Spatial Parser Generatorを持ったビジュアルシステム

馬場昭宏† 田中二郎††

ビジュアルプログラミングシステムや，特定用途の図形エディタでは対象となる図が図形を用いた言語（図形言語）であるために図形言語の解析を行う部分（Spatial Parser）を実装する必要があった．しかし個々のアプリケーションごとにSpatial Parserを実装するのは困難で時間のかかる仕事であった．また，これまでのSpatial Parsingアルゴリズムは解析を行うことに主眼がおかれ，その結果に基づいて何らかのコードを実行するというものは少なかった．本研究ではSpatial Parsingアルゴリズムの1つであるCMGにactionの概念を追加することで，実際に行った解析結果を利用した動作を行えるようにした．次に本研究ではSpatial Parser Generatorを持つことにより，図形言語の文法を動作を与えることで様々な図形言語に対応することができるシステム「恵比寿」を作成した．恵比寿ではまずはじめに図形を用いて入力し，それからCMGを半自動的に生成するため，直感的に図形言語の文法と動作を定義できる．さらに恵比寿では一度解析が終了すると文法に基づいて図形間に制約が課せられる．本論文では最後に恵比寿上でアプリケーションを作成する例を3つ示した．

A Visual System Having a Spatial Parser Generator

Akihiro Baba† and Jiro Tanaka††

Visual programming systems and special purpose graphic editors need to have spatial parsers, which analyze shapes and configurations of figures, because they treat figures as visual languages. However, creating a spatial parser for every system is a difficult and time consuming job. Main purpose of traditional spatial parsing algorithms is analyzing figures, and few of them execute the statements according to the result of the analysis. We extended a spatial parsing algorithm CMG by adding the concept of "actions." We can execute the statements using the result of spatial parsing. We have actually implemented "Eviss" system. As Eviss has a spatial parser generator, it can generate various visual applications, if provided with grammars and actions for such applications. In Eviss, users roughly define grammars in a visual way at the definition window at first. Eviss translates them into CMG form. Then users modify them and write actions. After that, users draw figures which should be parsed at the execution window. When spatial parsing succeeds, Eviss gives constraints to figures according to the grammars. We show three examples which denote visual applications can easily be created using Eviss.

1. はじめに

プログラミングの初期の過程においてデータ構造やデータの流れ，処理の流れなどを表すのに図を用いることが多い．ビジュアルプログラミングシステムはこのような図をそのまま使ってプログラミングをしようというものである．しかし，既存のビジュアルプログラミングシステムは特定のビジュアルプログラミング言語の仕様に固定されており，変更は困難で，かつ時間のかかる仕事であった．一方，回路図[1]やERダイアグラム[2]，OMTのオブジェクト図[3]など，特定の用途の図は専用のエディタを用いて描くことが多かった．汎用のエディタ（ドローツール）を用いて描くと，エディタがその図の意味を知らないために使用者が多くの操作をしなくてはならないためである．たとえば，円の中にラベルとして文字が書いてあるものをノード，それらのノード間をつなぐ直線をエッジとしてグラフを表現するようなものを考えると，ノードを動かすことを意図して円を動かしても，文字と直線はその円の動きに追随しない．

データ構造の図や回路図などは，構造を持っている，いわば図形を用いた言語（**図形言語**）であるため，空間的な文法を有しているといえる．

最近ではTcl/Tk[4]やJAVA[5]など，単純なグラ

† 筑波大学工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 筑波大学電子情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

ソィックスやGUIを簡単に扱うことができる環境が整ってきており，これらを用いてビジュアルシステムを作成し，システムごとに図形言語を解析する部分も作成していた．しかしながらこのようなアドホックな方法ではなく，テキスト言語に対してそれぞれの言語仕様に対応した構文解析器を作るように，図形言語の文法に従って図形言語を解析する部分（Spatial Parser）を作るという考え方に基づけば，より普遍的な方法で図形言語を解析することができる．このことを利用して図形言語の文法を記述することによりSpatial Parserを生成するSpatial Parser Generatorを実装することができる．

しかし実際のアプリケーションでは，図形言語の解析のみならず，解析結果に応じた動作を行い，図形間に幾何制約を課すことによって意味的関係を保存し，さらには描いた図形へのフィードバックを行うことが望まれる．我々はSpatial Parser Generatorと制約解消系を持つことでこれらの要求を満たすようなシステム「**恵比寿**」を作成した．恵比寿では図形言語の文法とその動作を与えることでアプリケーションを記述できる．

2. 図形言語

2.1 テキスト言語との比較

単語を構成するためにテキスト言語では文字を一次元的に配置していくが，図形言語では円や直線などの図形を二次元，もしくはそれ以上の次元に配置する．これらの基本的な図形のことを**図形文字**と呼ぶことにする．各図形文字は，種類，色，大きさ，位置などの属性を持つ．

テキスト言語ではある概念を単語として表すが，同様に図形言語にも単語に相当するものがあると考えられる．図形言語ではある概念をいくつかの図形を組み合わせて作った意味のある1つの「もの」として表すのが一般的である．本論文では図形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．また，1つの図形単語を作るためのいくつかの図形を構成要素と呼ぶことにする．

2.2 図形言語の文法の与え方

図形言語の文法を記述する方法とそれに基づいてSpatial Parsingを行うアルゴリズムとしては，Picture Layout Grammars（PLG）[6]，Positional Grammars（PG）[7]，Relation Grammars（RG）[8]，Constraint Multiset Grammars（CMG）[9]などが知られている．本研究ではCMGを用いることとした．

CMGでは表現できる図形言語のクラスが広い．PGでは1つの図形単語が持っているのは1つの点のみであるが，CMGでは任意個の点を図形単語の属性として持つことができる．RGでは2つの図形単語間の関係しか記述できないが，CMGでは任意の数の図形単語の属性間の関係を記述できる．CMGの考案者であるMarriottは文献10)でPG，RGおよびUnification Grammars[11]を用いて表した文法はすべてCMGを用いて書き換えることができることを示している．

また，我々はユーザが自分で図形言語を定義できることを目標としているので，記述，修正，理解が容易であることが重要であると考えた．CMG以外の記法では図形単語の属性，構成要素の種類と数，構成要素の属性間の制約などをすべて混在させて記述するので理解が困難であるが，CMGではこれらを別々に記述するため分かりやすい．

このようにCMGの記述力は高い．しかしながらそのアルゴリズムは解析を行うことに主眼をおいているため，解析はできるが解析した結果に基づいて何らかのコードを実行するものは少なかった．実際のアプリケーションでは解析した結果に基づいて何らかの仕事をするので，解析だけでは不十分である．一方，これまで図形言語の文法を記述する方法とそれに基づいてSpatial Parsingを行うアルゴリズムはいくつか提案されているが，実装となるとあまり多くはない[12]．SPARGEN[13]はPLGの拡張であるOOPLGに基づいている．SPARGENでは解析結果を利用することができるが，解析後にプログラムである図形の間に意味的な関係を保存するような機能はない．文献14)に述べられているシステムはCMGに基づいている．このシステムでは図形間に意味的な関係を保存するような機能はあるが，解析結果を利用することができない．

2.3 CMGへのactionの導入

本研究ではスクリプト言語の存在を仮定し，CMGにactionの概念を導入した．本研究ではCMGにおけるactionを「1つの図形単語が認識されたときにスクリプト言語のプログラムとして実行される文字列」として定義する．また，図形言語で書かれたプログラムへのフィードバックもアクション中に記述する．図形言語へのフィードバックについては3.1節で述べる．actionの中では，属性名の最初と最後を @ で囲んだ部分はスクリプト言語に渡される前に構成要素や新たに生成された図形単語の属性の値を表す文字列として扱われる．また，図形単語の名前の最初と最後を @ で囲むと，図形単語に付けられたidを表す文字列として扱われる．このidは図形単語の移動など，図形言語へのフィードバックを行うときに用いられる．また，

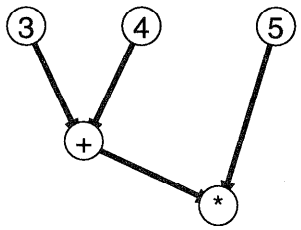


図1 計算の木
Fig. 1 Computation tree.

script 指令も導入するが，これについては例の中で詳しく述べる．

例として，計算の木を実行するようなものを記述することを考える．これはたとえば，**図1** のような図を与えると結果として 35 を返すというものである．ただし，図1において矢印の両端は円の中心にあるが，文字と重なると見にくくなるので円の下に描かれている．

図1の例のような図形言語は自然言語による次のような2つの規則によって表すことができる．

(1) ノードは円の中に数字が書いてある．
(2) ノードは2つのノードが矢印によって円につながれ，その円の中には演算子が書かれている．

これを我々の拡張 CMG で表すと次のようになる．

```
 1:node(string sort,point mid,
 2:      integer value) ::=
 3:   C:circle,T:text
 4:   where (
 5:      C.mid == T.mid
 6:   ) {
 7:      sort = {string "number"};
 8:      mid = C.mid;
 9:      value = to_int(T.text);
10:   } {
11:      display("@sort@ was parsed.");
12:      display("value = @value@");
13:   }
14:
15:node(string sort,point mid,
16:      integer value) ::=
17:   N1:node,N2:node,L1:line,
18:      L2:line,C:circle,T:text
19:   where (
20:      N1.mid == L1.start &&
21:      N2.mid == L2.start &&
22:      C.mid == L1.end &&
23:      C.mid == L2.end &&
24:      C.mid == T.mid
25:   ) {
26:      sort = {string "operator"};
27:      mid = C.mid;
28:      value = {script.integer {
29:         @N1.value@@T.text@@N2.value@}};
30:   } {
31:      display("@sort@ was parsed.");
32:      display("value = @value@");
33:   }
```

1行目から13行目までが数字を表すノードの定義である．1，2行目ではその図形単語の名前と属性を定義している．名前は node であり，3つの属性 sort，mid，value を持ち，それらの型はそれぞれ string，point，integer である．3行目では図形単語がどのような構成要素から成るのかを定義している．ここでは，1つの円（circle）と，1つの文字列（text）から成るということを定義している．4行目から6行目は，図形単語の構成要素間の制約を表している．ここでは円 C の mid という属性の値と，文字列 T の mid という属性の値が等しいという制約を表している．7行目から10行目は制約が成り立ったときに，定義中の図形単語の属性にどのような値を与えるかということを定義している．7行目では属性 sort に「number」という文字列定数を与えることを，8行目では mid と円 C の mid がつねに等しくなることを，value には文字列 T のテキストを integer 型に変換して代入することを定義している．11行目と12行目が CMG を拡張した部分「action」である．この図形単語の属性 sort の値が「number」で，属性 value の値が「3」であり，display が表示するための命令であれば，この図形単語が認識されたときには，

```
number was parsed.
value = 3
```

と表示される．15行目以降では，演算子に2つのノードがつながっているものを定義している．ほとんどは数字を表すノードと同じであるが，28，29行目で script 指令を用いている．script 指令は複数の構成要素の値から図形単語の属性の値を生成するのに用いられる．script 指令中では action と同様の方法によって複数の構成要素の値を参照し，1つの文字列にする．生成された文字列をスクリプト言語によって解釈させ，その結果を得る．結果の型はここでは特に規定しない．得られた結果を図形単語の属性の型に変換したものをその図形単語の属性の値とする．script 指令を使った属性値の書き方は次のような形式である．

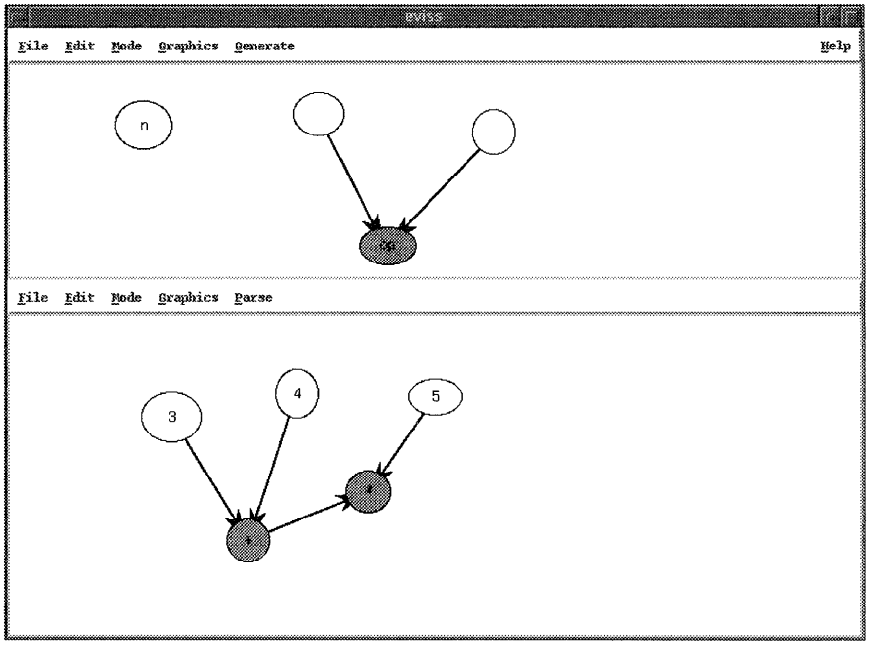


図2 恵比寿の実行例
Fig. 2 A snapshot of the Eviss system.

{**script**.*type* {*script*}}

ただし，*script* はスクリプト言語に解釈させる文字列で，*type* は得られた結果を今後どの型として扱うかを示す型の名前である．script 指令がないとすると，あらかじめ CMG 自体の解析系に準備しておいた関数を用いて値を生成するしかないが，script 指令を用いることによって，スクリプト言語が解釈できる文字列さえ生成できれば，後はスクリプト言語によって値を求められる．このことは属性の値に応じて動的に関数を変更できることを意味し，そのために script 指令を用いると記述量を削減できる．たとえば，N1.value が「3」，T.text が「+」，N2.value が「4」という値を持っているとすると，「**@N1.value@@T.text@@N2.value@**」は「**3+4**」という文字列に変換される．これをスクリプト言語によって解釈させて「**7**」という値を得る．属性 value は integer 型なので，「7」は integer 型に変換されて value の値となる．script 指令がない場合，where の中で演算子の種類のチェックを行い，それに対応した演算により属性 value の値を求めるようなルールを演算子の種類の数だけ書かなければならない．

3. システム「恵比寿」の実装

我々は Spatial Parser Generator を持ったビジュアルシステム「恵比寿」を Tcl/Tk と一部 C 言語を用いて実装した．Tcl/Tk はインタプリタ言語であるので，2.3 節で仮定したスクリプト言語として Tcl/Tk をそのまま利用できるという利点がある．

Marriott らは文献 10) の中で Chomsky の分類に対応させて CMG のクラス分けを行っている．今回実装を行ったのは CMG のうち exist を扱わないものであり，type2（文脈自由）に属する．

3.1 恵比寿の機能

恵比寿は次の機能を備える．

(1) 図形エディタ
(2) 図形による CMG の定義
(3) 図形単語間の意味的な関係を保存
(4) action の実行
(5) 図形言語へのフィードバック

システムの画面例を**図2** に示す．図 2 の画面の上半分を定義窓と呼び，図形言語の作製者はここで文法の定義をする．下半分を実行窓と呼び，ユーザはここで実際に解析，実行したい図形を入力する．これは通常のテキスト言語ではエディタを用いてプログラミングすることに相当する．定義窓，実行窓における図形の入力では，通常のドローツールにあるような，コピー，削除，整列といった操作を行うことができる．

CMG はその他の図形言語の記述方法に比べれば分かりやすいとはいえ，図形を文字を用いて表すために直感的に分かりにくい．本システムはエンドユーザでも簡単にビジュアルシステムを記述できるようにする

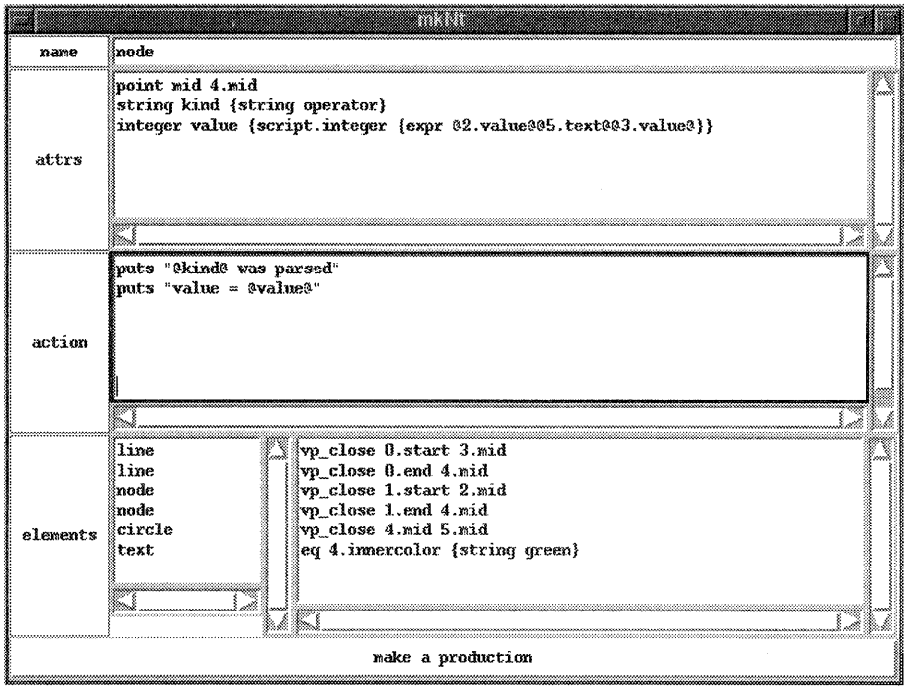


図 3 CMG 入力部
Fig. 3 CMG input windows.

ことを目的としているために，まず図形を用いて大まかな文法を与えた後に，それから CMG ベースの属性，制約を生成し，CMG 入力部と呼ばれるウィンドウに書き出す（**図 3**）．そのあとユーザはそれを修正して文法を決定するという方式で図形言語の文法と action を記述する．なお図 3 のウィンドウの各部に書かれている内容は 2.3 節の CMG による記述の 15 行目から 32 行目に書いたものとほぼ同様である．CMG 入力部は上から順に名前（name），属性（attr），action（action），構成要素の数と種類（elements の左側），構成要素間の制約（elements の右側）を書く欄に分かれている．これは 2.2 節で述べたように CMG がそれぞれを別々に記述しているために可能なことである．1 つの図形単語としたい図形を選択して CMG 入力部を開くと図形から構成要素とそれらの間に成り立っている制約，構成要素の属性の具体的な値が生成され，CMG 入力部に書き出される．ここで，構成要素の属性の具体的な値のうち，座標は通常意味を持たないので，後から削除しなくていいように書き出されないこととした．ユーザはここに新たに名前，属性，action を追加し，システムによって生成された制約のうち必要のないものを削除していく．

CMG 入力部における図形単語の 1 つの属性は型，属性名，属性値から成る．属性値は以下の 3 つのうちのいずれかである．

(1) 1 つの構成要素の属性をコピーする場合：*v-id.* 属性名
(2) 複数の構成要素から script 指令によって新たな値を作る場合：`script.`*type* と *script* から成るリスト．*type* は新たに作られた値の型，*script* はスクリプト言語に渡すための文字列．「`@`*v-id.* 属性名`@`」の形をした部分はその属性値に置換される．
(3) 定数：型と値から成るリスト

ここで，(1) の場合は構成要素の属性の値が変化したときには新たに作られた属性の値も変化するが，(2) の場合は構成要素から新たに定数を作っているので構成要素の値が変わっても属性の値は変わらない．なお，*v-id* は 1 つの生成規則の中で構成要素に順番に付けられた id である．

図形言語の定義が終わったら実行窓に実際に Spatial Parsing したい図形を入力する．恵比寿では Spatial Parsing のモードは 2 種類ある．新たな図形の入力があるたびに Spatial Parsing されるモードと，ユーザからの要求があったときのみに Spatial Parsing されるモードである．Spatial Parsing によって図形単語が認識されると，その図形単語の生成規則に書かれた制約が図形間に課せられる．そのためにユーザは以降

の編集においては図形単語ごとに移動を行うことができる．

また，図形単語が認識されたときにはその時点で図形単語の生成規則に書かれたactionが実行される．このためにプログラムが完成していなくても部分的に実行されていく．action中にはTcl/Tkの任意のスクリプトを記述できるので，値の計算や，ウィジェットの生成なども行うことができる．

ビジュアルプログラミングシステムではプログラムの視覚化表現を描き換えることによって実行の視覚化を行うことが多い．そのため恵比寿では図形言語へのフィードバックを行うための手続きも提供している．これらの手続きをaction中に記述することで図形単語の移動や削除，図形単語の属性の値（色や文字列の値，線の太さなど）を変更することができる．

3.2 CMGの解析アルゴリズム

Spatial ParsingのアルゴリズムにはCMGに付属のものを使用した．後の説明のために文献14)からアルゴリズムを引用し，簡単に説明する．なおここでは変数はイタリックで表記する．

```
procedure Parse(t)
  AddToken(t)
  for each SCC in order do
    repeat
      for each rule R in the SCC do
        EvalRule(R)
      end for
    until ParseForest is unchanged
  end for
end
```

これは新たな図形文字が入力されるたびにインクリメンタルに解析が行われるアルゴリズムである．*ParseForest*は図形単語のデータベースである．*SCC*は生成規則のデータベースで，*SCC*中では低い階層の図形単語を作るための生成規則から順に並べられている．手続きAddTokenは図形単語のデータベースに与えられた図形文字tを挿入するものである．手続きEvalRuleは生成規則Rを使って新たな図形単語を*ParseForest*に挿入し，生成に用いられた図形単語を*ParseForest*から削除するものである．

3.3 恵比寿中での実装

本節では，主にどのようにして図形間に制約を課すのかについて述べる．本節では変数名など実際の文字列はタイプライタ体で，値が変わるものはイタリックで表記する．

恵比寿の実装では3.2節での*ParseForest*を連想配列D，Rを連想配列Pとしている．Pのそれぞれの要素は**表1**のようになっている．ただし，*p-id*は1つの生成規則に付けられたidである．

表1 生成規則の内部表現
Table 1 Internal representation for the rules.

名前	値
P(*p-id*.name)	図形単語の名前
P(*p-id*.attrs)	図形単語の属性のリスト
P(*p-id*.script)	図形単語が解析されたとき実行されるaction
P(*p-id*.var)	図形単語の構成要素のリスト
P(*p-id*.constraints)	構成要素の属性間の制約のリスト

制約解消系にはSkyBlue[15]のC言語による実装に手を加え，これをTclから呼び出すインタフェースを作成し，使用した．このインタフェースを用いてTclからSkyBlueの変数や制約を操作できる．変数の操作には，作成，削除，値の変更，値を得る，型を得る，などがある．SkyBlueの変数はid（以下，*c-id*），値，型を持っている．型としては整数型，浮動小数点数型，文字列型，点型（x，y座標の組）を実装している．また，制約の操作には，作成，削除，制約している変数の*c-id*を得る，などがある．

恵比寿では各図形単語の属性の値をSkyBlueの変数とすることで各属性の値が変わったときの制約解消をSkyBlueに行わせている．*c-id*は図形単語のデータベースであるDに保存されている．Dの添字は，「*f-id*.属性名」の形をしている．ここで，*f-id*は図形単語のidである．たとえば，3という*f-id*を持つ図形単語の「mid」という属性は，D(3.mid)によって表される．この属性の値と型は，*c-id*を用いてSkyBlueの変数の値，型を得ることによって知ることができる．

構成要素となる図形単語間には次のようにして制約が課せられる．EvalRule中では図形単語のデータベースDに現在ある図形単語を構成要素として，生成規則のデータベースPにある制約を満たすかどうかチェックされる．構成要素となる図形単語間にSkyBlueによって実際に制約を課すのはPの各々の制約が満たされたときであってはならない．すべての制約が満たされたときにのみ新たな図形単語として認識されるためである．したがって，すべての制約が満たされたときにSkyBlueによって実際に図形単語間に課せられるべき制約のリストCを準備し，Pの制約のうち満たされているものから順にCに加えていき，新たに認識された図形単語をDに挿入するときにまとめてCに蓄えられたSkyBlueの制約を生成する．

新たに作られた図形単語の属性には次のようにして制約が課せられる．3.1節で述べた「1つの構成要素の

属性をコピーする場合」，「複数の構成要素からscript指令によって新たな値を作る場合」，「定数」の3種類それぞれについて考えなければならない．まず，1つの構成要素の属性をコピーする場合は，新たな図形単語の属性を表すものを D(f_1.a_1-*name*)，コピーしたい構成要素の図形単語の属性を表すものを D(f_2.a_2-*name*) とすると，D(f_2.a_2-*name*) に入っていた *c-id* を D(*f1*.a_1-*name*) に入れることによって実現できる．次に，複数の構成要素からscript指令によって新たな値を作る場合，下に述べる方法で属性名から実際の値を得，それをスクリプト言語によって解釈させて計算結果を得る．この結果を値とし，script指令によって与えられた型を型とする新たなSkyBlueの変数を作成し，その *c-id* を D(*f1*.a_1-*name*) に入れることによって実現できる．定数の場合，与えられた型と値を用いて新たなSkyBlueの変数を作成し，その *c-id* を D(*f1*.a_1-*name*) に入れることによって実現できる．

script指令やaction中で @*v-id*.*a-name*@ のような属性名（*v-id* は3.1節と同じもの）から実際の値を得る手順を以下に示す．

(1) P(*p-id*.var) の *v-id* 番目の要素を取り出すことにより *f-id* を得る．
(2) 属性名を得る．これは *a-name* そのものである．
(3) D(*f-id*.*a-name*) によって *c-id* を得る．
(4) *c-id* を用いてSkyBlueの変数の値を得る．

4. 実行例

本章では恵比寿の実行例を示す．

4.1 実行例1

恵比寿の実行例として，2.3節に示したことをするアプリケーションを恵比寿を用いて作成する過程を述べる．ただしプログラムを書くユーザが円の中心と文字列の中心を合わせる操作を行うのはわずらわしいので，「==」としては位置がほぼ等しいという制約である「vp_close」を用いることとする．また，sortがnumberであるnodeの色は白で，sortがoperatorであるnodeの色は濃いグレーで表すことにする．

まず数字を表すノードを定義するために，定義窓に円（circle）を入力し，その中心付近に適当な文字列（text）を入力する．これらをまとめて選択し，メニューバーのGenerateからMake New Production Ruleを選ぶとCMG入力部が開く．この時点でシステムによって構成要素の種類の欄には

```
circle
text
```

と書かれ，構成要素間の制約の欄には，

```
vp_close 0.mid 1.mid
eq 0.radius {integer 13}
...
```

などと書かれる．1行目はcircle（0）の中心（mid）とtext（1）の中心（mid）がほぼ一致しているという制約であり，2行目以降は実際に入力されたcircle，textが持っていた半径，色，書かれている文字列，フォントの種類などの具体的な値である．ここではこれらの具体的な値は必要ないので削除する．さらに名前を書く欄にnodeと書き，属性値を書く欄には，

```
string sort {string number}
point mid 0.mid
integer value 1.text
```

と書く．1行目はsortの属性値が「number」であることを，2行目はmidの属性値がcircle（0）の中心（mid）であることを，3行目はvalueの値がtext（1）の文字列（text）と同じであることを表している．actionを書く欄にはたとえば，

```
puts "@sort@ was parsed."
puts "value = @value@"
```

と書く．これによりこの図形単語が認識されると，@で囲まれた部分が実際の属性の値に置換された後，Tclのインタプリタに渡されて実行される．

これで数を表すノードの定義が完了したのでCMG入力部のウィンドウを閉じる．ウィンドウを閉じると定義した図形単語に対応した生成規則の内部表現が作られる．同様にして演算子に2つのノードがつながっているものを定義する．

作成した図形言語で描いたものを実際に実行するときには実行窓にSpatial Parsingしたい図形（たとえば図1）を入力していくと，Spatial Parsingのモードが Autoであれば図形を入力するたびに，モードがOn demandであればメニューバーのParseからParseを選ぶたびに，定義された図形言語の文法に従って実行窓の図形がSpatial Parsingされる．仮にすべて入力してからOn demandモードでSpatial Parsingしたとすると，図形単語が認識されるたびにactionが実行され，

```
number was parsed.
value = 3
number was parsed.
value = 4
number was parsed.
value = 5
operator was parsed.
value = 7
```

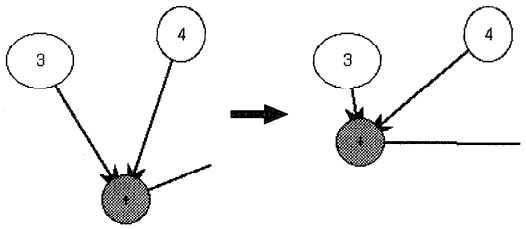


図4 ノードとしての移動
Fig. 4 Movement as a node.

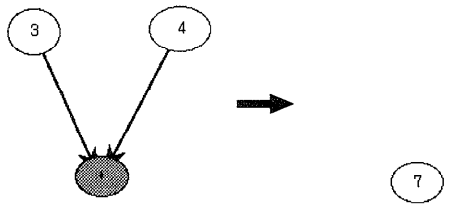


図5 図の描き換え
Fig. 5 Redrawing of a figure.

```
operator was parsed.
value = 35
```

と表示される．一度 Spatial Parsing が成功すると，図形間には実際に制約が課せられる．実行窓のノード（たとえば中に「+」と書かれたノード）の円を移動させようとすると文法に従ってその中に書かれている「+」という文字列と，それにつながっている矢印とが移動する（図4）．

4.2 実行例2

2つ目の例として，作成したプログラムを描き換えるようなものをあげる．規則としては 2.3 節に示したものとほとんど同じであるが，33 行目以降を次のように変更したものであるとする．

```
33:     delete {@N1@,@N2@,@L1@,@L2@};
34:     alter @C@ text to_string(@value@);
35:  }
```

これは恵比寿では次のように記述する．

```
delete {@0@ @1@ @2@ @3@}
alter @4@ text @value@
```

手続き `delete` は図形単語の id のリストを受け取り，それらを画面から削除し，関連する SkyBlue の変数と制約を削除するものである．手続き `alter` は図形単語の id と属性名と新しい値を受け取り，その属性を表す SkyBlue の変数の値を受け取った値に更新するものである．これにより，属性 sort が operator であるような node が認識されると，つながっていた2つのノードと，2つの矢印が消え，円の中に書かれていた演算子は計算結果に描き換えられる（図5）．現在の実装では内部の図形単語のデータベース（3.3 節）の書き換えは行われないので，内部の表現と画面上に表示されるものとは必ずしも一致していない．

4.3 実行例3

3つ目の実行例として，action を利用してボタンを作るようなものをあげる．図形言語の文法は「円の中にテキストを書いたものをボタンである」とし，action は「ボタンに書かれたテキストで示されるアプリケーションを起動するような Tk のボタンウィジェットを作る」とする．このような図形言語を定義することにより，図形単語である「ボタン」が認識されると認識された順番に Tk のボタンウィジェットが作られる．この例では Spatial Parsing を開始する前にボタンウィジェットをのせる台紙を作らなければならない．恵比寿では現在このような初期化動作をさせるために Tcl スクリプトが書かれた初期化ファイルを準備しておいてそれを評価するという方式をとっている．この例を作成するためには4行の初期化ファイルと1つのルール（ルール中の action は2行）を定義すればよい．実行例を**図6** に示す．この例では文法と動作を定義した後に実行窓に3つの円と3つのテキストを描き（図の左下付近），Spatial Parsing を行った結果，中に定義したテキストが書かれた3つのボタンウィジェットが生成されている（図の右上付近）．このうち真ん中の「xclock」と書かれたボタンを押したところそのアプリケーションが起動されている（図の右下付近）．

5. 関連研究

恵比寿は図形を用いたインタラクションを主眼としているために図を解析してテキストの生成を行うことはできるが，逆にテキストから図を生成するのは困難と思われる．TRIP2[16]では2つの中間表現を準備することによってテキストから図，図からテキストへの相互変換を可能としている．しかし，TRIP2 で行えるのはルールに従ってテキスト，図を生成することのみであり，恵比寿のように action を実行することによって任意のスクリプトを実行できるわけではない．また，TRIP2 ではテキストから図を生成するときに制約に基づいて位置を決定しているが，そのあと図形間に制約が課せられるわけではないので，恵比寿のように図形単語単位での移動や変形はできない．

恵比寿がすべてを円などの基本的な図形文字を描いて Spatial Parsing していき，図形間に制約を与えることで図形単語を生成するのに対し，ThingLab[17]ではオブジェクト指向の概念に基づいて図形単語を構築

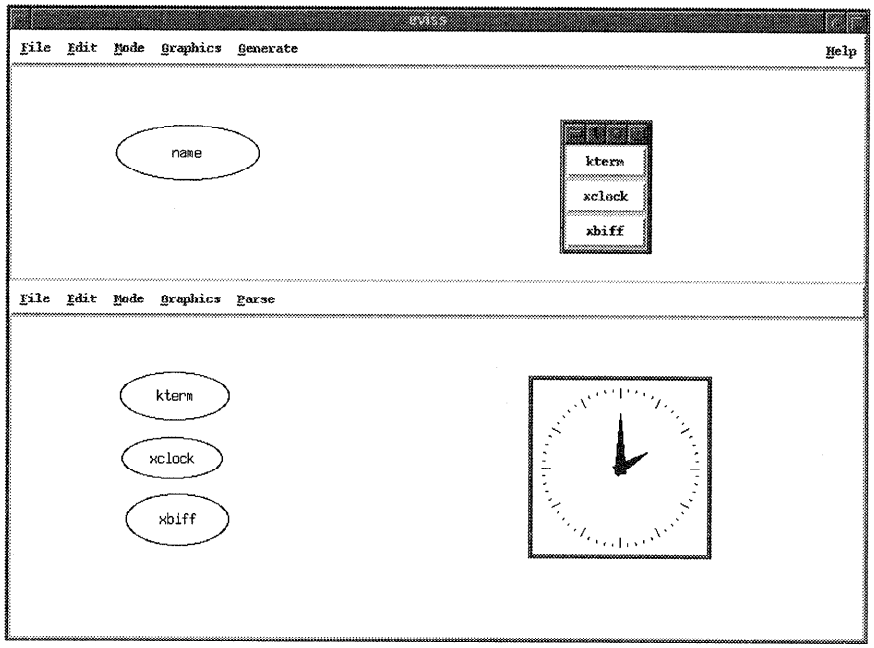


図6 ボタンを作る例
Fig. 6 An example of button creation.

し，図形文字（基本図形）だけでなくすでに定義した図形単語（ThingLab ではオブジェクト）を直接生成することができる．この方法は Spatial Parsing の効率化や図形入力の効率化の面において恵比寿でも参考にしたいと考えている．しかしながらこの方法では文法に基づいて図形間に制約を動的に追加するわけではないので，他のツール，もしくはフリーハンドで描いた図を解析して制約を与えるということはできない．今後この2つの方法のバランスをどこでとるかということが問題となると考えられる．

実行可能制約を用いた図形エディタ[18]では実行可能制約を用いることによって編集がなされたときには提供されている任意の操作が実行されるように指定できる．このことで図形単語ごとの操作を行うことができるが，恵比寿のように図形言語の意味を解析することはできない．

6. おわりに

本研究では図形言語の文法の記述方法の1つである CMG にスクリプト言語を用いて action の概念を導入し，action 付きの CMG を用いて図形言語の文法と動作を与えると定義された図形言語の処理系となるようなシステム「恵比寿」を作成した．本論文では CMG への action の導入と恵比寿について述べ，恵比寿上でアプリケーションを作成する例を示した．

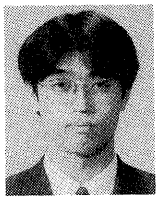
今後は我々が現在開発を行っているビジュアルプログラミングシステム PP[19),20)]など，より実用的で実際的なビジュアルシステムを恵比寿を用いて作成する予定である．

参考文献

1) 垂井忠明，田丸啓吉：デジタル計算回路，朝倉書店 (1969).
2) Korth, H.F. and Silberschatz, A.: *DATABASE SYSTEM CONCEPTS*, McGraw-Hill (1986).
3) Rumbaugh, J., Blaha, M., Premerlani, W., Frederick E. and Lorensen, W.: *Object-Oriented Modeling and Design*, Prentice-Hall International (1991).
4) Ousterhout, J.K.: *Tcl and the Tk Toolkit*, Addison-Wesley (1994).
5) JAVA WHITE PAPERS. Available from `http://java.sun.com:80/docs/white/`
6) Golin, E.J.: Parsing Visual languages with Picture Layout Grammars, *Journal of Visual Languages and Computing*, No.2, pp.371–393 (1991).
7) Costagliola, G., Orcficc, S., Polese, G., Tortora, G. and Tucci, M.: Automatic Parser Generation for Pictorial Languages, *Proc. 1993 IEEE Symposium on Visual Languages*, pp.306–313 (1993).
8) Ferrucci, F., Tortora, G., Tucci, M. and

Vitiello, G.: A Predictive Parser for Visual Languages Specified by Relation Grammars, *Proc. 1994 IEEE Symposium on Visual Languages*, pp.245–252 (1994).
9) Marriott, K.: Constraint Multiset Grammars, *Proc. 1994 IEEE Symposium on Visual Languages*, pp.118–125 (1994).
10) Marriott, K. and Meyer, B.: Towards a Hierarchy of Visual Languages, *Proc. 1996 IEEE Symposium on Visual Languages*, pp.196–203 (1996).
11) Wittenburg, K., Weitzman, L. and Talley, J.: Unification-based Grammars and Tabular Parsing for Graphical Languages, *Journal of Visual Languages and Computing*, No.2, pp.347–370 (1991).
12) Haarslev, V. and Wessel, M.: GenEd – An Editor with Generic Semantics for Formal Reasoning about Visual Notations, *Proc. 1996 IEEE Symposium on Visual Languages*, pp.204–211 (1996). Available from `http://kogs-www.informatik.uni-hamburg.de/~haarslev/publications/`
13) Golin, E.J. and Magliery, T.: A Compiler Generator for Visual Languages, *Proc. 1993 IEEE Symposium on Visual Languages*, pp.314–321 (1993).
14) Chok, S.S. and Marriott, K.: Automatic Construtions of User Interfaces from Constraint Multiset Grammars, *Proc. 1995 IEEE Workshop on Visual Languages*, pp.242–249 (1995).
15) Sannella, M.: Constraint Satisfaction and Debugging for Interactive User Intefaces, Technical Report, University of Wasington (1994). Available from `http://www.cs.washington.edu/research/constraints/sannella-phd.html`, C code is available from `ftp://ftp.cs.washington.edu/pub/constraints/code/SkyBlue/c/`
16) Matsuoka, S., Takahashi, S., Kamada, T. and Yonezawa, A.: A General Framework for Bidirectional Translation between Abstract and Pictorial Data, *ACM Trans. Information Systems*, Vol.10, No.4, pp.408–437 (1992).
17) Borning, A.: The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, *ACM Trans. Programming Languages and Systems*, Vol.3, No.4, pp.353–387 (1981).
18) 服部隆志：編集操作におけるマクロと制約の統合，インタラクティブシステムとソフトウェアIV，田中二郎（編），日本ソフトウェア科学会WISS '96, pp.41–49, 近代科学社 (1996).
19) Tanaka, J.: Visual Programming System for Parallel Logic Languages, *The NSF/ICOT Workshop on Parallel Logic Programming and its Program Environments*, pp.175–186, The University of Oregon (1994).
20) 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，竹内彰一（編），日本ソフトウェア科学会WISS '94, pp.205–214, 近代科学社 (1994).

(平成9年 6 月30日受付)
(平成9年12月 1 日採録)



馬場 昭宏（学生会員）

1973年生．1996年筑波大学第三学群情報学類卒業．現在同大学工学研究科電子・情報工学専攻．ヒューマンインタフェース，ビジュアルシステムの自動生成等に興味を持つ．日本ソフトウェア科学会会員．

田中 二郎（正会員）

1951年生．1975年東京大学理学部卒業．1978年から米国ユタ大学計算機科学科博士課程に留学．Ph.D. in Computer Science．新世代コンピュータ技術開発機構および富士通研究所を経て，1993年より筑波大学電子・情報工学系助教授．研究領域はビジュアルプログラミング，インタラクティブコンピュテーション，ヒューマンインタフェース．また，最近はオブジェクト指向に基づくソフトウェアの設計論にも興味を持っている．1998年7月に開催される本学会主催のAsia Pacific Human Computer Interaction 1998（APCHI '98）プログラム委員長．日本ソフトウェア科学会インタラクティブシステムとソフトウェア（ISS）研究会主査．ACM, IEEE Computer Socicty, 日本ソフトウェア科学会，電子情報通信学会，人工知能学会，計測自動制御学会各会員．