

自由ストロークに基づく電子白板システムのための ソフトウェアアーキテクチャ

五十嵐 健夫 †, W. Keith Edwards ‡, Anthony LaMarca ‡, Elizabeth D. Mynatt ‡

† 東京大学 情報工学専攻,
東京都文京区本郷 7-3-1
03-5841-7413
takeo@mtl.t.u-tokyo.ac.jp

‡ Xerox PARC
3333 Coyote Hill Rd. Palo Alto, CA 94304, USA
+1 650-812-4405
kedwards,lamarca,mynatt@parc.xerox.com

1. はじめに

ホワイトボードはオフィスにおいて最も重要な役割を果たしている道具の一つである。メモを書き付けたり、スケッチを描いて考えを整理したり、文章を練る際に要点を書き留めたり、同僚との議論の際に使用したりと様々な目的に使用される。デスクトップコンピュータが分析や清書といったアウトプットに近い作業に使われるのに対し、一般的にホワイトボードはより思考に近いレベルでの創造的な活動に使用されていると考えることができる[16]。

我々は、このような観察に基づき、計算機によって機能を強化された電子的なホワイトボード(電子白板, 通称 Flatland)の研究開発を行っている。本研究の目的は、「手書きの線を書き付けるだけ」という本来のホワイトボードの簡便さを失うことなく、情報の保存や計算・図の整形といった計算機によって可能となる拡張能力を付加していく手法を確立することである。このような電子白板は、現在使われているデスクトップコンピュータでは困難な、柔軟な知的活動の支援が可能になると期待できる。

本稿では、[17]で紹介された Flatland の機能を実現しているソフトウェアアーキテクチャーについて詳しく説明する。

本稿で提案するソフトウェアアーキテクチャは、デスクトップコンピュータで使用されているマウスとキーボード入力に基づいた GUI システムの、ペン入力版と見ることができる。通常の GUI システムと同様に、我々の紹介するシステムも画面全体をいくつかの領域に分けて使用する機構や、それぞれの領域内で特定の計算機能を付与する機構を提供している。基本的

“An Ink-oriented Software Framework for Personal Electronic Whiteboards”, Takeo Igarashi (Univ. of Tokyo), W. Keith Edwards, Anthony LaMarca, Elizabeth D. Mynatt (Xerox PARC)

な違いは、従来のシステムが文書生成といったアウトプットを志向した活動の支援を目的としているのに対し、我々のシステムはアイデアの書き留めといった、より創造的な活動を支援することを目的としている点である。

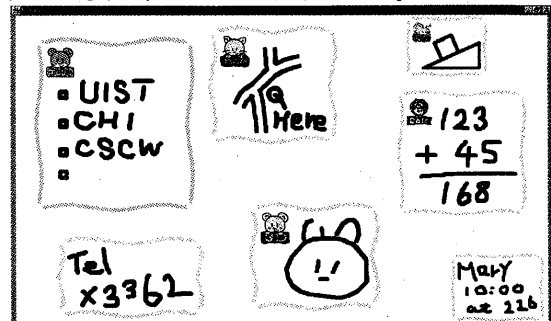


図1: Flatland システムの画面例

より詳細には、本システムは以下のような3つの大きな特徴を持っている。まず第一に、創造的活動を支援するために、入力はできるだけ簡単なもの、出力はできるだけインフォーマルな形のものを採用している。具体的には、従来の GUI では、ユーザの入力はマウスによる GUI 部品の操作あるいはキーボードによるものであったが、本システムではより“軽い”入力方法である手書きストロークのみを入力とする。同様に、従来の GUI において表示されるものは整形されたテキストや詳細にデザインされた部品であったが、本システムでは柔軟な発想を妨げないようユーザの入力からシステムの出力まですべて自由ストロークの形で表現される。すなわち、本システムでは、入力から内部表現、出力まで一貫して「自由ストローク」を基本単位として処理されている。

第二に、形のはっきり定まっていない創造段階での情報操作を支援するために、操作対象の情報とそれらの持つ内部構造の間の関係が非常に柔軟になっている。具体的には、通常の GUI システムにおいて静的な関係であったウィンド

ウとアプリケーションに対し、本システムにおいては情報素（自由ストローク）の集合関係（作業領域の区切り）は必要に応じて変更可能であり、また個々の作業領域に割り当てられたアプリケーションは取り外しや付け替えが可能となっている。

最後に、ホワイトボード上での長期的な活動を効率的に支援するために、自動的な情報の保存、およびコンテキスト情報による検索機能を提供している。従来のデスクトップシステムでは、ユーザは明示的に文書をファイルシステムから取り出し、編集し、また保存するといった作業が必要であった。しかし、このような操作は、「手書きでメモを取る」といった処理が主体の電子白板においては手間がかかりすぎる。本システムでは、あらゆる時点におけるすべての画面上のストロークが自動的に時間や色といった付加情報とともに記録されており、明示的な保存操作をすることなく、後から取り出すことが可能となっている。

以下、関連研究について述べた後、システムの動作について簡単に説明し、ついでアーキテクチャについて各特徴毎に詳細な説明を行う。最後に実装について触れ、結論を記す。

2. 関連研究

本研究に非常に関連の深いものに **Kramer** の [10] および [11] がある。これらは、従来一対一であった画面上の表現（ここではストローク）と計算機内部での解釈とのつながりを一対多にして、同一の表現の柔軟な構造化 [10]、および多様な計算処理の適用を可能にする枠組みを示したもので [11]、本研究の出発点となるものである。本研究では、先行研究の発展として、具体的なアプリケーションの例を数多く示す他、画面領域の管理方法やアプリケーションの動作など具体的な問題についてより詳細な検討および提案を行う。

ペン入力を用いた計算機システムに関する研究は数多く存在している。すでに商業化されている文字認識に関するものの他に、ペンによるテキストの高速入力 [18] やジェスチャ入力 [9] に関するものがある。システムとしては、創造的な活動を支援することを目的としたものが様々な応用分野で提案されている他 [14, 6, 23, 13]、会議での利用を想定した電子白板システムが多く開発されている [8, 19, 15]。これらの研究が主に特定の入力手法を提案するものであったり、特定の作業のためにデザインされたシステムであ

るのに対し、我々の研究は **Kramer** らの研究と同様にペンを利用した多様なアプリケーションを動かすための基本的なフレームワークを対象としている点を特徴とする。

従来の GUI の枠組みにかわるインタフェースのためのソフトウェアフレームワークに関するものとしては他に **Pad++** [1] や **MagicLens** [2] といったものが挙げられる。

3. ユーザインタフェース

本節では、ユーザからの視点でシステムの動作を概説する。細かい動作およびデザイン上の議論については [17] を参照のこと。

3.1. 基本操作

本システムは基本的には従来のホワイトボードと同様に動作する。すなわち、ペンでボード上に自由に線を描くことで入力を行う。描かれた線は空間的な近接関係に基づき自動的にグループ化される (図 2a, b)。グループ化されたストロークはセグメントと呼ばれる作業領域を構成する。セグメントをまたぐ線を引くと、それらのセグメントを融合することができる (図 2c)。一方、ペン上のボタンを押しながら線を引く (具体的な機構はハードウェアによる。マウスにおける右ボタンに相当) ことで制御ジェスチャが入力される。制御ジェスチャでぐちゃぐちゃした線を描くとその下にある線の消去が消去される。またセグメントを横切る線を引くことでセグメントの分割が実現される (図 2d)。

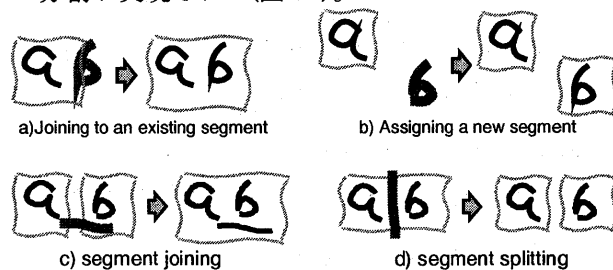


図2: ストロークのグループ化処理

セグメントの囲いの上から制御ジェスチャを開始することで、移動が実現される (ドラッグ操作)。領域同士の重なり合いを防ぐため、他の領域にぶつかった場合にはその領域を押し分け、さらに場所がなくなるとその領域は押し潰されて小さくなる (図 3)。小さくなった領域は、その上で作業を開始すると、もとの大きさに戻る。その他、カーテンを引くようにして隣のパネルに移動するといったことも可能となっている。

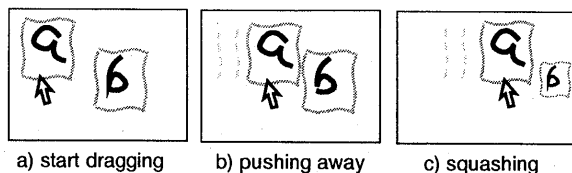


図3: 領域の移動と押し潰し

制御ジェスチャによるタッピング操作により図4のようなパイメニュー[22]が表示され、undo や redo, アプリケーションの付与といった操作を行うことができる。これはマーキングメニューとしても動作するので、慣れたユーザは短い線を制御ジェスチャにより各方向に向けて引くだけでコマンド操作を起動できる。

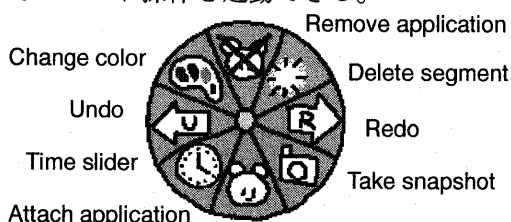


図4: パイメニュー

3.2. アプリケーション

Flatland は、上記のように単に手書きストロークを書き付ける機能に加えて、さまざまなアプリケーションを走らせることができるようになっている。アプリケーションは、ユーザによって明示的にセグメントに対して付加される。通常のウィンドウシステムと異なり、このアプリケーションは随時取り外しおよび付け替えが可能である。また、アプリケーション毎にボタンやメニューが表示されるのではなく、通常の描画操作に対して各アプリケーションが適切な処理を行なう。以下に現在実装されているものを列挙する。

リスト管理: 縦に並んだ手書きの項目の列を管理する。小さい四角を起点とする操作ジェスチャで順番を変更したり、項目を削除したりすることができる(図5)。

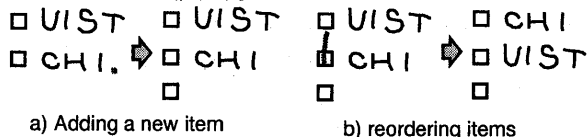


図5: リスト管理

地図描画: 線を引くと、自動的に2本の線からなる道路が描かれる。交差点や消去操作等も適切に処理される(図6)。

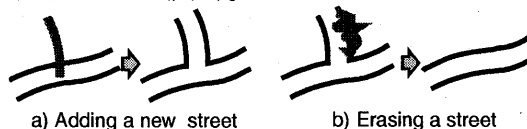


図6: 地図描画

幾何図形描画: 手書きの自由ストロークを自動的に整形して提示する。複数の候補の自動生成や次の描画操作の予測等も行う[12](図7)。

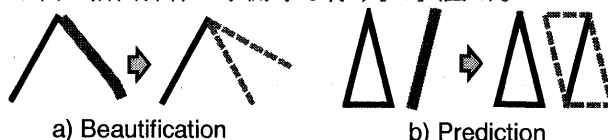


図7: 幾何図形描画

3次元描画: 入力された2次元図形が3次元物体が生成される。手書きストロークによる切断や突起生成といった編集も可能[13](図8)。

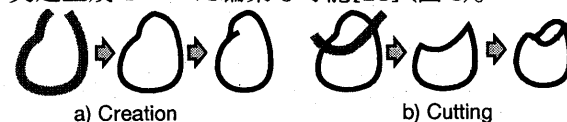


図8: 3次元描画

計算機: 手書きの数式を入力として計算を行う(図9)。長い横線が描かれると、その上にある数式を認識して計算する。手書き文字認識を利用している。計算結果は手が気風に表示される。

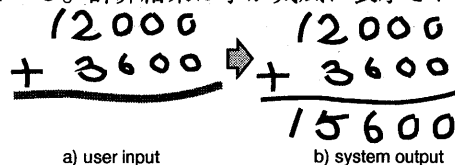


図9: 計算機

3.3. 履歴管理

Flatland におけるユーザの操作はすべてシステムによってコンテキスト情報と共に自動的に記録され、後から取り出すことができるようになっている。

一つの方法は、スライダーによって領域や画面の状態を任意の時点に逆戻しするものであり[21](図10左)、もう一つは時間や位置、色などのコンテキスト情報を元に検索するものである。

これにより、作業の度に明示的に保存操作を行うことなく、「先週、右の隅に描いた地図」といった情報を取り出すことができる。現在の実装では、表示されるボタン類を操作して検索条件を設定すると、検索結果が縮小表示される(図 10 右)。

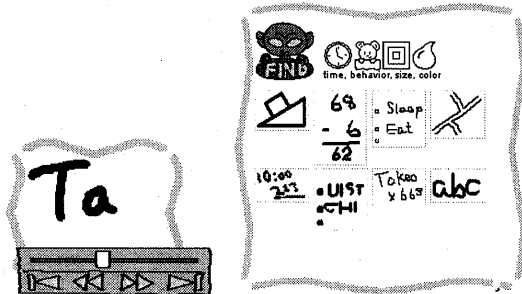


図10: 時間およびコンテキスト情報による検索

4. アーキテクチャ

本節ではアーキテクチャの全体像を示し、以下の節で各特徴について詳しく述べる。

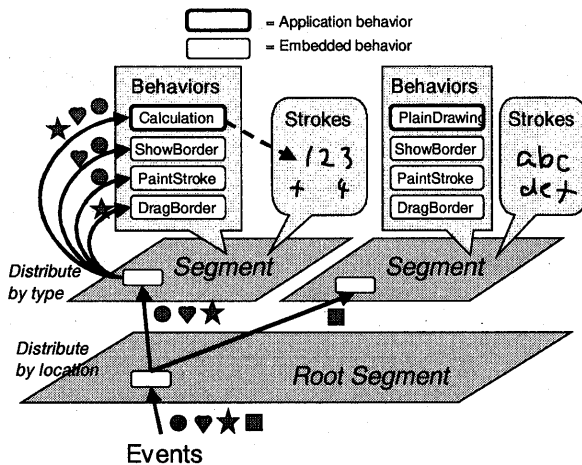


図11: アーキテクチャの概要

全体像を図 11 に示す。ルートセグメントが複数のセグメントを管理し、セグメントは各種の処理モジュール(Behavior)とストロークの集合を管理する構造になっている。

システム全体を通してストローク(具体的には点列で表現される折れ線)を基本的な情報の単位として取り扱う。ユーザの入力、画面表示、内部での表現がすべてストロークを基本として表現され、計算処理はストロークの集合に対する編集操作として記述される。これは、アプリケーション固有の内部表現を基本とし、画面表示やユーザ入力とはまた別の表現を用いている通常の GUI システムと対照的である。

画面上のストロークはグループ化されて、セグメントとして管理される。セグメントは移動や押し潰しなどの領域管理の基本となる他、アプリケーションの付与や検索の際の単位となる。セグメントは、互いに重ならない、自由に分割されたり統合されたりするといった点で通常のウィンドウと異なっている。

セグメントには任意の処理モジュールを付与することができる。処理モジュールは特定のサービスを提供するもので、セグメントでおきたイベントを監視し、適切な処理を行う。一つのセグメントに対しては、常に一つのアプリケーションレベル処理モジュールと、複数の基礎処理モジュールが付与されている。アプリケーション処理モジュールは、ユーザが直接選択することのできるアプリケーションに相当するものである。基礎処理モジュールは、入力中のストロークを画面に表示したり、セグメントの枠を表示するといった基礎的なサービスを提供する。GUI システムにおけるウィンドウとアプリケーションの関係が固定的なのに対し、セグメントと処理モジュールはセグメントから動的に取り除いたり付加したりすることができる。

入力されたストロークは、まずルートセグメントによって対応するセグメントに分配され、ついでそのセグメントに付与されている処理モジュールに分配される。処理モジュールは必要に応じてセグメントによって管理されているストロークの集合に変更を加える。

5. ストロークを基本単位とした入出力

Flatland はストロークを中心とした設計となっている。ユーザの入力はすべて自由ストロークによって行われ、システムのフィードバック(処理結果の提示など)はすべてストロークの集合として表示される。これによって、システム内部の処理はすべてストロークを対象とした計算処理として記述することが可能となる。さらに、システムが処理結果として生成したストロークをさらに後の処理への入力として使用するということも自然に実現される。

5.1. 入力ストロークの処理

ユーザがストロークを描き終わると、まずルートセグメントがそのストロークの送り先となるセグメントを検出する。もし入力ストロークが単一のセグメント内に含まれている、あるいは十分に近接している場合には、そのセグメントが送り先になる。近くにセグメントがない場

合には、新しいセグメントを生成してそこを送り先とする。もし2つ以上のセグメントにまたがっている場合には、それらのセグメントを融合し、できあがったセグメントを送り先とする。

入力ストロークを受け取ったセグメントは、それをそのままセグメントの管理するストローク集合に加えるのではなく、セグメントに含まれているアプリケーションレベル処理モジュールに処理を委託する(具体的には、`addInputStroke` メソッドが呼ばれる)。アプリケーションレベル処理モジュールは、必要に応じて、セグメントに表示ストロークを加えたり変更したりする。このような構成とすることで、低レベルでの動作を気にすることなく、基本的には `addInputStroke` メソッドを記述するだけでアプリケーション処理モジュールを設計することができるようになる(図12)。

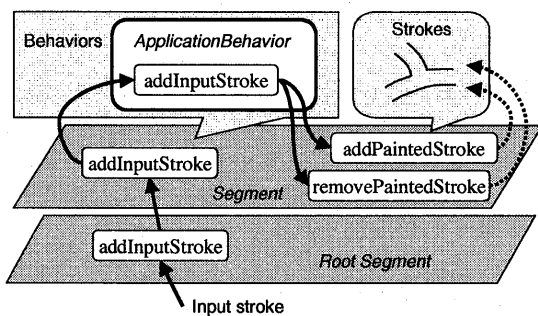


図12: 入力ストロークの処理

5.2. 処理結果のストロークによる表現

システムにおける情報の提示はすべてストロークとして表示される。これは主に、手書きによるホワイトボードの外観を保持し、インフォーマルなインタラクションを促進するといった美術デザイン上の理由からであるが、同時にシステム全体の実装をわかりやすくするという効果もある。

アプリケーション処理モジュールは、`addPaintedStroke` や `removePaintedStroke` といったメソッドを呼び出すことで、親セグメントが管理している表示ストロークの集合に変更を加える。これは、通常のアプリケーションプログラムが `drawText` や `drawLine` といった直接描画操作を呼び出すのと比較して間接的な方法である。これによってアプリケーション処理モジュールの側で特に表示に関して特別な処理ルーチンを書くことなく、セグメントの移動や押し潰し、表示内容に基づく検索といった動作

を行うことが可能となる。

6. 取り外し可能な処理モジュール

Flatland における処理モジュールは、ウィンドウシステムにおけるアプリケーションに近いが、いくつかの重要な違いがある。

まず、通常のウィンドウとアプリケーションの関係は静的で、システム動作中に関係が変化するのではないが、処理モジュールは動的に取り外しおよび付け替えが可能である。次に、ウィンドウに対応するアプリケーションは通常1つであるが、セグメントは複数の処理モジュールを持つことができる。最後に、ウィンドウの視覚的な表現はアプリケーション固有の描画ルーチンで生成されるが、セグメントではストロークの集合として統一された方法で管理される

6.1. イベント処理

各種の異なる処理モジュールに対して、イベントを知らせるために、Java におけるイベントリスナーによる実装を採用している。`SurfaceListener` はセグメントの移動や分割統合といったイベントを受け取る。ほとんどの基礎処理モジュールは `SurfaceListener` を実装している。`StrokeListener` はユーザによって描かれたストロークを処理するためのもので、アプリケーション処理モジュールに必ず実装されている。`MetaStrokeListener` は制御ジェスチャーを処理するためのものである。

6.2. 基礎処理モジュール

基礎処理モジュールは、セグメントに関わる各種の基本的な計算処理を提供するもので、ユーザによる指示ではなく自動的に付与される。このような計算処理は、処理モジュールとして独立させるのではなく、セグメント自体の機能として実装する方法もありうるが、独立させることでセグメントの実装をストロークの保持と処理モジュールへのイベントの配布という単純なものとするができる。現在の実装では、画面の描画要求に応じてストロークを画面に描画する `PaintStroke` 処理モジュール、枠をつかんで移動する制御ジェスチャーを処理する `DragBorder` 処理モジュール、セグメントの分割や統合を管理する `Segmenting` 処理モジュールなどが実装されている。

6.3. アプリケーション処理モジュール

アプリケーション処理モジュールは、通常の

アプリケーションプログラムに相当するものである。Flatland はアプリケーションプログラマが、システムの低レベル処理を意識することなく新しいアプリケーション処理モジュールを記述できるような API を提供している。

アプリケーション処理モジュールの基本的な動作は、ユーザによる入力ストロークを受け取って、それを処理してセグメントによって管理されているストロークの集合へ加えたり、すでにあるストロークに変更を加えたりするというものである。また、消去ジェスチャーを受け取って、対応するストロークをセグメントから取り除くといった処理も行う。

アプリケーション処理モジュールは、ストローク自体を管理することはないが、ストローク同士の関係や構造といった付加的な情報を管理する(図 13)。例えば、リスト管理アプリケーションは複数のストロークをまとめて一つの項目とするグループ構造を持つ。また地図描画アプリケーションでは道を表す内部表現と画面に表示されるストロークとの関係を表す内部表現を持つ。この付加的な構造情報は、アプリケーションが取り外された時に消滅し、再付与された時点で再構成される。



図13: アプリケーション固有の内部構造

以下に各アプリケーション処理モジュールの実装について具体的に説明する。

手書き描画: このアプリケーション処理モジュールは、ユーザが明示的にアプリケーションを指定していない時に、デフォルトとして自動的に付与されるものである。動作は、入力ストロークをそのままセグメントに加え、また消去ジェスチャーに対しては最寄りのストロークをセグメントから取り除くといった単純なものである。また実装上、他のアプリケーション処理モジュールのプロトタイプとして使用される。

リスト管理: ストロークの集合である項目の列を内部表現として管理する。また各項目の左端にあるチェックボックスは、このモジュールによって生成されセグメントに追加されたストロークである。

地図描画: このアプリケーション処理モジュールは、道路と交差点からなるネットワーク構造を内部表現として管理する。各道路は画面に表示される2本のストロークへのポインタをもち、各交差点はそこへつながる道路へのポインタを持つ。新しいストロークが入ってきたり、消去ジェスチャーが描かれた場合には、ネットワーク構造を適切に変更し、同時にセグメント内の対応するストロークを変更する。

幾何図形描画: 整形エンジン[12]が入力ストロークとセグメントに含まれるストロークの集合を受け取り、整形結果の候補を返す。整形結果の候補は、ピンク色のストロークとしてセグメントに加えられる。短いストロークが描かれると(タップ操作に相当)、ピンクの候補が削除され、対応する候補が黒い確定したストロークとしてセグメントに加えられる。

3次元描画: ポリゴンメッシュが内部表現として管理される。形状変形や回転などによって見えかたが変る度に、すべてのストロークがセグメントから削除され、新しい見えかたに対応するストロークが追加される。

計算機: 長い横線が引かれた時点で、すでに描かれているストロークの集合を文字認識エンジンに渡して数字に直し計算を実行する。計算結果は、手書き風のストロークに変形されセグメントにつけ加えられる。これらのシステムによって生成された数字も手書きで描かれた数字と同様に認識できるため、さらに後の計算で利用することもできる。

6.4. アプリケーション処理モジュールの再付与

すでに述べたように、付加的な構造情報は、アプリケーションが取り外された時に消滅し、再付与された時点で再構成される。この節では、このようなアプリケーション固有の構造情報の再構成について説明する。当初の実装では、これらの内部構造は、処理モジュールが取り外された時点でセグメントに埋め込む方針であったが、その後でのセグメント分割統合処理が複雑化するため、最終的には、内部構造を個々のストロークに断片を「埋め込む」方針を取った。これによって、アプリケーションが取り除かれた後でのストロークの消去やセグメントの分割

統合の際には、アプリケーション固有の内部構造に関して一切触れなくて済む。当該アプリケーションが再付与されると、そのセグメントに含まれているストロークに埋め込まれている内部構造の断片を集めて一貫して内部構造が再構成される。ただし、この埋め込みと再構成に関しては、各アプリケーションごとに注意深く設計する必要があり、実装の手間が大きくなる点が問題となる。

図14に、リスト管理アプリケーションにおける再付与の例を示す。3)の時点での画面表示は大きく崩れているが、個々のストロークにグループ構造が埋め込まれているため、4)で再付与されると正しいリスト構造が復元される。

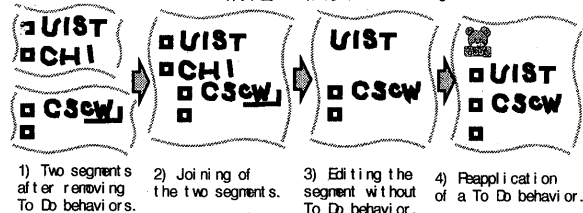


図14: リスト管理アプリケーションの再付与。

図15は、地図描画の場合の例である。同様に、3)で大きく崩れた画面表示が、4)で再構成されている様子が分かる。

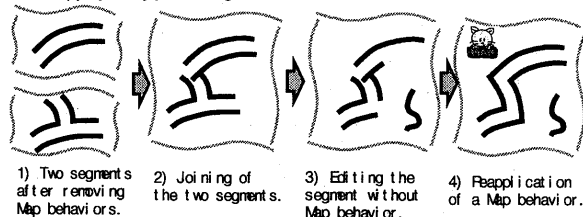


図15: 地図描画アプリケーションの再付与

7. 自動的な履歴管理

Flatlandでは、ユーザが明示的に保存を行うのではなく、各時点でのセグメント状態を自動的にデータベースに保存し、後から時間情報や位置情報などを元に検索する方法を取っている。

7.1. コマンドオブジェクトによる履歴

操作履歴を保存する基礎的な機構として、コマンドオブジェクトのモデル[7]に基づく無限Undo/Redo機構を採用している。ユーザの操作の系列はUndo可能なコマンドオブジェクトの列として保存される。この列をたどっていくことで任意の時点でのセグメントの状態を復元することができる[4,5]。スライダーによる時間操作はこの機構を直接利用したものである。

コマンドオブジェクトは通常、ユーザの操作一つに対して一つ割り当てられる。しかし、そのような方法では、コマンドオブジェクト内のコマンドを実行するたびに、すべてのイベント処理を再現しなければならず、オーバーヘッドが大きい。本システムでは、ユーザの操作をそのまま記録するのではなく、その結果としてアプリケーション処理モジュールがセグメントに対して起こした処理をコマンドとして記録する。この結果、セグメントの外見だけを再生したい場合には、アプリケーション処理モジュールを意識することなく、このセグメントに対する操作だけを再生していくだけで十分となる。このセグメントに対する操作に加えて、アプリケーション処理モジュール内部にある内部データ構造の対する変更も同様にして記録され、外見だけでなく内部構造をすべて復元することも可能になっている。

以上のように、履歴そのものはユーザの操作よりも細かいレベルでの系列となっている。しかし、実際にユーザがUndo/Redoを行ったり、時系列をたどる場合には、ユーザの操作毎にまとめておく必要がある。そのため、ユーザの操作開始・終了毎に、履歴内にトランザクションの開始および終了マークを挿入している(図16)。

```

121 OpenTransaction
122 BehaviorSpecificCommand
    (Map, addstreet, street#12a)
123 AddPaintedStrokeCommand (stroke#23a1)
124 AddPaintedStrokeCommand (stroke#23a2)
125 CloseTransaction
  
```

図16: トランザクションの例

7.2. 局所的履歴と全体的履歴

ユーザの側からみた場合、履歴には特定のセグメントに注目した場合の局所的なもの、画面全体に注目した場合の全体的なもの2種類がある(図17)。本実装では、履歴はセグメント毎に管理されており、全体的履歴はセグメント毎の履歴を自動的につなぎあわせることによって動的に再構成される。

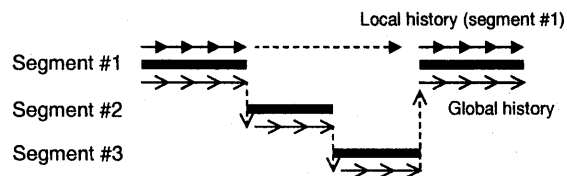


図17: 局所的な履歴と全体的な履歴

7.3. コンテキスト情報による検索

履歴の永続性を保存し、コンテキスト情報による検索を実現するために、本システムでは Presto データベース[3]を利用している。Presto はあらゆる Java オブジェクトをタグづけして保存、検索できる機能を持つ。システムは、定期的にセグメントの状態をそのセグメントが生成されてからの履歴としてコンテキスト情報と共に Presto へ保存する。検索時には、検索キーと一致するコンテキスト情報を持つセグメントの履歴が取り出され、その履歴を実行することでセグメントの状態が再構成される。検索結果を表示するだけであれば、履歴のうちアプリケーション処理モジュール固有のデータ構造書き換えに関するものは無視して、セグメントのストローク書き換えに関するものだけを扱えばよいので、再生がアプリケーション処理モジュールを経由せず高速に行える。検索の結果見つかった過去のセグメントで作業を続けたい場合には、内部構造書き換えも含めた完全な履歴の再実行が行われる。

8. まとめ

手書きストロークに基づく電子白板システム、およびその実装について紹介した。本システムは、従来のデスクトップシステムを補い、より自由に創造的な活動を支援することを目的としている。今後の発展の一つとしては、本アーキテクチャを他のペンベースのシステムに適用することなどが考えられる。

参考文献

1. Bederson, B.B., Hollan, J.F., Pad++: A Zooming graphical interface for exploring alternate interface physics, *UIST'94*.
2. Bier, E.A., Stone, M.C., Pier, K., Buxton, W., DeRose, T., Toolglass and magic lenses: The see-through interface, *SIGGRAPH'93*.
3. Dourish, P., Edwards, W.K., LaMarca, A., Salisbury, M., Using Properties for Uniform Interaction in the Presto Document System. *UIST'99*.
4. Edwards, W.K., Flexible Conflict Detection and Management in Collaborative Applications. *UIST'97*.
5. Edwards, W.K., Mynatt, E.D., Timewarp: Techniques for Autonomous Collaboration. *CHI'97*.
6. Forsberg, A., Dieterich, M., Zeleznik, R.C., The Music Notepad, *UIST'98*.
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing, 1995. Reading, Mass.
8. Geissler, J., Shuffle, throw or take it! Working efficiently with an interactive wall, *CHI'98*.
9. Gross, M.D., Do, E.Y., Ambiguous intentions: a paper-like interface for creative design, *UIST'96*.
10. Kramer, A., Translucent Patches—dissolving windows, *UIST'94*.
11. Kramer, A., Dynamic Interpretations in Translucent Patches, Representation-Based Applications, *AVI'96*.
12. Igarashi, T., Matsuoka, S., Kawachiya, S., Tanaka, H., Pegasus: A Drawing System for Rapid Geometric Design, *CHI'98* summary, pp.24-25.
13. Igarashi, T., Matsuoka, S., Tanaka, H., Teddy: A Sketching Interface for 3D Freeform Design, *SIGGRAPH'99*.
14. Landay, J.A., Myers, B.A., Interactive sketching for the early stage of interface design, *CHI'95*.
15. Moran, T.P., Chu, P., van Melle, W., Kurtenbach, G., Implicit structures for pen-based systems within a freeform interaction paradigm, *CHI'95*.
16. Mynatt, E.D., The writing on the wall, *INTERACT'99*.
17. Mynatt, E.D., Igarashi, T., Edwards, W.K., LaMarca, A., Flatland: New Dimensions in Office Whiteboards, *CHI'99*.
18. Perlin, K., Quikwriting: Continuous Stylus-based Text Entry, *UIST'98*.
19. Prderson, E., McCall, K., Moran, T.P., Halasz, F., Tivoli: An electronic whiteboard for informal workgroup meetings, *INTERCHI'93*, pp.391-399.
20. Rekimoto, J., A Multiple Device Approach for Supporting Whiteboard-based Interactions, *CHI'98*.
21. Rekimoto, J., Time-Machine Computing: A Time-centric Approach for the Information Environment, *UIST'99*.
22. Tapia, M.A., Kurtenbach, G., Some Design Refinements and Principles on the Appearance and Behavior of Marking Menus, *UIST'95*, pp.189-195.
23. Zeleznik, R.C., Herndon, K.P., Hughes, J.F., SKETCH: An interface for sketching 3D scenes. *SIGGRAPH'96*.