Spatial Parser Generator の Tcl/Tk を用いた実装

馬場 昭宏
筑波大学工学研究科
baba@softlab.is.tsukuba.ac.jp

田中 二郎
筑波大学電子情報工学系
jiro@softlab.is.tsukuba.ac.jp

概要

ビジュアルプログラミングシステムには図形の形状、配置の解析を行う部分 (Spatial Parser) があるが作成は困難で時間がかかるので、Spatial Parser Generator が必要になる。

本論文では現在我々が作成している仕様を与えて Spatial Parser を作るシステムについて述べる。生成された Spatial Parser は必要があれば対話的に再定義することが出来る。

Implementing A Spatial Parser Generator in Tcl/Tk

abstract

A visual programming system requires a *spatial parser*, which analyzes shapes and configurations of figures. Creating spatial parsers is a difficult and time consuming job, so we need a *spatial parser generator*.

In this paper, we describe a system that generates spatial parser from specifications. The resulted spatial parser can be tested and the specifications can be redefined, if needed, in an interactive way.

1 はじめに

ビジュアルプログラミングシステムとは、「2次元もしくはそれ以上の次元でプログラムを定義することが出来る環境」である [1]。我々は並列論理型言語 GHC[2] に基づき、ビジュアルプログラミングシステム PP[3][4] の提案を行ってきた。ビジュアルプログラミングシステム PP では、すでにいくつかの試験的な実装が成されているが、図形の形状や配置に関して、最終的な仕様が確定しておらず、仕様の変更が常に行われている段階である。

一般に、ビジュアルプログラミングシステムには図形の形状、配置の解析を行う部分 (Spatial Parser) がある。これはテキスト言語の処理系における字句解析、構文解析にあたる。通常、Spatial Parser は特定の図形の形状や配置の仕様に特化した形で実現されるが、このような形で Spatial Parser を作成するのは試験的な実装には不向きである。そこで、図形の形状や配置に関しても、仕様を入力すればその仕様に対する parser が自動的に合成される (テキスト言語における) lex や yacc のようなもの、すなわち Spatial Parser Generator があればよい、という考えが生まれる。

これまでに提案されてきた Spatial Parser Generator[11] は図形の形状や配置に関する仕様を与えた後にコンパイルし、生成された Spatial Parser をビジュアルプログラミングシステムに組み込み、動作のテストを行ってきた。

本論文では Tcl/Tk と Prolog を用いて定義とテストを繰り返しながら対話的に Spatial Parser を作成するシステムを提案する。本システムにおいては、仕様を対話的に与え、仕様を Prolog プログラムにコンパイルする。Test window から parse されるべき図形を与え、それを Prolog の query の形式に変換し、動作テストを行うことができるので、Spatial Parser をビジュアルプログラミングシステム本体に組み込む必要はない。

2　テキスト言語とビジュアル言語の構成要素の対比

ビジュアルプログラミングシステムで用いられる言語をビジュアル言語と呼ぶ。単語を構成するためにテキスト言語では文字を1次元的に配置していくが、ビジュアル言語では円や直線などの図形を2次元、もしくはそれ以上の次元に配置する。これらの図形のことを図形文字と呼ぶことにする。図形文字としては、例えば楕円、円弧、長方形、直線、文字などが考えられる。各図形文字は、形状に関わる情報として種類、色、大きさ、線の太さなどの属性を持つ。また、形状以外の情報として、ある座標空間における位置を属性として持つ。

テキスト言語ではある概念を単語として表すが、同様にビジュアル言語にも単語に相当するものがあると考えられる。ビジュアル言語ではある概念を図形文字を組み合わせて作った、意味のある一つの「もの」として表すのが一般的である (図1)。したがって本論文ではビジュアル言語におけるこのような「もの」を図形単語と呼ぶことにする。最も基本的な図形単語は1個以上の図形文字の集合として表される。テキスト言語では例えば「顔」という概念の視覚的な側面を説明する時に「顔は、輪郭があって、目が二つとまゆげが二つと口がある」という文ができるが、ビジュアル言語においては下位の概念を表す文字や単語を空間的に組み合わせることによって上位の概念を表す単語を作ることができる (図2(a))。このことから一般に図形単語は1個以上の図形文字または図形単語の集合として定義することができる。

図形単語自身の属性と図形単語を構成している下位の構成要素の属性とは異なっている。すなわち、一般には下位の構成要素が持っていなかった属性を図形単語が持つことになる。例として図2(b) をあげる。図2(b) では楕円や直線などから、「顔」という名前の図形単語を作っている。このとき、「顔」はその形状から「肥満度」という属性を持つ。これはどの構成要素にもなかった属性である。このように図形単語の属性はテキスト言語における形容詞の役割を果たしているものと考えることができる。

テキスト言語もビジュアル言語もある事柄を記述するための方法であるという点においては共通である。本論文ではテキスト言語で「文」と呼んでいる概念のことをビジュアル言語では図形文と呼ぶことにする。図形文は1個以上の図形単語の集合である。図形文と図形単語は見かけ上の違いはない。しかし、テキスト言語に変換したときに図形文は文になり、図形単語は単語になる。以下、本論文では単に文字、単語と言ったときは通常のテキスト言語における文字、単語を指すものとする。

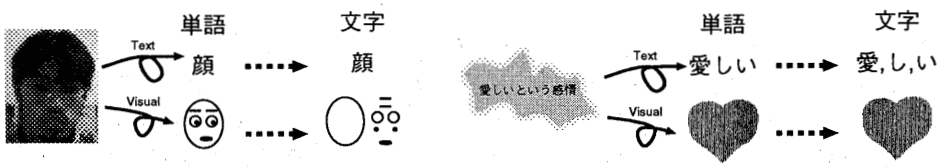


図 1: テキスト言語とビジュアル言語の対比

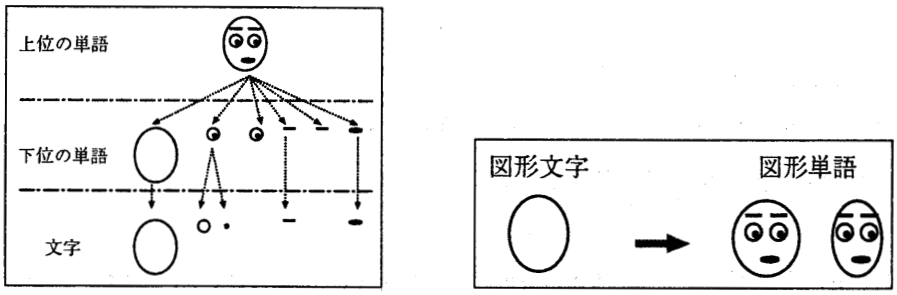
また、テキスト言語の字句解析、構文解析をビジュアル言語ではそれぞれ Spatial Lexical Analyzing、Spatial Parsing と呼ぶことにする。これまでビジュアル言語の構成要素は基本部品とそれを組み合わせることで作られるものとの2つに分類するのが一般的であったが ([9][10])、本論文ではテキスト言語の構成要素 (文、単語、文字) と対応付けて3つに分類した。この分類によりビジュアル言語の言語としての側面がよりはっきりした。

3　仕様の記述の方式

テキスト言語において構文解析する際に文法を形式的に記述する必要があるのと同様に、Spatial Parsing を行なうためにはビジュアル言語の仕様を形式的に記述する必要がある。

3.1　他の研究の動向

これまで Spatial Perser に仕様を与える方法としては文法的な記法 [5][6][7][8][9]、や宣言的な記法 [10] などが用いられてきた。しかし、これらの記法はいずれも textual なものである。図形文字や図形単語の属性は、図



(a) 単語の階層構造　　(b) 図形単語の属性

図 2: 図形単語

形として表現すれば自ずと備わるが、文字にしてしまうとそれらを明示的に記述する必要が出てくる。そのため、textual な記法は煩雑であり、しかも直観的に理解しにくい。

3.2 図形を用いる方式

そこで、図形を用いて仕様を記述したいという欲求が出てくる。しかし図形を用いると逆に必要のない属性までが記述されてしまう。例えば、図 3のような例を考える。図 3では「楕円の中に文字が書かれている」図形単語を定義したいとする。しかしそれを図を用いて記述しようとすると、「線の太さが 1 で長軸の長さが 30 で ··· の楕円の中心から x 軸方向に -10、y 軸方向に -5 だけ移動した点にフォントは Helvetica で色は黒で ···append と書かれた文字列がある」となってしまう。右側の 6 個の例はいずれも「楕円の中に文字が書かれている」図形単語であるが、左側の定義と完全に一致するのはいちばん右側のものだけである。あとのものは大きさや線の太さなどが異なるために一致しない。本当は右側の 6 個の図形単語すべてが一致して欲しいのこのような定義は表現したいものとは明らかに異なっている。

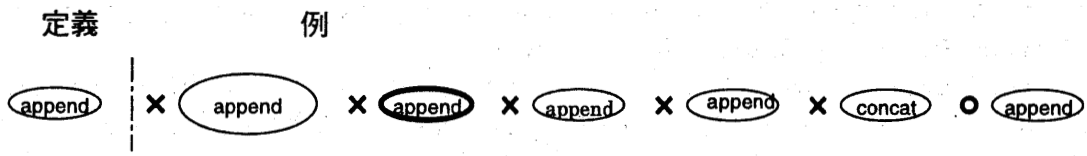


図 3: 図形のみを用いて仕様を与える例

3.3 図形と文字を併用する方式

3.2節で述べた問題の解決法として本論文で提案する仕様の記述の方式は、まず図形を用いて大まかな仕様を与えた後にそれを文字に変換し、必要のない属性を削除したり、複数の属性間の関係を記述することで仕様を決めるというスタイルである。この方式ではまず図形を用いているために定義したいものの概要を直観的に理解、記述することができ、その後文字に変換して必要のない属性をなくし、さらには図形の状態では必ずしも明らかではなかった図形間の関係を記述する。

4 実装するシステム

これらの考察をふまえて現在 Tcl/Tk を用いて実装を行っているシステムについて述べる。

4.1 機能

本システムが目指すものは

1. 3.3節で述べた方式によってビジュアル言語の仕様を記述し、
2. その仕様から Spatial Parser を自動生成し、
3. 生成された Spatial Parser のテストを行なう

ことをサイクルとして対話的に Spatial Parser を完成させるためのエディタである。システムの概要を図4に示す。図4の各行が上述の 1、2、3 に対応する。

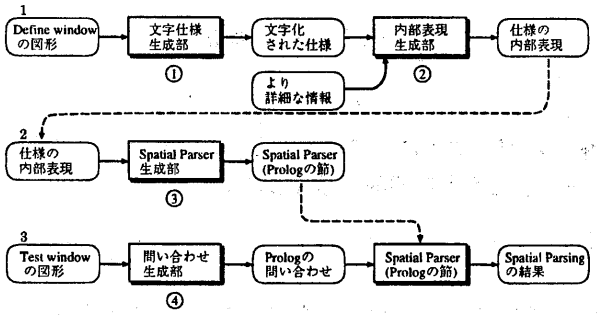


図 4: システムの概要

システムの画面例を図 5(a)、(b) に示す。図 5(a) の画面の上半分を Define window 下半分を Test window と呼ぶ。Define window のメニューバーには、入力した図形の保存、呼び出し、システムの終了などを行なう File、図形の消去、コピーなどを行なう Edit、入力する図形の種類を変える Mode、入力する図形の色、線の太さなどの変更の他、水平、垂直方向の整列を行なう Graphics、図形情報から文字情報、さらに Spatial Parser まで変換する Lex、各メニューの説明を表示する Help がある。Test window のメニューバーには Define window のメニューバーと同様の File、Edit、Mode、Graphics の他、Test window に描かれた図形を現在作られている Spatial Parser により Spatial Parsing する Parse がある。

ユーザはまず Define window に図形を入力する。入力した図形のうち一つの図形単語としたいものをまとめて選択し、メニューバーの Lex から Make Non-Terminal を選ぶと文字仕様を入力するためのウィンドウ (図5(b)) が開く (図4の①)。ここでユーザは必要のない属性を削除したり属性間の関係を記述する。ウィンドウを閉じると定義した図形単語が内部表現として登録される (図4の②)。

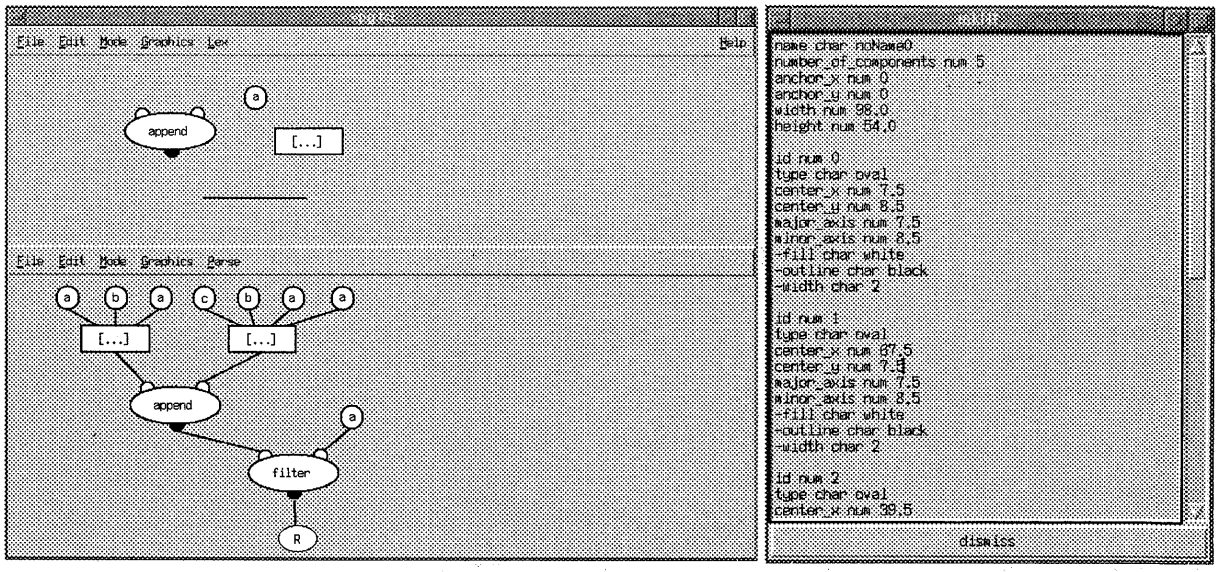
同様にしていくつかの図形単語を作成していく。メニューバーの Lex から Genarate Clause を選ぶと Spatial Parser として Prolog の節が作られる (図4の③)。生成された Spatial parser はファイルに書き出される。

Test window に実際に Spatial Parsing したい図形を入力し、メニューバーの Parse から Parse を選ぶと Test window の図形は Prolog の問い合わせに変換され (図4の④)、生成された Spatial Parser によって Spatial Parsing が行なわれる。このときに期待通りの出力が得られなかったら Spatial Parser の再定義を行ない、再びテストを行なう。

Spatial Parsing が成功すると仕様として与えられたアクションが実行される。アクションについては 4.2.4節で述べる。

4.2 文字による仕様

Spatial Parsing する際、上位の図形単語を構成するために下位の構成要素を組み合わせていくが、当然、下位の構成要素の属性値と上位の図形単語の属性値とは異なっている。図 5(b) において、いちばん上のブロックが現在定義中の図形単語自体の属性である。2 番目以降の各ブロックはその図形単語の構成要素の属性である。属性の各行は属性名、その属性の型、そして値を表している。現在のところ型としては、文字列型、数値型



(a) 図形仕様入力部およびテスト部　　(b) 文字仕様入力部

図 5: 作成中のエディタ

を用意している。また、定義している図形単語とその構成要素間で互いに属性の型や値を参照するため、参照型を用意している。

4.2.1 数値型

座標、長さなどを記述するために数値型を用いる。数値型の値は数値の取り得る値の範囲を指定できるようにするために次のように記述することとした。

{}	任意の値
n	値 n
{n_1 n_2}	n_1 以上 n_2 以下の値
{n_1 -}	n_1 以上の値
{- n_2}	n_2 以下の値

4.2.2 文字列型

図 3の例にあったように、図形の一つとしての文字の記述や、図形の種類、色などの記述には文字列を使う。これらには「任意の文字列」といったことが記述できると便利である。文字列型の値の記述には Tcl で用いることのできる正規表現を用いることができる予定である。

4.2.3 参照型

参照型を使うことによって、属性の値を記述する際に同一の単語内の他の構成要素の属性の値を参照する。例えば、ある構成要素 (id 1) の type を別の構成要素 (id 2) の type と同じものであるようにするときは次のように記述する。

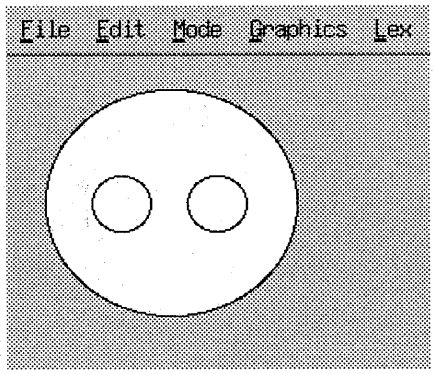
```
id num 1
type at 2.type
```

```
name char noName1
number_of_components num 3
anchor_x at 1.center_x
anchor_y at 0.center_y
width num 120.0
height num 121.0

id num 0
type char oval
center_x num 60.0
center_y num 60.5
major_axis num 60.0
minor_axis at 0.major_axis

id num 1
type char oval
center_x num 35.5
center_y at 0.center_y
major_axis num 12.5
minor_axis at 1.major_axis

id num 2
type char oval
center_x num 81.5
center_y at 0.center_y
major_axis at 1.major_axis
minor_axis at 1.major_axis
```



定義する図形の例

4.2.4 アクション

アクションはその図形単語が見つかったときに実行されるTclのスクリプトである。アクションは属性の一つとして記述される。

4.2.5 図形間の関係の記述の例

以下に参照型を用いて図形間の関係を記述する例をあげる。

この例では、右側に示すような3つの円を表している。id 0のものが外側の大きな円、id 1のものが内側左の円、id 2のものが内側右の円に対応する。まず、それぞれの円は参照型を用いて一つの楕円の長軸と短軸の長さを同じにすることで作られている。内側左の円は内側右の円の半径を参照することで内側右の円と同じ大きさになっている。内側の二つの円の中心のy座標は、外側の円のy座標を参照することでy座標を揃えている。さらに、定義された図形単語自体の属性anchor_x、anchor_yも参照型を用いて内側左の円の中心に設定している。

4.3 生成されたSpatial Parser

ビジュアル言語の定義はエディタによって図形的に、対話的に行なう。定義された文法から最終的に生成されるSpatial ParserはPrologの節の集合である。定義されたビジュアル言語の文法はテキスト言語の文法とは異なり、左から右へたどっていく、というような規則はない。そのためマッチングを行っていく過程において頻繁にやり直し(バックトラック)が必要となる。このような処理系を手続き型の言語として生成するのは非常に面倒である。そこで本システムではSpatial ParserとしてPrologの節の集合を生成することとした。このため、生成されるSpatial Parserは処理の流れではなく、ルール自体を記述したものとなった。文字情報化された図形単語はいったん内部表現に変換された後、さらに次のようなPrologの表現に変換される。ただしここに

示すのは字句解析部の概要のみである。

```
lex(Item,[Res|Result],Interp):-
        word1(Item,Res,Rest,Interp),
        lex(Rest,Result,Interp).
lex(Item,[Res|Result],Interp):-
        word2(Item,Res,Rest,Interp),
        lex(Rest,Result,Interp).

   ⋮

lex([],[],_).

word1(Rest0,Result,Rest3,Interp):-
        component1_1(Rest0,Result1,Rest1,[],Interp),
        component1_2(Rest1,Result2,Rest2,[],Interp),
        component1_3(Rest2,Result3,Rest3,[],Interp),
        Result=[[name|word1],[size_x|30],[size_y|40],
        [anchor_x|0],[anchor_y|0],
        [comp1|Result1],[comp2|Result2],[comp3|Result3]].

component1_1([X|Item],Result,Rest,Restt,Interp):-
        X=[Id|Rest0],
        comp_char(Rest0,[type|rectangle],Res1,Rest1,[],Interp),
        comp_num(Rest1,[coord_x|10],Res2,Rest2,[],Interp),
        comp_num(Rest2,[coord_y|20],Res2,Rest3,[],Interp),
        comp_char(Rest3,[-fill|white],Res4,_,[],Interp),
        Result=[[id|Id],[type|Res1],[coord_x|Res2],[coord_y|Res3],[-fill|Res4]],
        append(Item,Restt,Rest).
component1_1([X|Item],Result,Rest,Restt,Interp):-
        component1_1(Item,Result,Rest,[X|Restt],Interp).

component1_2([X|Item],Result,Rest,Restt,Interp):-

   ⋮

word2(Rest0,Result,Rest3,Interp):-

   ⋮
```

lex は与えられた図形に定義済みの図形単語を探し、再帰的にlex を呼び出すことによりすべての図形のマッチングを行なう。lex はさらに上位の述語parse から呼び出されることになる。

word$l$ は図形単語 l の定義である。図形単語 l を構成する図形文字 $l_1 \cdots l_m$ があるかどうかを探し、結果として図形単語 l 自体の属性のリストを作る。

componentl_m は図形文字 l_m の定義である。与えられた図形の集合の中から図形文字 l_m にマッチするものを探し、結果として図形文字 l_m の属性のリストをつくる。

comp_char、comp_num はそれぞれ文字型、数値型が Prolog の節に変換されたものである。

4.4 図形文

このようにして生成された Spatial Parser に与えるデータ (図形の集合) はやはり Prolog の問い合わせの形をしていなければならない。本システムでは Tk の Canvas Widget に描かれた図形を生成された Spatial Parser への問い合わせに変換するための Tcl のライブラリ (図 4の④) も提供する。

実際に変換されたものは次のような形式である。

lex([[図形文字 1],[図形文字 2], 図形文字 3],$\cdots$,[図形文字 n]],Result,Interp)

ここで、図形文字 i は次の形式をしている。

[Id,[属性名 1| 値 1],[属性名 2| 値 2],[属性名 3| 値 3],$\cdots$,[属性名 m| 値 m]]

5 おわりに

本論文では仕様を図形で与えてから詳細化していくことによりSpatial Parserを自動生成し、生成したSpatial Parserを用いてテストを行なうことができるようなシステムについて述べた。生成されたSpatial ParserはPrologの節の形をしており、これと、このシステムが同時に提供している図形の集合をPrologの問い合わせに変換するTclのライブラリを組み込むことにより、ビジュアルプログラミングシステムの実装はこれまでよりも容易になるはずである。

関連研究としてはGolinによるSPARGEN[11]があるが、本論文で述べた方法は仕様を与えるのにまず最初に図形を直接用いることができるという点と、生成されたSpatial Parserを用いてテストしながらさらに仕様を変えていくという対話的な方法でSpatial Parserを作成できるという点において異なる。

現在Spatial Lexical Analyzerのみではあるが図4の①、②を実装し、③、④を実装中である。実装中のシステムでは距離の範囲を指定して図形間の位置関係を記述している。しかしこの方法では「接している」、「包含している」、「交わっている」などの抽象的な位置関係を記述するのは難しい。そのため、より抽象的な方法で図形間の関係を記述できるようにする必要がある。また、図形単語の正規表現のようなものを用いて図形のみで仕様を記述する方法についても考察する予定である。

参考文献

[1] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1(1):97–123, 1990.

[2] K.Ueda: Guarded Horn Clauses, ICOT Technical Report TR-103, ICOT, 1985.

[3] J.Tanaka: Visual Programming System for Parallel Logic Languages, The NSF/ICOT Workshop on Parallel Logic Programming and its Program Environments, the University of Oregon, pp.175-186, 1994.

[4] 中野勝次郎, 田中二郎: ビジュアルプログラミングシステムにおけるモデルの視覚化アルゴリズム, インタラクティブシステムとソフトウェアII, 日本ソフトウェア科学会 WISS'94, 竹内彰一 編, 近代科学社, pp205-214, 1994.

[5] J.Rekers and A.Schurr. A graph grammar approach to graphical parsing. Technical report, Leiden University, 1995. available via ftp from ftp-server ftp.wi.leidenuniv.nl,file `/pub/CS/TechnicalReports/1995/tr95-15.ps.gz`

[6] C.Crimi, A.Guercio, G.Nota, G.Pacini, G.Tortora, and M.Tucci. Relation grammars and their application to multi-dimensional languages. *Journal of Visual Languages and Computing*, pages 333–346, 1991.

[7] K.Wittenburg, L.Weitzman, and J.Talley. Unification-based grammars and tabular parsing for graphical languages. *Journal of Visual Languages and Computing*, pages 347–370, 1991.

[8] S-K.Chang, M.J.Tauber, B.Yu, and J-S.Yu. A visual language compiler. *IEEE Transactions on Software Engineering*, 15(5):506–525, 1989.

[9] E J.Golin. Parsing visual languages with picture layout grammars. *Journal of Visual Languages and Computing*, pages 371–393, 1991.

[10] R.Helm and K.Marriott. Declarative specification of visual languages. In *Proceedings 1990 IEEE Workshop on Visual Languages*, pages 98–103, 1990.

[11] E.J.Golin and T.Magliery. A compiler generator for visual languages. In *Proceedings 1993 IEEE Symposium on Visual Languages*, pages 314–321, 1993.