ビジュアル言語の空間問題

原田康徳
NTT CS 基礎研究所

ビジュアル言語はトークンを空間に配置することで，わかりやすい，初心者向き，などと思われている．しかし，テキスト言語と比べて面積あたりの情報量が少なく，プログラムが複雑になると広い領域が必要になってしまう．そこで，それを解決するための一手法を提案する．

１．はじめに

我々は，GUI のプログラミングをビジュアルに行うシステムを開発している．そこでは，すべての情報が見えていて，見えているものは何でも自由に触って操作できる，という考え(Visibility)を基本としている．

従来のオブジェクト指向をベースとした GUI システムでは，オブジェクトが内部状態を持ち，メッセージをやり取りすることでその内部状態を更新し，それらの計算の結果，画面の表示を作り出す．マウスによる操作によってそこに仮想物体があるかのように，オブジェクト群を作り出す．しかし，この仮想物体は，操作と表示との間にギャップがあり，そこがうまくプログラムされていないと，動きそうで動かない，というものができる．

それに対して，Visibility に従うと，まずコンピュータのメモリと画面上の表現を双方向に関係づけるような可視化を作り，人間とプログラムはその可視化された情報を操作することで計算が進められる．Scott Kim はビットマップをその表現形式としてビットマップをメモリとするコンピュータでその可能性を示した．萩谷らはテキストをメモリとする Boomborg 計算を開発した．

我々が開発した Vispatch では，ベクトル図形をメモリとしたモデルであり，ベクトル図形によってプログラム(書き換えルール)とデータが表現されている．マウス操作によって書き換えルールが起動し，対象のデータを書き換えることで，計算が進む．図形の色で記号とメタ記号とを区別し，2 色を用いて自分自身を編集できるシステム(自己拡張可能な図形エディタ)が実現できた．

Vispatch は GUI の基本的なプログラムを図形だけで実現できたので，この方法を拡張することで，原理的には Microsoft Office に代表されるようなアプリケーションも実現可能である．

最初の Vispatch はデモシステムとして Java の上で開発された．しかし，巨大なシステムに発展させるためには，根本的なアーキテクチャをこの Visibility システムに合うように設計し直す必要があった．このアーキテクチャとは，まず，共通のデータ表現形式があり，

その形式を可視化するツール群を用意する．次にそのデータを操作するプロセス群がそのデータを書き換えてゆき，全体のアプリケーションが構成される．
スペルチェックを例にとって説明すると，オブジェクト指向によるアーキテクチャではすべてのテキストオブジェクトはスペルチェックの方法を知っている，ということになる．それに対し我々の方法では，テキストデータとそれを操作するプログラム群は独立して存在し，スペルチェックの計算だけをするプロセスがテキストデータに対して起動し，チェックをする．この方法では，後から様々な計算の機能を簡単に追加することができるのである．このようなプログラミングモデルを Visibility Programming と名付けた．ここで，開発されたシステムは，グラフをメモリとするシステムで，それを複数のプロセスで共有できるようにするものとなった．
オブジェクト指向はデータ構造の詳細を隠蔽した，というだけではなく，継承や多相といったプログラミング上の重要な技術も同時に実現した．Visibility Programming では，データ構造は共通化し，誰でもアクセスができるようにしている．つまり，データ抽象はない．しかし，プログラミング上，継承や多相は行いたい．そこで，ダイナミックデータ抽象という概念を提案した．これは，与えられたデータをひとつのオブジェクトのまとまりのように解釈し，その解釈をクラスと定義する．解釈の階層がクラス階層となり，自然に継承が実現できる．また，クラスごとにメソッドを定義できるため，多相も実現できる．
オブジェクト指向との違いは，最初にオブジェクトがあるのではなく，最初にデータの集合があり，その一部をオブジェクトとして見なしたいときに，ある解釈を与えて，それに対応するメソッドを実行する，ということである．
この考えは，どのような言語にでも実装可能であるから，C 言語に対して実装した CCC を提案した．CCC の場合，解釈は条件式で行う．クラスはひとつの条件式で定義され，その式が成立したときに，データはそのクラスであると判断する．メソッドが呼び出されると，あるクラス C の条件が成立し，そのすべてのサブクラスの条件が成立しないとき，クラス C のメソッドを実行する．
同様に，Vispatch の場合はルールのヘッド部がオブジェクトの解釈と見なすことができ，それを書き換えるということがオブジェクトの状態を変化させることになる．

２．Vispatch を拡張する際の問題点

Vispatch は次のような問題を含んでいた．

a)空間の狭さ

そもそも一般のビジュアル言語が持っている問題として，面積あたりの情報量の低さが挙げられる．Vispatch で作った自己拡張可能な図形エディタは，画面の大部分をプログラムで占めているので，図形エディタの本来の目的である，図形を作成するためのスペースが少ないのである．
もちろん，多くのビジュアル言語が採用している，プログラム空間と実行空間に分けるな

ど，複数の空間を用いる方法も考えられる．しかし，空間を分けてしまうことで，Vispatch の最大の特徴であるプログラムとデータを区別しないシステムの特徴が失われてしまう．

b)包含関係を用いた構文

Vispatch では見ためのプログラムが簡単になるように，図形間の包含関係を言語の要素として暗に用いていた．たとえばルールは 2 つの矩形から構成されるが，左の矩形の中に描かれた図形をルールのヘッドとし，それに対象がマッチすると，右の矩形の中に描かれた図形を生成する．また，マウスにより発生したイベントは，それを囲む矩形領域に対応づけられたルールで書き換えが行われていた．

このような包含関係を用いているにもかかわらず，Vispatch のパターンマッチは図形の接続関係(端点を共有していること)でのみ行っていた．そのため，Vispatch の構文の意味を考慮したような書き換えることはできなかったし，別の書き換えの結果，偶然に他の領域を侵してしまうことで，意味が変わってしまう，という問題もあった．

例えば Vispatch でスタックを表現することは簡単で,矢印図形を連結して行くだけでよい．しかし，それが伸びていった先に他の図形があった場合，その領域を侵してしまい，双方の意味が変わってしまう可能性がある．そのため，あらかじめ誰も描画してはいけない空間を予約する，といったことをしなければならない．

以上の問題をまとめると,Vispatch は 2 次元空間をメモリモデルとしてもつ言語であるが，その空間の仮想化がうまくいっていないということになる．これは，C と Lisp のメモリの違いと似ている問題である．C でも配列によるスタックを作成できるが，あらかじめ予約した領域を超えた場合には，他のデータを破壊してしまうため，それをチェックするプログラムが必要であった．一方で，Lisp ではメモリ構造を仮想化しているので，論理的に無限に成長するデータ構造を簡単に実装できた．

３．画面問題

上で述べた問題は，2 次元的な見てくれの問題と，論理的な仮想化が必要という問題に分けられる．それぞれを解決するために，2 つの仕組みを新たに導入した．

３．１．Trans 図形

Trans 図形は，矩形を構成する 2 点 A,B と，対象の位置を指示する 1 点 C から構成され，C が指している領域に描画されている図形を矩形 AB に映し出す．また，逆に矩形 AB で生じたイベントは C が指す領域に変換される．この仕組みは表示上の効果しかなく，もともとの画面に描画されていた論理構造には影響を与えない．Trans 図形は入れ子にすることができ，それにより再帰的な構造を作り出すことができる．

Trans 図形は次のように用いる

a)スクロール

ボタンを押す事に，Trans 図形の対象点を上下に動かすような書き換えをさせることで，対

象の領域をスクロールして表示させる効果を持つ.

b)プログラムの隠蔽

Trans 図形を用いると，紙を切り貼りするように，その巨大な紙の好きな領域を自由にレイアウトして見せることができる. Trans 図形は対象の論理的な構造には一切影響を与えないので，その配置は見ためのわかりやすさにのみ専念できる.

c)ツールキット

特定の領域で，マウスの操作に反応する様々なプログラムが用意されている．その際，上記のプログラムの隠蔽を行うことで，特定の領域での動作だけに注目することができる．これは，ツールキットのボタン，ラジオボタンなどとして使うことができる．すなわち Vispatch によってツールキットを実現することができる.

３．２．点とスコープ

これまで包含関係を用いて，構文を表現していたが，それをすべて，関係を明示することにした. つまり，点がある領域に含まれているという，暗黙のルールは用いないのである. そのかわりすべての点はどれかのスコープに属しており，ルールのヘッドとボディはそれぞれ独立したスコープを持っている，と考える.

一般に，GUI でのスコープは，矩形領域を定義して，座標の平行移動の機能と同時に図形のクリッピングの機能も持っている．しかし，すでにクリッピングの機能は上述の Trans 図形が行っているので，ここでは座標の平行移動の機能のみを持たせることとする.

以上のことから，点をスコープとして用いることとした．すなわち，

1) すべての点は一つのスコープに属する.

2) スコープは点である.

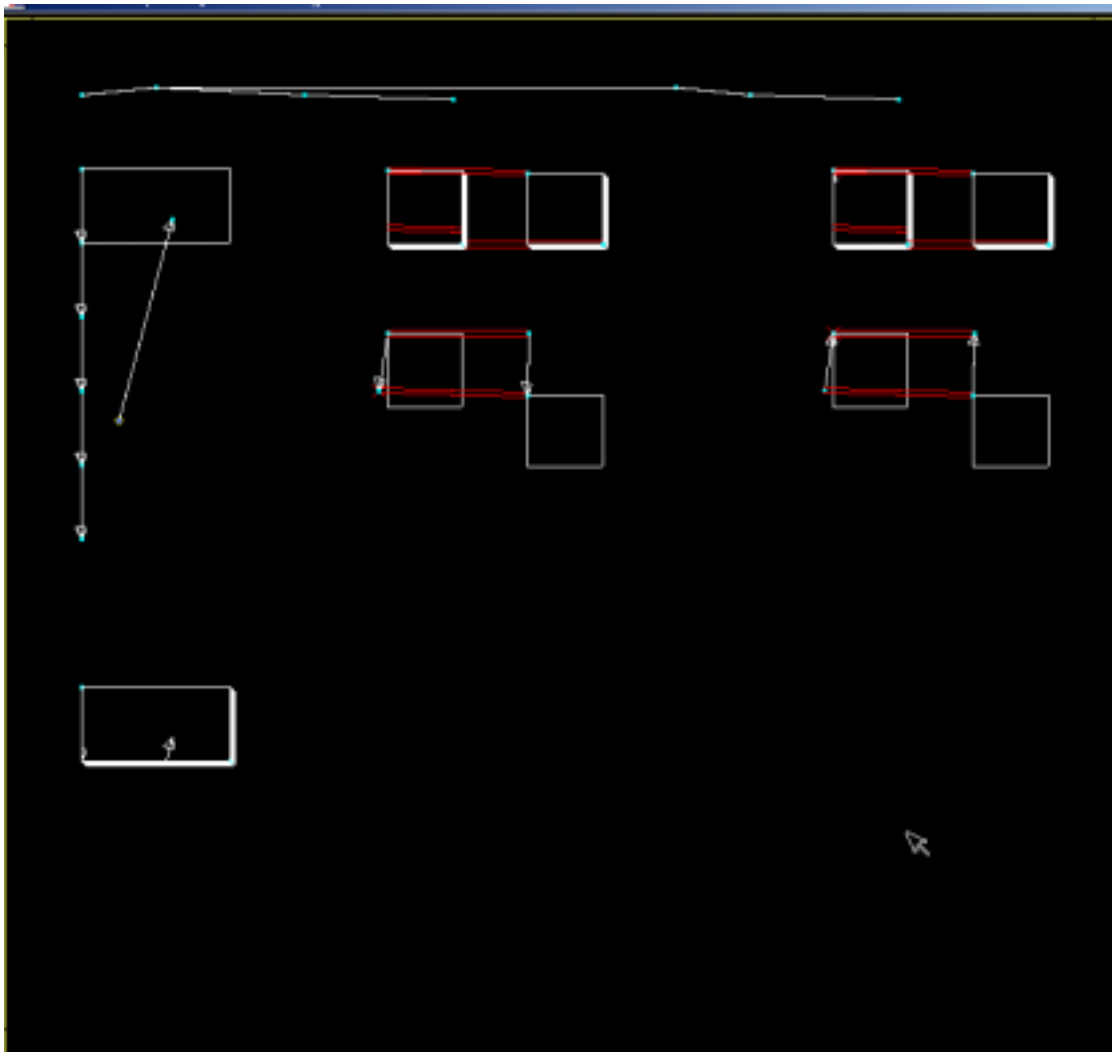
となり，結果的にある 1 点をルートとする，点の木構造ができる．この点の木構造は，ビジュアル言語の構文の一部として用いられることと，点の大域座標をそれが属しているスコープの点の座標から相対的に決められる，ということに用いられる.

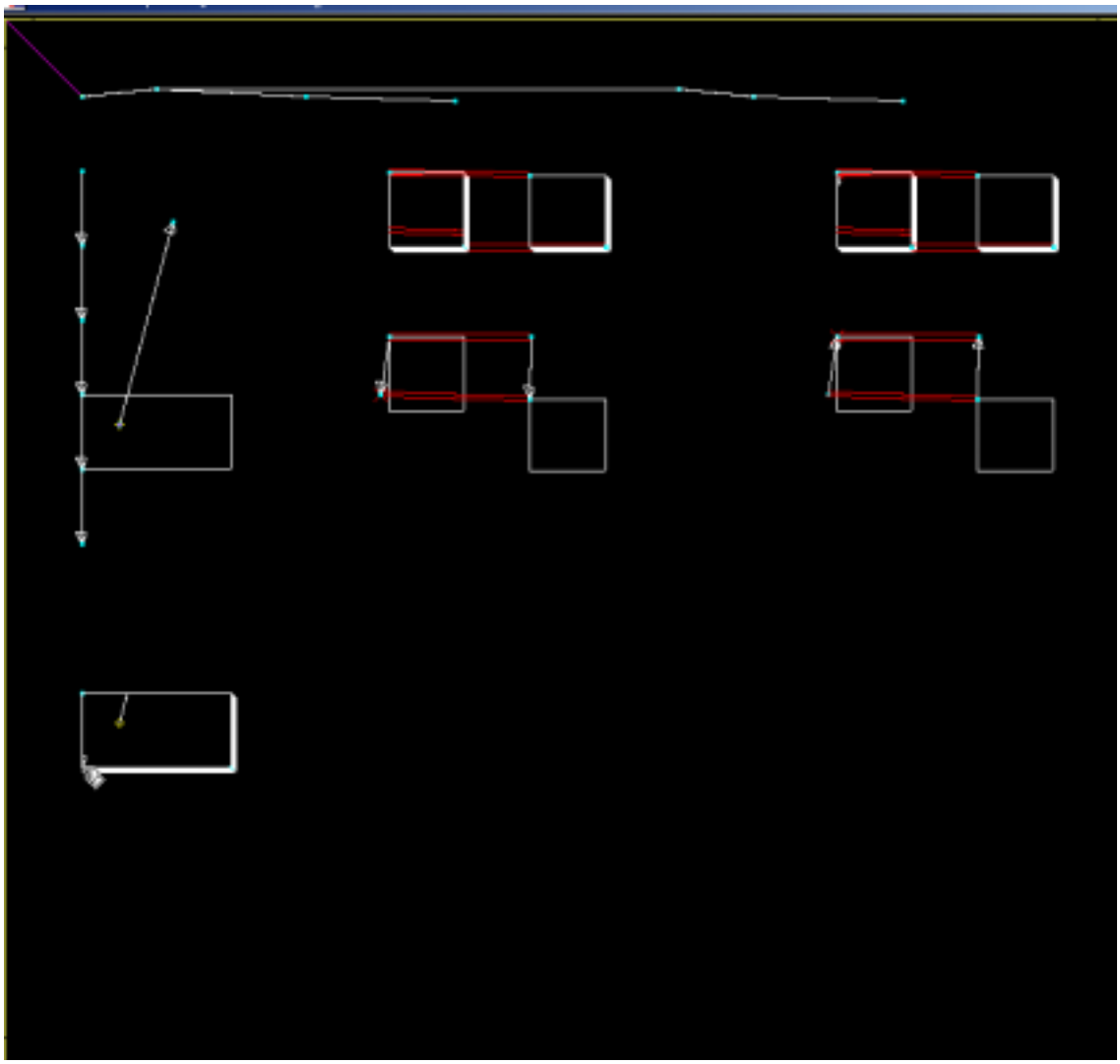
点のスコープが，包含関係では決められないということは，インタラクション上の問題を引き起こす．たとえば，点が何もない領域でマウスの操作をした場合には，イベントが生じた場所で，点の構造を生成する必要があるが，その際にどのスコープに属する点を生成するべきなのかがわからないからである.

ある点がどのようなスコープの階層に属しているのか，という情報はきわめてわかりにくい．そこで，それらの情報がわかりやすく提示されるように，ユーザとのインタラクションを工夫することにする.

４．実装

上記の機能を実装した Vispatch を紹介する．これは，Visibility Programming の一つの応用として実装した.





図は，新しい Vispatch におけるプログラムの例である．
各点がどのスコープに含まれているか，という情報は，その点をマウスで触れることにより，色の線が表示されて示される．また，その情報は直後の点の生成時のスコープの情報としても用いられる．
1 つのルールは，ルール自身の ID となる点の他に，書き換え対象，ヘッド，ボディをあらわすスコープをそれぞれ点として持っている．すなわち，4 点からなる図形である．
陰付きの矩形と影なし矩形は組になっていて，Trans 図形の表示部と対象部を示している．
この例では，画面のスクロールの機能を実装したプログラムを示している．大きく 4 つの部分に別れている．上の部分は 2 つのルール図形が並んでいる．中央と右側にある図形がルールを示している．左側にある図形が書き換え対象となる図形である．書き換え対象，それぞれのルールのヘッドとボディに，Trans 図形が使われている．このプログラムの意味は，書き換え対象の Trans 図形の対象部から出ている矢印図形の矢をクリックすると，対象部が矢の先に移動する(中央のルール)．また，対象部に入っている矢の先をクリックす

ると，今度はその矢の付け根に移動する(右のルール)．このクリックする場所によって対象部が矢印に沿って上下に移動することになり，その横に表示されている図形をスクロールして見ることができる．
Trans 図形は対象の図形を表示部に映すだけではなく，表示部で起きたイベントを対象部に変換する働きもする．そのため，左下の表示部の矩形内で，左下の角をクリックすると，それは，対象部から出た矢印の先でクリックしたことになり，中央のルールが発火する．また，表示部の矩形内で，左上の角をクリックすると，対象部に入る矢印の先でクリックしたことになり，右のルールが発火する．これらのクリックのたびに対象部が上下に移動するので，連続したクリックは連続したスクロールになる．

５．まとめ

見えているものが，コンピュータのすべての状態である，という Visibility の考えかたでシステムを設計する際の，2 次元空間の問題を解決するプリミティブを示した．
今後は，大規模な応用を考えている．