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

丁 錫泰[†] 田中二郎[††]

我々は、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、図形を解釈しながらインタラクティブに図形をレイアウトするビジュアルシステム生成系として「Rainbow」を開発した。

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。軟かいレイアウト制約は、図形の全体を自動描画アルゴリズムに従って分りやすくレイアウトする制約である。また、硬いレイアウト制約は、特定の図形の座標や図形間の距離などを具体的に与える場合に用いる制約である。我々は、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、リスト構造制約、木構造制約などを実装した。

また、「Rainbow」のアプリケーション作成例として、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」と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」の例を示した。

Rainbow:Implementing Layout Constraints in Visual System Generator

Sucktae Joung[†] and Jiro Tanaka[††]

We have developed a visual system generator "Rainbow," which interactively lays out figures while parsing them. The system handles layout constraints, which were not addressed in "Eviss."

We have implemented two kinds of layout constraints: "soft-layout constraints" and "strict-layout constraints." Soft-layout constraints use graph drawing algorithms to layout the figures in an understandable way. Strict-layout constraints are applied when coordinates of the figure elements and distances between them are provided. We have implemented four kinds of soft-layout constraints: spring model constraint, magnetic spring model constraint, list constraint and tree constraint.

We present two application examples: "E-R diagrams" which describe entities and their relationships of the real world in the database, and "object diagrams" which are used for object-oriented software design.

1. はじめに

図形は工程図、データの流れ図、回路図、および、テレビドラマの登場人物の関係を表す図など様々な分野で使われる。こうした図形は図形要素間に関係構造を持つ図形言語であるため、空間的な文法を持っていると考えることができる。

空間パーサ生成系とは、図形の文法を定義することで図形の空間パーサを生成するシステムのことである[1)2)3)4)5)]。空間パーサが図形を解釈すると文法に基づいて図形間に制約が課せられ、それらの関係を保存したまま編集を行うことができる。制約というのはある図形要素間になり立っている関係である。例えば、円の中心にラベルとしてテキストが書いてあるものをノードとする。ある図形がノードとして解釈されると円を動かしてもラベルがついてくるような編集を行うことができる。

空間パーサ生成系についての研究として、SPARGEN[1)] や Penguins[2)3)] などの研究がある。Golin らにより提案されている SPARGEN は、OOPLG（Object-Oriented Picture Layout Grammars）を用いて図形言語の文法を定義することで空間パーサを生成するシステムである。OOPLG では、図形の属性や制約を C^{++} を用いて定義している。また、Marriot らの Penguins では、図形の文法を Constraint Multiset Grammars（CMG）[2)6)] を用いて空間パーサを生成している。これらの空間パーサ生成系は、テキストを用いて図形言語の文法を定義するので、ユーザは文法を理解している必要があり、一般のユーザにとって使いやす

† 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

いものとはいえない。また、これらの空間パーサ生成系は、バッチ処理的な要素が強いが、本来、図形を処理するシステムは、もっとインタラクティブであるべきだと考える。

そこで、我々が研究を行っている恵比寿[4)5)7)]では、ユーザが入力した図形を用いて大まかな図形の CMG の文法を自動的に生成するようにしている。そのあとユーザは、生成された文法を見ながら、制約の追加または削除などをおこない修正する。また、 VIC[8)] では視覚的な制約入力インターフェイスを恵比寿上に実装し、テキスト編集を行わずに CMG の文法を生成している。

本研究では、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、図形の一部または全体を解釈しながらバランスよく分かりやすくレイアウトすることができる「Rainbow」を開発した。「Rainbow」は、我々がこれまで研究を行ってきた恵比寿にレイアウト制約を追加することにより実現している。

最近、 Marriot らは空間パーサ系 Penguins[9)] にレイアウト制約を追加することで図形のレイアウトを行うことを提案している。 Penguins が提供するレイアウト制約は、「Rainbow」の硬いレイアウト制約に相当するものである。

しかし、レイアウトでは、図形の全体を把握しやすくバランスよくレイアウトすることが大事であるが、Penguins では、図形の全体をバランスよく分りやすくレイアウトするための軟らかいレイアウト制約を提供してない。

「Rainbow」の新しい点は、 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、図形を解釈しながらインタラクティブに図形をバランスよくレイアウトできるビジュアルシステムを生成することである。「Rainbow」では、レイアウト機能の追加によって、Penguins と比べより広い範囲のビジュアルシステムを扱うことができる。

例えば、ビジュアルシステムの例として、組織図、家系図、テレビドラマの登場人物の関係を表す図、ソフトウェア開発において重要視される各種の設計図などを考えることができるが、これらの空間パーシングにレイアウト機能を追加することにより、編集中に、図形を整形し、自動描画すること、すなわち図形をよりインタラクティブに処理することが可能となる。

本論文の構成は次のとおりである。まず、2 章では図形の文法を記述する方法である CMG について述べる。3 章ではシステム「Rainbow」について説明を行う。 4 章では我々が実現したレイアウト制約について説明する。また、 5 章ではレイアウト制約の例を示し、 6 章で関連研究に関して述べる。

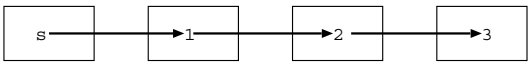


図 1 リスト構造
Fig. 1 List structure

2. CMG

CMG[2)6)] は終端シンボルの集合、非終端シンボルの集合、開始シンボル、および、生成規則の集合から構成される。すべての終端シンボルと非終端シンボルは属性を持っている。生成規則はトークン（終端シンボルもしくは非終端シンボルのインスタンス）のマルチセットとそれらの属性間の関係を保つ制約により、新たなトークンに書き換えるルールである。本論文では右辺から左辺への書き換えを生成と呼ぶ。生成規則は次のように定義される。

$$T(\vec{x}) ::= T_1(\vec{x_1}), \cdots, T_n(\vec{x_n})\ where$$
$$exists\ T'_1(\vec{x'_1}), \cdots, T'_m(\vec{x'_m})$$
$$where\ C\ and$$
$$\vec{x}\ =\ F(\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m})$$

すなわち、トークンのマルチセット T_1, …, T_n（normal の構成要素）、T'_1, …, T'_m（exist の構成要素）の属性が制約 C を満たす場合、T_1, …, T_n が非終端シンボル $T(\vec{x})$ に書き換えられることを意味している。但し、 exist の構成要素は、定義したい $T(\vec{x})$ を認識するために、存在する必要がある構成要素である☆。F は、構成要素の属性 $\vec{x_1}, \cdots, \vec{x_n}$ と $\vec{x'_1}, \cdots, \vec{x'_m}$ を引数とする関数であり、定義中の非終端シンボルの属性に値を与えることを定義している。

図 1のようなリスト構造を考えてみる。リスト構造の文法を定義するためには次の二つの生成規則が必要になる。

生成規則 1 四角の中心にラベルとして s が書いてあるものをリストとする。

生成規則 2 一つのリストが矢印によって四角につながれ、その四角の中心にラベルとして数字が書いてあるものをリストとする。

これらを CMG で記述すると次のようになる。

```
1:list(point mid) ::= R:rectangle, T:text
2:   where (
3:      R.mid  == T.mid  &&
4:      T.text == ‘‘s’’
```

☆ その他に CMG は構成要素として not_exist、 all などを持っている。詳しくは文献[2)4)6)] を参照。

```
 5:   ) {
 6:   mid = R.mid;
 7: }
 8:
 9:list(point mid) ::= R:rectangle, T:text,
10:           L:line, LL:list
11:   where (
12:      R.mid   == T.mid   &&
13:      R.mid   == L.end   &&
14:      LL.mid  == L.start
15:   ) {
16:   mid = R.mid;
17: }
```

生成規則 1 は図 1のラベル s の四角をリストとして解釈するための生成規則で、 1 行目から 7 行目までが CMG の定義である。 1 行目は四角（R）とテキスト（T）で構成されている非終端シンボル「リスト（list）」を定義している。リストは、属性として中心（mid）を持つ。 2、 3、 4、 5 行目は、リストの構成要素間の制約を定義している。 3 行目は、四角の中心（R.mid）とテキストの中心（T.mid）が等しいという制約を表す。 4 行目は、テキストの文字列（T.text）が「s」であることを表している。この二つの制約を満たすときに 6 行目が行われる。 6 行目はリストの中心として四角の中心の値を代入することを表している。

生成規則 2 はラベルが付いている四角に一つのリストがつながっているものをリストとして解釈するための生成規則で、 9 行目から 17 行目までが CMG の定義である。 13 行目は四角の中心と直線の終点（L.end）が一致する制約、 14 行目はリストの中心（LL.mid）と直線の始点（L.start）が一致する制約を意味している。

3. システム「Rainbow」

「Rainbow」では図 2のような図形エディタを備えている。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、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。

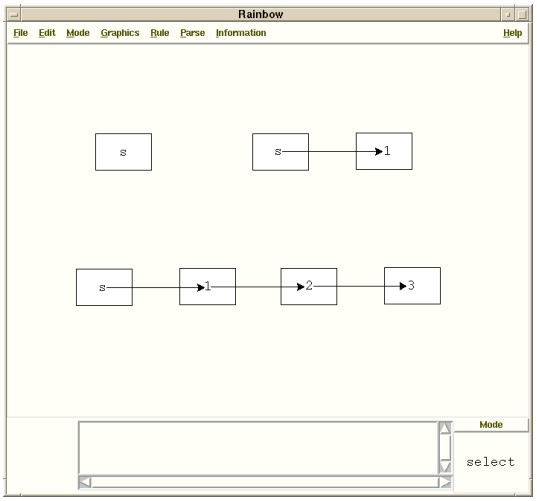


図 2 「Rainbow」の図形エディタ
Fig. 2 Graphic editor of the Rainbow

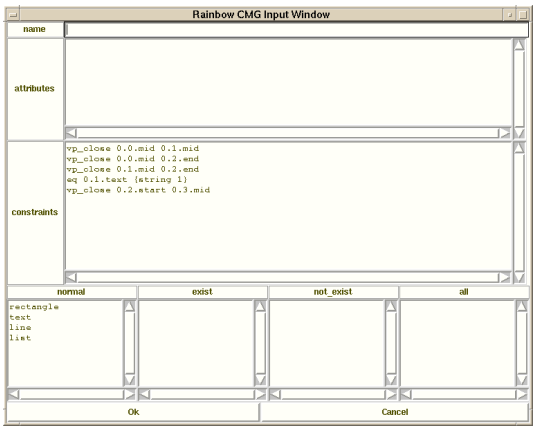


図 3 「Rainbow」の CMG 入力ウィンドウ (1)
Fig. 3 CMG input window(1) of the Rainbow

「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に描く。次に、その図形を選んで CMG 入力ウィンドウ（図 3）を開く。そうすると「Rainbow」は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を CMG 入力ウィンドウに書き出す。 CMG 入力ウィンドウは上から順番に名前、属性、制約、構成要素を書く欄になっている。ここにユーザは制約を修正し、また非終端シンボルの名前、属性を追加することにより、生成規則を定義していく（図 4）。図 2の上左の図形はリスト構造の生成規則 1 を定義するため入力した図形で、上右の図形は生成規則 2 を定義するための図形である。上右の図形を選択して開かれた CMG 入力ウィンドウが図 3である。図

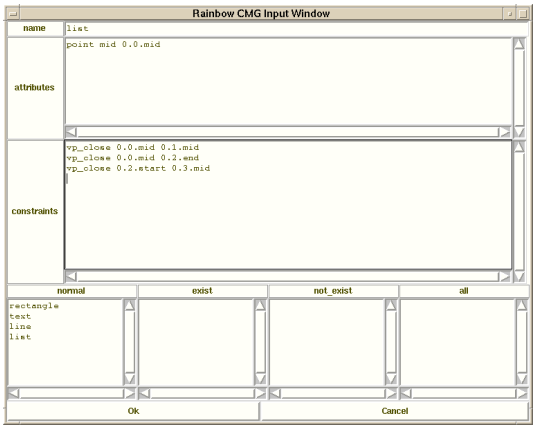


図 4 「Rainbow」の CMG 入力ウィンドウ (2)
Fig. 4 CMG input window(2) of the Rainbow

2の下の図形は実際に解釈したい図形である。

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** ☆がある。これらの制約を常に成り立たせるための機構を制約解消系と呼ぶ。制約解消系としては SkyBlue[10] を用いている。

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。

図形言語の生成規則の定義が終わったら実際に図形を図形エディタに入力し、パーシングすることが可能になる。一般に「Rainbow」では、図形を解釈するモードとして自動モードと要求モードの二種類を用意している。新たな図形の入力があるたびにパーシングを行うのが自動モード、また、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のが要求モードである。

☆ **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
☆☆ not_exist は 2、 all は 3 になる。

4. レイアウト制約

「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して、スプリングモデル[11] 制約（**spring**）、マグネティックスプリングモデル[12] 制約（**magnetic**）、リスト構造制約（**listStructure**）、木構造[13] 制約（**treeStructure**）などを実装した。

ここで、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、リスト構造制約と木構造制約は、それぞれグラフをリスト構造と木構造にレイアウトする場合に用いる。本論文では特に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、リスト構造制約について詳しく述べる。

一方、硬いレイアウト制約は、図形の座標や図形間の距離などの制約を具体的に与えたい場合に用いる。

4.1 軟かいレイアウト制約

4.1.1 CMG と軟かいレイアウト制約の関係

軟かいレイアウト制約は、図形の全体を自動描画アルゴリズムに従ってバランスよく分りやすくレイアウトする制約である。この制約は、特定の図形要素間に与えられる制約と性質が異なり、図形の全体の配置を変更する制約である。

我々は、軟かいレイアウト制約を CMG の一つの生成規則として以下のように定義する。

$$\begin{aligned} S ::= & \ nodes\ S_1, \cdots, S_n \\ & edges\ S'_1, \cdots, S'_m \\ & where\ SC\ and\ GS. \end{aligned}$$

ここで、S は非終端シンボル、S_1，…，S_n は $node$ の構成要素の名前、S'_1，…，S'_m は $edge$ の構成要素の名前、SC はレイアウトの種類、GS は図形の構造を再帰的に定義する時に生成される非終端シンボル（再帰的に生成される非終端シンボル）を示している。

この生成規則は、**node** の構成要素 S_1，…，S_n のマルチセットと **edge** の構成要素 S'_1，…，S'_m のマルチセットを非終端シンボル S として認識し、指定されたレイアウトの種類 SC を与えることを意味する。

各非終端シンボルは、自分の構成要素の情報を持っているので、軟かいレイアウトの生成規則が適用される時に、GS の構成要素が GS を持ったなくなるまで繰り返し検査し、**node** や **edge** の構成要素のマルチセットを動的に求めて、SC に従ってレイアウトを行うことができる。このため、図形エディタに **node** や **edge** の構成要素になっている非終端シンボルが追加・削除されて

も、きちんとレイアウトを行うことができる。

4.1.2 軟かいレイアウト制約の種類

スプリングモデル[11] は無向グラフ描画アルゴリズムの一つであり、ノードは鉄のリングに、エッジは力学系を形成するバネに置き換える。このモデルは 2 種類のバネが使われる。すなわち隣接するノード間をつなぐバネと隣接しないノード間に斥力だけを与えるバネである。隣接するノード間に働く力 f_s は

$$f_s = c_1 \log(d/c_2)$$

により与えられる。ここで d はノード間の距離、c_1 と c_2 は定数とする。$d > c_2$ のとき f_s は引力、$d < c_2$ のとき斥力、$d = c_2$ のときノード間に力は働かない。また、隣接しないノード間に働く力 f_r は

$$f_r = c_3/d^2$$

により与えられる。ここで c_3 は定数とする。このような力 f_s と f_r をできるだけ緩和するように各ノードを (c_4 × そのノードに働く力) ずつ移動させることにより、すべての隣接するノード間の距離を c_2 の近傍に収束させる。c_4 は定数とする。

マグネティックスプリングモデル[12] は、スプリングモデルに磁場の概念を入れたものである。エッジを磁針と見なし、グラフの置かれた磁場から回転力を受ける。マグネティックスプリングモデルの磁場には、有向エッジに働くものと無向エッジに働くものの二種類がある。有向エッジの場合はエッジの終点が磁場の北を向くように回転力を受ける。無向エッジの場合は磁場の向きは関係なくノードは南北の向きに近い方を向くように回転力が働く。回転力は次のように定義される。

$$f_m = c_5 b d^{\alpha} \mid t \mid^{\beta}$$

ここで、b は基準点（エッジの中心）における磁場の強さ、d は現在のエッジの長さである。t はエッジの基準点における磁場の北からの終点のずれの角度である。すなわち、有向エッジの場合は $0 < t \leq \pi$、無向エッジの場合は $0 < t \leq \pi/2$ となる。また、α は辺の長さの回転力への影響を制御する定数、β は t の回転力への影響を制御する定数である。また、隣接するノード間に働く力と隣接しないノード間に働く力は、f_s と f_r を用いる。

我々は、軟かいレイアウト制約の生成規則を定義するとき、まずユーザは解釈したい図形を図形エディタに描いてその図形またはその一部を選択し、「軟かいレイアウト CMG 入力ウィンドウ（図 5）」を開く。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（c_1、c_2 など）を設定するメニュー（Layout Constant Input）、名前、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。「Rainbow」では、図形を選択すると、**node** と **edge** の構成要素の欄には、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がシステムにより自動的に書き出されるので、**node** や **edge** の構成要素の欄には、ユーザがそれぞれの構成要素にしたい非終端シンボルの名前を選ぶ。次に、非終端シンボルの名前、SC、GS を定義する。また、指定するレイアウトの定数を設定する。レイアウトの定数を設定しなかった場合には、「Rainbow」で用意した定数の値が使われる。

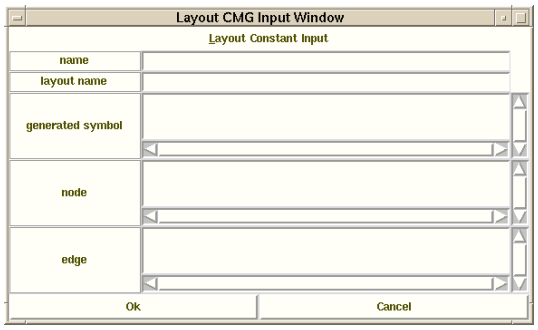


図 5 軟かいレイアウト CMG 入力ウィンドウ
Fig. 5 CMG input window for the soft-layout

軟かいレイアウト制約モジュールの実行により軟かいレイアウト制約の処理が行われる。スプリングモデル制約モジュールでは、GS の構成要素が GS を持ったなくなるまで繰り返し検査し、**node** や **edge** の構成要素のマルチセットを求め、図形要素間のグラフ構造を得る。グラフ構造を用いて各ノードに働く力 f_s、f_r が求められる。

マグネティックスプリングモデル制約モジュールでは、スプリングモデルに必要なグラフ構造に加えて、それぞれのエッジの種類を決めることとそのエッジが磁場に対して指す方向が必要である。これらを用いて各ノードに働く力 f_s、f_r、f_m が求められる。

リスト構造制約モジュールでは、GS の構成要素が GS を持ったなくなるまで繰り返し検査し、**node** や **edge** の構成要素のマルチセットを求め、図形要素間のグラフ構造を得る。ヘッドのノード（テキストとして **s** を持つノード）を決められた位置に配置し、あらかじめ与えられた順序に従いヘッドのノードの平行線上にノードを左から右に並べる。ノードの **y** 座標はヘッドのノードの **y** 座標と同じであり、ノードの **x** 座標は連続するノードの間を一定の間隔に開けるようにする。

4.2 リスト構造制約の例

リスト構造制約の例として、2 章で挙げられたリスト

構造を用いる。まず、リスト構造の構成要素を定義するための生成規則を定義する。

生成規則 1 四角の中心にラベルとしてテキストが書いてあるものをノードとする。

```
1: lNode(point mid, string text) ::=
2:             R:rectangle, T:text
3:   where (
4:      R.mid == T.mid
5:   ) {
6:   mid  = R.mid;
7:   text = T.text;
8: }
```

生成規則 2 ノード間を結ぶ直線をエッジとする。

```
1: lEdge(point start, point end) ::=
2:             L:line
3:   where ( exist N1:lNode, N2:lNode
4:      where (
5:         L.start == N1.mid  &&
6:         L.end   == N2.mid
7:   ) {
8:   start = L.start;
9:   end   = L.end;
10: }
```

次は、lNode や lEdge を用いてリスト構造を再帰的に定義する生成規則と軟かいレイアウトの生成規則を定義する。

生成規則 3 2 章のリスト構造の生成規則 1 と同じであるが、構成要素がノードである。

```
1:list(point mid, string text) ::=
2:           N:lNode
3:   where (
4:      N.text == ‘‘s’’
5:   ) {
6:   mid  = N.mid;
7:   text = N.text;
8: }
```

生成規則 4 2 章のリスト構造の生成規則 2 と同じであるが、構成要素がリスト、エッジ、ノードである。

```
1:list(point mid, string text) ::=
2:           L:list, E:lEdge, N:lNode
3:   where (
4:      E.start == L.mid  &&
5:      E.end   == N.mid
6:   ) {
7:   mid  = N.mid;
8:   text = N.text;
```

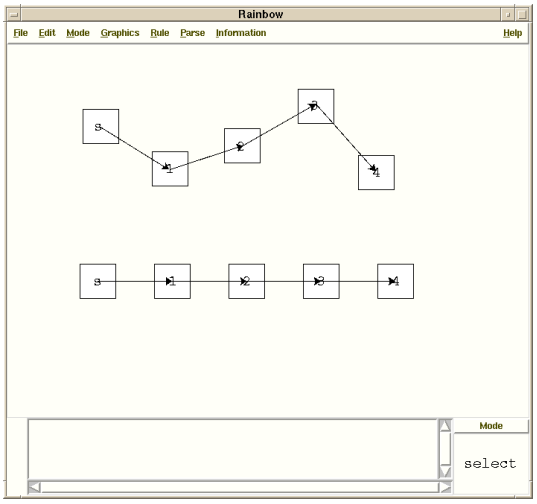


図 6 レイアウト前後のリスト構造
Fig. 6 List structure before and after layout

```
9: }
```

生成規則 5 軟かいレイアウトの生成規則は、以下の通りである。

```
1: layoutList() ::= nodes:lNode,
2:                  edges:lEdge
3:   where (
4:      listStructure  &&
5:      list
6:   )
```

ここで、4 行目の listStructure は軟かいレイアウト制約のリスト構造制約（SC）、5 行目の list は再帰的に生成される非終端シンボル（GS）を示している。

これらの生成規則を用いて、図 6の上の図形を解釈すると下の図形のようにレイアウトされる。

4.3 硬いレイアウト制約

4.3.1 CMG と硬いレイアウト制約の関係

硬いレイアウト制約は、CMG に基づいた制約の拡張として考えることができる。その理由は、硬いレイアウト制約は通常の制約のように生成規則が適用される特定の図形要素間に与えられる制約だからである。

すなわち、生成規則の定義、

$$T(\vec{x}) ::= T_1(\vec{x_1}), \cdots, T_n(\vec{x_n})\ where$$
$$exists\ T'_1(\vec{x'_1}), \cdots, T'_m(\vec{x'_m})$$
$$where\ C\ and\ HC\ and$$
$$\vec{x}\ =\ F(\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m})$$

の中でレイアウトの種類（HC）は、通常の制約（C）の後ろに記述する。これは、硬いレイアウト制約が、CMG に基づいた制約と同様の性質を持つものだからで

あり、通常の制約の延長として扱うことができる。

4.3.2 硬いレイアウト制約の種類

図形の座標を一致させて図形を描画す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。例えば、 `layout_eq` 0.0.`mid_y` 0.1.`mid_y` は、「0.0」と「0.1」の構成要素が解釈された後、「0.1」の構成要素の属性 `mid_y`（中心の `y` 座標）の値を「0.0」の構成要素の属性 `mid_y` の値と等しくして「0.1」の構成要素を描画する。

図形間の距離を具体的に与えて図形を描画する制約は、

`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。例えば、 `layout_dist` 0.0.`mid_x` 0.1.`mid_x` 100 は、「0.0」と「0.1」の構成要素が解釈された後、「0.0」の構成要素の属性 `mid_x`（中心の `x` 座標）と「0.1」の構成要素の属性 `mid_x` との距離を 100 ドットにして「0.1」の構成要素を描画する。

4.4 通常の制約と硬いレイアウト制約の違い

図形の解釈が成功するためには、文法に書かれた通常の制約を満す必要がある。また、通常の制約は図形を解釈することにより、図形間の関係をそのまま維持する制約である。それに対して、レイアウト制約は図形が解釈された後、もともとの図形要素間になかった位置関係を作り出す制約である。

`layout_eq` と `eq` との違いは、 `eq` は図形を解釈するときに予め二つの変数の値が等しいことを意味するが、 `layout_eq` は異なっている二つの変数の値を等しくするところにある。

硬いレイアウト制約を導入した理由は、生成規則により生成される非終端シンボルの一部をローカルにレイアウトするためである。また、硬いレイアウト制約と軟かいレイアウト制約を混ぜて用いることにより、空間パーサの応用が広がると考えられる。

4.5 硬いレイアウト制約の応用

通常の制約は、いわば生成規則を適用するための条件として扱われるが、硬いレイアウト制約と軟かいレイアウト制約は、一種の表示の工夫と考えられる。硬いレイアウト制約と軟かいレイアウト制約を混ぜて用いることにより、空間パーサの応用が広がると考えられる。例えば、ノードやエッジの構成要素をユーザが考えた通りに硬いレイアウト制約を用いてローカルなレイアウトを行ってから、図形のノードやエッジの全体を自動描画アルゴリズムに従ってバランスよく分りやすくレイアウトすることなどができる。

具体的な応用としては、家系図などが考えられる。親ノードの構成要素の二つのノードを同じ平行線上に配置し、ノード間の距離を一定にすることが望まれるので、硬いレイアウト制約を用いる。また、家系図の全体は木描画アルゴリズムに従って分りやすくレイアウトすることが望まるので、全体に木構造制約を与える。

最近、 Marriot らは空間パーサ系 Penguins[9] にレイアウト制約を追加することで図形のレイアウトを行うことを提案している。 Penguins が提供するレイアウト制約は、「Rainbow」の硬いレイアウト制約に相当するものであるが、 Marriot らは、硬いレイアウトの具体的な応用例として二分木、数学の方程式、状態遷移図などを挙げている。

4.6 通常の制約と硬いレイアウト制約、軟らかいレイアウト制約の処理

CMG で記述された生成規則の集合として文法を記述し、解釈の対象となる図形を与えると通常は以下のように解釈が進む[2]。まず、ユーザが記述した生成規則と軟かいレイアウトの生成規則は、生成規則データベース SCC に保存される。また、解釈の対象となる図形を構成する全ての終端シンボルのトークンは、トークンデータベース ParseForest に格納される。さらに、それらのトークンの内部属性間に存在する制約は制約解消系 SkyBlue に追加される。

実際の図形の解釈は、以下の [1] を行い、 [1] が終了したあとで [2] を行うことにより実行する。

[1] ParseForest が変更されなくなるまで、 SCC の各通常の生成規則に対して、構成要素の候補になれるトークンの組合わせリストを ParseForest から求めることを繰り返す。次に、各トークンの組合わせに対して、生成規則に記述された制約を満し、さらに硬いレイアウト制約が存在する場合にその解が存在するかを繰り返し検査を行う。もし、生成規則に記述された制約および硬いレイアウト制約を満す場合には、その生成規則が適用され、硬いレイアウト制約モジュールが実行される。そし

て、新たな非終端シンボルのトークンを ParseForest に挿入し、生成に用いられたトークンを ParseForest から削除する（トークンに「削除マーク」を付けるだけで実際には ParseForest から削除されない）。また、全ての制約を SkyBlue に追加する。

[2] SCC の各軟かいレイアウトの生成規則に対して、GS（再帰的に生成される非終端シンボル）が ParseForest に存在するかを検査し、GS の構成要素が GS を持ったなくなるまで繰り返し検査し、**node** や **edge** の構成要素のマルチセットを求める。

求められた **node** や **edge** の構成要素のマルチセットに SC（軟かいレイアウト制約）が与えられ、レイアウトが行われる。さらに、S（新たに生成された非終端シンボル）のトークンは ParseForest に挿入される。

5. レイアウト制約の例

5.1 スプリングモデル制約の例

スプリングモデル制約の例として、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[14] 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。これらの構成要素を解釈する生成規則を「Rainbow」を用いて定義する。例えば、実体ノードを解釈する生成規則の場合、ユーザは「Rainbow」の図形エディタに四角を描いてその中にテキストを書いて CMG 入力ウィンドウを開く。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書く。制約の欄には、四角の中心とテキストの中心を等しくする制約を選び、非終端シンボル **entityNode** の定義を完了する。さらに、非終端シンボル **ERGraph** を生成するように、E-R ダイアグラムを再帰的に定義する生成規則を定義する。

次に、解釈したい図 7の図形を選んで軟かいレイアウト CMG 入力ウィンドウ（図 5）を開くと、システムは、**node** と **edge** の構成要素の欄に、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

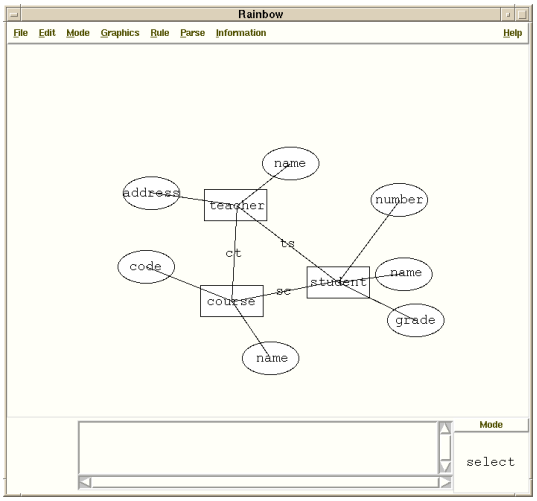


図 7 レイアウト前の E-R ダイアグラム
Fig. 7 E-R diagrams before layout

entityEdge
attributeEdge

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。

次に、ユーザは「Layout Constant Input」メニューを選択してスプリングモデルの定数を決める。非終端シンボルの名前の欄には **ERModel**、レイアウトの名前の欄には **spring**、再帰的に生成される非終端シンボルの欄には **ERGraph** を書いて軟かいレイアウト生成規則を定義する。これらの生成規則を用いて、図 7の図形を解釈すると図 8のようにレイアウトされる。

5.2 マグネティックスプリングモデル制約の例

マグネティックスプリングモデル制約の例として、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[15)16)] の自動レイアウトについて考える。オブジェクト図の構成要素として、クラス（**class**）、関連（**association**）、汎化（**generalization**）、および、集約（**aggregation**）が存在する。ここで、クラスをノード、3 種類の関係をエッジとして見なすことができ、3 種類のエッジをそれぞれ違う方向に向けるようにすれば、オブジェクト図の自動レイアウトが可能である[17]。

それぞれのエッジへどの種類の磁場を与えるかは、オ

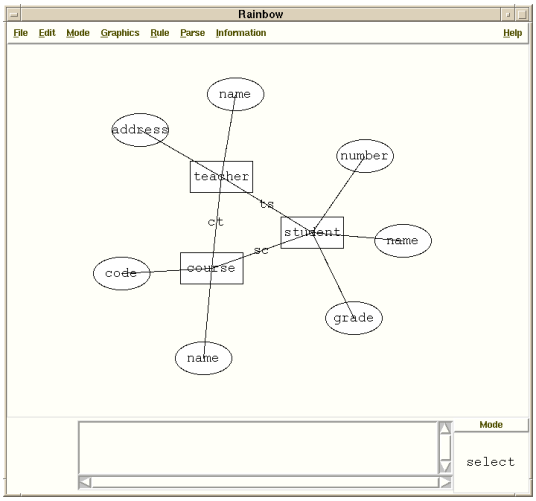


図 8 レイアウト後の E-R ダイアグラム
Fig. 8 E-R diagrams after layout

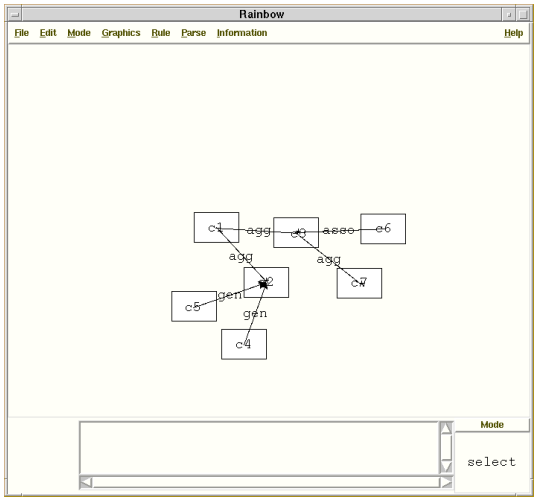


図 10 レイアウト前のオブジェクト図
Fig. 10 Object diagrams before layout

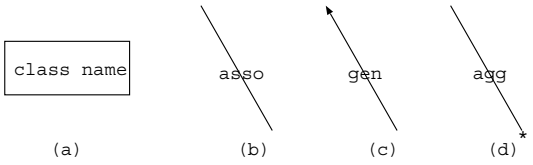


図 9 オブジェクト図の構成要素
Fig. 9 Components of the object diagrams

ブジェクト図におけるそれぞれのエッジの特性や意味的な要素から以下のようにする。関連は、オブジェクト同士の対等な参照あるいは利用関係を表すもので、方向のない関係である。従って関連エッジは平行磁場とし、横向きに磁場を与える。汎化は、クラス間の概念的な包含関係、すなわちスーパークラスとサブクラス間の継承関係を表すものなので、上から下の有向関係である。従って汎化エッジは下向の磁場を与える。集約は、一方のオブジェクトが他方の部分となるような構造的な包含関係を表すものである。集約エッジは汎化エッジと区別するために斜め右下 45 度の磁場を与える。

我々は、オブジェクト図の構成要素を区別するため図 9のように定義する。（a）は四角の中にクラス名が書いてある図形をノードとして定義する。（b）は直線の中心に「`asso`」が書いてある図形を関連エッジとして定義する。（c）は直線の始点に矢印があって直線の中心に「`gen`」が書いてある図形を汎化エッジとして定義する。（d）は直線の終点に「＊」があって直線の中心に「`agg`」が書いてある図形を集約エッジとして定義する。まず、これらのオブジェクト図の構成要素を解釈する生成規則を定義する必要がある。定義方法は、図 9のように図形を図形エディタに描いて開いた CMG 入力ウィンドウで行う。例えば、集約エッジの場合は、非終端シンボルの名前として CMG 入力ウィンドウの名前の欄に `aggregation` と書き、属性 `start`、 `end` は直線の始点、終点にするように書く。制約の欄には、直線の中心とテキストの中心、直線の終点と「＊」の中心を等しくする制約を選び、非終端シンボル `aggregation` の定義を行う。さらに、非終端シンボル `ObjectGraph` を生成するように、オブジェクト図を再帰的に定義する生成規則を定義する。

また、解釈したい図 10の図形を選んで軟かいレイアウト CMG 入力ウィンドウ（図 5）を開く。ユーザは自動的に書き出された非終端シンボルの名前から `node` と `edge` の構成要素を選び、マグネティックスプリングモデルの定数を決める。名前として `ObjectModel`、レイアウトの名前の欄には `magnetic`、再帰的に生成される非終端シンボルの欄には `ObjectGraph` を書く。次に、それぞれの `edge` の構成要素にエッジの種類と磁場を与える。これらの与え方は、 `edge` の構成要素の欄に選ばれた構成要素の名前の最初にエッジの種類と磁場の角度をユーザが付け加える。この例では次のように記述する。

```
{undirect(0) association}
{direct(90) generalization}
{direct(45) aggregation}
```

これらの生成規則を用いて図 10の図形を解釈すると図 11のような結果が得られる。

6. 関 連 研 究

空間パーサ生成系の関連研究としては、 1 章で説明

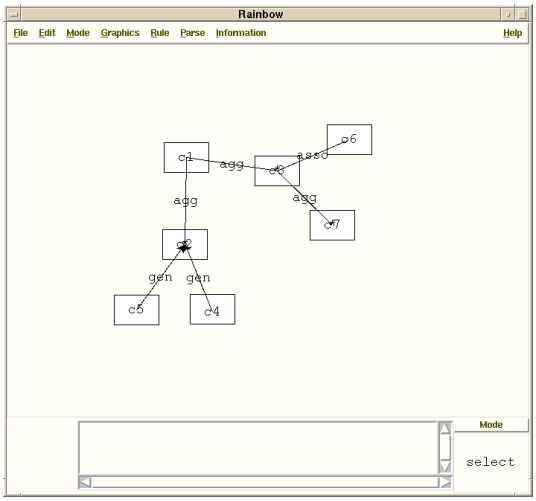


図 11 レイアウト後のオブジェクト図
Fig. 11 Object diagrams after layout

したように SPARGEN[1]、 Penguins[2)3)] などがある。我々は空間パーサ生成系にとして恵比寿[4)5)7)]、 VIC[8)] を開発している。

その他のレイアウトシステムとして TRIP[18)] がある。 TRIP とは、テキスト形式の抽象オブジェクトとその関係からレイアウトされた図形を生成するシステムである。抽象オブジェクトとその関係はユーザが定義するマッピング規則により、図形オブジェクトとその関係に変換され、レイアウトシステムより図形が生成される。マッピング規則は、ユーザが Prolog で記述している。 TRIP では、制約に基づいて図形要素の位置が決められてから図形が生成されるが、そのあと図形要素間に制約が課せられないので、制約を保つつつ図形を動的に編集することができない。 TRIP でもグラフレイアウトシステムが存在し、無向グラフの描画アルゴリズムを使って図形のレイアウトを行うが、図形要素の位置を決める図形関係と混じってレイアウトすることはできない。

7. 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」について述べた。

レイアウト制約は、軟かいレイアウト制約と硬いレイアウト制約を実装した。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、リスト構造制約、木構造制約などを実装した。ここで、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、リスト構造制約と木構造制約は、それぞれグラフをリスト構造と木構造にレイアウトする場合に用いる。硬いレイアウト制約は、図形の座標や図形間の距離などの制約を具体的に与えたい場合に用いる。

また、「Rainbow」のアプリケーション作成例として、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」と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」の例を示した。

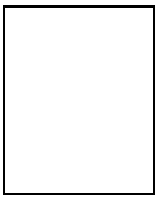
現在は通常の制約と硬いレイアウト制約を同じ強さで扱っているが、今後は制約階層の導入が考えられる。制約階層とは、強さと呼ばれる制約の優先度を用いて、強い制約をできるだけ多く満し、より弱い矛盾する制約を無視するように制約系の解を求める方法である。通常の制約を硬いレイアウト制約より強い制約として定義することにより、硬いレイアウト制約を起動することで、以前に充足していた通常の制約が満されなくなる場合の問題を解決することができると考えられる。

参 考 文 献

1) Eric J. Golin and Tom Magliery: A Compiler Generator for Visual Languages, *Proceedings of the IEEE Symposium on Visual Languages*, pp. 314–321, (1993).
2) 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).
3) 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).
4) 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).
5) 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).
6) Kim Marriott: Constraint Multiset Grammars, *Proceedings of the IEEE Symposium on Visual Languages*, pp. 118–125 (1994).
7) Akihiro Baba and Jiro Tanaka: Eviss : a Visual System Having a Spatial Parser Generator, *Proceedings of Asia Pacific Computer Human Interaction*, pp. 158–164 (1998).
8) Kenichirou Fujiyama, Kazuhisa Iizuka and

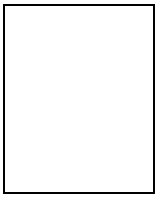
Jiro Tanaka: VIC:CMG Input System Using Example Figures, *Proceedings of the International Symposium on Future Software Technology*, pp. 67–72, (1999).
9) Sitt Sen Chok, Kim Marriott and Tom Paton: Constraint-based Diagram Beautification, *Proceedings of the IEEE Workshop on Visual Languages*, pp. 12–19 (1999).
10) Michael Sannella: Constraint Satisfaction and Debugging for Interactive User Intefaces, Technical report, University of Wasington (1994).
11) Peter Eades: A Heuristic for Graph Drawing, Congressus Numerantium, Vol. 42, pp. 149-160 (1984).
12) 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).
13) John Q. Walker: A Node-positioning Algorithm for General Trees, Software Practice and Experience, Vol. 20, No. 7, pp. 685-705, (1990).
14) Peter Chen: The Entity-Relationship Model: Towards a Unified View of Data, *ACM Transactions Database System*, Vol. 1, No. 1, pp.9-36 (1976).
15) James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen: *Object-Oriented Modeling and Design*, Prentice-Hall International (1991).
16) 中島哲, 田中二郎: オブジェクト指向方法論に基づくオブジェクト図の自動レイアウト, 情報処理学会論文誌, Vol. 39, No. 12, pp. 3282-3293 (1998).
17) Takayoshi Noguchi and Jiro Tanaka: New Automatic Layout Method based on Magnetic Spring Model for Object Diagrams of OMT, *Proceedings of Future Software Technology*, pp. 89–94 (1998).
18) Tomihisa Kamada and Satoru Kawai: A General Framework for Visualizing Abstract Objects and Relations, *ACM Transactions on Graphics*, Vol. 10, No. 1, pp. 1–39 (1991).

(平成8年2月 4 日受付)
(平成8年5月11日採録)



丁　　錫泰（学生会員）

1989 年韓国全南大学電算学科卒業. 1996 年筑波大学理工学研究科修士課程終了. 現在同大学工学研究科博士課程在学中. ヒューマンインタフェース, ビジュアルシステムなどに興味を持っている. 日本ソフトウェア科学会会員.



田中　二郎（正会員）

1975 年東京大学理学部卒。 1977 年同大学院修士課程修了。 1984 年米国ユタ大学計算機科学科博士課程修了、 Ph.D. in Computer Science。 1993 年より筑波大学、電子・情報工学系に勤務。現在、筑波大学、電子・情報工学系教授。プログラミング一般やヒューマンインタフェースに関する研究を行っている。最近では、スクリプト言語に興味を持っている。 ACM、 IEEE Computer Society、電子情報通信学会、人工知能学会、日本ソフトウェア科学会、各会員。