VispatchKids
― 子供向け図形書き換え言語 ―

原田 康徳

ルールに従って図形書き換えを行う子供向けビジュアル言語 Vispatch Kids を紹介する．この言語の設計方針は，Visibility に従ってすべての状態を見えるようにし，プログラムの実行を分かりやすくしたこと，また，図形のパターンマッチングを極めて柔軟にし，エンドユーザが想像しやすい動作を心がけたことである．その自然さを追求した結果，あるプログラムに期待する動作は，それをどのような対象に対して適用したかで変化する．

VispatchKids
― A Graphics Rewriting Language for Kids ―

HARADA, Yasunori

We introduce a new visual language for kids, Vispatch Kids, that rewrites graphics according to rules. The features of this language are visualization of the state of computation and flexible pattern matching of graphics. They make an execution of this language easy to understand.

1. はじめに

コンピュータが広く普及したが，コンピュータの真の能力である，プログラムを変更することで様々な動作ができる魔法の機械，はまだエンドユーザに開放されていない．エンドユーザがコンピュータを使用する理由は様々であろうが，たとえば電子メールを遊びで利用するような感覚で，プログラミングを遊びに利用するという需要もあるだろう．現におもちゃのロボットの動作などをエンドユーザが記述できるシステムもある．
エンドユーザ向け，特に子供向けの言語を開発する理由は大きく２つある．一つは，コンピュータの動作の基本を理解させるという教育的な効果，もう一つは究極のマルチメディアマシンの追求である．これらは背反する理由ではないが，その位置付けの違いが言語のデザインに大きな影響を与える．
例えば教育という観点を全面に押し出すと，レジスター（変数) や条件分岐，といったノイマン型コンピュータの動作をいかに分かりやすく教えるかというところに焦点がおかれる．その結果，既存のプログラミング言語に制約を与えて可視化したというものになりがちである．
一方，究極のマルチメディアマシンを求めるということは，これまでのコンピュータのメカニズムにとらわれずに，設計しなおすことになるかもしれない．これでできるものは，結果的に教育効果があるかもしれないけれど，今のコンピュータのサブセットを教えるというのではなく，計算の根底にある基本的な考え方を身につける，ということになるであろう．
ここで目標にしたデザインポリシーのは次の通り．

1. 計算の課程が目に見える．
2. 計算機固有の知識を必要としない．
3. 適当にやってもなんらかの動作をする．
4. 楽しい．

VispatchKids は Visibility の考え方にした

NTT コミュニケーション科学基礎研究所
NTT Communication Science Labs.

がって設計された．Visibility とは計算機の内部状態をすべて見せ，触れるようにするというもので，Scott Kim が最初に提案し [5]，彼のシステムVIEWPOINTで，それが実証された．その後，いくつかのシステムがこの Visibility に従って開発され，その有効性が確かめられた．

Vispatch[6]はVisibility に従った図形書き換え言語である．VispatchKidsはこのVispatchを子供向けに設計しなおしたものである．この論文の構成は次の通りである．２節では，VispatchKidsがどのような動作をするのかを簡単な例によって示している．３節ではそれを設計する．４節ではより具体的に実装する方法について述べ，５節で考察し6節で比較する．

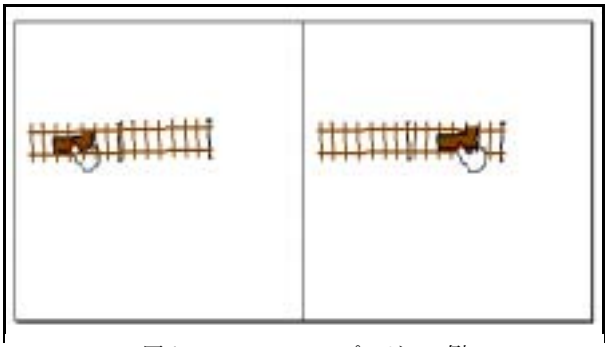

図１VispatchKidsのプログラム例
左側が書き換え前，右側が書き換え後のパターンを表している．ここでは2つの線路があり，電車の上でクリック（手の図形はクリックの意味）すると電車は進み，ひきつづきクリックをするという意味である

2. VispatchKids

ここではプログラム例によってVispatchKidsを紹介する．プログラムは図形の書き換えルールの集合で構成される．図１は一つの書き換えルールの例である．左側が書き換え前，右側が書き換え後の図形である．ここで手の形をした図形はクリックを表している．このルールの意味は線路が２つつながっていて，その上の電車でクリックをすると，電車が移動して，さらにその電車の上でクリックする，ということである．

さて，このようなプログラムを図２のような図形の上で動作させる．実行するには，図２の電車の上でクリックする．一回の書き換えで電車が隣の線路に移動し，新たにクリックが生成されることから，連続して書き換えが生じ，電車が線路上を移動するアニメーションができる．さらに，図３では，線路が交差しているが，この部分では電車はまっすぐ進んで欲しい．

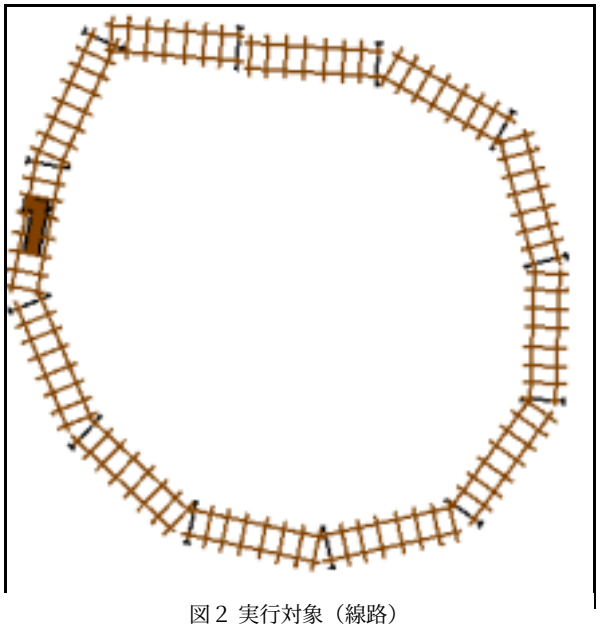

図２ 実行対象（線路）
この図が図１のプログラムに関係付けられているとき，この電車の上でクリックすると，図１のルールが発火し，アニメーションが動き出す．ここでは，電車が線路上を移動する．

重要なことは，用意されていた書き換えルールは，線路が１直線につながっていたものであったが，書き換えの対象となる図形では，線路が曲がっていたり，交差していたりするということである．このように用意したパターンにマッチしなかった場合柔軟にマッチングを行うのである．

図４では，亀がクリックによって配置されるプログラムの例である．上のルールで，亀をクリックすることで，1匹が3匹に分割する．下のルールで重なった2匹の亀をクリックするとその2匹は消える．最初画面には1匹の亀を置

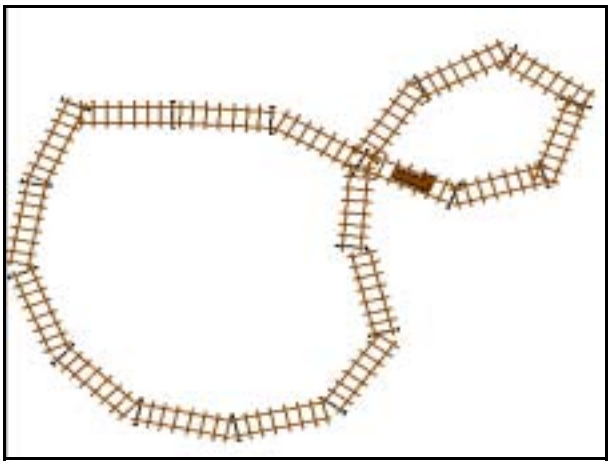

図３ 交差した線路

たとえ線路が交差していても，図１のプログラムは予想するような動き方をする．電車はできるだけ直線的に進もうとする．

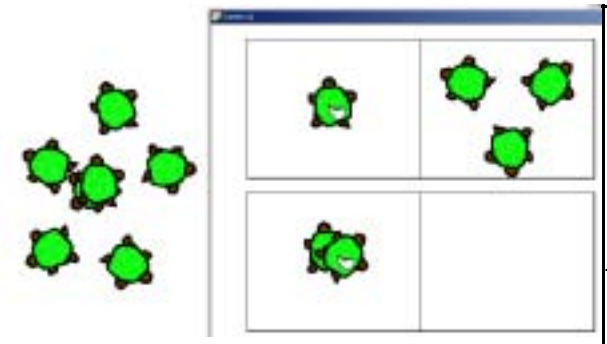

図４ 亀の増殖
上のルールは亀をクリックすると，３匹に増殖する
下のルールは重なった2匹の亀をクリックすると2匹は消える
右の実行例は何度かのクリックで亀が増減している

き，それを適当にクリックしてゆくことで，亀の並びが変化して行く．

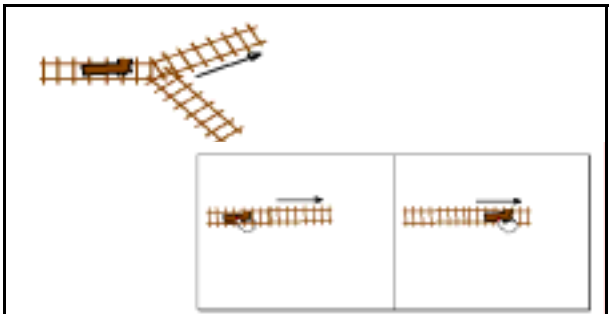

図５ 分岐
ルールに矢印を追加したので矢印のある線路に分岐する

図５では，線路の分岐のあいまい性を無くすために，矢印記号を導入している．ルールには矢印が追加され，線路の分岐で矢印のある方向に進むように動作する．これはさらに，図６のようなルールを追加することで，クリックで矢印の向きを変更することができる．図５と組み合わせることで，分岐に置かれている矢印の向きを変えることができ，その結果電車の進路を自由にコントロールできることになる．
図７ではインターネットのシュミレーションの例が書かれている．ここではトラックの荷台に

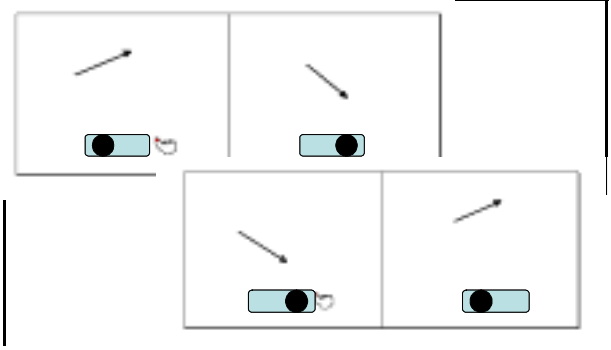

図６ ポイント切り替え
スイッチの図形をクリックすると矢印の向きが変わる

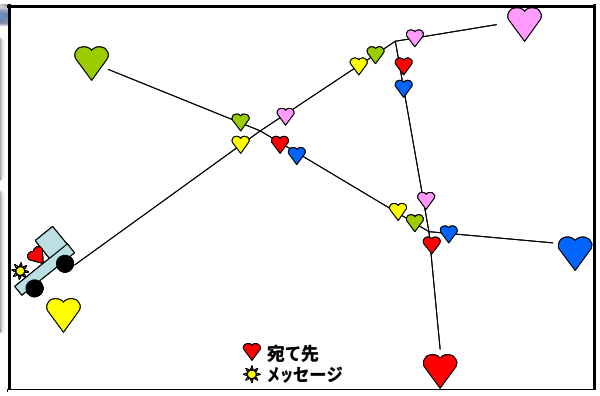

図７ インターネットのシュミレーション
トラックには宛て先とメッセージを積んである．交点（ルータ）には各宛て先への行き先（ルーチングテーブル）が書いてありそれにしたがって，トラックは移動する

メッセージとその宛て先が積まれてあり，直線に従って移動する．直線の交点はルータを表現しており，そこから出ている直線につながっている宛て先を示す記号が書かれている．これはルーチングテーブルに相当する．このような図に対して与えるルールは，図８のようになる．図８では，荷台にあるハートと同じ色の線にトラックは進む，という意味である．
これらのプログラムとその実行のストーリはエンドユーザにとって自然に感じると仮定する．もちろん，図５の矢印の導入などは恣意的であるし，図６に至っては，プログラミングの経験のある人独特の発想であるかもしれないし，エンドユーザはそのような発想はできない，という意見もあるであろう．しかし，これはすべてのエンドユーザ向け言語が抱えている問題であり，教育という観点からみると難しい議論が必要である．しかし，従来のビジュアル言語から比較すると，十分障壁は低いと考えている．そこで以下の議論は，これ以上エンドユーザに対する自然さに踏み込まずに，この仮定を踏まえて進めることにする．

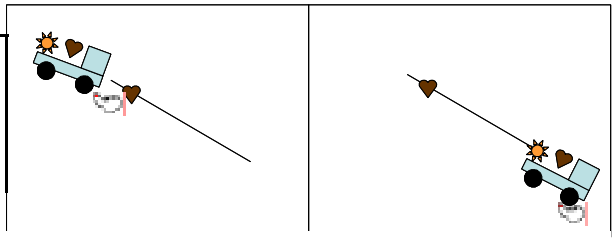

図８ シュミレーションのルール
トラックでクリックされると，トラックの荷台にあるハートマークと同じ色の線を進む

3. 設計

以上のプログラムとその実行例をどのように実装するかを考える．もっとも単純で無意味な解は，上で述べた例に限りなく近い場合しか動作しないという実装である．そうならないように設計したつもりでも，気が着きにくいものである．そこで，もう少し問題を整理して見る必要がある．

図１では２つの線路が接続されているというルールが書かれ，それは図２や図３のような線路の接続に対しても動作する必要がある．図１が表現したかったことを一言でいうと，滑らかに連結している２つの線路があり…，ということになる．決して，２つの線路が直線状に並んでいてそれらが水平になっている，という意味でこのルールを用いているのではないのである．一方，図５のポイント切り替えでは，クリックによって矢印の向きが変更されるというルールが書かれている．このルールで表現したいことは，右上に向かっている矢印を右下に向かうように書き換える，という意味である．言い換えると，矢印の向きは絶対的な方向を用いている．

この二つのルールの例をみても，それらが互いに矛盾している点に気づく．一方は角度を相対的に扱っているのに対し，もう一方では絶対的に扱っているからである．このような場合，これまでの言語設計で行われてきたアプローチでは，角度をどのように扱うのかを明示してきた．例えば，図１のルールには相対座標を示すアイコンや２つの線路間の角度は厳密ではないというアイコンを追加するし，図５のルールでは，角度は絶対座標を示すアイコンを追加するのである．相対座標と絶対座標のように背反する条件の場合はいずれか一方をデフォルト値として，その指定の負担を減らす方法をとることも可能であろう．いずれにしても，このようなアプローチでは，ユーザにとって，理解しなければならないことが増えてしまう．

同様に図８で宛て先を示すハートの色は対象の図７では使われていない色を使うことが重要である．この色がある特定の色を指すのか，それともルール内で閉じている（同じ色であることさえ守られればよく，同一の色の変数を示している）ことでよいのか，ということを区別しなければならない．多くの解はここで変数と定数を区別する構文を用意する．ビジュアル言語的には，変数を示す特別な色を導入することになろう．しかし，この場合もユーザに強いるよけいな知識が増えてしまう．

図９曲がった線路
線路が左に曲がっていると電車は移動するが，新たなクリックはしないので，アニメーションは停止する

また，他のルールの存在によって解釈が異なってくる場合もある．例えば，図９は線路が左に曲がって置かれている場合のルールが書かれている．ここでの動作は，電車は進むが，新たにマウスクリックを生成しないので，電車のアニメーションはここで停止する，という意味である．さて，このルールの他に線路が右に曲がって置かれているルールや線路が直線になっている図１とがいっしょにあった場合，どのように解釈されるべきであろうか．図１が単独であった場合には，「線路が滑らかに連結していた場合」という解釈であったが，曲がった線路と一緒にあると「線路が直線で連結している場合」という解釈をして欲しい．このように，単独では広い意味のルールとなるが，近いパターンを持つルールがあると，それらの違いを際立たせるような解釈が必要となる．

問題を整理すると，ルールをどのように解釈するかは，そのルールがどのように使われるか，他にどのようなルールが存在するか，によって異なる，ということである．

このような動作を，ユーザにとって不自然にならないように実装する必要がある．これには，例外的な処理を数多く追加して行くというよりは，むしろ単純な処理の組み合わせによってシステムが構築されていることが重要と考える．このことで，未知のルールと対象の組み合わせに対しても，容易にその動作を想像できるようになる．

4. アルゴリズム

以下にVispatchKidsの動作の概略をのべる．

1. それぞれのルールの左辺に対して，そこから抽出できる図形制約をすべて列挙する．たとえば，線路Aと線路Bの端点が一致している，線路Aと線路Bの向きが同じ，線路Aは水平，線路Bは水平，線路Aは茶色，線路Bは茶色，線路Aと線路Bは同じ色，線路A

の中心に電車がある，電車と線路Aは同じ向き，電車と線路Bの中心とは５０ピクセルはなれている，といった具合にである．ここで抽出された制約は冗長であることに注意する．

2. あらかじめ各制約の種類に応じて点数を決めておく，例えばXの色がYという制約を満足すると５０点，X上の特徴点AとY上の特徴点Bが一致しているという制約は，その２点の距離の関数として定義され，一致したときに最大値，そこからずれると急速に減少するもの，などである．
3. 実行対象でマウスイベントが生じると，現在有効なルールがすべて集められ，イベントをマッチングの種として，すべてのルールでマッチングを試みる．
4. マッチングは，１で抽出した制約が満足しているかどうかを調べる．ルール全体を通じて，ルール上の図形が対象の図形とですべての組み合わせを試みる．各ルール，各図形の組み合わせごとに，２で定義した点数の合計を求め，最も点数の高いルール，図形の組み合わせが選択され，発火する．

マッチングは全ての制約を満足する必要はないので，１．で抽出する制約は冗長であっても意味がある．Eviss [9] はこれと似た動作をするが，抽出した制約はすべて図的構文として用いられ，全ての制約が満足する必要がある．

VispatchKidsはここでマッチしたルールの左辺（ヘッド図形）に対応した対象の図形を削除し，ルールの右辺（ボディ図形）を対象上に生成する．図形を生成する，ということは，その図形のすべての属性（位置，傾き，色など）を決めなければならないということである．それらの属性は左辺と右辺の関係を抽出することによって決められる．まず，ある属性が右辺内および左辺と右辺間とで強い関係を持っている場合は，その関係をそのまま用いる．例えば，図１では右辺に３つのオブジェクトとクリックを新たに生成することになっている．生成される２つの線路は左辺と右辺とでまったく同じ属性を持っているので，左辺にマッチした対象の図形をそのまま使う．それに対して生成される電車は線路の中心で線路と同じ向きに配置されているし，色は，左辺にあった電車と同じ色をしている．これらの関係を用いて例えば図２に対して動作させたとき，線路の傾きに沿って電車が配置されるように属性が決められなければならない．

また，図４の上にあるルールでは生成される３匹の亀に対しては位置と傾きの属性のヒントが殆どない．この場合は左辺の亀との相対的な関係を利用してなんとか属性を決めなければならない．その際，左辺と右辺の関係，マッチングで各制約の点数などをもとに，属性を計算する必要がある（図１０）．

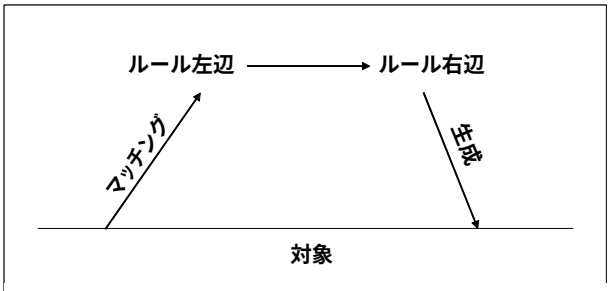

図１０ ルールと対象の関係
対象，ルール左辺，右辺の関係で，様々な属性がどれくらいの強さで関係を持っているかに依存して，対象に生成される図の属性が決まる．

5. 考察

VispatchKids を用いると様々な計算機の基本的な仕組みを表現できる．例えば，クリックイベントは計算のステップを表現しているが，右辺で複数のイベントを生成することができるし，実行中にユーザが別の図に対してクリックすることもできる．これらはイベントがスレッドのように振舞っていることに相当する．また，図形のパターンには包含関係が表現できるので，ルールの継承が実現できる．さらにマウスに色の状態をもたせ，赤いクリック，青いクリック，のようにすることで，メッセージの種類が表現でき，多相が実現できる．図形の個数で数を表現し，加算などが再帰的に定義できる．これらは，一般に書き換え言語であれば持っている性質であるが，VispatchKidsの柔軟なパターンマッチのおかげで，それらの性質がいっそう強調される．

マッチングや生成の処理が極めてコストのかかるという点に注意して欲しい．例えば，変数と定数，絶対と相対といった従来ならばプログラマに明示させていたような情報も実行する対象や，他のルールの存在によって左右されてしまう．このことはユーザが考えなければならないことを減らすけれども，ユーザの期待していた動作と異なってしまう可能性も残されている．さらに，このように動的にプログラム解釈が変

わるという性質は，新たな polymorphism を示唆している．

VispatchKids は子供向けということで実行環境の工夫も凝らされている．図形は自由に回転平行移動して配置されるのであるが，従来のエディタでは，モードを切り替える（回転移動モードと平行移動モード），つまみを用意する（ここをつまむと平行移動，ここをつまむと回転移動）という方法がとられていた．それに大してVispatchKidsではオブジェクトを引きずるという操作法を導入した．これは，机の上に置いたオブジェクトを指一本で引きずったように動作する．つまり，平行移動と回転移動が混合した動作をする．この利点は，余計な説明をせずにすぐに操作できるということである．

6. 比較

CoCoa[2]とAgentSheet[7]はアイコン書き換え型のビジュアル言語である．アイコンは格子状にならび，アイコンの並びでルールを表現し，対象全体で，その並びを発見したら，書き換える．これらの言語で問題なのは，この基本的な仕組み（アイコンの並びで情報を表現する）が貧弱であるため，アイコンに内部状態やスクリプトを導入してしまっている点である．そのため，純粋なビジュアル言語というよりは，テキストとのハイブリッド言語といえるし，通常のテキスト言語が持つ問題をそのまま継承してしまっている．さらにこの手のハイブリッドシステム（例えば Micromedia 社の製品など）に共通して言える欠点は，スクリプトが各オブジェクトに分散して存在し，他人の作ったプログラムがどのような動作をするのかが分かりにくくなっている．これはオブジェクト指向そのものの問題とも言える．

BITPICT[3] と Visulan[8]はビットマップの書き換え言語である．対象からビットマップのパターンを見つけだし，書き換えて行くことで計算がすすむ．Visulanはマウスとのインタラクションを表現するために，現在のマウスボタンの状態が反映されるパターンを用意し，そのパターンをルールに書くことで，マウスボタンの状態を書き換え言語の中に導入することに成功している．これのアプローチはVIEWPOINTのキーボードとのインタラクションの表現にも見られる．これら２つの言語は安易にスクリプトを導入せずにすべてビットマップで頑張っている点は評価できる．しかし，パターンマッチが単純なビットマップのマッチングであり，ピクセルの変数の概念などもないため，この言語が楽に表現できる範囲は限られている．

ChemTrains[1]はグラフ書き換え言語である．グラフで表現されたパターンを対象から見つけると，それを書き換えて行く．グラフの持つ柔軟性と情報表現能力の高さから，高いプログラム表現能力を持つ．

Vispatchはマウスイベントで書き換えを制御したグラフ書き換え言語である．また，プログラムを図形だけで表現したため，プログラムを書き換えることプログラムが実現できた．Vispatch以外の書き換え言語が対象からパターンにマッチする部分を探し，見つけると書き換えるという，いわゆる一般に書き換えと呼ばれる実行形式であるのに対し，Vispatchではマウスイベントによって書き換えを制御している点が新しい．これによって，インタラクティブリフレクションが実現できたのである．

ChemTrainsとVispatchはグラフを書き換えの対象としたことで，計算の能力は格段に向上した．しかし，ユーザの分かりやすさという点ではあまりよくない．グラフの接続関係という情報は分かりにくいからである．それに対してVispatchKidsは図形の位置，向き，色によってパターンマッチする．より，人間の持っている図形の知識に近い動作をすると言ってよい．しかし，色や図形の種類を増やすことで，グラフ表現の能力に近づけることは可能であるので，潜在能力は高い．

7. まとめ

本稿では子供向け図形書き換え言語 VispatchKids を紹介した．基本的な設計方針は，実装の難しさよりも，エンドユーザの直感を重視した．現在一部が実装されており，簡単なデモを実行することができる．

参考文献

[1]B.Bell, and C.Lewis : ChemTrains: A Languages for Creating Behaving Pictures, VL'93.

[2]A. Cypher, and D.C. Smith : KidSim: End User Programming of Simulations, CHI'95.

[3]G.W. Furnas : New Graphical Reasoning Models for Understanding Graphics Interfaces, CHI'95.

[4]Ken Kahn: ToonTalk — An Animated Programming Environment for Children, Journal of Visual Languages and Computing, pp.197-217, June, 1996.

[5]S.Kim : Viewpoint : Toward a Computer for Visual Thinkers, Stanford University, 1988.

[6]Y.Harada, K.Miyamoto, R.Onai: VISPATCH: Graphical rule-based wwwlanguage controlled by user event, VL'97.

[7]Repenning, A. and J.Ambach, Tactile Programming : A Unified Manipulation Paradigm Supporting Comprehension, Composition and Sharing, VL'96.

[8] 山本格也 ： ビットマップに基づくプログラミング言語 Visulan, WISS'95.

[9] A. Baba J. Tanaka : Eviss : a Visual System Having a Spatial Parser Generator, IPSJ Journal Vol.39 No.05—023.