーザのプロファイルを作成し、これを基に検索を行う。Java のプログラミングという領域に特化しており一般的なアプリケーションの知識については考慮されていない。

Ackerman らが開発した Answer Garden[57, 58] は、質問者から寄せられる質問とその回答を組織の知識として順次格納することで、利用者や回答者の手間を軽減するシステムである。質問と回答は枝分れしたネットワーク構造に分類されており、各分岐点には利用者を誘導するための質問 (アンケート) が含まれている。まず利用者はアンケートに答えながらネットワークの中を探索し、最終的に関連する質問と回答の集合に辿りつく。もしその中に目的の回答が含まれなかったり、回答に満足しなかった場合は、質問メールを作成して有識者に配送する。有識者の回答は質問者に送られると共に、自動的に蓄積される。メールによる社会的インタラク션을用いて動的に知識を蓄積し、検索行為を支援する先駆的研究の 1 つである。しかし、質問者と有識者を完全に分離しており、知識を獲得した質問者による参加はできない。このことは知識の質を保つ効果がある反面、権威を持つ者と持たざる者との間に隔差が生じる。また、参加者全員が納得するネットワークを作成するのは困難な作業である。実際、複

数の有識者の間でネットワーク構成の方針が分かれることがあったと報告している。

Zephyr[59] は、MIT の Athena プロジェクトで開発された同期チャットシステム (一種のインスタントメッセージシステム) である。Zephyr では利用者間でメッセージが交換できるほか、インスタンス (Instance) と呼ばれるコミュニティを作成できる。Zephyr は基本的にメッセージを交換・公開する簡潔なインタフェースしか提供していないにもかかわらず、その中に作られた互助コミュニティ (Help Instance) では運営するための規律や質問に回答する参加者が自発的に現れ、有効に機能することが確認された [60]。このことは、適切なインタフェースを用意することでヘルプを交換・流通するコミュニティを形成できる可能性を示唆している。

4.7 結言

オンラインヘルプに社会的なインタラクションを導入し、検索行為から質問行為へと円滑に移行できる仕組みを提案した。質問に必要なアプリケーションの状態や最近繰り返された操作などの情報を自動的に含める機能によって、質問者の心理的不安と労力を軽減できると考えている。今後の課題としてはブラウザとアプレット間の連携部分を構築し、実際に運用を行いながら、問題点を洗いだしその原因を明らかにしていくことを考えている。

第 5 章

結論

本論文では、ネットワークアプリケーションの使い方をアニメーションで具体的に見せるための方法と、文書挿入機能を用いて構造化文書の閲覧における負荷を軽減する方法を中心として述べた。

第 2 章の主要な内容は以下の通りである。我々が提案したアプレットビューアモデルを用いることによって、操作を説明する機能を持たない一般的な Java アプレットに変更を加えることなくアニメーション機能を追加し、操作の記録・再現を行うことができることを述べた。また、操作列と意味とを結び付ける実装知識 (コマンドルール) を用意することによって記録した操作の意味付けを行い、アニメーションの内容を表す注釈文字列を生成する方法について述べた。アプレット開発者は利用者に見せたい操作を行うだけで注釈付きの操作説明が準備できる。また利用者も簡単な操作でアニメーションを参照し操作の概要を直感的に理解できる。我々が実装したアニメーション機能が使用するリソースは多くのアプレットが使用するリソースに比べて少ないため、アニメーション機能の追加によって動作する環境を限定することはない。

第 3 章の主要な内容は以下の通りである。HTML 文書に代表されるハイパーテキストを閲覧するときには、文書を読む行為に加えてナビゲーション行為が要求されることを述べた。ナビゲーション行為を軽減し本来の目的である文書を読む行為に集中する方法として、リンク先文書をアンカー位置に挿入する「文書内挿機能」と、単一のアンカーで文書の展開状態を制御する「機能アンカー」を備えるリンク機能 **inlineLink** を提案し、従来のブラウザにおいて実現した。実験の結果、従来のリンク機能に比べてマウスのクリック数と移動量を軽減できることを確認した。

第 4 章は、第 2 章 と第 3 章で述べた技術を用いた発展例という位置付けであり、その内容は以下の通りである。ヘルプの検索に失敗した場合、利用者が検索方法の間

題とヘルプ自身の問題を切り分けることが困難であり心理的な負担を感じることもある。この心理的負担を軽減するために、オンラインヘルプの検索機能にチャットや Web 掲示板などの社会的インタラクション機能を導入することを提案した。この枠組みを用いることで、第 2 章で述べた具体的な操作を用いた操作説明を効果的に流通させることができると考えられる。また、Web 掲示板や構造を持つヘルプ文書に第 3 章で述べた **inlineLink** を適用することでヘルプ利用者が効率良く情報を閲覧できると考えている。

本論文で述べたこれらの技術を用いることで、従来のヘルプにあった問題点が解消され、すべての人がヘルプをより有効に利用できるようになるものと思われる。また、これらの技術はヘルプ利用者の視点だけでなく、ヘルプ記述者の負担を軽減するように設計・実装している。具体的には、アニメーションで示す操作の説明をコマンドルールによって付加する機能や、JavaScript によって動的にページ内のリンクを書き替える機能などである。これらの機能によって、ヘルプ記述者がヘルプの内容に集中でき、結果として質の高い情報が増加することが期待できる。

謝辞

約5年間にわたり厳しくも温かい指導を賜りました筑波大学電子・情報工学系田中二郎教授に深く感謝します。田中先生には、様々な場面での的確な御助言をいただきました。また研究の方向性について数多くの示唆をいただきました。

筑波大学電子・情報工学系 西原清一教授，福井幸男教授，北川博之教授，山本幹雄助教授には，本論文の審査にあたって数多くの貴重な助言・討論をいただきました。ここに感謝の意を表します。

実験方法や論文の書く上での細かな相談に逐一助言をいただきました筑波大学電子・情報工学系 志築文太郎助手に深く感謝します。

データ収集や実験，研究を進める上での議論に参加・協力していただいた田中研究室の皆様及びOBの皆様に深く感謝します。

最後に，父一夫，母世津子及び姉由希子，礼美の今日に至るまでの温かい支援に深く感謝いたします。

参考文献

- [1] 情報処理学会（編）． エンサイクロペディア情報処理 2000-2001, 第 1.7 章, pp. 28–30. オーム社出版局, 2000.
- [2] Richard E. Cullingford, Myron W. Krueger, Mallory Selfridge, and Marie A. Bi-enkowski. Automated Explanations as a Component of a Computer-Aided Design System. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-12, No. 2, pp. 168–181, March/April 1982.
- [3] Piyawadee Sukaviriya. Dynamic Construction of Animated Help from Application Context. In *Proceedings of the ACM SIGGRAPH symposium on User Interface Software*, pp. 190–202, October 1988.
- [4] Piyawadee “Noi” Sukaviriya and James D. Foley. Coupling A UI Framework with Automatic Generation of Context-Sensitive Animated Help. In *Proceedings of the third annual ACM Symposium on User Interface Software and Technology (UIST'90)*, pp. 152–166, October 1990.
- [5] Krishna Bharat and Piyawadee “Noi” Sukaviriya. Animating User Interfaces Using Animation Servers. In *Proceedings of the sixth annual ACM Symposium on User Interface Software and Technology (UIST'93)*, pp. 69–79, November 1993.
- [6] Susan Palmiter and Jay Elkerton. An Evaluation of Animated Demonstrations for Learning Computer-based Tasks. In *Human factors in computing systems conference proceedings on Reaching through technology (CHI'91)*, pp. 257–263, May 1991.
- [7] Charles Crowley. TkReplay: Record and Replay for Tk. In *USENIX Tcl/Tk Workshop Toronto*, pp. 131–140, July 1996. <http://www.cs.unm.edu/%7Ecrowley/papers/replay.tk95.html>.

- [8] Krishna Bharat, Piyawadee “Noi” Sukaviriya, and Scott Hudson. Synthesized Interaction on the X Window System. Technical Report 95-07, Graphics and Usability Center, Georgia Institute of Technology, 1995.
- [9] 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*, pp. 112–121, 1990.
- [10] Motoki Miura and Jiro Tanaka. A Framework for Event-driven Demonstration based on the Java Toolkit. In *Asia Pacific Computer Human Interaction (APCHI-98)*, pp. 331–336, July 1998.
- [11] Motoki Miura and Jiro Tanaka. Jedemo: The Environment of Event-driven Demonstration for Java Toolkit. In *International Symposium on Future Software Technology (ISFST-98)*, pp. 215–218, October 1998.
- [12] 三浦元喜, 田中二郎. Java アプレットののためのアニメーションヘルプシステム. 情報処理学会論文誌：プログラミング, Vol. 41, No. SIG4 (PRO7), pp. 56–64, June 2000.
- [13] Motoki Miura and Jiro Tanaka. Jedemo: Demonstrational Authoring Tool for Java Applets. In *Proceedings of the fifth World Conference on Integrated Design and Process Technology (IDPT2000) (CD-ROM), Copyright 2000 by the Society for Design and Process Science (SDPS)*, June 2000. 8 pages.
- [14] 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.
- [15] Microsoft Corporation. Microsoft Office – Microsoft Excel. <http://www.microsoft.com/office/excel/default.htm>.
- [16] Apple Computer, Inc. Introduction to the Macintosh Family — Second Edition.
- [17] Allen Cypher, editor. *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.

- [18] Allen Cypher. Eager : Programming Repetitive Tasks by Example. In *Human factors in computing systems conference proceedings on Reaching through technology (CHI'91)*, pp. 33–39, May 1991.
- [19] David L. Maulsby, Ian H. Witten, and Kenneth A. Kittlitz. Metamouse: Specifying Graphical Procedures by Example. In *Conference proceedings on Computer Graphics (SIGGRAPH'89)*, pp. 127–136, July 1989.
- [20] David Kurlander and Steven Feinter. A History-Based Macro By Example System. In *Proceedings of the fifth annual ACM Symposium on User Interface Software and Technology*, pp. 99–106, November 1992.
- [21] Henry Lieberman. Mondrian: A Teachable Graphical Editor. In [17], chapter 16, pp. 341–358. The MIT Press, 1993.
- [22] Henry Lieberman. A demonstrational interface for recording technical procedures by annotation of videotaped examples. *International Journal of Human-Computer Studies*, Vol. 43, pp. 383–417, 1995.
- [23] Richard Potter. Triggers: Guiding Automation with Pixels to Achieve Data Access. In [17], chapter 17, pp. 361–380. The MIT Press, 1993.
- [24] Philippe P. Piernot and Marc P. Yvon. The AIDE Project: An Application-Independent Demonstrational Environment. In [17], chapter 18, pp. 383–401. The MIT Press, 1993.
- [25] Henry Lieberman. Tinker: A Programming by Demonstration System for Beginning Programmers. In [17], chapter 2, pp. 48–64. The MIT Press, 1993.
- [26] Brad A. Myers. Creating Dynamic Interaction Techniques by Demonstration. In *CHI/GI 1987 conference proceedings on Human factors in computing systems and graphics interface (CHI + GI '87)*, pp. 271–278, April 1987.
- [27] 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.

- [28] Brad A. Myers, Rich McDaniel, Rob Miller, Alan Ferreny, 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.
- [29] Roberto Moriyon, Pedro Szekely, and Robert Neches. Automatic generation of help from interface design models. In *Conference proceedings on Human factors in computing systems: "celebrating interdependence" (CHI'94)*, pp. 225 – 231, April 1994.
- [30] Sun Microsystems Inc. JavaHelp Homepage. <http://java.sun.com/products/javahelp/>.
- [31] Tim Berners-Lee. WorldWideWeb. (Web Page). <http://www.w3.org/People/Berners-Lee/WorldWideWeb.html>.
- [32] NCSA Mosaic Home Page. (Web Page). <http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html>.
- [33] 三浦元喜, 田中二郎. ハイパーテキストにおける文書挿入型リンク機能の提案. 日本ソフトウェア科学会第 17 回大会, September 2000.
- [34] 三浦元喜, 志築文太郎, 田中二郎. inlineLink: 文書挿入型リンク機能を用いたハイパーテキスト閲覧. インタラクティブシステムとソフトウェア VIII, pp. 231–232. 近代科学社, December 2000.
- [35] Motoki Miura, Buntarou Shizuki, and Jiro Tanaka. inlineLink: Inline Expansion Link Methods in Hypertext Browsing. In *The 2nd International Conference on Internet Computing (IC'2001)*, June 2001. (to appear).
- [36] World Wide Web Consortium (W3C). (Web Page). <http://www.w3.org/>.
- [37] Document Object Model (DOM) Level 2 Specification. (Web Page), May 2000. <http://www.w3.org/TR/DOM-Level-2/cover.html>.

- [38] Netscape Communications Corporation. JavaScript Documentation. (Web Page). <http://developer.netscape.com/docs/manuals/javascript.html>.
- [39] World Wide Web Consortium (W3C). HTML 4.01 Specification. (Web Page), December 1999. <http://www.w3.org/TR/html401/>.
- [40] Linda Tauscher and Saul Greenberg. How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems. *International Journal of Human-Computer Studies*, Vol. 47, No. 1, pp. 97–138, 1997.
- [41] Benjamin B. Bederson and James D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of the ACM symposium on User interface software and technology (UIST'94)*, pp. 17–26, November 1994.
- [42] Masashi Toyoda and Etsuya Shibayama. Hyper Mochi Sheet: A Predictive Focusing Interface for Navigating and Editing Nested Networks through a Multi-focus Distortion-Oriented View. In *Proceedings of the CHI 99 conference on Human factors in computing systems (CHI'99)*, pp. 504–511, May 1999.
- [43] Microsoft Windows Update. (Web Page). <http://windowsupdate.microsoft.com/>.
- [44] Java House Topics. (Web Page). <http://java-house.etl.go.jp/ml/topics/dhtml/>.
- [45] 三菱総合研究所科学技術研究本部. 新技術 Today. (Web Page). <http://www.internetclub.ne.jp/TECH/Ztoday/UD/index.html>.
- [46] Theodorich Kopetzky and Max Mühlhäuser. Visual Preview for Link Traversal on the WWW. In *Proceedings of the 8th International World Wide Web Conference (WWW8)*, 1999. <http://www8.org/w8-papers/4b-links/visual/visual.html>.
- [47] Harald W. R. Weinreich and Winfried Lamersdorf. Concepts for Improved Visualization of Web Link Attributes. In *Proceedings of the 9th*

- International World Wide Web Conference (WWW9)*, 2000. <http://vsys-www.informatik.uni-hamburg.de/projects/hyperscout/>.
- [48] Polle T. Zellweger, Bay-Wei Chang, and Jock Mackinlay. Fluid Links for Informed and Incremental Link Transitions. In *Proceedings of HyperText'98*, pp. 50–57, 1998.
- [49] Polle T. Zellweger, Susan Harkness Regli, Jock D. Mackinlay, and Bay-Wei Chang. The Impact of Fluid Documents on Reading and Browsing: An Observational Study. In *Proceedings of the CHI 2000 conference on Human factors in computing systems (CHI'00)*, pp. 249–256, April 2000. <http://www.parc.xerox.com/istl/projects/fluid/>.
- [50] Eser Kandogan and Ben Shneiderman. Elastic Windows: A Hierarchical Multi-Window World-Wide Web Browser. In *Proceedings of the 10th annual ACM Symposium on User Interface Software and Technology (UIST'97)*, pp. 169–177, October 1997.
- [51] Eser Kandogan and Ben Shneiderman. Elastic Windows: Evaluation of Multi-Window Operations. In *conference proceedings on Human factors in computing systems (CHI'97)*, pp. 250–257, March 1997.
- [52] Atsushi Sugiura. Web browsing by example. In Henry Lieberman, editor, *Your Wish is My Command – Programming by Example –*, chapter 4, pp. 61–85. Morgan Kaufmann Publishers, 2001.
- [53] 三浦元喜, 田中二郎. 社交的ヘルプ：社会的インタラクション機構を備えた協調型オンラインヘルプシステム. インタラクティブシステムとソフトウェア VII, pp. 181–186. 近代科学社, December 1999.
- [54] Noboru Iwayama, Masahiko Murakami, and Masahiro Matsuda. Consulting Search Engines as Conversation. In *Proceedings of the 7th Conference on Human-Computer Interaction (INTERACT'99)*, August 1999.
- [55] 岩山登. 検索行為の自発的開示によるコラボレーション. インタラクション'99, pp. 17–24. 情報処理学会, March 1999.

- [56] Adriana Vivacqua and Henry Lieberman. Agents to Assist in Finding Help. In *Proceedings of the CHI 2000 conference on Human factors in computing systems (CHI'00)*, pp. 65–72, April 2000.
- [57] Mark S. Ackerman and Thomas W. Malone. Answer Garden: A Tool for Growing Organizational Memory. In *Proceedings of the conference on Office Information Systems*, pp. 31–39, April 1990.
- [58] Mark S. Ackerman. Augmenting the Organizational Memory: A Field Study of answer Garden. In *Proceedings of the conference on Office Information Systems*, pp. 31–39, April 1990.
- [59] Zephyr on athena (ac-34) draft. (Web Page). <http://web.mit.edu/olh/Zephyr/Zephyr.html>.
- [60] Mark S. Ackerman and Leysia Palen. The Zephyr Help Instance: Promoting Ongoing Activity in a CSCW System. In *Conference proceedings on Human factors in computing systems (CHI'96)*, pp. 268–275, April 1996.

著者論文リスト

【本研究に関する論文】

- [1] Motoki Miura and Jiro Tanaka. A Framework for Event-driven Demonstration based on the Java Toolkit. In *Proceedings of third Asia Pacific Computer Human Interaction (APCHI-98)*, pp. 331–336, July 1998.
- [2] Motoki Miura and Jiro Tanaka. Jedemo: The Environment of Event-driven Demonstration for Java Toolkit. In *Proceedings of the International Symposium on Future Software Technology (ISFST-98)*, pp. 215–218, October 1998.
- [3] 三浦元喜, 田中二郎. Java アプレットのためのアニメーションヘルプシステム. 情報処理学会論文誌：プログラミング, Vol. 41, No. SIG4 (PRO7), pp. 56–64, June 2000.
- [4] Motoki Miura and Jiro Tanaka. Jedemo: Demonstrational Authoring Tool for Java Applets. In *Proceedings of the fifth World Conference on Integrated Design and Process Technology (IDPT2000) (CD-ROM)*, Copyright 2000 by the Society for Design and Process Science (SDPS), June 2000. 8 pages.
- [5] Motoki Miura, Buntarou Shizuki, and Jiro Tanaka. inlineLink: Inline Expansion Link Methods in Hypertext Browsing. In *Proceedings of 2nd International Conference on Internet Computing (IC'2001)*, June 2001. (to appear).
- [6] 三浦元喜, 田中二郎. 社交的ヘルプ：社会的インタラクション機構を備えた協調型オンラインヘルプシステム. インタラクティブシステムとソフトウェア VII, pp. 181–186. 近代科学社, December 1999.

【その他参考文献】

- [8] Koji Miyagi, Motoki Miura, and Jiro Tanaka. A graph layout and a multi-focus perspective display in the flowgraph editor for the legal articles. In *Proceedings of the International Symposium on Future Software Technology (ISFST-97)*, pp. 378–385, October 1997.
- [9] Koji Miyagi, Motoki Miura, and Jiro Tanaka. Flowgraph Editor for Legal Articles. In *Journal of Advanced Computational Intelligence*, pp.34–41, Vol.2, No.1, February 1998.

付録 A

inlineLink 実験詳細

3.6で述べた **inlineLink** の評価実験の詳細について示す.

A.1 実験に用いた文書の詳細

表 A.1 と **表 A.2** に, 実験に用いた文書の文字数と行数を示す. ここで「文字数」は各 HTML 文書に含まれるタグを取り除いた後のテキストの文字数を示す¹. また「行数」は, 実験に用いた Web ブラウザでテキストのフォントサイズを「中 (Medium)」にし, 最大化表示して計測した.

A.2 t 検定の結果

表 A.3, **表 A.4**, **表 A.5** に t 検定の結果を示す. これらの検定はすべて有意水準 5% ($\alpha = 0.05$) にて行った.

¹例えば, index.html というファイル名だった場合, % w3m -dump index.html | wc -c として計算した. w3m の出力文字コードは EUC であるため, 実際の日本語の文字数とは異なる.

文書名	文字数	行数
(インデックスページ)	634	18
dired の起動	800	8
dired バッファ内のコマンド	780	8
dired でのファイル削除	1457	18
一度に多数のファイルにフラグを付ける	2263	25
dired でのファイルの訪問	1225	16
dired の印とフラグ	3527	43
ファイルの操作	3957	44
dired でのシェルコマンド	1643	17
dired でのファイル名の変換	2073	27
dired でのファイルの比較	639	7
dired のサブディレクトリ	1375	17
サブディレクトリへの移動	947	11
サブディレクトリの隠蔽	1077	10
dired バッファの更新	2323	29
dired と find プログラム	1524	16

表 A.1: inlineLink 実験に用いた文書 (日本語翻訳版)

文書名	文字数	行数
(Index Page)	684	18
Entering Dired	794	8
Commands in the Dired Buffer	827	8
Deleting Files with Dired	1526	18
Flagging Many Files at Once	2500	27
Visiting Files in Dired	1308	12
Dired Marks vs. Flags	3655	43
Operation on Files	4306	46
Shell Commands in Dired	1592	16
Transforming File Names in Dired	2232	27
File Comparison with Dired	600	6
Subdirectories in Dired	1362	17
Moving Over Subdirectories	957	15
Hiding Subdirectories	1041	10
Updating the Dired Buffer	2168	29
Dired and find	1453	16

表 A.2: inlineLink 実験に用いた文書 (英語版)

	normal	inlineLink
平均	26.16666667	16.8
分散	388.8333333	113.5448276
観測数	30	30
ピアソン相関	0.389598517	
仮説平均との差異	0	
自由度	29	
t	2.787838253	
$p(T \leq t)$ 片側	0.004633886	
t 境界値 片側	1.699127097	
$p(T \leq t)$ 両側	0.009267773	
t 境界値 両側	2.045230758	

表 A.3: t 検定：一対の標本による平均の検定ツール：クリック数

	normal	inlineLink
平均	167.1666667	126.5
分散	18193.52299	7792.12069
観測数	30	30
ピアソン相関	0.676067578	
仮説平均との差異	0	
自由度	29	
t	2.240169197	
$p(T \leq t)$ 片側	0.016454685	
t 境界値 片側	1.699127097	
$p(T \leq t)$ 両側	0.032909369	
t 境界値 両側	2.045230758	

表 A.4: t 検定：一対の標本による平均の検定ツール：移動量

	normal	inlineLink
平均	155.5	140.1333333
分散	3966.051724	2413.085057
観測数	30	30
ピアソン相関	-0.018915523	
仮説平均との差異	0	
自由度	29	
t	1.044267026	
$p(T \leq t)$ 片側	0.152494467	
t 境界値 片側	1.699127097	
$p(T \leq t)$ 両側	0.304988933	
t 境界値 両側	2.045230758	

表 A.5: t 検定：一対の標本による平均の検定ツール：作業時間

A.3 被験者に与えた問題 (日本語版)

SESSION 1

[1] dired の起動

dired を起動するには、_____ か M-x dired を使います。

[2] ファイル削除

_____ このファイルに削除フラグを付ける。

_____ 削除フラグが付いたファイルを (本当に) 削除する。

[3] バックアップファイルに自動的に削除フラグを付ける

_____ (名前が '~' で終る) すべてのバックアップファイルに削除フラグを付ける。

[4] ファイルの訪問

_____ M-x view-file を用いて現在行が表すファイルを閲覧する (dired-view-file)。

[5] ファイルに印を付ける

_____ カレントファイルに '*' で印を付ける (dired-mark)。

_____ dired バッファのすべてのファイルの印を消す (dired-unmark-all-files-no-query)。

[6] ファイルのコピー

_____ new RET 指定したファイルをコピーする (dired-do-copy)。

SESSION 2

[7] ファイルの比較

—— diff プログラム (dired-diff) を用いて、(ポイントがある箇所の) カレントファイルを (マークがある箇所の) 他のファイルと比較する。

[8] 指定したファイルの圧縮

—— 指定したファイルを圧縮する (dired-do-compress)。ファイルがすでに圧縮済みと思われるときは展開する。

[9] 指定したファイルの内容を置換

—— *from* RET *to*RET 指定したファイル群のそれぞれについて、query-replace-regexp (問い合わせながら正規表現を置換する) を実行し、*from*(正規表現) に一致する部分を *to* に置換する (dired-do-query-replace)。

[10] サブディレクトリの内容 (リスト) も同時に表示

—— 指定したサブディレクトリの内容をバッファの終りに追加する。

[11] サブディレクトリの隠蔽

—— ポイント位置にあるサブディレクトリを隠蔽、あるいは、再表示して、ポイントをつぎのサブディレクトリへ移動する (dired-hide-subdir)。

[12] ファイルリスト (バッファ) の更新

—— dired バッファの全内容を更新する (revert-buffer)。

—— アルファベット順の表示と日付／時間順の表示を切り替える (dired-sort-toggle-or-edit)。

A.4 被験者に与えた問題 (英語版)

SESSION 1

[1] Entering Dired

To invoke Dired, do `_____` or **M-x dired**.

[2] Deleting Files

`_____` Flag this file for deletion.

`_____` Delete the files that are flagged for deletion.

[3] Flag all backup files

`_____` Flag all backup files (files whose names end with `~'`) for deletion.

[4] Visiting Files

`_____` View the file described on the current line, using **M-x view-file** (`dired-view-file`).

[5] Marking Files

`_____` Mark the current file with ``*'`` (`dired-mark`).

`_____` Remove all marks from all the files in this Dired buffer (`dired-unmark-all-files-no-query`).

[6] Copying Files

`_____ new RET` Copy the specified files (`dired-do-copy`).

SESSION 2

[7] File Comparison

____ Compare the current file (the file at point) with another file (the file at the mark) using the diff program (dired-diff).

[8] Compressing Files

____ Compress the specified files (dired-do-compress). If the file appears to be a compressed file already, it is uncompressed instead.

[9] Replace specified file contents

____ *from* **RET** to **RET** Perform query-replace-regexp on each of the specified files, replacing matches for *from* (a regular expression) with the string *to* (dired-do-query-replace).

[10] Show Subdirectories List at the same time

____ Insert the contents of a subdirectory later in the buffer.

[11] Hiding Subdirectories

____ Hide or reveal the subdirectory that point is in, and move point to the next subdirectory (dired-hide-subdir).

[12] Updating the File List (buffer)

____ Update the entire contents of the Dired buffer (revert-buffer).

____ Toggle between alphabetical order and date/time order (dired-sort-toggle-or-edit).