図形文法の定義と実行インタフェースを統合した空間解析器生成系 Viola

Viola: Spatial parser generator which unifies a definition and execution interface.

亀山 裕亮 † 志築 文太郎 † 田中 二郎 †
Hiroaki KAMEYAMA Buntarou SHIZUKI Jiro TANAKA

† 筑波大学システム情報工学研究科
Department of Computer Science, University of Tsukuba
{kame,shizuki,jiro}@iplab.is.tsukuba.ac.jp

我々は，ビジュアルシステムの図式表現を直接操作することにより図形文法を定義し，同時にビジュアルシステムを実行することが可能な空間解析器生成系 Viola を作成した．Viola では，図形文法の各要素を視覚的に表現し，それらの要素に対して直接操作を行うことで，図形文法の定義を行うことができる．また，図形言語の定義と生成したビジュアルシステムの実行を一貫した一つの図形エディタ上で行うため，ビジュアルシステムの実行結果を確認しながら文法の定義を行うことができる．Viola を用いることで，空間解析器生成系を用いたビジュアルシステムの開発作業を効率よく行うことが可能になった．

1 はじめに

近年，PDA やタブレット PC などの情報端末が普及し，そのような環境下で用いるビジュアルシステムの開発に対する要求は高まっている．ビジュアルシステムにおいて，図形が決められた規則のもとに組み合わされている図式を図形言語と呼ぶ．従来，ビジュアルシステムは個々の図形言語に特化した形で実現されてきた．すなわち，ビジュアルシステムを開発する際には，始めにそのシステム固有の図形言語の仕様を決定し，次にその図形言語を解析するためのビジュアルシステムを一から実現する，という手順を踏む．このような実現形態では，図形言語の仕様に変更が生じた場合，ビジュアルシステムをはじめから開発し直す必要がある．

これに対して，図形言語における図形間の規則を定義するための文法を図形文法と呼び，図形文法記述からビジュアルシステムのための空間解析器を自動的に生成する生成系を空間解析器生成系と呼ぶ．空間解析器とはビジュアルシステムのユーザが入力した長方形，円，線分，テキストなどの基本図形に対して，それらの図形間の空間的な構造の解析を行う解析器のことである．空間解析器生成系を用いることで，ビジュアルシステムに組み込んで利用することが可能な空間解析器を簡単に作成できる．また，図形言語の仕様に変更が生じた場合にも，文法記述を修正するだけで新しい空間解析器が得られるため，ビジュアルシステムの開発を効率的に行うことができる．

我々は，ビジュアルシステムの図式表現を直接操作することにより図形言語を定義し，同時にビジュアルシステムを実行することが可能な空間解析器生成系 Viola を作成した．Viola では，図形文法の各要素を視覚的に表現し，それらの要素に対して直接操作を行うことで，図形文法の定義を行うことができる．また，図形文法の定義と生成したビジュアルシステムの実行を一貫した一つの図形エディタ上で行うため，ビジュアルシステムの実行結果を確認しながら文法の定義を行うことができる．これにより，空間解析器生成系を用いたビジュアルシステムの開発作業を効率よく行うことが可能である．

2 従来の空間解析器生成系の問題点

空間解析器生成系とは，図形言語における図形間の規則を図形文法を用いて記述することで，ビジュアルシステムのための空間解析器を記述から自動的に生成するシステムである．これまでに空間解析器として SPARGEN[6]，VLCC[4]，Penguins[3], 恵比寿 [1, 2] などが提案されている．

SPARGEN，VLCC，Penguins では，ユーザは生成された空間解析器をシステムに組み込むことで，定義した図形言語を扱うビジュアルシステムを簡単に作成することができる．

恵比寿は空間解析器生成系を備えたビジュアルシ

ステムであり，図形言語の定義だけではなく，定義に基づいた図形の解析も同じシステムで行うことができる．

しかし，従来の空間解析器を用いてビジュアルシステムを作成する際に以下にあげる問題があった．

文法記述を定義する際の問題 図形文法の記述はテキストを用いて行われていたが，図形言語の文法記述は，図形言語を構成する図形同士が満たすべき制約 (含む，接する，中央に存在する等の位置関係や色など) の 2 次元的な構造を記述した規則の集合である．そのような 2 次元の構造をテキストのみを用いて記述する場合，複雑な文法を記述する際に規則全体を把握し理解することや，矛盾している規則を発見することが困難になるといった問題が生じる．

図形の解析を行なう際の問題 従来の空間解析器生成系では図形言語の定義だけを行い，図形言語の解析を行なうビジュアルシステムは生成された空間解析器を用いて別に実現する必要があった．そのため，実行時に問題が見つかった場合には，定義を正しく修正した後で生成系により修正された空間解析器を生成し，生成された空間解析器をまたシステムに組み込むという作業が必要になってしまう．図形言語の定義を行う際には全体を一度に定義するのではなく，必要な部分から順に定義することで，文法の正しさを検査するという作業を行う．従来の空間解析器生成系では，図形言語の定義とビジュアルシステムの実行が別のシステムとして提供されているため，この作業手順が阻害されるという問題があった．

3 空間解析器生成系 Viola

我々は従来の空間解析器の問題点を解決し，以下の特徴を持つ空間解析器生成系 Viola を作成した．Viola の動作画面を図 1 に示す．Viola の実装には Java(j2sdk 1.4.1_01) を用いた．

- 図形言語の図式表現を編集するという例示操作により図形文法の定義を行うことができる．また，成立している制約についてはシステムにより画面上に制約をインタラクティブに図示することにより，現在定義を行っている制約を直感的に理解することが可能である．
- 図形エディタに空間解析器生成系の機能の統合

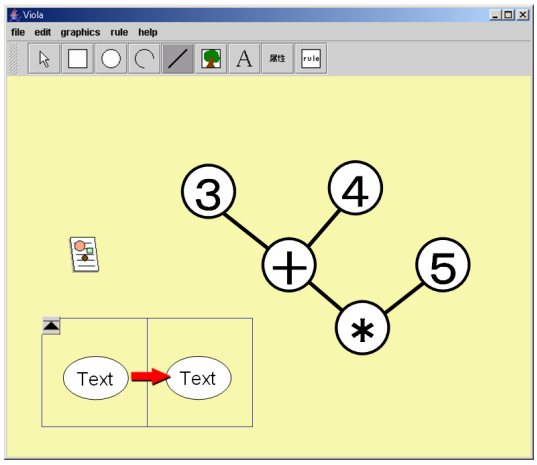


図 1: 空間解析器生成系 Viola

することで，図形エディタ上で図形言語の定義を確認しながらビジュアルシステムの実行を行うことができる．

Viola は図形言語の定義とビジュアルシステムの実行の両方に用いられる図形エディタを備えている．図形エディタで扱うことのできる図形の種類は長方形，楕円，円弧，直線，画像，テキスト文字列があり，画面上部にある各ボタンで描画する図形を選択することができる．また，マウスの右ボタンを使用することで手書きストロークも入力することができる．線の色や太さ，フォントなどの属性についてはメニューから変更することができる．また一般の図形エディタと同様に，描いた図形に対して移動，コピー，削除といった操作を行うことも可能である．Viola では図形文法として拡張 CMG[2] を用いている．拡張 CMG は Marriott らにより提案された CMG[10] にアクションと呼ばれる図形の書き換え規則を追加した図形文法である．

3.1 図式表現を用いたグラフィカルな定義

拡張 CMG の文法記述は構成要素，制約，属性，アクションにより構成されている．Viola では，ユーザは以下の手順により図形言語の規則をグラフィカルに定義することが可能である．

規則の定義範囲の指定 まず，規則を定義する範囲を指定する．ツールパネルにある rule と書かれたボタンを選択し，図 2 のように画面上に範囲を示す枠を描画する．この枠の中で図式を編集することにより規則の定義が行える．

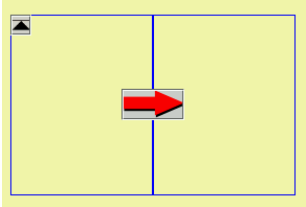


図 2: 規則の定義範囲

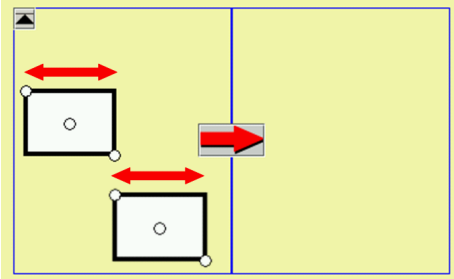


図 3: 制約の図示

構成要素の定義 規則により解析される図形要素を構成要素と呼ぶ．構成要素については，左側の枠の中に直接図形を描くことにより定義を行う．

制約の定義 規則が適用されるために，構成要素となる図形要素が満たさなければならない条件を制約と呼ぶ．制約は，左側の枠の中に描いた構成要素の図形を操作し，満たさなければなら条件を例示することにより定義を行う．この時，2 点の座標が一致している，幅が同じであるなどといった制約が図 3 のように画面上に図示される．

アクションの定義 図形の削除や移動など，規則が適用された時に実行される動作をアクションと呼ぶ．構成要素及び制約の定義を終えた後，中央にある矢印をクリックすることで，構成要素として描いた図形が右側の画面に複製される．複製された図形を編集し，アクションが実行された後の図式を作成することで，アクションの定義を行うことができる．

3.2 ビジュアルシステムの実行

ビジュアルシステムの実行を行なうには，ユーザは前述の手順で図形言語の定義を行なった後，画面上に解析したい図を描く．描かれた図形に対し Viola がインクリメンタルに解析を行い，図形間に制約を課す．解析が行なわれた後では，図形を編集しても制約関係が保持される．Viola では制約解消系に，幾

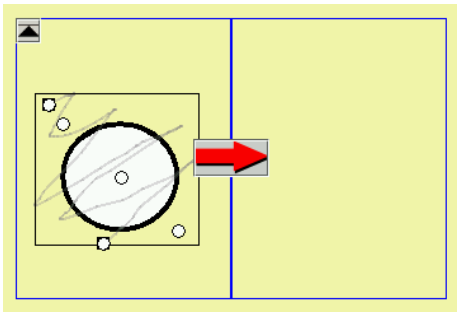


図 4: 円を削除するジェスチャーの定義

何制約を扱うことができる Chorus[9] を用いている．

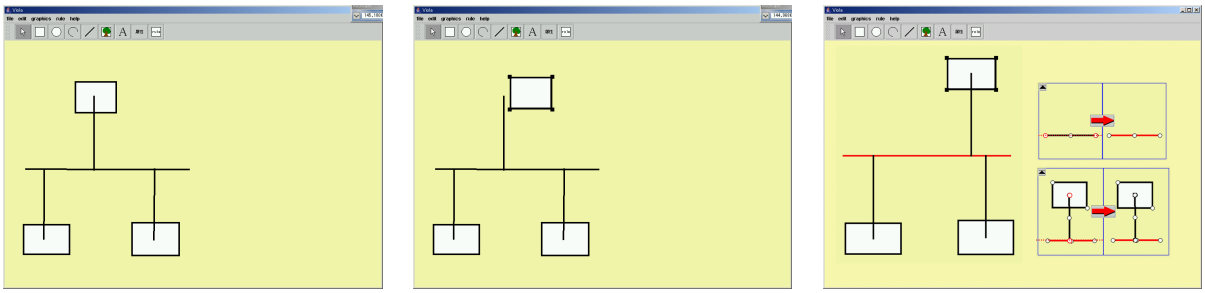
3.3 手書きストロークによるジェスチャの定義

PDA やタブレット PC などではジェスチャーと呼ばれる手書きストロークが用いられることが多い．ジェスチャーとはペンやマウスを用いて特定の軌跡を描くことにより，図形の削除や追加などの操作を行う機能のことである．Viola では手書きストロークを図形と同様に扱うことにより，ジェスチャーを定義することができる．例えば，円の上でギザギザの手書きストロークを描いた時に円を削除するというジェスチャを定義する場合，まず定義領域に円を入力した後，その上にギザギザの手書きストロークを入力する．その後で定義領域の右側で図 4 のように円と手書きストロークを削除することで円と手書きストロークが解析された後に円が削除される規則が定義できる．手書きストロークの認識には SATIN[8] を用いている．

4 図形言語の定義とビジュアルシステムの実行の例

Viola を用いて図形言語を定義しビジュアルシステムを実行する手順を解説するためにネットワーク図を表わす図形言語を用いる．

Viola では通常のドローツールと同様に自由に図を描くことができる．この状態で図 5–(A) のようにネットワーク図を描くことは可能であるが，ネットワーク図のための規則が何も定義されていない状態であるので，描いた図はネットワーク図としては認識されていない．この状態で図 5–(B) のように四角形を動かすとネットワーク図が崩れてしまう．描いた図をネットワーク図として認識させるためには，適切な規則を定義する必要がある．



(A) (B) (C)

図 5: ネットワーク図

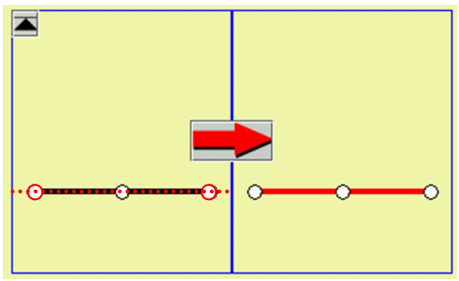


図 6: ネットワーク図の定義 (1)

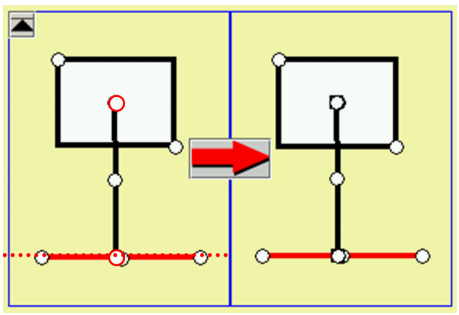


図 7: ネットワーク図の定義 (2)

まず，ネットワーク図の幹線となる線を定義する．規則の定義範囲を描画し，左側の領域に直線を描画する．直線の始点と終点の Y 座標を揃えて制約として定義すると，赤い点線によりその制約が図示される．さらに，右側の領域で直線の色を赤色に変更しアクションとして定義する (図 6)．

次にネットワーク機器と幹線を結ぶ線を定義する．新しく規則の定義範囲を描画し，左側の領域に直線を描画する．すると描いた直線が先程の規則により幹線として認識される．ネットワーク機器を表わす長方形を 1 つ入力し，その長方形と幹線を結ぶように直線を入力する．直線の始点を長方形の中心と一致させ，終点の Y 座標を幹線の Y 座標と一致させる (図 7)．

以上の 2 つの規則を定義することにより，図 5–(A) の図式がネットワーク図として認識され，図 5–(C) のようにネットワーク機器を表わす四角形を動かしても，Viola により図形間の関係が保持されるため，ネットワーク図のレイアウトが崩れることはない．

5 関連研究

Viola では図形の書き換え前と書き換え後を表現する 2 つの定義インタフェースを用いて図形文法の定義を行う．このような図式表現を用いて書き換え規則を定義する研究として Visulan [11] や VISPATCH[7], KIDSIM[5] などがある．KIDSIM では物体を動かすことにより物体の移動規則を例示で定義することができる．移動前と移動後の状態の絵の組が変換規則であり，パターンにマッチする変換規則があればそれに従って画面が書き換えられる．Visulan では変化前と変化後の組により絵の変化を表し，その組を一つのルールとみなしてプログラムを構築することができる．画面の一部が変化前の絵にマッチしたらその部分を変化後の絵に書き換えを行うことでプログラムが実行される．VISPATCH はルールをもとに図形の書き換えを行うビジュアル言語である．マウスによるイベントが発生すると，入力された図形によって表される条件が成立しているルールを探す．もし条件が成立しているルールがあれば，そのルールで定義されている図形へと書き換えを行う．

書き換えが行われる前後の画面を使って文法を定義するという点ではこれらの研究と Viola は類似している．しかし，Viola では図形の書き換え規則 (アクション) 以外にも，規則の適用条件である制約や，属性についても図式表現を例示することにより，グラフィカルに文法を定義し，その文法に基づいてビ

ジュアルシステムを生成することができるという点が異っている．

6 まとめ

従来の空間解析器生成系では，図形言語の定義にテキストを用いている，図形言語の定義とビジュアルシステムの実行を同時に行うことができない，という問題点があった．そこで，我々は図式表現を用いてグラフィカルに図形言語を定義し，同じ画面上で実行を行うことができる空間解析器生成系 Viola を作成した．Viola を用いることで図形を用いて図形言語の定義を行いながら文法の正しさを検査するという作業を同時に行うことが可能になった．

参考文献

[1] 馬場昭宏, 田中二郎. Spatial parser generator を持ったビジュアルシステム. 情報処理学会論文誌, Vol. 39, No. 5, pp. 1385–1394, May 1998.

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

[3] Sitt Sen Chok and Kim Marriott. Automatic construction of intelligent diagram editors. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pp. 185–194, 1998.

[4] Gennaro Costagliola, Filomena Ferrucci, Giuseppe Polese, and Giuliana Vitiello. Supporting hybrid and hierarchical visual language definition. In *Proceedings of IEEE Symposium on Visual Languages*, pp. 236–245, September 1999.

[5] Allen Cypher and David Canfield Smith. Kidsim: End user programming of simulations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHi' 95)*, pp. 27–34, May 1995.

[6] Eric J. Golin and Tom Magliery. A compiler generator for visual languages. In *Proceedings of IEEE Symposium on Visual Languages*, pp. 316–321, August 1993.

[7] Yasunori Harada, Kenji Miyamoto, and Rikio Onai. Vispatch:graphical rule-based language controlled by user event. In *Proceedings of the 1997 IEEE Symposium on Visual Languages*, 1997.

[8] Jason I. Hong and James A. Landay. Satin: a toolkit for informal ink-based applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 63–72, 2000.

[9] Hiroshi Hosobe. A modular geometric constraint solver for user interface applications. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST'01)*, Vol. 3, pp. 91–100, 2001.

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

[11] Kakuya Yamamoto. Visulan: A visual programming language for self-changing bitmap. In *Proceedings of International Conference on Visual Information Systems, Victoria University of Tech. cooperation with IEEE (Melbourne, Australia)*, pp. 88–96, 1996.