「恵比寿」を用いたビジュアルシステムの作成

馬場昭宏† 田中二郎††

我々はすでにSpatial Parser Generatorを持ったビジュアルシステム「恵比寿」の提案を行っている[1]．恵比寿ではビジュアル言語の文法と動作を記述することによってビジュアルシステムを定義することができる．我々は恵比寿を用いていくつかのシステムを作成したが，その過程においていくつかの必要な機能が欠けていることに気が付いた．恵比寿は文脈自由であるようなCMGしか扱えなかったが，文脈依存であるようなCMGを記述できるようにした．また，部分的な生成規則の保存および追加的な生成規則の呼び出しを可能にした．さらにGUIの作成には必須であるビットマップイメージを図形の1つとして使えるようにした．我々は恵比寿の有効性を示すためには実際にシステムを作成することが重要であると考える．本論文では新たに追加された機能を用いて3つのシステムを恵比寿上に作成した実例について述べる．1つ目は「GUIを作るビジュアルシステム」である．この例ではGUIの構成要素であるウィジェットの配置と，それに関連づけられたバインディングを視覚的に定義することができる．2つ目は「VISPATCH[2]のサブセット」である．この例では図形によるルールの定義および描き換えが実装されている．最後は「HI-VISUAL[3),4)]のサブセット」である．この例では2つのアイコンを重ね合わせることによって動作を実行させることができる．

Creating Visual Systems with "Eviss"

AKIHIRO BABA† and JIRO TANAKA††

We have already suggested the system Eviss, which has a spatial parser generator[1]. We can create visual systems with Eviss by describing grammars and actions of them. We realized that Eviss lacks some necessary functions while we created some systems with Eviss. Since Eviss could only treat context free CMGs, we extended Eviss so that it can treat context sensitive CMGs. We also extended Eviss so that it can save production rules partially, and load production rules incrementally. Final extension is to make Eviss possible to treat bitmap images as a figure element. We believe that it is important to create visual systems in order to show that Eviss is useful. In this paper, we describe three examples of creating visual systems with new facilities of Eviss. The first example is a "GUI creator." Using the "GUI creator," we can define configurations of widgets and their bindings in a visual way. The second example is a subset of VISPATCH[2]. In this example, defining rules visually and redrawing figures are implemented. The third example is a subset of HI-VISUAL[3),4)]. In this example, actions can be executed by overlapping two icons.

1. はじめに

ビジュアル言語（2次元もしくはそれ以上の次元を用いる言語）は様々なフィールドにおいて使用され，テキスト言語と相補的な関係にある．ERダイアグラム，OMTのオブジェクト図[5]，数式，楽譜，テレビドラマの登場人物の関係を表す図などはすべてビジュアル言語の例である．これらはテキスト言語だけでは理解しにくい情報を視覚的に表すことによって理解を容易にしている．

今後ビジュアル言語は計算機上においてもテキスト言語と相補的な関係になり，その実装技術はますます重要なものになっていくものと思われる．現在ビジュアル言語の処理系（ビジュアルシステム）を作成，もしくは変更するのは困難で時間がかかる仕事である．なぜならそれらのシステムは特定の仕様に特化しているためである．

我々はすでに文法と動作を与えることでビジュアルシステムを生成するようなシステム「恵比寿」を提案している[1]．恵比寿はCMG（Constraint Multiset Grammars）[6]に基づいており，Tcl/TkとC言語によって実装されている．Marriottらは文献7)においてChomskyの分類に対応させてCMGをtype0から

† 三菱電機株式会社
Mitsubishi Electric
†† 筑波大学電子情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

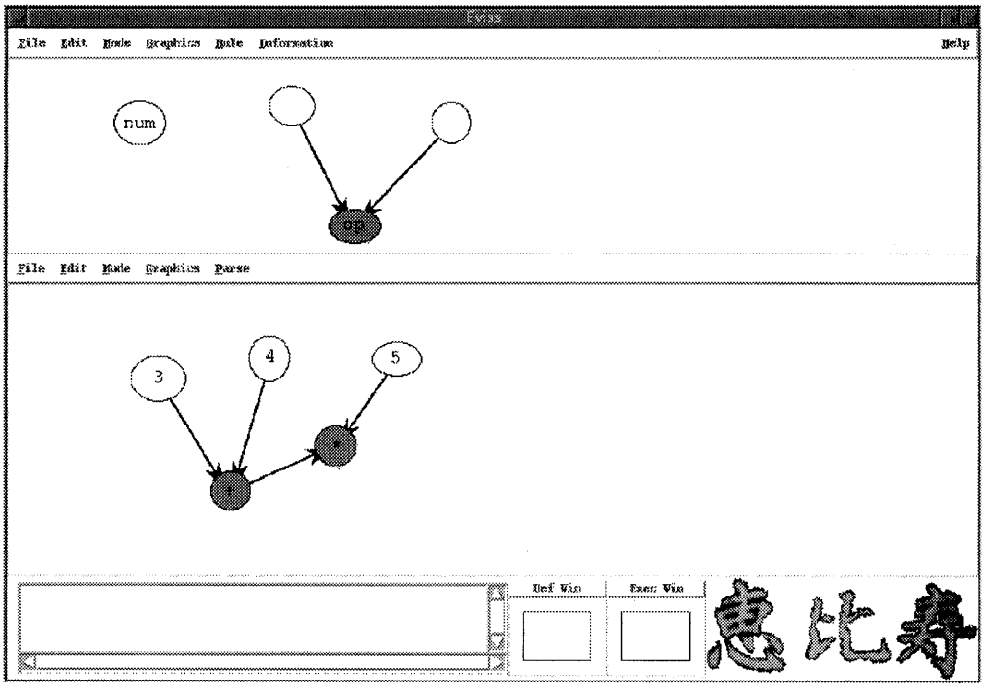


図1 恵比寿
Fig. 1 A snapshot of Eviss.

type3 までにクラス分けしている．この分類によると，文献 1) の時点で恵比寿が扱えるのは type2（文脈自由）である．システムの外観を**図1** に示す．画面の上半分は定義窓と呼ばれる．ビジュアルシステムの作成者は定義窓でビジュアル言語の文法を記述する（定義フェーズ）．画面の下半分は実行窓と呼ばれる．ユーザは解析したい図を実行窓に描く（実行フェーズ）．まず定義フェーズについて述べる．最初に作成者はおおまかな文法（定義するものが何からできているか，およびそれらがどのような関係にあるか）を図を用いて定義する．恵比寿はその図を CMG に似た記法に変換する．次に作成者はそれをモディファイし，必要があれば定義したものが認識されたときに実行されるアクションを記述する．実行フェーズにおいてはユーザは円や長方形などの図形を入力していくことで解析したい図（本論文では図形文と呼ぶ）を描く．解析はインクリメンタルに行うことができる．解析が行われると図形間には制約が課せられ，それらの関係を保存したまま編集を行うことができる．

しかし，実際のさまざまなアプリケーションを作成していくには扱える CMG のクラス，生成規則，図形などに関して恵比寿の機能は不足していた．本論文では 2 章において CMG の説明をした後，3 章ではどのように恵比寿を拡張するかについて述べ，4，5，6 章では拡張した機能を用いてビジュアルシステムを作成した実例について述べる．

2. CMG

本章ではこの後の章で用いられる用語を定義することを目的として，恵比寿における文法の記述の基本となっている CMG とその分類について述べる．

2.1 CMG の構成要素

CMG においては図形文はトークンのマルチセットである．トークンには終端トークンと非終端トークンがある．終端トークンは円やテキスト文字列のような図形である．トークンは半径や位置のような属性を持つ．トークンの種類は記号（もしくはシンボル）と呼ばれる．オブジェクト指向の用語を用いると，記号はクラスであり，トークンはオブジェクトである．CMG は複数の生成規則により構成される．通常は生成規則というと左辺から右辺を生成するものであるが，本論文では生成という言葉で右辺から左辺に書き換えることを表すものとする．

ここでは例として計算の木を考える．図 1 の実行窓にある図は計算の木の例である．この文における終端トークンは 5 つの円，5 つのテキスト文字列と 4 つの直線である．計算の木の文法を定義するためには次に示す 2 つの規則が必要である．

```
1: node ::= C:circle, T:text where (
2:   not_exist L:line where (
3:     C.mid == T.mid &&
4:     C.mid == L.end
5:   )
6: ) {
7:   mid = C.mid;
8: }
```

図2 数を表すノードの生成規則
Fig. 2 A production rule for a node which represents a number.

(1) ノードは円とその中に描かれたテキスト文字列から構成される．ノードに入ってくる直線はない．

(2) ノードは円とその中に描かれたテキスト文字列から構成される．その円に入ってくる2つの直線がなければならない．各々の直線は他のノードにつながっている．

(1)は数を表すノードの定義であり，(2)は演算子を表すノードの定義である．

(1)は**図2**に示すCMGの生成規則として表される．1行目はこの生成規則が「ノード（node）」という非終端記号を定義していることと，ノードが円とテキスト文字列から構成されていることを表している．この例における円とテキスト文字列をnormalの構成要素と呼ぶことにする．normalの構成要素は実際に左辺の非終端記号の部品となるものである．生成規則が適用されると，normalの構成要素として使われたトークンは非終端トークンと置き換えられる．3行目はテキスト文字列の中心と円の中心が一致していることを表している．これは「制約」と呼ばれる．制約は生成規則中でトークン間の関係を表すのに用いられる．

2行目は文のどこかに直線が存在してはならないことを表している．我々はこの例における直線をnot_existの構成要素と呼んでいる．not_existの構成要素は生成規則をより決定的にするために用いられる．もしも直線が図形文のどこかに存在し，制約を満たしていたらこの生成規則は適用されない．制約中でnot_existの構成要素が用いられることがある．このような制約はネガティブ制約と呼ばれる．ネガティブ制約は満たされてはならない制約である．4行目はnot_existの要素を含んでいるのでネガティブ制約である．xをnot_existの構成要素，Cを制約とすると，$\neg(\exists x\, s.t.\, C) \Rightarrow \forall x\, s.t.\, \neg C$である．

7行目はこの生成規則で定義されるノードの属性`mid`（中心）が`C`（円）の属性`mid`と同じ値を持つ，つまり中心が一致していることを表している．右辺のシンボルのトークンが存在し，制約を満たすときに非終端トークンが生成される．

```
1: node ::= C:circle, T:text where (
2:   exist N1:node, N2:node,
3:       L1:line, L2:line where (
4:     N1.mid == L1.start &&
5:     N2.mid == L2.start &&
6:     C.mid == L1.end &&
7:     C.mid == L2.end &&
8:     C.mid == T.mid
9:   )
10: ) {
11:   mid = C.mid;
12: }
```

図3 演算子を表すノードの生成規則
Fig. 3 A production rule for a node which represents an operator.

(2)は**図3**に示すCMGの生成規則として表される．2行目と3行目は2つのノードと2つの直線が図形文のどこかに存在しなければならないことを表している．我々はこれらをexistの構成要素と呼んでいる．existの構成要素はnormalの構成要素と図形文のどこかに存在するその他のトークンとの間に成り立つ制約を記述するのに用いられる．もしこの例においてノードと直線がnormalの構成要素として用いられると，それらは生成規則の左辺のノードの一部として扱われてしまう．ノードは1つの円と文字列から構成されるものとしたいので，この扱いは明らかに適切ではない．

計算の木の例では使用されていないが，allと呼ばれる構成要素もある．allの構成要素は図形文の中にある指定された種類のすべてのトークンから構成されるマルチセットを作るために用いられる．計算の木の例では引数の数を2つと決めているが，不定個の引数をとる場合はallの構成要素を用いると簡単に生成規則を記述することができる．allの構成要素を使わないとしたら再帰的な生成規則を書かなくてはならない．

2.2 CMGの分類

文献7)におけるCMGの分類について述べる．ただし，ここで示すのは主要な部分のみである．

type3 正規文法．

type2 文脈自由文法．normalの構成要素のみ用いることができる．

type1 文脈依存文法．normalに加えてexistの構成要素を用いることができる．

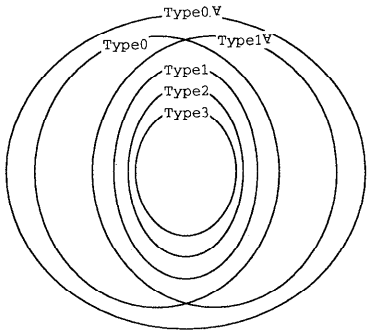


図4 CMG のクラスの包含関係
Fig. 4 Inclusive relation between CMG classes

type0 （本論文でいうところの）生成時に1つの非終端トークンの他に複数のトークンを生成するような生成規則を書くことができる．

type1∀ type1 に加えて全称記号を用いた制約を書ける．すなわち normal，exist の他に not_exist の構成要素を用いることができる．

type0∀ type0 に加えて全称記号を用いた制約を書けるもの．

これらの包含関係は図4のようになる．

3. 恵比寿の改良

3.1 扱える言語のクラスの拡大

文献1) に示したこれまでの恵比寿は type2 であるような CMG しか扱えなかった．これを改良し，最も記述力の強い type0∀ であるような CMG をも扱えるようにした．文献1) では type1∀ であるはずの計算の木を扱っているが，これは数を表すノードと演算子を表すノードの色を変えて非決定性をなくすことによって not_exist の使用を回避したり，直線をノードの一部とする（つまり normal の構成要素とする）ことによって exist の使用を回避して実現していた．

本研究と関連した研究として文献8) では type1∀ であるような CMG の実装が試みられている．このシステムは図形間に制約を課すことはできるが，生成規則中にアクションを記述することができないために，実用的であるとはいえない．

3.2 追加的な生成規則の読み込みと生成

生成規則についてはそれをオブジェクト指向的に解釈することが可能である．すなわち，新しい生成規則を定義することは normal の構成要素を親クラスとするようなサブクラスを定義することである．これを陽にサポートすることによって以前作成したクラスを再利用しやすくなり，効率的に文法を定義できるようになる．改良した恵比寿では指定した生成規則のみをファイルに保存することができるので，クラスライブラリのようなもの（実際は生成規則のライブラリ）を作ることができる．また，呼び出しはファイル単位で追加的に読み込むことができる．

ここで，この追加的な読み込みは解析中も含めて随時行うことができる．すなわち，トークンの持つ値を用いて解析中に新たな生成規則を追加することもできる．

3.3 扱える終端記号の追加

GUI を作成する際にビットマップイメージは必須であるといえる．我々は恵比寿で GIF イメージを終端記号の1つとして扱えるようにした．

4. GUI を作るビジュアルシステム

本章では恵比寿における「GUI を作るビジュアルシステム」の構築について述べる．この例では拡張された恵比寿で type1∀ の CMG を記述できる機能を利用している．この例を type2 もしくは type1 で記述できる可能性もあるが，以前の恵比寿で計算の木を定義したようにトリッキーな方法が必要となる．より広いクラスの CMG を扱えることにより，言語の定義を自然に行うことができる．

4.1 GUI の構成要素

GUI の構成要素としてはウィジェットとバインディングを考える．

ウィジェットとしては以下の4つを考える．

フレームウィジェット 中に他のウィジェットを持つことでウィジェットの階層を作る．

スクロールバーウィジェット テキストウィジェットの表示範囲を制御する．

ボタンウィジェット マウスのクリックによって押されるとあらかじめ指定された手続きを呼び出す．

テキストウィジェット 内部にテキスト文字列を表示する．

ウィジェットにはバインディングを施すことができる．バインディングは特定のイベントが発生したときにそのイベントに関連付けられた手続きを呼び出す機構である．

4.2 言語の定義

本節では 4.1 節で述べた GUI を記述するためのビジュアル言語の定義について述べる．このビジュアル言語を定義するためには全部で 13 個の生成規則を必要とする．

テキストウィジェット以外のウィジェットで以下の3つの生成規則が必要である．

フレームウィジェット（Frame） 中が塗り潰されてい

ない長方形.

スクロールバーウィジェット（Scroll） 中がオレンジ色で塗り潰された長方形.

ボタンウィジェット（Button） 中が黄色で塗り潰された長方形の中にテキスト文字列が書かれたもの. テキスト文字列はボタンのラベルと押されたときに呼び出される手続き名を表す.

テキストウィジェット 中が赤で塗り潰された長方形.

スクロールバーウィジェットはテキストウィジェットと組み合わせたときのみ意味を持つので，スクロールバーウィジェットとテキストウィジェットは1つの非終端記号 **TextW** として定義する. **TextW** はスクロールバーなし，水平スクロールバー付き，垂直スクロールバー付き，両方のスクロールバー付きの4つの生成規則がある.

ウィジェットを表す長方形に色を付けるのには次の2つの理由がある.

(1) ビジュアル言語のそのものの定義を容易にするため.

(2) 図形文の可読性を高めるため.

バインディング（**binding**）はテキスト文字列がウィジェットの左上の点に直線で結び付けられたものによって表される. バインディングのための生成規則は1つである.

このように定義されたビジュアル言語によってGUIを記述する例を示す. **図5** はGUIの例であり，**図6** はそれを表すビジュアルプログラムである. ここで，図5のテキストウィジェットでマウスの左ボタンをクリックすることで手続き **clear_text** が呼び出されることとする.

次にウィジェットを組み合わせてGUIを作るための生成規則について述べる. 最も基本的なGUIを定義するためには3つの生成規則がある. それぞれの生成規則はフレームウィジェット，テキストウィジェット，ボタンウィジェットを normal の構成要素として持つ. さらに，GUIは2つのGUIを再帰的に水平または垂直に並べることで作られる. そのためGUIの再帰的な生成規則は2種類ある. それぞれの生成規則は1つのフレームウィジェットと2つのGUIを normal の構成要素として持ち，構成要素である2つのGUIはフレームウィジェットの中になければならない.

前述のように，GUIにはバインディングを施すことができる. バインディングのための生成規則はテキスト文字列と直線を normal の構成要素として持つ. 直線の一方の端点はテキスト文字列の中心にあり，他方の端点はGUIの左上の点になければならない

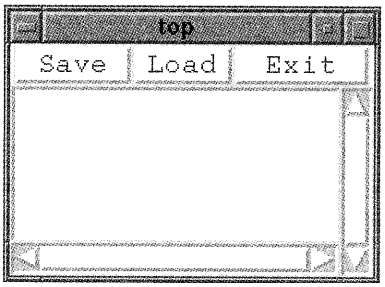


図5 GUIの例
Fig. 5 An example of GUI.

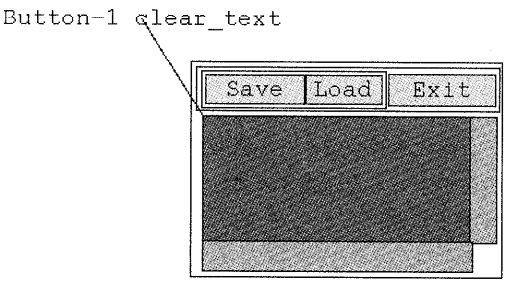


図6 図5を表すビジュアルプログラム
Fig. 6 A visual program that represents Fig. 5.

（図6）. 図6において，テキスト文字列は「**Button-1 clear_text**」であり，GUIはテキストウィジェットである. テキスト文字列はイベントの名前とそのイベントが発生したときに呼ばれる手続きの名前である. 図6において，イベントは **Button-1** であり，手続きの名前は **clear_text** である.

4.3 実　　装

以上述べた生成規則を恵比寿上に作成した. 作成したシステムでは解析が行われる過程でGUIの内部表現が作られ，解析終了後に内部表現をもとに実際のGUIを生成する. 我々は内部表現から実際のGUIを生成するために約250行のTcl/Tkの補助コードを書いた.

5. VISPATCHのサブセット

本章では恵比寿上における「VISPATCH[2)]のサブセット」の構築について述べる. フルセットの記述も可能であると思われるが，恵比寿の新しい機能の有効性を示すという観点からはサブセットで十分である.

この例では解析中に新たな生成規則を生成する機能を用いている. また，この例はtype0∀である. つまり，解析中に新たなトークンを生成する. そのため，より記述力の低いクラスで実現することはできない. これまでの恵比寿でも新たな図形を生成することは

できたが，これは画面の表示だけであり，内部の表現（トークンのデータベース）はそのままであった[1]．今回の拡張により内部の表現も書き換えられるようになり，この例を実現することが可能となった．

5.1 VISPATCH

VISPATCHは図によって表されたルールに基づいて図を描き換えていくビジュアルシステムである．描き換えはユーザまたはシステムによって起こされたマウスのクリックやドラッグのようなイベントによって開始される．VISPATCHプログラムの例を図**7**に示す．VISPATCHのプログラムは大きく3つ（イベントセンサ，ディスパッチ図形，ルール領域）に分けることができる．

図7において，左側の影が付いた長方形がイベントセンサ，中央にある2重線の矢印がディスパッチ図形，右側にあるのがルール領域である．ルール領域では複数のルールを定義することができる．1つのルールは水平に並べられた2つの長方形から成る．左側の長方形はヘッドと呼ばれ，右側の長方形はボディと呼ばれる．ルールの中で細い矢印はドラッグイベントを表す．イベントセンサにおいてイベントが発生すると，ディスパッチ図形によって結び付けられたルール領域にあるルールの中でヘッドに表された条件が成り立っているものがあるかどうかチェックされる．もしもあれば，VISPATCHはイベントセンサにあるその図形をボディにある図形と置き換える．図7において，ドラッグイベントがイベントセンサ内で発生したら，直線が描かれる．なぜならルール領域にある上の方のルールではドラッグイベントのみがヘッドにあり，ボディには直線があるからである．直線の始点はドラッグイベントの始点であり，直線の終点はドラッグイベントの終点である．これはルール中のヘッドとボディの間を結んでいる線（Equivalence Point）によって表されている．もしも直線の終点からドラッグイベントが発生した場合は，直線とドラッグイベントは長方形に置き換えられる．ここで，この長方形の対角線は直線の始点とドラッグイベントの終点を結んだものである．なお，図7はこの2つの図形がイベントセンサに描かれた状態を示す．

図7の例ではボディの図形を指定するための点はヘッドの図形によって与えられていたが，図**8**ではポイントジェネレータを用いて新たな点を生成することによって星型を描いている．ポイントジェネレータは基準となる図形との関係（回転，拡大縮小，その両方など）から新たな点を生成する．たとえば図8では円の中に×印が描かれたポイントジェネレータによって，基準となる直線（これはヘッドのドラッグイベントによって決められる）に対しての角度と，その直線との長さの比を指定して新たな点を生成する．

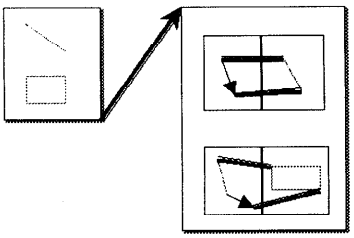


図**7** VISPATCHプログラムの例
Fig. 7 A VISPATCH program.

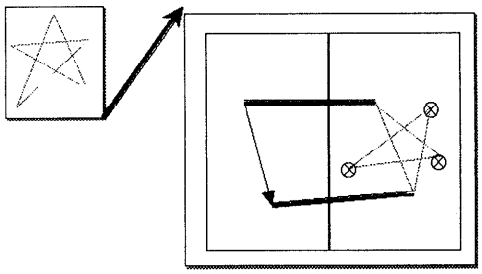


図**8** VISPATCHにおける点の生成
Fig. 8 Generating a point on VISPATCH.

5.2 実　　装

我々は恵比寿上にVISPATCHのサブセットを実装した（図**9**）．実装の対象となる機能は以下のものとした．

扱う図形 直線，長方形，ディスパッチ図形（ディスパッチ図形は二重矢印ではなく，太い矢印とする）

扱うイベント ドラッグイベント，クリックイベント（「C」という文字列で表す）

ポイントジェネレータ 図8で述べた回転および拡大縮小を行うもの（我々の実装では円の中に×印が書かれたものではなく，ただの円で表現する）

また，イベントセンサおよびルール領域は影付きの長方形ではなく，ただの長方形とした．

前述のように，VISPATCHは描き換えをイベントによって開始する．つまりVISPATCHは*event driven*である．一方，恵比寿は解析および描き換えを図形の描画，削除，変更などによって開始する．つまり恵比寿は*data driven*である．恵比寿が解析を開始するために，イベントセンサでイベントが発生したらルールのヘッドにあるのと同じ図形をイベントセンサに描くこととした．

恵比寿上にVISPATCHのサブセットを作成するために必要な生成規則の数は24個である．まずヘッドおよびその内部のもののための生成規則が次の6個であ

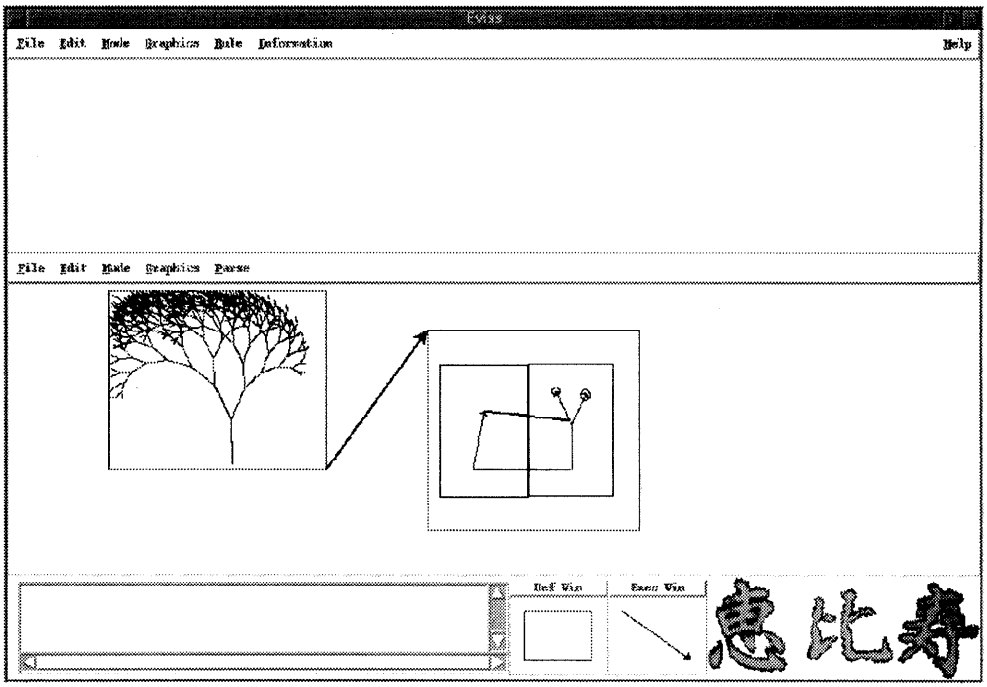


図9 恵比寿上に実装されたVISPATCHのサブセット
Fig. 9 A snapshot of a subset of VISPATCH implemented with Eviss.

る．ヘッド（**Head**）はVISPATCHにおいては1つの長方形で表されるので，1つの長方形をnormalの構成要素として持つ．ヘッドのすぐ右側にはボディとなるべき長方形がなくてはならない．また，ヘッドおよびボディとなるべき長方形はルール領域となるべき長方形に囲まれていなければならない．ヘッドの中の直線，長方形，ディスパッチ図形はそれぞれ非終端記号**LineH**, **RectH**, **DispatchH**として定義される．ヘッドの中のドラッグイベント，クリックイベントはそれぞれ**DragH**, **ClickH**として定義される．

ボディおよびその内部のもののための生成規則は次の6個である．ボディ（**Body**）は1つの長方形である．ボディのすぐ左にはヘッドがなければならない．ボディの中の直線，長方形，ディスパッチ図形はそれぞれ**LineB**, **RectB**, **DispatchB**として定義される．ボディの中のドラッグイベントは**DragB**として定義される．ポイントジェネレータ（**PointGen**）はボディの中の円として定義される．

ルール領域関係のための生成規則は次の4個である．ルール（**Rule**）は隣どうしのヘッドとボディとして定義される．ヘッドとボディの図形の点の対応を示す直線（**EqPoint**）はヘッドに始点を持ち，ボディに終点を持つ直線として定義される．ルール領域の枠（**RuleFrame**）は左上の点にディスパッチ図形の終点を持つ長方形として定義される．ルール領域（**RuleArea**）はルール領域の枠およびその中のすべてのルールとして定義される．

ディスパッチ図形関係のための生成規則は次の3個である．ディスパッチ図形（**Dispatch**）を定義するためにまず折れ線矢印（**Arrow**）を定義する．折れ線矢印は再帰的に定義される．再帰の終了条件となる生成規則は「ただの矢印は折れ線矢印である」というものである．もう1つの生成規則は「折れ線矢印はただの矢印の終点に折れ線矢印がある」というものである．ディスパッチ図形は折れ線矢印の始点と終点に長方形があるものとして定義される．

その他，ルール関係では次の2個の生成規則がある．イベントセンサ（**EventSensor**）は右下にディスパッチ図形の始点がある始点を持つ長方形として定義される．ルールの中の図形およびイベントセンサすべてを集める生成規則が**AllRule**である．

イベントおよびそれにともなう描き換えを表す生成規則として次の3個がある．ドラッグイベント（**Drag**）は終点と始点がイベントセンサの中にある矢印として定義される．描き換えの結果として生成される，始点のみがイベントセンサの中にある矢印は**Garbage**という生成規則により削除される．ドラッグイベントのうち，マウスの移動距離が小さいものがクリックイベン

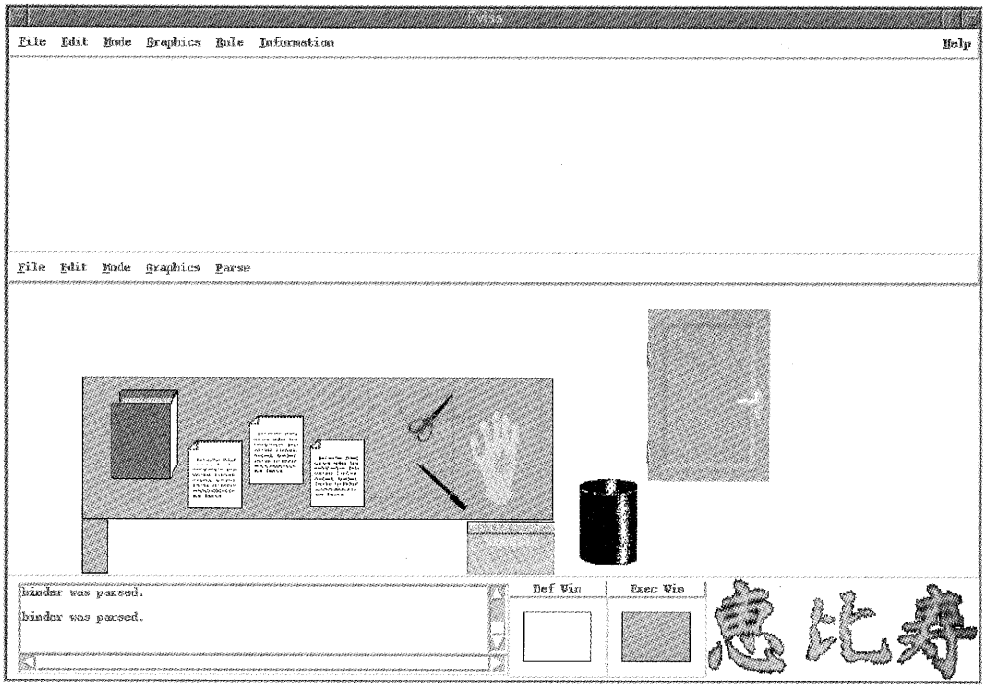


図10 恵比寿上に実装したHI-VISUAL
Fig. 10 A snapshot of a subset of HI-VISUAL implemented with Eviss.

トである．したがって，十分短いドラッグイベントをクリックイベントに描き換える生成規則**DragToClick**を定義した．

AllRuleの中ではアクションとしてVISPATCHの各々のルールから恵比寿の生成規則を生成する手続きを呼び出す．1つのVISPATCHのルールに対して1つの恵比寿の生成規則が生成される．このための補助コードはTcl/Tkで500行弱である．ルール領域は自分がどのイベントセンサとつながっているかを知っているので，生成される恵比寿の生成規則では自分がつながっているイベントセンサの中でイベントが発生したときのみ制約が成り立つように記述されている．したがって，2つのイベントセンサA，Bがあるとして，Aでイベントが発生してもBでは描き換えは起こらない．ヘッドに描かれた図形が恵比寿の生成規則ではnormalの構成要素となる．また，ボディに描かれた図形は恵比寿の生成規則ではアクション中で新たに生成される図形単語となる．

もともとVISPATCHはルール（つまりVISPATCHプログラム）もデータ（VISPATCHによる描き換えの対象）として扱うことができたが，恵比寿上に実装することによってVISPATCHの処理系そのものとVISPATCHプログラムとを恵比寿の生成規則という同じレベルで記述したことになる．

6. HI-VISUALのサブセット

本章では恵比寿上における「HI-VISUAL[3),4)]のサブセット」の構築について述べる．フルセットの構築については5章と同様の理由によりここでは考えない．この例では終端記号としてGIFイメージを使用する機能を用いている．

6.1 HI-VISUAL

HI-VISUALは2つ以上のアイコンの重ね合せによって動作を指定するビジュアル言語である．紙アイコンの上に鉛筆アイコンを重ねるとエディタが起動し，紙アイコンをコピー機アイコンの上に重ねるとファイルをコピーする，という具合である．この関係は一意に定まるとは限らない．たとえば紙と鉛筆の組合せとしては紙で鉛筆を包むというものも考えられる．アイコンはすべて物体（はさみ，鉛筆など）を表し，動作（切る，書くなど）を表すものはない．アイコンはオブジェクトであり，アイコンの重ね合せはオブジェクト間のメッセージの送信であると見ることができる．

6.2 実　　装

HI-VISUALの最も中心的な概念である2つのアイコンの重ね合せによる動作の指定を恵比寿上に実装した（**図10**）．1つのアイコンは1つのGIFイメージをnormalの構成要素として持ち，各アイコンは1つの

```
1: edit_file ::= exist F:file, P:pen where (
2:   overlap(F,P)
3: ) {
4:   emacs F.name
5: }
```

図11 ファイルとペンが重なったときに適用される生成規則の概略
Fig. 11 An outline of a production rule which is applied when a file icon and a pen icon are overlapping.

表1 恵比寿上に実装したHI-VISUALが持つ機能
Table 1 Functions of HI-VISUAL implemented with Eviss.

下のアイコン	上のアイコン	動作
ファイル	ペン	エディタを開いてファイルを編集
バインダ	ファイル	ファイルをバインダにしまう
バインダ	手	バインダの中にあるファイルを見せる
ゴミ箱	ファイル	ファイルを捨てる
ゴミ箱	バインダ	バインダを捨てる

生成規則として表される．アイコンのうち，同種のアイコンでもそれぞれを区別する必要があるもの（ファイル，ディレクトリなど）は属性として「名前」を持っている．1つの動作もまた1つの生成規則として表される．動作を表す生成規則はnormalの構成要素として2つのアイコンを持つ．2つのアイコンが重なっているときにその生成規則が適用され，アクションとして指定した動作が実行される．例としてファイルアイコンとペンアイコンが重なったときにエディタemacsを起動するという生成規則の概略を図**11**に示す．動作を表す生成規則には重なりの順番（どのアイコンが画面の手前に描かれているか）が考慮される．これによってたとえば「紙で鉛筆を包む」というのと「鉛筆で紙に書く」というのとを区別する．アイコンはファイル，バインダ，手，ペン，ゴミ箱，ドア，引き出し，はさみの8個を実装した．これらを重ね合わせたときの動作としては**表1**に示す5個を実装した．したがって，このシステムは合計13個の生成規則から成る．

2つのアイコンを重ね合わせるHI-VISUALのサブセットはtype1である．なぜならばオブジェクトは使用後も再び使用可能でなければならない（たとえば鉛筆と紙を重ねた後にそれらが1つのものになってしまっては困る）ので，existの構成要素が必須になるためである．3つ以上のアイコンの重ね合せを実現するためには，生成規則を一意なものにするためにnot_existの構成要素が必要になる．したがって，この場合のクラスはtype1∀となる．

7. 関連研究

恵比寿はトークンを解析して，それらの間に制約を与えることで非終端トークンを作る．ThingLab[9]では基本オブジェクト（恵比寿における終端トークン）だけでなく，複合オブジェクト（恵比寿における非終端トークン）を直接生成できる．しかし逆にThingLabでは複数のオブジェクトをまとめて1つの複合オブジェクトとすることはできない．もし我々がThingLabのように非終端トークンを直接生成することができたら，より効率的な解析と図形入力が恵比寿で可能となるだろうと思われる．

SPARGEN[10]は文法の定義にOOPLGと呼ばれるオブジェクト指向の記法を用いている．恵比寿はCMGを基にしているが，1つの生成規則を1つのクラスの定義であると見なすとSPARGENと同様にオブジェクト指向の方法を用いてビジュアルシステムを作成できる．SPARGENはコンパイラ型であるが恵比寿はインタプリタ型であるという点において2つのシステムは大きく異なる．恵比寿がインタプリタ型であることの利点は，アクションの実行中に動的に命令の生成，変更が可能であるという点である．

8. おわりに

本論文ではシステム恵比寿の拡張された機能（文脈依存文法への対応，部分的な生成規則の保存および追加的な生成規則の呼び出し，ビットマップイメージの使用への対応）について述べた．また，拡張された機能を用いて「GUIを作るビジュアルシステム」，「VISPATCHのサブセット」および「HI-VISUALのサブセット」という3つのシステムを恵比寿上に作成した実例について述べた．これらの実例を見ても分かるように，いずれも13から24個程度の生成規則を記述するだけでかなり実用的なビジュアルシステムを記述できることが分かる．

今後の課題の1つとしては解析と図形入力の効率化があげられる．前章で述べたように，今後ThingLabのように非終端トークンを直接入力できるようにしたいと考えている．これによって生成規則を図形のみを用いて定義できるようになる可能性もある．

本論文で述べた恵比寿およびアプリケーションの例は完全に実装されており，WWW経由で以下のURLから入手可能である．

`http://www.softlab.is.tsukuba.ac.jp/~iplab/software-j.html`

なお，本研究は著者の1人である馬場が筑波大学

工学研究科在籍中に田中と行った研究に基づくものである．

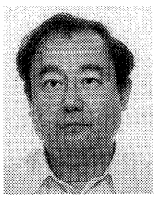
参考文献

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 (1998).
2) Harada, Y., Miyamoto, K. and Onai, R.: VISPATCH: Graphical rule-based language controlled by user event, *Proc. 1997 IEEE Symposium on Visual Languages* (1997).
3) Yoshimoto, I., Monden, N., Hirakawa, M., Tanaka, M. and Ichikawa, T.: Interactive Iconic Programming Facility in HI-VISUAL, *Proc. 1986 Workshop on Visual Languages*, pp.34–41 (1986).
4) Hirakawa, M., Nishimura, Y., Kado, M. and Ichikawa, T.: Interpretation of Icon Overlapping in Iconic Programming, *Proc. 1991 IEEE Workshop on Visual Languages*, pp.254–259 (1991).
5) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: *Object-Oriented Modeling and Design*, Prentice-Hall International (1991).
6) Marriott, K.: Constraint Multiset Grammars, *Proc. 1994 IEEE Symposium on Visual Languages*, pp.118–125 (1994).
7) Marriott, K. and Meyer, B.: Towards a Hierarchy of Visual Languages, *Proc. 1996 IEEE Symposium on Visual Languages*, pp.196 203 (1996).
8) Chok, S.S. and Marriott, K.: Automatic Construction of User Interfaces from Constraint Multiset Grammars, *Proc. 1995 IEEE Workshop on Visual Languages*, pp.242–249 (1995).
9) Borning, A.: The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, *ACM Trans. Prog. Lang. Syst.*, Vol.3, No.4, pp.353–387 (1981).
10) Golin, E.J. and Magliery, T.: A Compiler Generator for Visual Languages, *Proc. 1993 IEEE Symposium on Visual Languages*, pp.314–321 (1993).

(平成10年 5 月12日受付)
(平成10年12月 7 日採録)

馬場 昭宏（正会員）

1973年生．1996年筑波大学第三学群情報学類卒業．1998年筑波大学大学院工学研究科中退．修士（工学）．現在三菱電機株式会社PC統括事業部にてデスクトップPCの製品化業務に従事．ヒューマンインタフェース，特にビジュアルシステムの自動生成に興味を持つ．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．1984年Utah大学計算機科学科博士課程修了．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年より筑波大学に勤務．現在，電子・情報工学系教授．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．その他に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．ACM，IEEE Computer Society，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．