文法を用いた手書きストローク認識のための枠組み

志築 文太郎　　田中 二郎　　飯塚 和久

筑波大学　電子・情報工学系
〒 305-8573　茨城県つくば市天王台 1-1-1
{shizuki,jiro,iizuka}@iplab.is.tsukuba.ac.jp

図形言語を手書き入力するためのエディタを開発することを目的として、手書きストローク認識のための文法を用いた枠組みを提案する。本枠組みは、文法を、開発対象のエディタが扱う図形言語のシンタックスを記述する目的だけでなく、手書きストロークの解釈を行うためのルールを記述することにも用いる。この結果、手書きストロークそのものの形状だけではなく、既に描かれている図形との位置関係、および他の手書きストロークとの位置関係に応じて、手書きストロークを解釈するルールを記述することが可能である。我々は、本枠組みを「恵比寿」を拡張することによって実現した。

A Unified Framework for Interpreting Handwritten Strokes using Grammars

Buntarou Shizuki　　Jiro Tanaka　　Kazuhisa Iizuka

Institute of Information Sciences and Electronics, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573 JAPAN

To support rapid development of pen-based structured diagram editors, we propose a unified framework to describing such editors. The framework uses grammars to describe the context, i.e. the positional relationship between handwritten strokes and other objects, that can be used to interpret ambiguous results of pattern matching, as well as to describe the syntax of the target diagrams. We have implemented the framework by extending Eviss, our visual system which supports rapid prototyping of structured diagram editors.

1　はじめに

図形言語が用いられる機会は多い。UML を用いたソフトウェア設計を行う際にはクラス図、シーケンス図、状態遷移図等の図形言語が用いられる。また、データ構造を表現する際にも様々な図形言語が用いられる。さらに、組織図は一般においてもよく用いられる図形言語である。したがって、これらの図形言語をペン型デバイスを用いてタブレット型コンピュータや電子ホワイトボードから自由に入力することができれば好都合である。

しかしながら、手書きされた図形言語を認識す

るシステムを構築するには、手書きストロークが持つ曖昧性を除去することが課題となる。なぜならば、個々の手書きストロークの形状のみからはその意味を解釈することが不可能な場合が多いからである。

ただし、手書きストロークが描かれたコンテキストを参照することによって、解釈が可能となる場合がある。例えば、二分木という図形言語を描くという前提があり、かつ円状の手書きストロークの内部にさらに別の手書きストロークが与えられた場合、後者の手書きストロークはノードのラベルであると解釈することができ、かつ文字を表している可能性が高い。

このような認識を行い、さらにインタラクティブな編集機能を提供する図形言語エディタを実装するには、以下に挙げる機能を実装することが求められる。

手書きストロークの解釈機能 入力された手書きストロークを、その形状、および既に描かれている図形あるいは手書きストロークとの位置関係に応じて解釈し図形として確定する機能

シンタックスに基づいた解析機能 入力された図形が図形言語のシンタックスに従っているか否かを解析する空間解析機能

後者に対しては、これまでにも、空間解析機能を文法記述から生成するシステムとしてSPARGEN[1]、VLCC[2]、Penguins[3] 等があった。また、図形言語エディタを文法記述から生成するシステム DiaGen[4] もある。いずれも、与えられた文法記述から空間解析機能を自動生成することによって、システム開発を容易にする方式をとっている。しかし、両機能を統一的に提供する試みは見られない。

本稿では、この両機能を持ったインタラクティブな図形言語エディタを容易に開発するための枠組みを提案する。提案枠組みは、文法を開発対象のエディタが扱う図形言語のシンタックスを記述する目的だけでなく、手書きストロークの解釈を行うルールを記述することにも用いる。文法を用いるため、その形状だけではなく、既に描かれている図形あるいは他の手書きストロークとの位置関係に応じて、手書きストロークを解釈するルールを記述することが可能となっている。

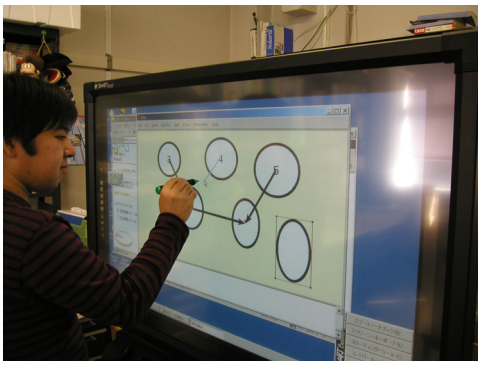

図 1: 提案手法を用いて実現した二分木エディタ

我々は、図形言語エディタのラピッドプロトタイピングを支援するシステム「恵比寿」[5] を拡張することによって、提案枠組みを実装した。本実装を用いて実現した、二分木エディタを図 1 に示す。この二分木エディタは、手書き入力された二分木の構成要素 (ノード、エッジ、ノードのラベル) を自動的に整形する。

以下では、まず 2 節において、本枠組み、および今回、その枠組みを実現するために用いた文法を説明する。その後、その枠組みにおいて、手書きストロークをコンテキストを用いて解釈をするルールの記述例を 3 節に示し、実装について 4 節に述べる。

2 文法を使う手書きストローク認識

提案枠組みにおける、手書きストロークの認識方法を図 2 に示す。「パターン認識」は、手書きストロークの形状に基づいた認識を行い、n-best リストを出力する。「文法記述に基づいた解析」は、文法記述として与えられる解析規則に従って n-best リストの要素を選別することにより、手書きストロークの解釈を決定する。

例えば、図 2 のような円形の手書きストロークに対して、パターン認識は、円の尤度は 0.90、数字の 0 の尤度は 0.90、数字の 6 の尤度は 0.78、といったリストを出力する。文法記述に基づいた解析は、場合によっては円、あるいは数字の 0 であると解釈を決定する。

なお、文法記述に基づいた解析は、1 個の手書きストロークをパターン認識した結果のみから、その手書きストロークに対する解釈を決定するの

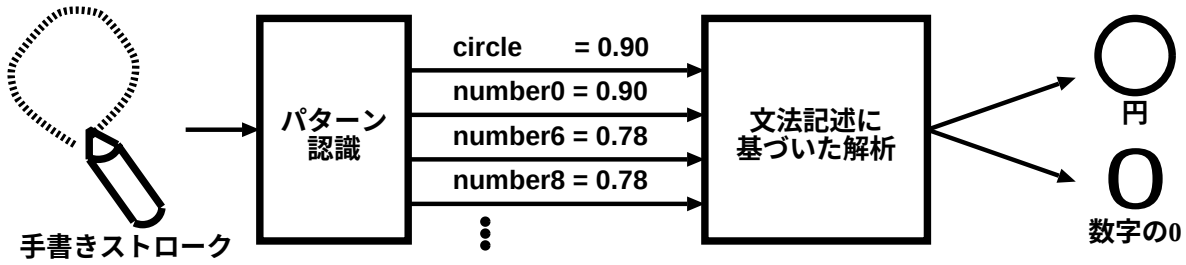

図 2: コンテキストを用いた認識結果の選別

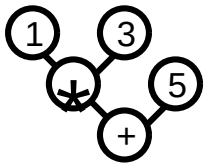

図 3: 図形言語の例「二分木」

ではない。重要な点は、他の図形との位置関係あるいは他の手書きストロークのパターン認識の結果などを考慮した解釈を可能とする点である。

本節では、今回用いた文法、および以降の例として用いる二分木エディタを 2.1 節に示し、手書きストロークのパターン認識の結果を文法記述において参照するために導入する特殊なトークンについて 2.2 節に示す。

2.1 文法を用いた図形言語エディタの記述

今回は CMG[6] を拡張した文法を用いた。以下に、文法記述を構成するルールを示す。

$$P \quad ::= \quad P_1, \cdots, P_n \\ where\ C\ with\ Attr\ and\ Action$$

これは、トークン $P_i (i = 1, \cdots, n)$ が存在し、かつ P_i の属性が制約条件 C を満たす場合、P_i を P に還元し、P の属性を $Attr$ によって定めることを示す。$Action$ は、我々が CMG に対して拡張した部分であり、還元が行われる際に行うべき処理内容である。

例として、二分木 (図 3) を定義する文法記述を以下に示す。

```
# Rule 1
Node::=C:Circle,T:Text where (
    close(C.mid,T.mid)
) {
    cp = C.mid;
    r  = C.radius;
} { }

# Rule 2
Node::=N1:Node,N2:Node,N3:Node,
        L1:Line,L2:Line where(
    inCircle (L1. start , N1.cp,N1.r) &&
    inCircle (L1.end,N2.cp,N2.r) &&
    inCircle (L2. start , N3.cp,N3.r) &&
    inCircle (L2.end,N2.cp,N2.r)
) {
    cp = N2.cp;
    r  = N2.r;
} { }
```

プログラマはこの 2 個のルールからなる文法記述、および close()、inCircle() というユーザ定義関数を空間解析器に与えることによって、二分木エディタを定義することが可能である。

第 1 ルールはテキストをラベルとして持つ円形のノードを定義する。このルールは、円 C とテキスト T が存在し、それぞれの中心が近い場合には、C と T をノードに還元し、ノードに 2 個の属性を与える。属性 cp は接続点、すなわちエッジの端点となり得る点を表す。また、属性 r は、点 cp から半径 r 内にエッジの端点が存在する場合にエッジを接続するために用いる。cp および r の値は、それぞれ C の中心および半径とする。close(P1,P2) は、2 点 P1 および P2 の距離がある閾値未満であるか否かを真偽値として返す。

第 2 ルールは 1 個の節ノード、および 2 個の葉ノードを組み合わせて、再帰的にノードを定義する。このルールは、3 個のノード N1、N2、N3、および 2 本の線分 L1、L2 が存在し、L1 が N1 と N2 の接続点付近を結び、かつ L2 が N3 と N2 の接続点を結ぶ場合には、これらをノードに還元し、2 個の属性を与える。属性 cp および r は節に相当するノード N2 のそれを継承する。inCircle(P,C,R)

表 1: Gesture トークンの属性

属性名	属性値
pattern	認識結果の n-best リスト
start	始点の座標
end	終点の座標
bound	バウンディングボックス
length	長さ
time	入力の所要時間

は点 P が点 C を中心とした半径 R の円の内部に存在するか否かを真偽値として返す。

2.2 パターンマッチの結果を保持する Gesture トークン

手書ストロークから直接得られる情報に加えて、認識エンジンから得られる認識結果を属性として保持する特殊なトークンを導入する。これを Gesture トークンと呼ぶことにする。手書きストロークがひとつ描かれる度に Gesture トークンをひとつ生成し空間解析器に渡すことによって、手書ストロークに関する情報を、円やテキストなどの通常のトークンと同様に文法記述から参照することが可能となる。

表 1 に Gesture トークンが保持する属性を示す。属性 pattern は認識エンジンが手書ストロークを解析した認識結果である。pattern 以外の属性は手書ストロークから直接得られる情報である。

2.3 Gesture トークンを用いたルールの記述例

Gesture トークンを用いたルールの記述法、およびそのルールによる手書きストロークの処理を説明するために、図 4 のように、円形の手書きストロークが描かれた場合に整形された円を描くルールを以下に示す。

```
_CreateCircle::=G:Gesture where(
    findGesture(G,"circle ",0.5)
) {} {
    createCircle(G.bound);
    delete(G);
}
```

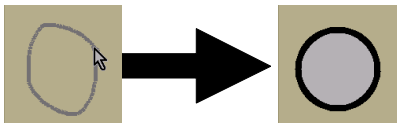

図 4: 手書きストロークの形状のみを用いた認識例

このルールは、Gesture トークン G が存在し、その形状が 0.5 以上の尤度で円形である場合には、G を_CreateCircle に還元する。制約条件の記述に用いている findGesture(T,N,P) は、Gesture トークン T がパターン N を pattern の中に P 以上の尤度で持っているか否かを真偽値で返すユーザ定義関数である。このルールが適用されると、Action として createCircle(G) を実行することにより、G のバウンディングボックスに内接する円を生成する。

なお、このルールはトークン G を Action において消去している。この操作によって、トークン G がキャンバスから消去されるとともに、一旦は生成された_CreateCircle トークンも存在しなくなる。なぜならば、この_CreateCircle トークンが存在するために必要な条件が成立しなくなるからである。この機構は、整形された図形を描いた後には不必要となる手書きストロークを画面から消去する操作と、ルールを記述する便宜上定義したトークン (この場合_CreateCircle トークン) を取り除く操作を同時に実現している。

3 コンテキストに応じた手書きストロークの認識

図形言語エディタにおいてユーザからある手書きストロークが与えられた時、対応して生成される Gesture トークン G のコンテキストには以下のような分類を与えることが可能である。

(1) 入力対象となっている図形言語のシンタックス

(2) G の周囲に存在する、図形言語を構成することが既に判明しているトークン

(3) G の周囲に存在する、図形言語を構成するか否か未だ不明のトークン

(1) のみから手書きストロークの解釈を決定するルールは、2.3 節に示した、二分木の構成要素

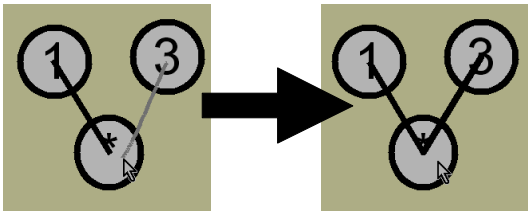

図 5: ノード間に描かれた直線状の手書きストロークの解釈

である円を生成するルールのように、Gesture トークンのみを存在条件としたルールを記述すれば良い。(2) については、既に存在する図形言語のトークンとの位置関係に応じて G を解釈するルールを記述する。(3) については、他の Gesture トークンとの位置関係に応じて G を解釈するルールを記述する。以下では、(2) および (3) のルールの記述例を示す。

3.1 既に描かれている図形との位置関係に応じた解釈

既に描かれている図形との位置関係に応じて手書きストロークの解釈を決定するためには、Gesture トークン、および図形言語を構成するトークンが存在することを条件としたルールを記述する。

例として、図 5 に示すように、二分木のノード間に描かれた直線状の手書きストロークをエッジとして解釈するためのルールを以下に示す。

```
_CreateEdge::=G:Gesture,
              N1:Node,N2:Node where(
    findGesture(G,"line",0.3) &&
    inCircle(G.start,N1.cp,N1.r) &&
    inCircle(G.end,N2.cp,N2.r)
){}{
    createLine(N1.cp,N2.cp);
    delete(G);
}
```

このルールは Gesture トークン G、および 2 個のノード N1、N2 が存在し、G の形状が直線状であり、かつ G が N1 の接続点付近から N2 の接続点付近に描かれている場合に、N1 と N2 の接続点を結ぶ線分を作成し、さらに G を消去する。

このルールが適用された結果として生成される線分は 2.1 節に挙げた第 2 ルールにおいて処理され、結果として 3 個のノードがつながったノード

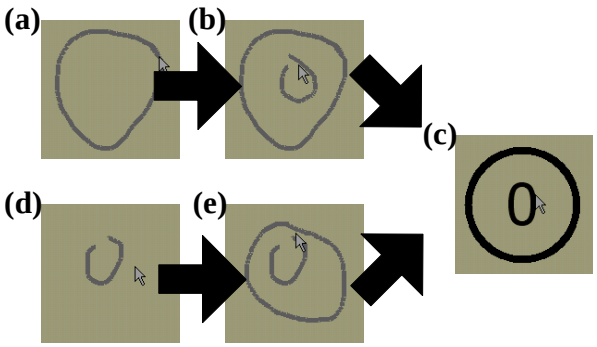

図 6: 円状の手書きストロークとその内部の手書きストロークの解釈

となる。

3.2 他の手書きストロークとの位置関係に応じた解釈

他の手書きストロークとの位置関係に応じて手書きストロークを解釈するためには、複数の Gesture トークンを存在条件として、その解釈を決定するルールを記述する。

例えば、図 6 のように、円状の手書きストロークの内部にさらに別の手書きストロークが存在する場合に、その 2 個の手書ストロークを、文字をラベルとして持った円形のノードとして解釈するためのルールを以下に示す。

```
_CreateNode::=G1:Gesture,G2:Gesture where(
    findGesture(G1,"circle",0.5) &&
    insideOf(G2.bound,G1.bound)
){}{
    C = createCircle(G1.bound);
    createText(getString(G2),C.mid);
    delete(G1);
    delete(G2);
}
```

上のルールは、2 個の手書きストローク G1、G2 が存在し、G1 が円状であり、かつ G2 が G1 の内部に存在する場合、G1 のバウンディングボックスに内接する円 C を作成し、さらにテキストをその円の中心に配置する。ここで、getString(G) は、G の認識結果のうち最も高い尤度を示した文字のパターンを文字列として返す関数であり、ラベルとして与える文字列を決めるために用いている。

なお、このルールは、2 個の手書きストロークの順序に依らない認識を可能とし、手書きストロー

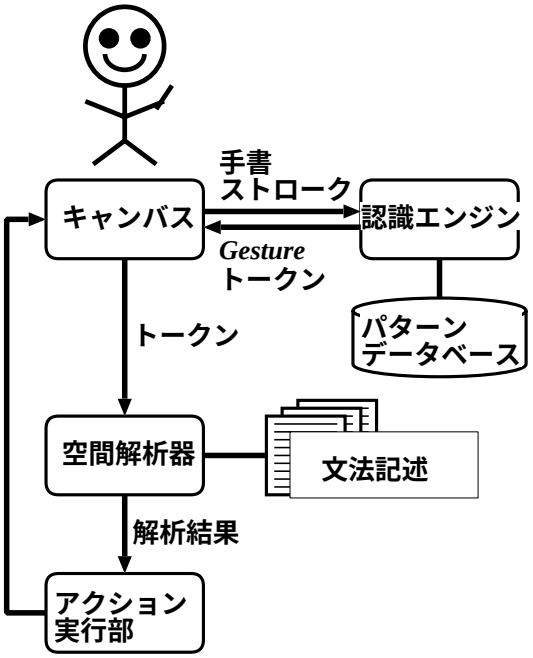

図 7: システム構成

クを与えるユーザにとって順序を強制しない自然な入力を可能とする。具体的には、外側のストローク (図 6a) を与えた後に内側のストローク (図 6b) を与えた場合に、上のルールによってその 2 個のストロークは図 6c のように変換されるが、順序を逆にして外側のストローク (図 6d) の後に内側のストローク (図 6e) を与えた場合にも、上のルールによって図 6c のように変換される。

4 実装

提案手法を実現したシステムの構成を図 7 に示す。各モジュールの動作は以下の通りである。

キャンバス キャンバスは、ユーザによって手書ストロークが描かれる度に、認識エンジンに処理を依頼し Gesture トークンを得る。また、新しいトークンが出現する、既存のトークンが消滅する、既存のトークンの属性の値が変わった等の変更があった場合には、再解析のために空間解析器に処理を依頼する。

認識エンジン 認識エンジンは、新規に入力された手書ストロークを、パターンデータベースに登録されているパターンデータと照合する。次に、得られた n-best リストと手書ストロークそのものの情報を束ねて Gesture トークンを生成し、キャンバスに返す。

空間解析器 空間解析器は、キャンバスに存在するトークンに対して、適用可能なルールを探索する。適用可能なルールを 1 個見つけたら、そのルールを適用する。ルールを適用した結果、Action を実行する必要がある場合には、アクション実行部に Action の実行を依頼する。探索の結果、適用可能なルールが見つからない場合には、そこで処理を終了する。未処理のトークンはそのままキャンバスに残す。これによって、他の手書きストロークとの位置関係に応じた手書きストロークの解釈 (3.2 節) が行えるようになっている。

アクション実行部 アクション実行部は、入力された解析結果と、ルールに記述された Action の内容を受け取り、その実行を行う。

我々は、恵比寿 [5] に認識エンジンを追加することによって、提案枠組みの実装を行った。恵比寿は、図形言語エディタのラピッドプロトタイピングを支援するシステムであり、CMG に基づいた空間解析器を有する。現実装では、SATIN[7] とともに配布されている認識エンジンを流用して、認識エンジンを実装した。また、システムの実装は Tcl/Tk、C、および Java を用いて行っている。

5 関連研究

手書きストロークをコンテキストに応じて処理する枠組みはこれまでにも幾つか提案されてきた。

Electronic Cocktail Napkin[8] はルールによって、手書きストロークを認識し、図形言語の構造に応じた編集を可能としているシステムである。しかし、ジェスチャについては幾つかの組み込みのものを提供するのみである。我々の提案手法は、構造的な編集を可能とするだけではなく、新たなジェスチャを定義することも可能である。

手書きストロークを使ったインタラクションを実装するためのツールキットである Artkit[9] は、「sensitive regions」を提供する。個々の sensitive region は、手書きストロークの形状、およびその形状に対する処理方法の組をセットとして持った画面

領域を定義するデータ構造である。また、Translucent patches[10] が示している「patch」は、patch が覆う領域に入力された手書きストロークを解釈し処理する方法を提供するコンテキストを提供している。また、Flatland[11] はユーザが入力する手書きストロークを解釈し処理するコンテキストを「behavior」として提供する。behavior を既に描かれた手書きストロークに対しても取り替えて適用し、その手書きストロークの解釈の方法を変えることが可能となっている。本稿で示した枠組みは、これらのシステムにおけるコンテキストを文法を用いて定義できるようにしていると位置づけられる。

なお、DiaGen[4]、および Electronic Cocktail Napkin は言語指向編集の機能を提供している。すなわち、入力された図形言語を、そのシンタックスを保った状態のまま、一部を移動する、大きさを変える等の編集を行えるようにする機能を提供する。本枠組みは、この機能を取り入れたシステムの実現も可能とする。我々の実装では、制約解消系を利用することによって実現している。例えば、二分木の例において、一度円およびテキストが一つのノードとして認識された後は、それぞれの中心は常に同じ値を保つようにしている。これによって、図形言語エディタのユーザが円の位置を変更した場合にも、この操作に伴ってテキストの位置も変わる。現在、制約解消系として SkyBlue[12] を用いている。

さらに、[13] に述べられている図形の整形機能の記述は、制約解消系の機能をルールの Action において用いることによって可能である。例えば、2 個のノード N1 および N3 を常に水平に並べるように配置する、という整形機能はそれぞれの cp の Y 座標を等しくする、という制約を与えれば良い。

6 議論

認識について 本枠組みは、他の図形認識のための手法、および文字認識のための手法を排除するものではなく、これらを組み合わせることによってさらに認識精度を高めるインタフェースを構築することが可能な枠組みである。すなわち、提案手法と、n-best 候補に基づく文字認識の絞り込みインタフェースとを併せて用いることによって、文字認識の絞り込みを行うインタフェースを自然に実現することが可能である。例えば、二分木エディタにおいて、円形の手書きストロークの内部に描かれた手書きストロークに対してのみ、認識エンジンが行った文字認識の n-best 候補をユーザに提示し、ユーザに明示的な選択操作をさせることによって絞り込みを行う。

ただし、現実装では、1 個の手書きストロークを 1 回だけ認識エンジンに処理させている。この方式は Unistroke[14] のような一筆書きで文字を入力する文字入力方式を採用することによって英文には対応することが可能であるが、複数ストロークからなる文字の入力機能を提供する際には問題となる。この問題は、枠組み自体は変更することなく、複数ストロークに対応した認識エンジンを用い、さらに複数個の Gesture トークンを明示的に認識エンジンに渡す Action を記述できるようシステムを拡張することによって解決することが可能である。この拡張によって、既存の手書きストロークに重ねて手書きストロークが描かれた場合に、重なる手書きストローク全てを認識エンジンに渡すルールを定義することが可能となる。

記述性について 現実装では文法記述の中に尤度を表す数値をパラメータとして直に記述する方法をとっている。したがって、パターンデータベースが持つパターンデータが変わると認識エンジンから得られる尤度の数字も変わるため、ルールのパラメータを変更する必要が生じる場合がある。記述性を高めるためにこの点を改善することは今後の課題である。

今回示した枠組みは、手書きストロークを解釈する方法、および解釈した結果行うべき処理のふたつをルールにひとまとめに定義することを可能としている。また、文法が提供する強力な空間解析機能を用いて手書きストロークを解釈するルールを記述できる点、およびひとつの記述言語を用いて図形言語のシンタックスおよび手書きストローク認識のルールを定義できる点が優れている。

7 まとめ

図形言語を手書き入力するためのエディタを開発することを目的として、手書きストローク認識

のための、文法を用いた枠組みを提案した。本枠組みは、文法を、開発対象のエディタが扱う図形言語のシンタックスを記述する目的だけでなく、手書きストロークの解釈を行うためのルールを記述することにも用いる。この実現のために、認識エンジンから得られる手書きストロークの認識結果を属性値として有する Gesture トークンを導入し、他の図形を表現するトークンと同じように文法において記述できるようにした。これによって、手書きストロークそのものの形状だけではなく、既に描かれている図形との位置関係、および他の手書きストロークとの位置関係に応じて、手書きストロークを解釈するルールを記述することが可能となった。

謝辞

高い実装能力をもって精力的に本システムを実現した山田英仁氏に深く感謝する。

参考文献

[1] Eric J. Golin and Tom Magliery. A compiler generator for visual langauges. In *Proceeding of the 1993 IEEE Symposium on Visual Langauges*, pp. 314–321. August 1993.

[2] Gennaro Costagliola, Genoveffa Tortora, Sergio Orefice, and Andrea De Lucia. Automatic generation of visual programming environments. *Computer*, Vol. 28, No. 3, pp. 56–66, March 1995.

[3] Sitt Sen Chok and Kim Marriott. Automatic construction of intelligent diagram editors. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pp. 185–194. November 1998.

[4] Mark Minas and Gerhard Viehstaedt. Diagen: A generator for diagram editors providing direct manipulation and execution of diagrams. In *Proceedings 11th IEEE International Symposium on Visual Languages*, pp. 203–210. September 1995.

[5] Akihiro Baba and Jiro Tanaka. Eviss: A visual system having a spatial parser generator. In *Proceedings of Third Asian Pacific Computer and Human Interaction*, pp. 158–164. July 1998.

[6] Kim Marriott. Constraint multiset grammars. In *Proceedings of 1994 IEEE Symposium on Visual Languages*, pp. 118–125. October 1994.

[7] Jason I. Hong and James A. Landay. SATIN: a toolkit for informal ink-based applications. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 63–72. November 2000.

[8] Mark D. Gross and Ellen Yi Luen Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pp. 183–192. November 1996.

[9] Tyson R. Henry, Scott E. Hudson, and Gary L. Newell. Integrating gesture and snapping into a user interface toolkit. In *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pp. 112–122. October 1990.

[10] Axel Kramer. Translucent patches. *Journal of Visual Languages and Computing*, Vol. 7, No. 1, pp. 57–77, March 1996.

[11] Takeo Igarashi, W. Keith Edwards, Anthony LaMarca, and Elizabeth D. Mynatt. An architecture for pen-based interaction on electronic whiteboards. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 68–75. May 2000.

[12] Michael Sannella. SkyBlue: a multi-way local propagation constraint solver for user interface construction. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pp. 137–146. November 1994.

[13] Sitt Sen Chok, Kim Marriott, and Tom Paton. Constraint-based diagram beautification. In *Proceedings of 1999 IEEE Symposium on Visual Languages*, pp. 12–19. September 1999.

[14] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 80–87. April 1993.