ビジュアルシステム生成系 Rainbow

丁 錫泰　　田中 二郎

ISE-TR-00-173

2000 年 8 月 3 日

〒305-8577 茨城県つくば市天王台 1-1-1
筑波大学電子情報工学系
インタラクティブプログラミング研究室(iplab)
E-mail: {joung jiro}@iplab.is.tsukuba.ac.jp

ビジュアルシステム生成系 Rainbow

丁 錫泰*　　田中 二郎

〒305–8577 茨城県つくば市天王台 1-1-1

筑波大学電子情報工学系インタラクティブプログラミング研究室 (iplab)

TEL: (0298)53-5165　FAX: (0298)53-5206

email: {joung jiro}@iplab.is.tsukuba.ac.jp

1 はじめに

ビジュアルシステム生成系とは、図形言語の文法を定義することで、ビジュアルシステムを生成するシステムのことである [1] [2] [3] [4] [5]。ビジュアルシステムは、空間パーサを用いて図形をパーシングし、図形を処理するシステムである。

情報を視覚化し図形として表示すると、理解や記憶が容易である。しかしながら、情報の視覚化が有用であるかどうかは、そのレイアウトの仕方に強く依存している [6]。ユーザが手作業により直接図形をレイアウトする方法は、時間がかかり、あまり良い結果が得られない。たとえ小さな図形でも、描かれた図形を見て修正することを繰り返すような場合などは、手作業には自ずから限界がある。

我々は、ビジュアルシステム生成系にレイアウト制約を導入すれば、図形をパーシングしながらインタラクティブにレイアウトするビジュアルシステムを生成することが可能ではないかと考えた。これらに基づいて、我々はビジュアルシステム生成系 Rainbow を開発した [7] [8] [9]。

Rainbow では、レイアウト制約として軟かいレイアウト制約と硬いレイアウト制約を実現している。軟かいレイアウト制約とは、図形の全体をグラフ描画アルゴリズムに従ってバランスよくレイアウトする制約である。軟かいレイアウト制約を提案した理由は、レイアウトでは、図形の特定な部分をレイアウトすることより、図形の全体を把握しやすくすることが大事だからである。一方、硬いレイアウト制約とは、図形の一部をローカルにレイアウトする制約である。

本論文では、ビジュアルシステム生成系 Rainbow の使用方法について述べる。

2 恵比寿

恵比寿とは、図形言語の文法を定義することで、ビジュアルシステムを生成するシステムである。本システムは、恵比寿を拡張したシステムである。

恵比寿のソフトウェアおよびアプリケーションの例は、WWW 経由で以下の URL から入手可能である。

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

*Current Address: Nambu University, 864-1 Weulke Kwangsan Kwangju 506-302 Korea

3 ビジュアルシステム生成系 Rainbow

3.1 Rainbow の動作環境

現在次の環境で動作を確認している。

- マシン /OS:SUNW,Ultra-1/SunOS5.5.1, SUNW,SPARCstation-5/SunOS5.4
- Tcl/Tk:7.5/4.1, 7.6/4.2, 8.0/8.0
- gcc:2.7.2.2

3.2 Rainbow のダウンロード

Rainbow のソフトウェアおよびアプリケーションの例は、WWW 経由で以下の URL から入手可能である。

```
http://www.iplab.is.tsukuba.ac.jp/~joung/rainbow.html
```

3.3 Rainbow のインストール

Rainbow は次の順番にインストールを行う。

1. `Makefile` 中のマクロを書き換える。
   - `VPG_DIR` : Rainbow がインストールされているディレクトリのパス名
   - `TCL_DIR` : この配布キットで提供される Tcl のライブラリのパス名
   - `WISH` : wish のコマンド名
   - `TCLSH` : tclsh のコマンド名
2. コマンドラインから `make` コマンドを実行する。そうすると `rainbow` という名前のファイルができる。

3.4 Rainbow の起動と終了

Rainbow を起動するには，Rainbow がインストールされているディレクトリでコマンドラインから次のコマンドを実行する．`%` はプロンプトである．

```
% rainbow
```

Rainbow を終了させるには，図形エディタ（図 1）の「File」メニューから「Exit」をクリックする．もし最後にルールをセーブしたのよりも後にルールが書き換えられていたら，実際に終了していいかどうか聞く。

3.5 Rainbow と恵比寿の違い

Rainbow と恵比寿の違いは、次の通りである。

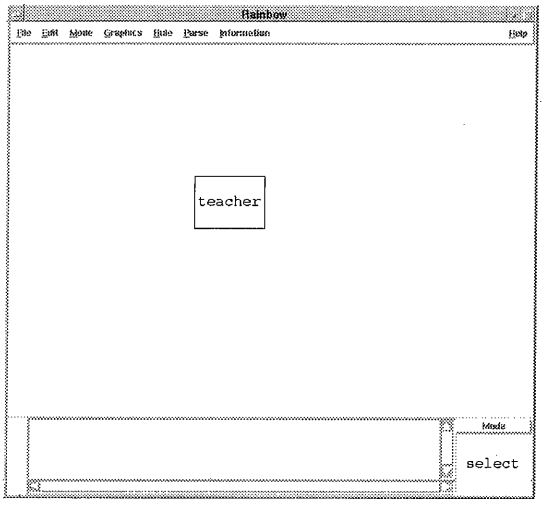

図 1: Rainbow の図形エディタ

1. 恵比寿では、図形の文法を定義する定義窓と実際に図形の解釈を行う実行窓に分れていたが、Rainbow ではこれらを一つの画面上で直接操作によって行うようにした。

   その理由は、1) ユーザは、文法を定義しながら図形を描いて文法の正しさを検査することの繰り返しによって文法を定義していくので、文法の定義モードと図形の実行モードを一緒にする方が便利である。2) 複雑な図形の実行結果（レイアウトされた図形）を表示するのに画面の空間を有効に使える。3) 一つの非終端シンボルとしたい図形の中に他の生成規則によって定義された非終端シンボルを認識するためである。これは、複雑な関係構造を持つ図形をレイアウトするために、多くの非終端シンボルを持つ図形の文法を定義するときに役に立つ。

2. Rainbow ではレイアウト制約が導入され、図形をパーシングしながらレイアウトするビジュアルシステムを生成することができるようになった。

   レイアウト制約として軟かいレイアウト制約と硬いレイアウト制約の二種類を実現した。

   (a) 軟かいレイアウト制約

   - スプリングモデル [10] 制約（`layout_spring`）
   - マグネティックスプリングモデル [11] 制約（`layout_magnetic`）
   - 木構造 [12] 制約（`layout_treeStructure`）

   (b) 硬いレイアウト制約

   - 図形の座標を一致させる制約（`layout_eq`）
   - 図形間の距離を具体的に与えり制約（`layout_dist`）

3.6 Rainbow による図形の実行モード

Rainbow では、図形を実行するモードとして

- 自動モード（新たな図形の入力があるたびにパーシングを行う）
- 要求モード（ユーザからの要求があったときのみにパーシングを行う）

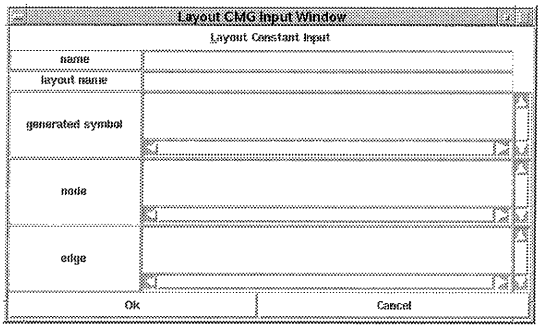

図 2: Rainbow の軟かいレイアウト CMG 入力ウィンドウ

の二種類を用意している。

自動モードにするには、図形エディタの「Parse」メニューから「Auto」をクリックする．それから、図形を入力するとパーシングが行われる。要求モードにするには、図形エディタの「Parse」メニューから「On demand」をクリックする．図形を入力し選択してから図形エディタの「Parse」メニューから「Parse Selected items」をクリックするとパーシングが行われる。ここで、図形の選択は図形エディタの「Mode」メニューから「select」をクリックする．それから、マウスの最初のボタンを用いて Drag-and-Drop し、範囲を指定することで図形の選択が行われる。

3.7 Rainbow の提供するレイアウト制約の種類

Rainbow では、レイアウト制約として軟かいレイアウト制約と硬いレイアウト制約がある。

3.7.1 軟かいレイアウト制約

軟かいレイアウト制約として、スプリングモデル制約、マグネティックスプリングモデル制約、木構造制約などを実装した。スプリングモデル制約は、無向グラフのレイアウトを行う場合に用いる。マグネティックスプリングモデル制約は、エッジの方向を考えて有向グラフのレイアウトを行いたいときに用いる。また、木構造制約は、それぞれグラフを木構造にレイアウトする場合に用いる。

軟かいレイアウト制約に従ってレイアウトするための軟らかいレイアウト生成規則の定義は以下の順番に行う。

1. ユーザはレイアウトしたい図形を図形エディタに描いて選択する。
2. 図形エディタの「Rule」メニューから「Make New Layout Production Rule」をクリックする。
3. そうすると、「軟かいレイアウト CMG 入力ウィンドウ（図 2）」が開く。このウィンドウは上から各軟かいレイアウト制約の定数を設定するメニュー（Layout Constant Input）、非終端シンボルの名前、軟らかいレイアウトの名前 SC、再帰的に生成される非終端シンボルの名前 GS、node の構成要素、 edge の構成要素を書く欄になっている。
4. ユーザは軟かいレイアウト CMG 入力ウィンドウの各欄を定義する。軟らかいレイアウトの名前を書く欄には、 `layout_spring`、 `layout_magnetic`、 `layout_treeStructure` などを書く。
5. 「OK」ボタンをクリックする。

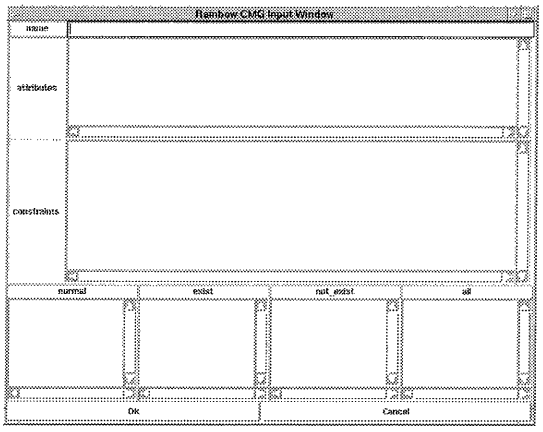

図 3: Rainbow の CMG 入力ウィンドウ

3.7.2 硬いレイアウト制約

硬いレイアウト制約として、図形の座標を一致させる制約と図形間の距離を具体的に与える制約実装した。硬いレイアウト制約は、通常の生成規則を定義するための「CMG 入力ウィンドウ（図 3）」の制約を書く欄に定義する。CMG 入力ウィンドウは上から順番に名前、属性、制約、構成要素（normal、exist などがある）を書く欄になっている。

図形の座標を一致させる制約は具体的に次のように記述する。

```
layout_eq　変数 1　変数 2
```

ここで、変数 1 と変数 2 は終端シンボルもしくは非終端シンボルの属性を示している。変数 1 と変数 2 の型としては integer、point を用いることができる。変数 2 の値として変数 1 の値を代入して、変数 2 を属性として持つ図形を変った位置に描画する。

図形間の距離を具体的に与える制約は、

```
layout_dist　変数 1　変数 2　distance
```

のように記述することができる。ここで、変数 1 と変数 2 は終端シンボルもしくは非終端シンボルの属性、distance は図形間の距離を表す定数である。変数 1 と変数 2 の型としては integer、point を用いることができる。変数 1 の値と distance ほど離れている座標を求めて、変数 2 の値に代入したあと変数 2 を属性として持つ図形を変った位置に描画する。

4 Rainbow のビジュアルシステム作成例

Rainbow のビジュアルシステム作成例として、データベース分野で実世界のデータ構造を記述するのに用いられる E-R ダイアグラム [13] を考える。

E-R ダイアグラムの構成要素を次のように定義する。1) 四角の中に実体名が書いてある図形を実体ノード（entityNode）とする。2) 円の中に属性名が書いてある図形を属性ノード（attributeNode）とする。3) 実体ノード間の関連を表す直線を実体エッジ（entityEdge）とする。その直線の中心には関連型が書いてある。4) 実体ノードと属性ノード間の関係を表す直線を属性エッジ（attributeEdge）とする。また、E-R ダイアグラムのレイアウトは、実体エッジや属性エッジの長さを一定し、属性ノードや属性ノード間に対称性を与えると分かりやすくレイアウトされると考えられる。そこで、我々はスプリングモデル制約を用いる。

まず、E-R ダイアグラム中のこれらの構成要素をパーシングするための生成規則が必要になる。Rainbow では、図形を用いて大まかな生成規則の定義を行っている。Rainbow で生成規則を定義するときに、ユーザは一つの非終端シンボルとしたい図形を図形エディタ（図 1）に描く。例えば、実体ノードをパーシングする生成規則の場合、ユーザは Rainbow の図形エディタに四角（rectangle）を描いてその中にテキスト（text）を書く。これらをまとめて選択し、図形エディタの「Rule」メニューから「Make New Production Rule」をクリックすると「CMG 入力ウィンドウ（図 4）」が開く。CMG 入力ウィンドウは上から順番に名前、属性、制約、構成要素（normal、exist などがある）を書く欄になっている。CMG 入力ウィンドウが開くとき Rainbow によって、図形から構成要素とそれらの属性間に成り立っている制約は CMG 入力ウィンドウに書き出される。構成要素の欄には

```
rectangle
text
```

が書かれる。制約の欄には、四角の中心とテキストの中心が等しいことを表す

```
vp_close 0.0.mid 0.1.mid
```

という制約が書かれる。ここで、ユーザはそれを修正して生成規則を定義する（図 5）。CMG 入力ウィンドウの名前の欄には、entityNode を書く。属性の欄には、実体ノードの属性 mid（中心）を四角の中心にするように

```
point mid 0.0.mid
```

を書いて「OK」ボタンをクリックすることにより、entityNode の生成規則の定義を完了する。。

CMG 入力ウィンドウに書かれる制約としては、eq（equal）、neq（not equal）、gt（greater than）、ge（greater or equal）、lt（less than）、le（less or equal）、vp_close [1]がある。CMG 入力ウィンドウの中で、制約の書き方は「制約名 変数 1 変数 2」である。ここで、変数 1 と変数 2 は定義している非終端シンボルの構成要素になる終端シンボルもしくは非終端シンボルの属性を示している。属性の参照は、「構成要素の種類. 構成要素の順番. 属性名」の形で行う。構成要素の種類は構成要素になる終端シンボルもしくは非終端シンボルが normal（exist）の構成要素だったら 0（1）になり、構成要素の順番は構成要素の種類の中で何番目の構成要素かを表す（0 から始まる）。例えば、normal の構成要素の 2 番目の構成要素の属性 mid（中心）を表す場合には「0.1.mid」のように記述する。

同様にして非終端シンボル attributeNode、entityEdge、attributeEdge の生成規則を定義する。さらに，E-R ダイアグラムを再帰的に定義する生成規則を記述する。すなわち、entityNode や attributeNode を ERGraph としてパーシングする生成規則、二つの organizationGraph とそれらの間を結ぶ entityEdge や attributeEdge を ERGraph としてパーシングする生成規則を記述する。

次に、ユーザは E-R ダイアグラムの構成要素をスプリングモデル制約に従ってレイアウトするための軟らかいレイアウト生成規則を定義する。ユーザはレイアウトしたい E-R ダイアグラムを図 6のように図形エディタに描き、図形を選択して図形エディタの「Rule」メニューから「Make New Layout Production Rule」をクリックする。そうすると、「軟かいレイアウト CMG 入力ウィンドウ（図 7）」が開く。Rainbow は、図 6の図形から四角、円、直線などの終端シンボルを除いて非終端シンボルの名前を自動的に書き出す。node と edge の構成要素の欄には

```
entityNode
attributeNode
entityEdge
```

[1]vp_close 制約は、変数と変数の値がある程度近い場合に eq 制約が課せられる

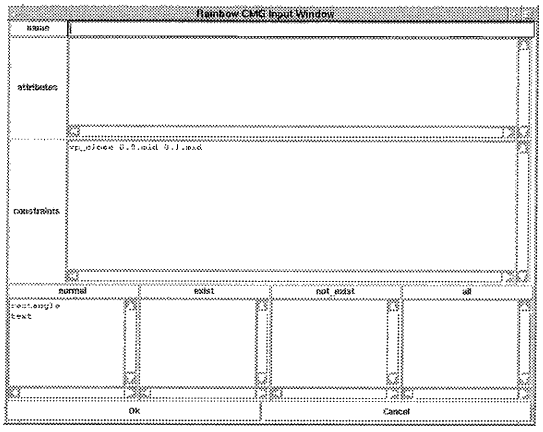

図 4: Rainbow の CMG 入力ウィンドウ (1)

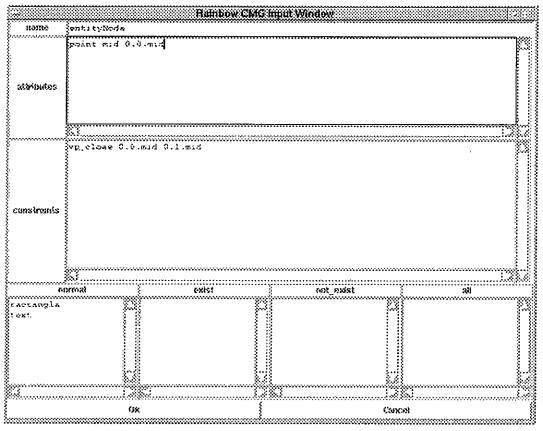

図 5: Rainbow の CMG 入力ウィンドウ (2)

attributeEdge
ERGraph

のように記述される。そうすると、ユーザは node の構成要素の欄には

entityNode
attributeNode

edge の構成要素の欄には

entityEdge
attributeEdge

を選ぶ。さらに、非終端シンボルの名前の欄には ERModel、軟らかいレイアウトの名前の欄には layout_spring、再帰的に生成される非終端シンボルの欄には ERGraph を書き、「OK」ボタンをクリックすることにより、軟らかいレイアウト生成規則の定義を完了する（図 8）。これらの生成規則を用いて、図 6の図形をパーシングすると図 9のようにレイアウトされる。

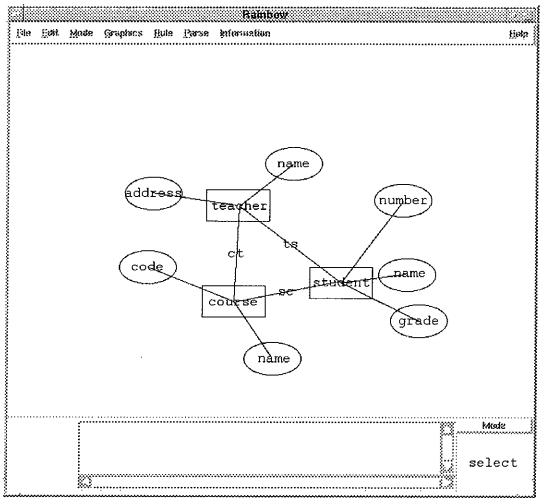

図 6: レイアウト前の E-R ダイアグラム

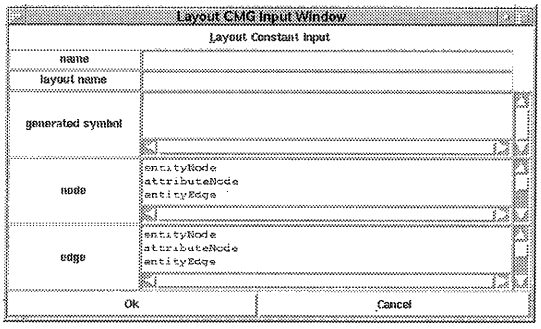

図 7: 軟かいレイアウト CMG 入力ウィンドウ (1)

なお、この例の生成規則は、Rainbow がインストールされているディレクトリにある Rule というディレクトリの下にある。ファイル名は以下の通りである。

- er.rule : 通常の生成規則
- er.lrule : 軟らかいレイアウト生成規則

また、例として実行に用いた図形は、Rainbow がインストールされているディレクトリにある Canvas というディレクトリの er.can にある。

5 他のビジュアルシステムの作成例

我々は、ビジュアルシステムの作成例として、E-R ダイアグラムの他に以下の例を実現した。また、通常の生成規則のファイル名（xxx.rule）、軟らかいレイアウト生成規則のファイル名（xxx.lrule）、実行に用いた図形のファイル名（xxx.can）を挙げる。生成規則のファイルはディレクトリ Rule、実行に用いた図形のファイルはディレクトリ Canvas にある。

- オブジェクト指向に基づくソフトウェア設計に用いられるオブジェクト図 [14]：
  object.rule, object.lrule, object.can

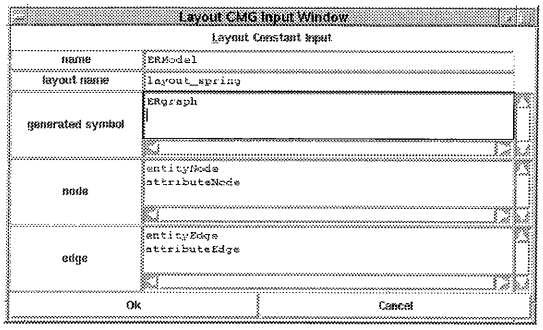

図 8: 軟かいレイアウト CMG 入力ウィンドウ (2)

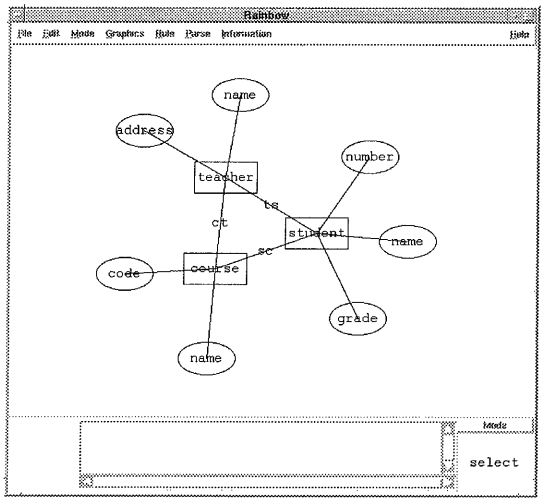

図 9: レイアウト後の E-R ダイアグラム

- 会社などの仕組みを表すのに用いられる組織図：
  `tree.rule, tree.lrule, tree.can`
  ただし、自動モードで実行するには `tree_first.can` をロードしてから行う。

- 親族の関係を表すのに用いられる家系図：
  `kinship.rule, kinship.lrule, kinship.can`
  ただし、自動モードで実行するには `kinship_first.can` をロードしてから行う。

6 まとめ

本論文では、ビジュアルシステム生成系にレイアウト制約を扱えるようにして、図形をバランスよく見やすくレイアウトできるビジュアルシステム生成系 Rainbow の使用方法について述べた。

参考文献

[1] Sitt Sen Chok and Kim Marriott, "Automatic Construction of User Interfaces from Constraint Multiset Grammars," *Proceedings of the IEEE Workshop on Visual Languages*, pp.

242–249, 1995.

[2] 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.

[3] 馬場昭宏, 田中二郎, "「恵比寿」を用いたビジュアルシステムの作成," 情報処理学会論文誌, Vol. 40, No. 2, pp. 497–506, 1999.

[4] Eric J. Golin and Tom Magliery, "A Compiler Generator for Visual Languages," *Proceedings of the IEEE Symposium on Visual Languages*, pp. 314–321, 1993.

[5] Sitt Sen Chok and Kim Marriott, "Automatic Construction of Intelligent Diagram Editors," *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 185–194, 1998.

[6] 杉山公造, "グラフ自動描画法とその応用 - ビジュアルヒューマンインタフェース -," 計測自動制御学会, 1993.

[7] 丁 錫泰, "ビジュアルシステム生成系へのレイアウト制約の導入," 筑波大学工学研究科博士論文, 7月, 2000.

[8] 丁 錫泰, 田中二郎, "Rainbow: ビジュアルシステム生成系におけるレイアウト制約の実現," 情報処理学会論文誌, Vol. 41, No. 5, pp. 1246–1256, 2000.

[9] Suchtae Joung and Jiro Tanaka, "Generating a Visual System with Soft Layout Constraints," *Proceedings International Conference on Information – Information'2000 –* , Fukuoka Japan, October, 2000.

[10] Peter Eades, "A Heuristic for Graph Drawing," Congressus Numerantium, Vol. 42, pp. 149–160, 1984.

[11] 三末和男, 杉山公造, "マグネティック・スプリング・モデルによるグラフ描画法について," 情報処理学会研究報告 ヒューマンインタフェース 55-3, pp. 17–24, 1994.

[12] John Q. Walker, "A Node-positioning Algorithm for General Trees," Software Practice and Experience, Vol. 20, No. 7, pp. 685–705, 1990.

[13] Peter Chen, "The Entity-Relationship Model: Towards a Unified View of Data," *ACM Transactions Database System*, Vol. 1, No. 1, pp. 9–36, 1976.

[14] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, "Object-Oriented Modeling and Design," Prentice-Hall International, 1991.