筑波大学大学院博士課程

工学研究科修士論文

インタラクティブ性を考慮した
オブジェクト図の自動レイアウト手法

電子・情報工学専攻

著者氏名　野口 隆佳

指導教官　西川 博昭

平成 12 年 2 月

要旨

本研究ではオブジェクト指向方法論で用いられるオブジェクト図の自動レイアウトアルゴリズムについて論じる。オブジェクト図はオブジェクト指向方法論でも大変重要な図であり、そのためのレイアウトアルゴリズムはCASEツールにおけるオブジェクト図の描画に大変有効である。本アルゴリズムは「グラフ自動描画法」に着目し、オブジェクト図のレイアウト手法として適用した。またレイアウトの基準として、いかにグラフをきれいに見やすくレイアウトするかという基準の他に、CASEツールにおいて、グラフの描画途中におけるユーザとのインタラクティブ性を考慮した新たなアルゴリズムを提案した。またレイアウトシステムの枠組を提案し、オブジェクト図の描画にとどまらず、3次元のグラフシステムへの応用についても述べる。

目次

図一覧

第 1 章

序論

近年、オブジェクト指向に基づくソフトウェア開発において分析・設計といった上流工程が特に重要視されてきており、それにともない上流工程を主に支援するオブジェクト指向方法論が各種提案されている [4]。

オブジェクト指向方法論には OMT（Object Modeling Technique）法 [1] や Booch 法 [5] など、すでに広く実用化されているものもあり、それらのモデル作成の為の統一言語として UML(Unified Modeling Language)[3] が提唱されている。また、それらの方法論や言語を支援する CASE ツールのソフトウェア開発への導入も活発化している。一般にオブジェクト指向方法論に基づくソフトウェア開発では、開発対象システムを構成するオブジェクトの静的構造を表現するモデル（オブジェクトモデル）を、図式表記を用いて視覚的に記述する。この図はオブジェクト図（クラス図）と呼ばれ、システム開発の基盤となる最も重要なダイヤグラムである。システム開発ではこの種のダイヤグラムは一般に複数の開発者により利用され、開発者間のコミュニケーションの道具ともなるため、見やすく可読性の高いレイアウトが要求される。既存のオブジェクト指向 CASE ツールの大半は、オブジェクト図を効率良く描くための図形エディタを備えている。しかし、それらの図形エディタでは、オブジェクト図を構成する基本部品をすべてユーザが配置するといった操作環境となっているため、ユーザは見やすいレイアウトを常に考慮しながら配置していかなければならず、オブジェクト図が複雑化するに従って作図に手間がかかる。

以前よりグラフ描画アルゴリズムを用いたダイヤグラムの自動レイアウト技術に関する研究は盛んに行われているが [22, 23]、最近では Rational Rose[11] に代表されるような商用のツールにおいても自動レイアウト技術が採り入れられ実用化が進みつつある。しかしながら、既存のグラフ描画アルゴリズムは単純なグラフに対しては良好なレイアウトが得られるが、現状ではオブジェクト図のような複雑なグラフに対しては有効なアルゴリズムは非常にまれである。そこでオブジェクト図の自動レイアウト

機能を持つツールでは、オブジェクト図を単純な構造のグラフとみなすことにより既存のアルゴリズムを利用している。そのため、オブジェクト図に特有な美的基準を考慮できず、高品質なレイアウトを得ることは難しい。そこで本研究では、オブジェクト図中のクラス間の関係の種類に着目し、関係の種類によってクラス間の位置関係を決定するアルゴリズムを提案した [16, 17]。さらに、レイアウトは完成したオブジェクト図に対して行うのみではない。エディタ上でオブジェクト図を描画する途中においてもその描画のサポートとなるレイアウトは必要である。そこで、レイアウトの品質よりも、インタラクティブ性を重視したレイアウトアルゴリズムを提案した [18, 19]。

第 2 章

オブジェクト指向方法論 OMT と統一モデリング言語 UML

オブジェクト指向によるシステムの開発は、従来の機能中心の手続き型言語でのシステム開発に対し、実世界の自然なモデル化、再利用・拡張の容易さ、仕様変更などによる実装変更の容易さなどの点で優れているとされ、注目されている [4, 6]。そこで 90 年代初期からオブジェクト指向のための方法論が数多く発表された [1, 5, 8, 9, 10]。しかしそれぞれの方法論は、それぞれ独自の表記法を使用していた。この独自性のためにシステムの設計者は方法論の選択が重要になっていた。

なかでも OMT(Object Modeling Technique) 法 [1] は十分な記述能力、実用的な作業プロセスの提示等の点で他の方法論よりも優れている。

また、 OMT 法 [1] の開発者である James Rumbaugh と、 Booch 法 [5] の開発者である Grady Booch と、 OOSE(Object-Oriented Software Engineering) 法 [9] の開発者である Ivar Jacobson により UML(Unified Modeling Language)[2, 3] という標準のモデリング言語が開発された。

2.1 オブジェクト指向方法論 OMT

数あるオブジェクト指向方法論 [5, 7, 8] の中でも OMT(Object Modeling Technique) 法 [1] は他の手法と比べ、図表やモデルが詳細に決っている事から、十分な記述能力を備えている点と、作業プロセスを具体的に指示しており実用的である点が優れている [6]。また現在主流になりつつあるモデリング言語である UML の基本となる方法論の一つである。

OMT 法ではシステムを 3 種類の異なる視点からモデル化し、その 3 種類のモデルを中心にシステムを開発していく。開発作業は分析、システム設計、オブジェクト設

計、実装の 4 段階のフェーズに分類され、それぞれの過程で 3 種類のモデルを進化させていくことで開発を進める。本節では OMT において中心となる 3 種類のモデルについて概説する。 OMT において用いられる 3 種類のモデルは、オブジェクトモデル、動的モデル、機能モデルの 3 種類である。また、これらのモデルを視覚的に記述するために各種の図が用いられる (図 2.1)。

オブジェクトモデル

システム内のオブジェクト、オブジェクト間の関係、そしてオブジェクトの各クラスを特徴づける属性や操作を示すことによって、システムの静的な構造をとらえるためのモデルである。またオブジェクトモデルはオブジェクト図を用いて表現される。オブジェクト図にはシステムを構築するオブジェクトの静的関係や、個々のオブジェクトの詳細 (属性、操作など) が記述される。

動的モデル

時間と操作の順序にかかわるシステムの側面を記述する。ここで、操作が何を行なうか、それらが作用する対象は何か、どのように実装されているかということには関係せず、 “制御” すなわち起こるべき操作の列を記述するというシステムの側面のみをとらえたものである。また動的モデルは状態図と事象トレース図によって表現される。状態図は個々のオブジェクトの状態遷移を記述し、動的モデルは複数の状態図から構成される。事象トレース図はオブジェクト間のメッセージ送受信により発生するイベントの推移を記述する。これもシナリオとよばれる、システムが実行している際に発生する事象系列の数だけ複数存在する。

機能モデル

値の変換にかかわるシステムの側面を記述する。それは関数、写像、制約そして機能依存性といったものからなる。機能モデルはシステムが何を行なうか、いつ行なわれるかとは無関係である。また機能モデルはデータフロー図によって記述される。データフロー図は、値の間の依存性、入力値からの出力値の計算、そして機能を記述している。

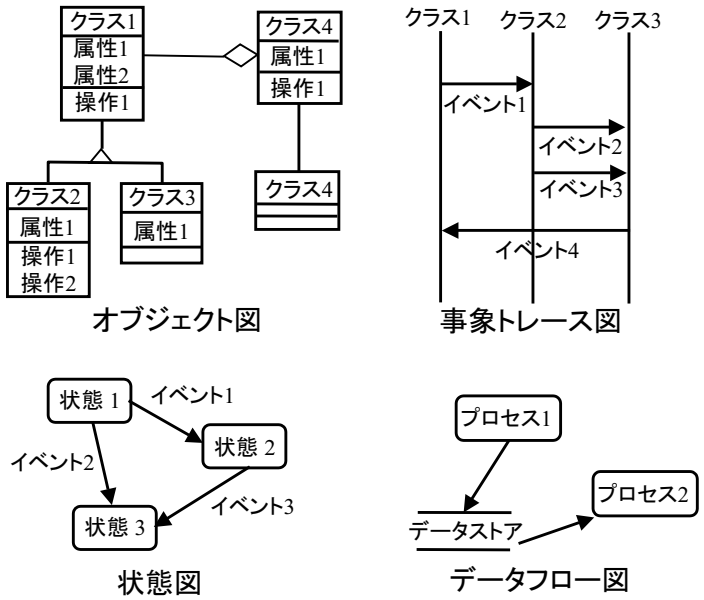

図 2.1: OMT 法で用いられる図

2.2 統一モデリング言語 UML

UML は多数あるオブジェクト指向方法論の統一モデリング言語として開発された。UML の主要部分は Booch、 OMT、 OOSE の方法論で用いられるモデルを基本としている。

UML には 5 種類のビューと呼ばれるものと 9 種類のダイアグラムによって構成されている。 9 種類のダイアグラムは、ユースケース図 (図 2.3)、クラス図 (図 2.4)、オブジェクト図 (図 2.5)、状態図 (図 2.6)、シーケンス図 (図 2.7)、コラボレーション図 (図 2.8)、アクティビティ図 (図 2.9)、コンポーネント図 (図 2.10)、配置図 (図 2.11) であり、 5 種類のビューはこの 9 種類のダイアグラムのいずれか、またはその組合せによって表現される (図 2.2)。ビューとはシステムを把握するために必要ないろいろな面からの視点である。通常システム全体を一つのグラフではっきりと定義し、理解や伝達をするのは不可能である。そこでシステム全体を把握するためには、機能面、機能以外の面、構成面など多くの違った側面からシステムを見る事が必要になってくる。そのために UML では５つのビューというものを定義し、その目的にあったシステムの側面を表現する。 5 つのビューとは以下の通りである。

- **ユースケースビュー** 外部アクターの観点からシステムの機能を表すビュー。ユースケース図またはアクティビティ図によって記述される。

- **論理ビュー** システムの静的構造と動的振る舞いに関して、システムの機能がどの

ように設計されているかを表すビュー。静的構造はクラス図とオブジェクト図を使って表現され、動的振る舞いは、状態図、シーケンス図、コラボレーション図、アクティビティ図によって記述される。

・**コンポーネントビュー** コード・コンポーネントの構成を表すビュー。コンポーネント図から構成される。

・**並行性ビュー** 並行システムの通信と同期の問題を処理するために、システムの並行性を表すビュー。並行性ビューはシステム開発者とインテグレーターのためのビューであり、動的ダイアグラム (状態図、シーケンス図、コラボレーション図、アクティビティ図) と実装ダイアグラム (コンポーネント図、配置図) から構成される。

・**配置ビュー** コンピュータと装置を含む物理アーキテクチャへのシステムの配置を表すビュー。配置図により構成される。

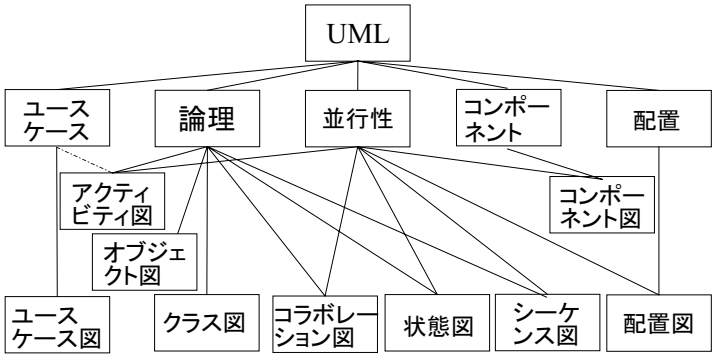

図 2.2: ビューとダイヤグラムの関係

2.3 OMT と UML との関係

2.3.1 OMT と UML でそれぞれ用いられる図の対応

前節でも述べた通り、 OMT は UML の基本となる方法論の一つである。よって OMT で用いられる主要な図は UML においても利用されている。図 2.12に OMT における各図と UML で扱われている図の関係を示す。

OMT におけるオブジェクト図は、 UML ではクラス図とオブジェクト図の総称とほぼ同じものを指す。 UML でのクラス図は OMT においてもクラス図と呼ばれるが、UML でのオブジェクト図は OMT ではインスタンス図と呼ばれる。 OMT ではオブ

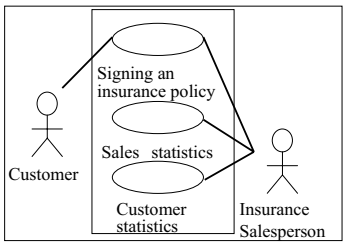

図 2.3: ユースケース図

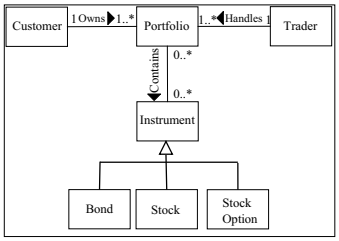

図 2.4: クラス図

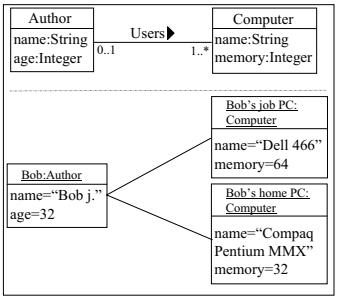

図 2.5: オブジェクト図

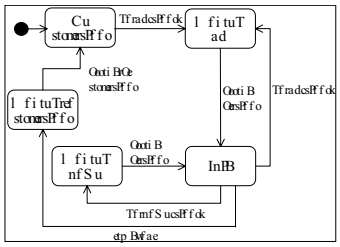

図 2.6: 状態図

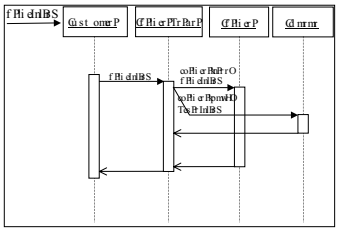

図 2.7: シーケンス図

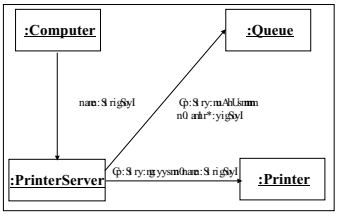

図 2.8: コラボレーション図

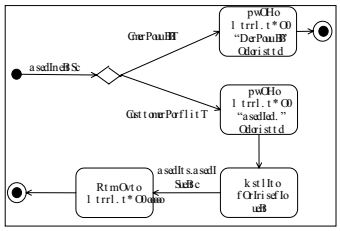

図 2.9: アクティビティ図

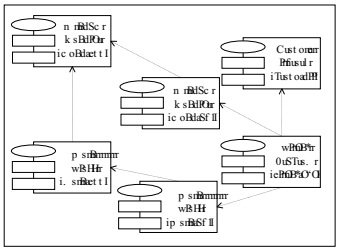

図 2.10: コンポーネント図

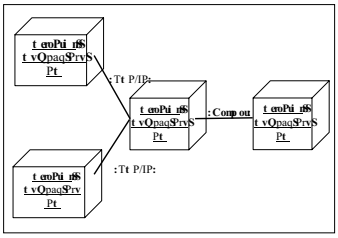

図 2.11: 配置図

OMT	UML
オブジェクト図	クラス図 オブジェクト図
事象トレース図	シーケンス図
状態図	状態図
データフロー図	なし

図 2.12: OMT と UML の図の関係

ジェクト図とはクラス図とインスタンス図両方を合わせた総称のことである。なお本論文でのオブジェクト図とはOMTにおけるオブジェクト図を指すものとする。

状態図は個々のオブジェクトの状態遷移を記述する図である。 OMTにおける状態図はUMLでの状態図と同じものである。

事象トレース図はオブジェクト間のメッセージ送受信により発生するイベントの推移を記述する。事象トレース図はUMLにおけるシーケンス図にあたるダイヤグラムである。データフロー図は、値の間の依存性、入力値からの出力値の計算、そして機能を記述している。

2.3.2 OMT、UMLにおけるオブジェクト図の重要性

OMTにおける3モデルは、それぞれがどのような問題においても同等に重要というわけではない。ユーザインターフェースやプロセス制御のような相互作用とタイミングにかかわる問題においては、動的モデルが重要であり、コンパイラや工学計算のような重要な計算処理を含む問題においては、機能モデルが重要になる。一方、オブジェクトモデルは初めの問題要求から必要なオブジェクトを取り出し、それらのオブジェクトを作業可能な単位の集まりとして組織化する全体を通して重要な図である。またオブジェクトモデルの静的構造は他の動的モデル、機能モデルの構築の基本となるものであり、またそれら2つのモデルの結果はオブジェクトの操作としてオブジェクトモデルに組み込むのである。

UMLはオブジェクト指向モデルを表現するための表記法と規則である。よってOMTのようにどのようにモデルを表現するのかというUMLを使用するためのプロセスは規定されていない。しかし、 UMLの設計の段階においていくつかのプロセスを考慮して設計されている。そのプロセスはOMT、 Booch、 OOSEの考え方に基づいたものにいなっている。決められたプロセスというものはモデリング言語であるために存在しないが、それぞれの図の果たす役割は決められている。なかでもクラス図は、オブジェクトの状態を示す状態図、オブジェクト間の協調を示すコラボレーション図などの動的ダイヤグラムを定義するための土台となる図である。つまりUMLにおいてもOMTのオブジェクト図を表すクラス図は他の図の土台を定義するための重要な図として用いられている。

第 3 章

グラフ自動描画アルゴリズム

図による視覚的な表現はインターフェースとして最も基本的なものの一つであり、使われる範囲も広く、簡便で親しみやすいという特徴を持っている。よって設計図、家系図、論理回路図など日常的にも多く利用されている。なかでも我々が普段、研究や仕事に扱う事の多い、フローチャート、組織図、システム構成図などは実体をノード、実体間の関係をエッジとしたグラフとして表現される。またこういったグラフはシステム工学、情報工学、ソフトウェア工学など様々な分野において、基礎的なモデルとして広く使われている。最近ではコンピュータの発達にともない、グラフを視覚的に表示したり、操作したりする事が強く求められるようになってきている。そのためこのような図の自動レイアウトを行う、グラフ描画アルゴリズムが数多く提案されている。

3.1 グラフの美的基準

グラフ描画アルゴリズムが対象とするグラフは、その図的性質によって様々である。それゆえグラフの種類によって美的基準もまちまちである。そこで各クラスの特徴を考慮した描画法がグラフの種類ごとに開発されている [24, 25, 26, 33, 34, 35]。

グラフでいう美的基準とは、描画規約と描画規則のことである [21]。描画規約はノードとエッジに関する基本的約束であり、描画の際に必ず満たされる制約である。描画規則は “良い” 描画の基準となるものである。ただどのようなグラフが “良い” かは個人により変化し、主観による部分が大きい。しかし、グラフの構成はノードとエッジというように、極度に単純化できるため、細かい違いはあるものの、グラフの性質からくる、共通で、基本的ないくつかの良い描画の基準を識別する事ができる [21]。それらの描画規則の例をいくつか挙げる。

- エッジの折れ点数を最少とする

- エッジの交差数を最少にする
- 対称性 (がある場合) を顕示する
- ノードとエッジの配置や配線の密度を一様化する
- 子ノードを対称に配置する
- 階層構造を垂直あるいは水平に顕示する

グラフ描画アルゴリズムとは一般的に、対象とするグラフの特長から優先度の高い順に、これらの描画規則を取り出し、最適基準または制約条件とする。そして描画規約を最適化問題のゴールとして、複数のステップにおいて順次最適化問題を解くことにより、多くの描画規則を満たしていくものである。しかし、木などの交差の無いグラフを除くと、規則を満たせる効率的な方法が無いことが多く、最適解を得ることは困難なことが多い。したがって、ヒューリスティクスを用いた種々の発見的方法によるアルゴリズムも開発されてきている [33]。

3.2 グラフの種類

グラフ描画アルゴリズムが対象とするグラフは、図 3.1に示すように大きく 4 種類に分けることができる。

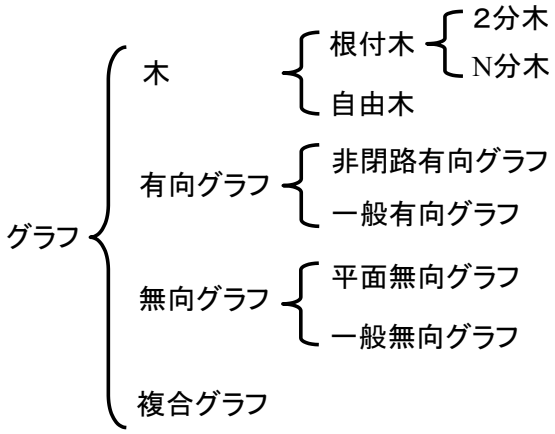

図 3.1: グラフのクラス分類

木には更に根付き木 (図 3.2) と自由木 (図 3.3) に分ける事が出来る。根付き木とは特定の根となるノードが存在する木であり、自由木とは特定の根となるノードを持たない木である。根付き木の描画アルゴリズムでは Walker の木描画アルゴリズム [26]

が最もよく知られている。このアルゴリズムはそれまでの 2 分木に対する描画アルゴリズム [27, 28, 29] の欠点を克服し、 n 分木に拡張したアルゴリズムである。

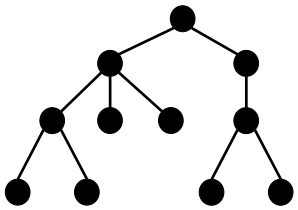

図 3.2: 根付き木

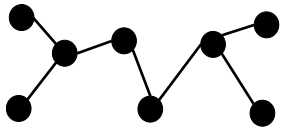

図 3.3: 自由木

有向グラフには閉路をもたない非閉路有向グラフ (図 3.4) と、閉路をもつことが許される一般有向グラフ (図 3.5) にわけられる。有向グラフの描画では、全ての辺の方向をある一つの流れの方向へ従わせる描画法が一般的である。このような描画法を同方向描画という。非閉路有向グラフでは同方向描画可能であり、階層的描画法 [30, 31] が最も一般的である。

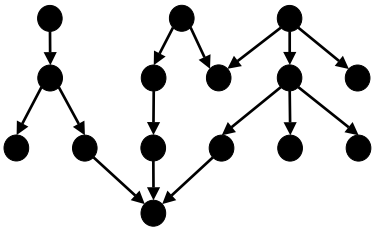

図 3.4: 非閉路有向グラフ

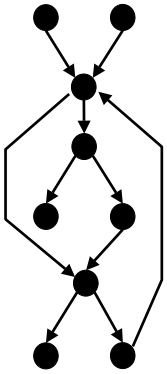

図 3.5: 一般有向グラフ

無向グラフは交差するエッジが無いように平面に描くことのできる平面グラフ (図 3.6) と、エッジの交差が必ず存在してしまう事を許す一般無向グラフにわけられる。しかし、一般無向グラフにおいてエッジの交差が存在する場合には、その交差を架空のノードに置き換えることによって平面グラフとみなすことが出来る。平面グラフの描画法では力学モデルに基づいたスプリングモデル [24] が有名である。

複合グラフ (図 3.7) は、上記の 3 種類のグラフがノード間の関係を表すエッジを 1 種類に限定しているのに対し、それらのグラフ構造を混在させたグラフである。複合グラフには同時に、有向エッジや無向エッジ、木構造が含まれる。実際に我々が研究開発等に用いる図は有向グラフ、無向グラフという風に限られてはおらず、ノード間

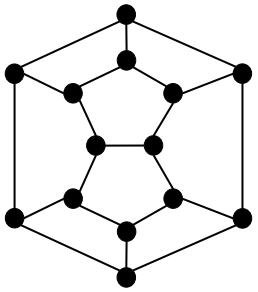

図 3.6: 無向グラフ

の関係は複数あるのが普通である。したがって複合グラフの自動描画法の開発は、人間の思考と機械による図の生成との効果的な統合を生み出し、 CAD などの様々なツールの優れたビジュアルインターフェースに役立つと考えられる。しかし、このような複合グラフのためのアルゴリズムは、そのアルゴリズムの難しさからあまり存在しない。複合グラフの描画法としては隣接関係と包含関係の 2 種類の関係を考慮できる描画法 [32] がある。この描画法では隣接関係を有向グラフで表し、包含関係を階層を表す矩形で表す。しかしこのアルゴリズムは関係の種類が 2 種類に限定されており、それ以上の種類の関係が存在する場合には適用できない。

そこで、三末・杉山によってエッジの向きや方向を柔軟に制御する事が出来るマグネティックスプリングモデル [25] という手法が開発された。本論文において我々はこのマグネティックスプリングモデルを拡張し、オブジェクト図のレイアウト手法として適用する。次節においてマグネティックスプリングモデルについて詳しく解説する。

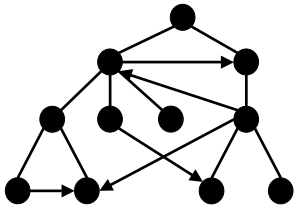

図 3.7: 複合グラフ

3.3 マグネティックスプリングモデル

マグネティックスプリングモデル [25] は無向グラフ描画法であるスプリングモデルを改良し、複数種類の有向エッジを含む複合グラフを描画する事が出来るアルゴリズムである。本節ではマグネティックスプリングモデルについて詳しく解説する。

マグネティックスプリングモデルにおいて考慮されている描画規則は以下のとおりである。

1. エッジの長さが均一

2. エッジの交差数が最少

3. 対称性を表示

4. ノードを均一に分配

5. エッジが特定の向きまたは方向に従う

1. 〜 4. はマグネティックスプリングモデルの基礎であるスプリングモデルで採用されている描画規則である。 5. がマグネティックスプリングモデルの最大の特徴を示す描画規則である。この規則により、元来無向グラフの描画法であったスプリングモデルを複合グラフの描画法に拡張できるのである。

マグネティックスプリングモデルの基礎であるスプリングモデルとは、ノードを鉄のリングに、エッジを力学系を形成するバネに置き換え、リングの安定状態を見つけることで適切なレイアウトを求めるものである (図 3.8)。それに対して、マグネティックスプリングモデルでは、スプリングの力の他にエッジに働く「回転力」を定義する。これによって、エッジの向きや方向の制御を行う (図 3.9)。回転力を定義するのにここでは「磁場」の概念と「磁針」の概念を導入している。エッジを「磁針」とし、グラフが、ある「磁場」の中にあるとする。それにより、磁針であるエッジはグラフが置かれた磁場の北を向くように回転力が働く。

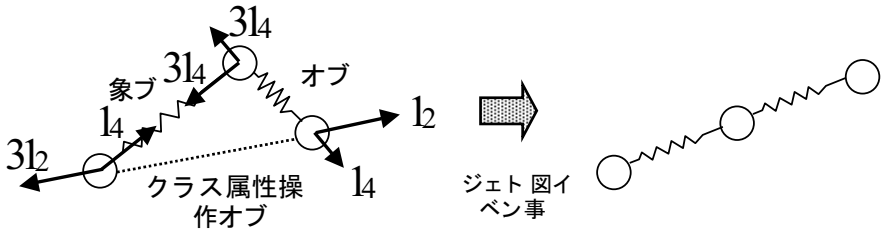

図 3.8: スプリングモデルによるグラフのレイアウト

ただし、ここでいう磁場は仮想的なもので現実のそれとは同じ性質を持たない。以下に、マグネティックスプリングで導入する力と磁場について説明し、マグネティックスプリングモデルのアルゴリズムを示す。

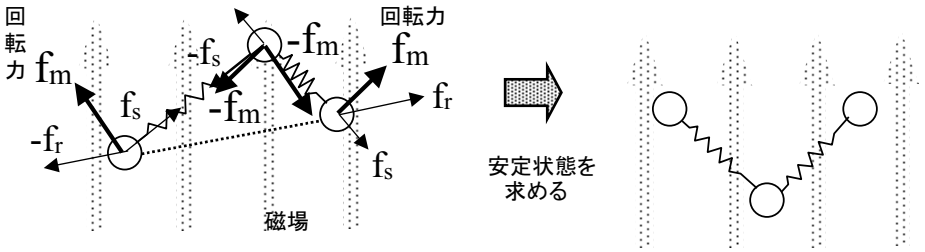

図 3.9: マグネティックスプリングモデルによるグラフのレイアウト

3.3.1 力

- スプリングによる力
  すべてのエッジはスプリングとみなされ両端の頂点にはそのスプリングによって引力あるいは斥力が働く。 $d > d_0$ の時は引力、 $d < d_0$ の時は斥力、 $d = d_0$ の時は力は働かない。

$$f_s = c_s log(\frac{d}{d_0})$$

  c_s は他の力とのバランスを制御する係数 (以下に現れる c_r、 c_m も同様)、 d は現在のスプリングの長さ、 d_0 はスプリングの自然長である。

- 非隣接ノード間の斥力
  非隣接ノード間には斥力が働く。

$$f_r = c_r \frac{1}{d^2}$$

  d はノード間の距離である。

- 磁場から受ける回転力
  有向エッジの場合、エッジ (磁針) は、定義された磁場における北の向きに揃えられる (図 3.10)。無向エッジの場合、エッジ (磁針) は磁場の向きは関係なく方向のみが揃えられればよいので、北か南の近い方への回転力となる (図 3.11)。このような磁針を無向磁針と呼ぶ。

$$f_m = c_m b d^{\alpha} |t|^{\beta}$$

  b は基準点 (実現上は辺の中点) における磁場の強さ、 d は現在のエッジの長さ、 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。定数 α はエッジの長さの、定数 β は t の、それぞれ回転力への影響を制御する。

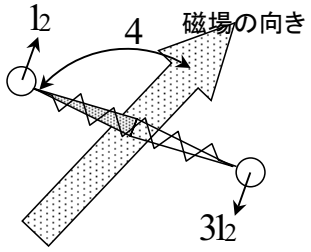

図 3.10: 有向エッジが磁場から受ける回転力

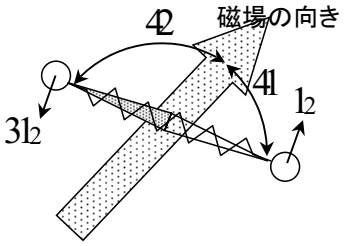

図 3.11: 無向エッジが磁場から受ける回転力

3.3.2 磁場

マグネティックスプリングモデルでは磁場として以下のようなものがある。

- 平行磁場
  平面上のどの場所でも向きが一定の磁場。

- 放射状磁場
  ある点を中心に放射状に外側あるいは内側に向く磁場。

- 同心円磁場
  ある点を中心に同心円状に向く磁場。

先程述べたようにここで扱う磁場は現実の磁場とは違い、人工的な性質を自由に与えることができる。その一つとして上記のような磁場をおたがいに影響し合わないように複数組み合わせて与えることができる。これを複合磁場という。

3.3.3 マグネティックスプリングモデルのアルゴリズム

各ノードを初期配置する;
(定められた回数のループ)
{
  for v=1 to (ノード数)
  {
    for w=1 to (ノード数)
    {
      if (ノード v と w がエッジ e で隣接)
        then
          エッジ e に関して v が受ける f_s を計算する;
          エッジ e に関して v が受ける f_m を計算する;
          $f_v := f_v + f_s + f_m$;
        else
          ノード w に対して v が受ける f_r を計算する;
          $f_v := f_v + f_r$;
    }
  }
  ノード v を δf_v 移動する;
}

以上がマグネティックスプリングモデルのアルゴリズムである。マグネティックスプリングモデルはヒューリスティックスを用いてそれぞれのノードの位置を一気に決めるアルゴリズムではなく、バネと斥力による力学モデルのシミュレーションによるレイアウトである。シミュレーションは微小移動のイタレーションによって行なう。各ノードに対してそれぞれにかかる力を第 3.3.1 節の定義式から求める。全てのノードにかかる力を求めた後、それぞれのノードを 3 つの力の合力の δ 倍だけ移動する。この作業を一定回数、あるいは全てのノードが安定するまで行なった結果をレイアウト結果として表示するのである。

第 4 章

オブジェクト図のレイアウト手法

本章ではまずオブジェクト図のグラフ構造について述べる。またその構造に対するマグネティックスプリングモデルの適合性について述べる。その後マグネティックスプリングモデルを適用するにあたり改良した 3 点について述べる。その 3 点とは

- 3 種類の関係エッジへの磁場の割り当て
- ノードの大きさを考慮し、ノードの重なりの防止
- 新たな力の導入による、レイアウトのインタラクティブ性の向上

である。

4.1 オブジェクト図のグラフ構造とマグネティックスプリングモデルの適合性

オブジェクト図とはシステムの静的な構造を表す図である (図 4.1)。

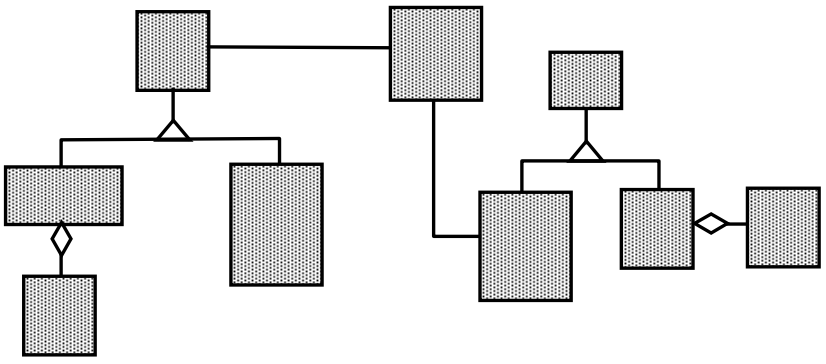

図 4.1: 一般的なオブジェクト図

オブジェクト図では、クラスを長方形で表現し、クラス間の関係をクラス間を結ぶ線で表す。クラスは 3 つの領域からなり、上から順に、クラス名、属性リスト、操作リストをそれぞれ内容として持つ (図 4.2。各属性名は型やデフォルト値といった追加的な詳細が後ろに付けられる。各操作名は引数リストや戻り値の型といった追加的な詳細が後ろに付けられる。よってクラスを表す長方形の大きさは属性リスト、操作リストの長さや、クラス名、属性名、操作名の長さによって決まり、各クラスによってまちまちである。

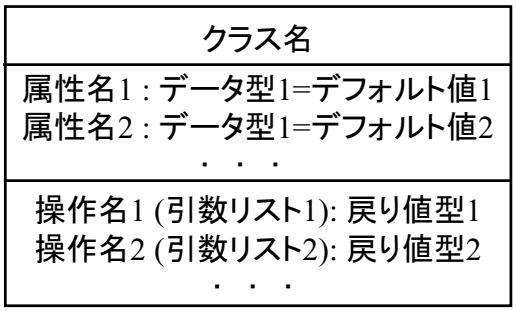

図 4.2: クラスの表記法

クラス間の関係は関連、継承、集約と 3 種類あり、それぞれ何も付かない線、三角形が間に付く線、菱形が間に付く線で表現される (図 4.3)。

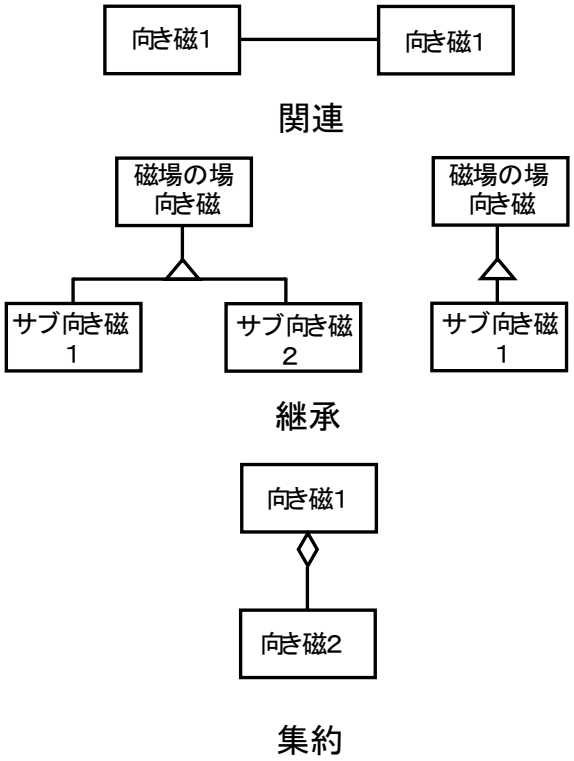

図 4.3: オブジェクト図の表記法

そこでクラスを表す長方形をノード、クラス間を結ぶ線をエッジとすることでオブジェクト図をグラフととらえ、グラフ描画アルゴリズムを適用することでオブジェクト図のレイアウトを行う。第 3 章で述べたように、一般にグラフ描画アルゴリズムは、その描く対象のグラフの種類によって木、有向グラフ、無向グラフなどに分類され、それぞれ様々なアルゴリズムが開発されている [21, 22]。

UML や OMT 法で用いる他の図もオブジェクト図と同様にアクターや状態などをノードととらえ、そのノードを結ぶ線をエッジととらえることでグラフ描画アルゴリズムを適用できる。しかし、それらの図はいずれもエッジの種類が多くなく、既存の有向グラフ、または無向グラフ描画アルゴリズムの適用が容易である。しかし、オブジェクト図では明確にエッジの種類がわけられており、関連関係は無向エッジ、継承関係は有向エッジ、集約関係は有向エッジまたは木構造と、エッジの種類も多様であり、他の図に比べて非常に複雑なグラフ構造となる。

そこで本研究ではオブジェクト図の関係エッジの種類に着目し、エッジの種類によってエッジの方向を変える事によってレイアウトを行う手法を提案する。これは以下の理由からオブジェクト図のレイアウトに有効であると思われる。

1. オブジェクト図の表記法として、

- 関連関係は左から右に読めるようにクラスの配置をする方が良い
- 継承関係はできる限りスーパークラスを上に書きサブクラスを下に書くべきである

などノード間の関係によって推奨されるノードの位置関係が定義されている。

2. エッジの種類が多いグラフ (3 種類) であることから、オブジェクト図が大きくなるほど方向のみでノード間の関係が把握できるレイアウトは判りやすいレイアウトであるといえる。

エッジの方向を決めるアルゴリズムとして、有向グラフのアルゴリズムがある。しかし一般に、有向グラフ描画アルゴリズムとして提案されているものは、できるだけ多くの線が一定方向を向くようなアルゴリズムであり、複数の種類の線をそれぞれ違う方向に向けるようなアルゴリズムは無い。そこで三末・杉山によって提案されたのがマグネティックスプリングモデル [25] である。第 3.3 節でも述べたとおりこのアルゴリズムの特徴は、複数種類のエッジをそれぞれ個別に制御しレイアウトを行える点である。この特徴は本研究でのクラス間の関係の種類に着目したレイアウトに大変適している。

4.2 関係エッジへの磁場の適用

マグネティックスプリングモデルによりオブジェクト図のレイアウトを行なうにあたり、オブジェクト図に存在する 3 種類の関係エッジにそれぞれ、違う種類の磁場を与える。違う磁場を与えることにより、それぞれのエッジの方向を制御し、見やすいレイアウトを実現する。

エッジの種類が 3 種類あることから、3 種類のエッジに別々に、影響し合わない磁場を与えることができる複合磁場を用いることにする。

それぞれのエッジへどの種類の磁場を与えるかは、オ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を考慮し以下のように決定した。

- 関連
  関連は本質的に双方向性を持つ。これは 2 クラス間の関係において、どちらの方向をたどるのも同じ意味を持つことから、方向の無い関係であるといえる。2 つのクラス間に向きが無いということから両者のクラスに優劣が無い。よって関連辺に関しては無向磁針を採用する。これにより関連辺の方向のみが決り、向きは磁場の北向きに近い方のノードが北を向くようにできる。また OMT の

記法として関連は左から右に読めるような配置が望ましいとされている。そこで関連関係の磁場は平行磁場とし、横向きに磁場を与える (図 4.4)。

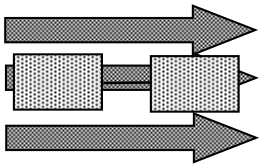

図 4.4: 関連エッジに対する磁場のかけ方

- 継承

  継承は、一つのクラスを特殊化し、その特殊化したクラス (サブクラス) と元のクラス (スーパークラス) の間の関係である。サブクラスは、スーパークラスの属性・操作を継承する。継承関係はスーパークラスからサブクラスへの有向線、またはサブクラスが複数ある場合は木構造によって表す。そして線の中間に、頂点とスーパークラスを結ぶ三角形を入れる。よって継承辺は有向磁針で表す。また OMT 記法において継承は、できる限りスーパークラスを上、サブクラスを下に書くべきであるとされている。よって磁場の向きを下向きにする (図 4.5)。

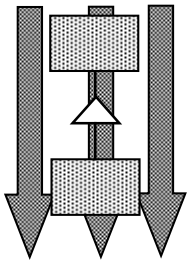

図 4.5: 継承エッジに対する磁場のかけ方

- 集約

  集約は「部分 - 全体」あるいは ”a-part-of” と表される関係である。一つのクラスがいくつかの部分クラスによって構成されている場合などがこれにあたる。集約関係は組み立てクラスから部分クラスへの有向線によって表し、組み立てクラスに小さな菱形をいれる。集約はある特別な意味づけを強固に結びついた関連の一特殊形態であること、また継承関係のように任意の深さの階層構造を持つことから、関連および継承の磁場の方向と区別するため、関連・継承の中間である斜め右下 45 ° の磁場を与える (図 4.6)。

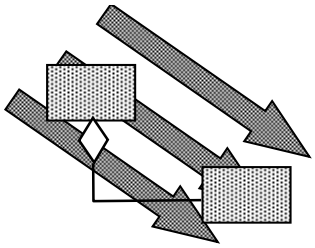

図 4.6: 集約エッジに対する磁場のかけ方

4.3 ノードの大きさを考慮したレイアウト手法

グラフ自動描画アルゴリズムでは通常ノードは全て無限小の点として扱われる。そのため、ノードの大きさの概念が無く、エッジは点から点までの距離となっている。そのため実際のグラフにそのまま適用すると、近すぎるノード同士が重なって描画されてしまう。マグネティックスプリングモデルにおいても例外ではなく、実際のシステムへ実装するためにはノードの大きさを考慮しなくてはならない。

ノードの大きさが一定のシステムである場合には、あらかじめその大きさを考慮し、エッジの長さを長めに設定する事で疑似的にノードの大きさを考慮することができる。

しかし、オブジェクト図では第 4.1 節で述べたとおり、クラスを表すノードの大きさは一定ではない。属性や操作の数が増えるとノードは縦に長くなり、クラス、属性、操作の名前が長くなるとノードは横に広くなってしまう。そこで、ノード間の距離、エッジの長さを計算する際には、対象となるノードの大きさを考慮にいれてそれぞれの長さを決めなければならない。このようにアルゴリズムをノードの大きさを考慮した手法に改良することによりノード同士が近寄りすぎ、重なりが生じることを避けることが出来る。ノードの重なりがあるレイアウトはユーザにとって見やすい美しいレイアウトとは言えず、重なりが生じないようにアルゴリズムを改良することは大変重要なことである。

そこでそのようなノードの重なりを除去する手法、またはノードの重なりが生じないようにする手法が以下のように提案された。

- レイアウト後にノードの重なりを直線的に除去する手法 [12]

この手法ではレイアウトが終わった後にノードの重なりがあった時、重なりのあるノード同士をそのノードの中心を結んだ直線上を移動させることでノードの重なりを除去する (図 4.7)。この手法には 2 つの大きな問題がある。

一つ目はこの手法はレイアウトアルゴリズムとは独立しているために、前もって Eades のスプリングモデルでレイアウトをしておかなければならない点である。よってレイ

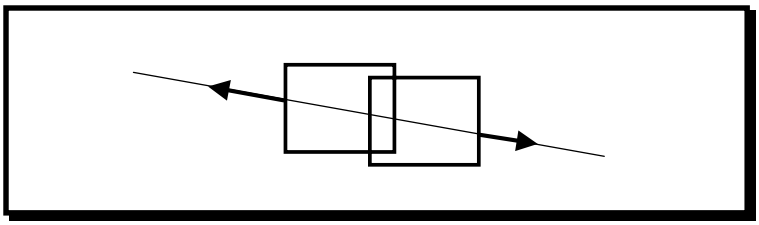

図 4.7: 直線的に重なりを除去する手法

アウトを行なう行程と、ノードの重なりを除去する行程の二つを行なって初めてレイアウトが完成する。行程の数が増える程レイアウトにかかる時間が増えるために、 CASE ツール上で頻繁に使うアルゴリズムとしてはあまり向いていない。

二つ目はレイアウトアルゴリズムによってレイアウトした結果をこの手法によって破壊してしまう点である。レイアウト後に重なりを除去するためにノードを移動すると、その移動先にまた別のノードが存在し、重なってしまう場合がある。そして、その重なりを除去するためにまたノードを移動するというように連鎖的にノードの移動が生じ、除去の前に行なったレイアウトが大きく崩れていってしまう場合が存在する。このようにこの手法ではノードの重なりを除去するために本来の目的であるレイアウトを悪化させてしまう欠点がある。

そこでこの問題を解決するための手法として以下のような手法が提案された。

- ノードの大きさによってスプリングの自然長を変化させる手法 [16, 17]

この手法はノードが大きい場合そのノードにつながっているエッジの自然長を長くするという手法である。大きいノードに連結されているエッジは長く設定されているために、それだけ大きな力が働き、エッジによってつながっているノードと重ならない。また斥力を計算するためのノード間の理想距離もノードの大きさによって長くする。そのため大きいノード同士は斥力が大きくなり、重なりを回避する。この手法はノードが円の場合はノード同士の相対位置によらずノード間の距離が決定できるため有効である。しかし、オブジェクト図で扱うノードは矩形であるため縦に長いノードや横に長いノードが存在する。横に長いノードの場合、他のノードとの相対位置が縦の場合にノード間の距離を適切な長さにすると、ノード間の相対位置が横になった時にノードが重なってしまう。また逆に相対位置が横になった時のノード間の距離を適切な長さに設定すると、ノード間の相対位置が縦になった時に、今度はノード間の距離が長くなりすぎてしまう。このようにノード間の理想距離がその相対位置によって変化してしまうために矩形のノードの重なりを除去する手法としては向いていない (図 4.8)。

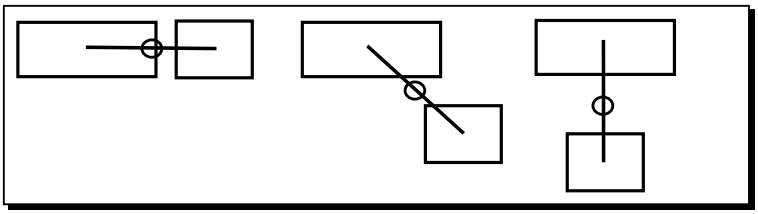

図 4.8: ノード間の中心間の距離と実際のノード間の距離の変化

先の手法ではノード間の距離をノードの中心から中心までの長さとしている。これはマグネティックスプリングモデルにおいてノードは無限小の点として扱われていたためである。このアルゴリズムを拡張したためにノード間の距離はもともとそのノードを示す中心の点を基準に決められていた。このことから、先の手法ではノードの相対位置によってノード間の距離が変わってしまう問題があったのである。

この問題を解決するために本手法では、スプリングの長さとノード間の距離をノードの端から端までの実際の距離に変更することで、任意の大きさの矩形のノードの存在するオブジェクト図に対応した (図 4.9)。この手法ではノードの形、大きさ、相対位置によらず常にノード間の理想距離が決定できる。よって先の手法ではノードの理想距離をノードの大きさにより変える必要があったが、本手法ではノードの理想距離を一定にし、ノードの相対位置によらず常にノード間の距離を一定の理想距離に近づける事が可能になった。

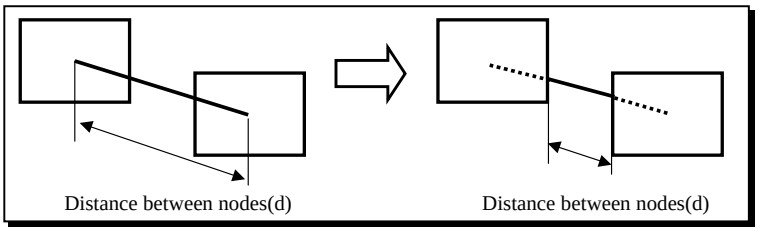

図 4.9: ノード間の距離の定義

しかしこの場合問題となるのがノードが重なっていた時のスプリングの長さやノード間の距離である。今までの手法ではノードが点として表現されていたために、実際にノードが重なっていても中心間の距離をはかることでノードの距離を計算することが出来た。しかし、本手法ではノードが重なっていた場合にはノード間の距離を算出することが出来ない。そこで、重なっていた場合のノード間の距離を 0 とすることが考えられる。しかしこの場合、ノードにかかる力が無限大になってしまう。なぜなら第 3.3.1 節の f_s や f_r の定義式においてノード間の距離である d を 0 にすることによっ

て f_s、 f_r が無限大になってしまうからである。これを回避するためにノード間の最低距離を定義した。このノード間の最低距離は、ノードがその距離以上近づいた場合、それ以上はノード間の距離を短くしない最低の距離である。この距離は十分に短いために、ノード間に働く力はノードの重なりを除去するには十分な大きさになる。また力が大きすぎてノードが過度に移動するのを防ぐことが可能である。

4.4 インタラクティブ性を重視したレイアウト手法

CASE ツールにおいてレイアウト機能を使用するのはオブジェクト図を描画し終えた後だけではない。むしろオブジェクト図を描画している途中の段階において、描画のサポートとなるレイアウト機能は重要である。本アルゴリズムはオブジェクト図を描画する際にその描画のサポートに主眼をおいている。このような描画の途中の段階で行うレイアウトにおいてはインタラクティブ性は非常に重要である。なぜなら描画の途中の段階で用いるということは、レイアウトの後、描画の作業をスムーズに行えることが必要だからである。インタラクティブ性のあるレイアウトとはレイアウトを行った後、ユーザが次の作業にスムーズに移行できるレイアウトである。ユーザが次の作業にスムーズに移行するためにはレイアウトの結果がユーザのメンタルマップを破壊しないことが重要である。メンタルマップを破壊するとは、レイアウトによりノードの位置関係が急激に変わってしまい、どのノードがどの位置にあるのかを認識できなくなってしまうことである。よってユーザのメンタルマップを破壊しないようなレイアウトを行うことがインタラクティブ性のあるレイアウトを行うことになる。

このようなインタラクティブ性を重視したレイアウトを実現するために、我々は 4 つ目の力を新たにマグネティックスプリングモデルに導入する。この力はレイアウトを行う前の位置を保存しようとする力である。本アルゴリズムではこの力を慣性力と呼び f_i とする。以下に定義式を示す。

$$
\begin{aligned}
&\text{if } (d_k > s) \\
&\quad \text{then} \\
&\qquad f_i = c_i(d_k - s)^2; \\
&\quad \text{else} \\
&\qquad f_i = 0;
\end{aligned}
$$

式における c_i は他の力とのバランスをとるための定数である。 d_k はノード k のレイアウト前の位置から現在の位置までの距離である。そして s はノードの対角線の半分の長さである。

f_i の定義の中で s を用いる理由は大きいノードほど慣性力が影響しはじめるのを遅くするためである。なぜなら動いた距離が同じであっても、大きいノードは前の位置と重なる部分が大きく見ための位置の変化が少ないからである (図 4.10)。

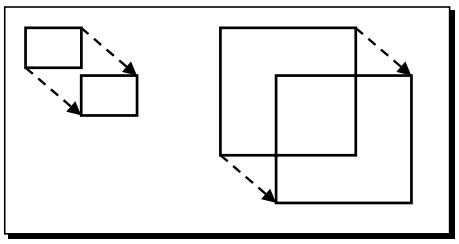

図 4.10: ノードの大きさによる見た目の移動量の変化

レイアウト前の位置を保存する慣性力とは、マグネティックスプリングモデルの 3 つの力がレイアウトを行ないノードを大きく移動させようとすると、それを抑制しノードの動きを小さく押える力である。よってこの力は他の 3 つの力とは性質が少し異なる。以下のアルゴリズムに示すように、他の 3 つの力が現在の位置のみを用いてレイアウトを行なうのに対し、慣性力はそれぞれのノードの力を計算し移動するイタレーションの前に全ノードの位置を保存し、その位置を利用する。その位置とイタレーションの各時点におけるノードの位置の差が大きい程慣性力は大きくなり、ノードの移動を抑制するのである。

全てのノードの位置を保存する;
(定められた回数のループ)
{
　for v=1 to (ノード数)
　{
　　for w=1 to (ノード数)
　　{
　　　if (ノード v と w がエッジ e で隣接)
　　　　then
　　　　　エッジ e に関して v が受ける f_s を計算する;
　　　　　エッジ e に関して v が受ける f_m を計算する;
　　　　　$f_v := f_v + f_s + f_m$;
　　　　else
　　　　　ノード w に対して v が受ける f_r を計算する;
　　　　　$f_v := f_v + f_r$;
　　　ノード v に対して最初に保存した位置と現在の位置を用いて f_i を計算する;
　　　$f_v := f_v + f_i$
　　}
　}
　ノード v を δf_v 移動する;
}

メンタルマップを破壊する一番の要因はノードが大きく動きノードの相対位置関係が大幅に変わってしまう点である。よってこのように現在の配置を保存するようにレイアウトをすることはメンタルマップを破壊しにくくし、ユーザが次の作業を行なうのに適したレイアウト手法であると考えられる。

ユーザが新しいノードをグラフに描くとレイアウト機能が動作し計算をはじめる。マグネティックスプリングモデルはイタレーションにより少しずつノード間に働く力を緩和するようにノードを移動していく。よってイタレーションの初期の段階ではノードはほとんど動いていないため f_i は微弱である。しかし計算が進むにつれノードが初期の位置から遠ざかると f_i が強く働くようになる (図 4.11)。そしてレイアウト終了後、ユーザはまた新たにオブジェクト図にノードを加える。その次に行なわれるレイアウトでは各ノードの初期配置は前のレイアウト結果の位置にリセットされているため f_i は前のレイアウト後の位置から計算される。

このインタラクティブ性を考慮したレイアウトでは 4 つ目の力を導入したために、3 種類のエッジの方向を考えたオブジェクト図の綺麗さよりも、次の作業を行なうためのインタラクティブ性を重視している。よってこのレイアウトを一回行なうと、ノー

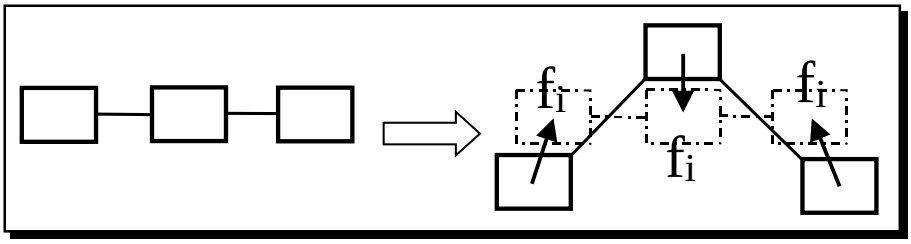

図 4.11: f_i の力のかかり方

ドの重なりを除去し、第 4.2 節で定義したそれぞれのエッジの方向へ少し移動する。移動が小量のためにユーザのメンタルマップの破壊は少ないが、レイアウトも完全にはならない。しかし、描画をはじめてから終了するまでにノードを何度も描画し、繰り返しこのレイアウトを用いることによって緩やかに 3 種類のエッジの方向を考えたレイアウトに収束してく。よって各レイアウトの段階ではインタラクティブ性を考慮し、最終的にはオブジェクト図の特徴を考えた綺麗なレイアウトへと近づけることが可能になるのである。また他の力に対する f_i の割合を変化させることにより、インタラクティブ性とレイアウトの美しさの重要度を調節することが可能である。 f_i の割合を大きくとることによって現在の位置を保存したレイアウトができメンタルマップを破壊せず、描画途中のメンタルマップを保存したレイアウトが可能になる。逆に f_i の割合を小さくすることによってインタラクティブ性よりも美しさ、見やすさを重視したレイアウトも可能になる。

第 5 章

レイアウトシステム

本研究で開発したシステムは本研究室の中島により開発されたシステム [12] を元に開発した。本章では本研究で開発したレイアウトシステムについて、システムの構成、本システムの特徴について述べる。

5.1 システム構成

本システムはクライアント - サーバシステムとして実装した。クライアントとしてオブジェクト図を描画するための OMT Editor(図 5.1)、サーバとしてクライアントからグラフ情報を受け取り、レイアウトを結果の座標情報を返す Layout Server からなる。そのシステム図を図 5.2に示す。

Layout Server は C++ によって、 OMT Editor は Tcl/Tk によって実装されている。両者の間はソケット通信によってデータのやりとりを行っている。 LayoutServer は、ある一定のフォーマットにしたがったデータを受け取るとそのデータからグラフ構造とレイアウトアルゴリズムを読み取り指定されたアルゴリズムを用いたレイアウト結果をクライアントに送信する。 OMT Editor と LayoutServer 間の通信のためのデータ形式は図 5.3に示す通りである。 OMT Editor は設計図記述言語 DSL 形式 [20] によって記述されたテキスト形式のオブジェクト図を読み取りエディタ上に表示することが可能である。またユーザからの編集を終えた後に DSL 形式によってのファイルへの出力が可能である。 OMT Editor からの Layout Server へのレイアウト要求はユーザが任意にレイアウトボタンによって行うことができる。ユーザがレイアウトボタンを押すことによって、その時点でのグラフ構造とユーザが選択したレイアウトアルゴリズムが Layout Server に送信され、その結果は直ちに OMT Editor 上のオブジェクト図に反映される。

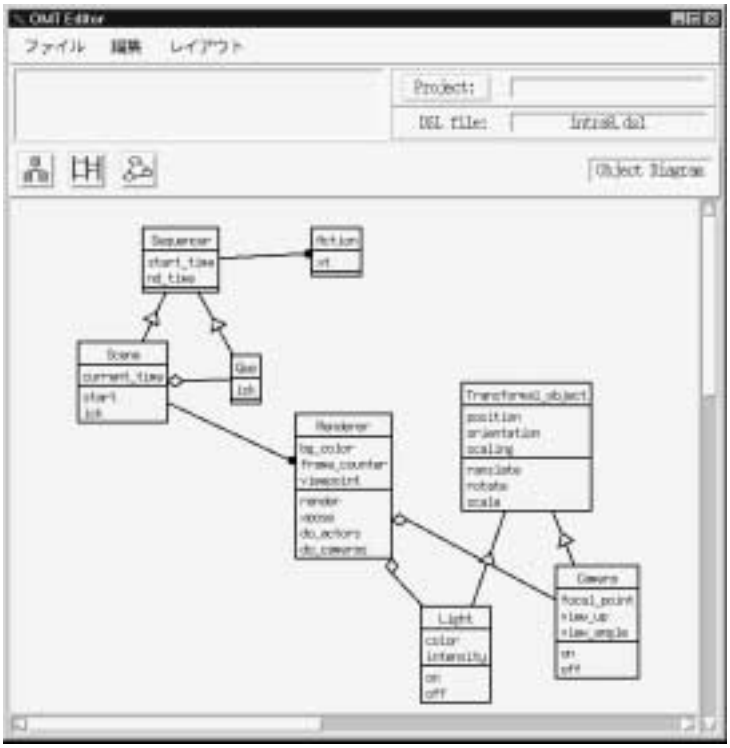

図 5.1: OMT Editor

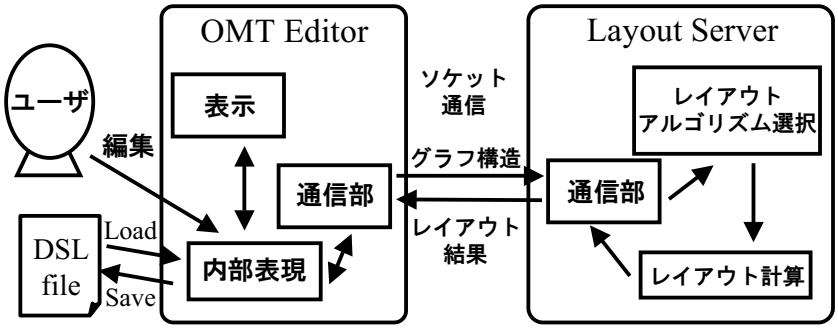

図 5.2: システム構成

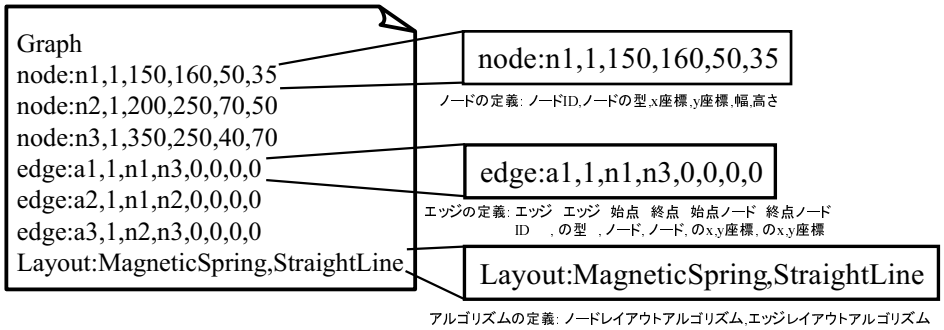

図 5.3: 通信データ形式

5.2 システムの特徴

本システムには以下の 2 点の新たな特徴がある。

- アニメーション機能
- 3D システムへの適用

前者のアニメーション機能は本研究で実装したインタラクティブ性を重視したレイアウトには大変重要な機能である。また後者の 3D システムへの適用は本システムがクライアント - サーバ方式を取っているメリットを活かし、 OMTEditor 以外への Layout Sarver を適用した例である。

5.2.1 アニメーション機能

本研究ではインタラクティブ性を重視したレイアウトではノードの動きを抑え、ユーザのメンタルマップの破壊を抑える手法を提案した。この手法を用いることでユーザのメンタルマップの破壊を抑えることができるが、アニメーションの機能を追加する事で、よりユーザのメンタルマップの破壊を抑えることが可能になる。

OMT Editor でのレイアウトは全て Layout Server との通信によって行われる。そのためアニメーション機能を Layout Server に持たせてしまうと、アニメーションのための OMT Editor 上での描画の回数だけ、 Layout Server との通信を行わなければならず、現実的ではない。そこで本システムではアニメーション機能を OMT Editor 上に実装した。 OMT Editor はレイアウトを行う際に Layout Server に送信するグラフ情報を一時保管する。そして Layout Sarver から送られてきたレイアウト結果と保管しておいたレイアウト前のグラフ情報を用いてアニメーションを実現する。

それぞれのノードに対して、レイアウト後の位置とレイアウト前の位置からその移動距離を計測し、その間を一定のコマ数で分割することによってアニメーションを実現している。レイアウトを、アニメーションを用いて実行する事により各ノードの動きがユーザに分かりやすく表現され、ユーザのメンタルマップの破壊を大きく抑制する事が出来る (図 5.4)。

5.2.2 3D システムへの適用

本システムはクライアント - サーバ形式を用いているのでサーバへ渡すデータ形式をあわせることで他のクライアントシステムへも応用することができる。現在計算機の発達にともない、 3 次元のグラフを扱ったシステムも多数存在している [36, 37, 38]。

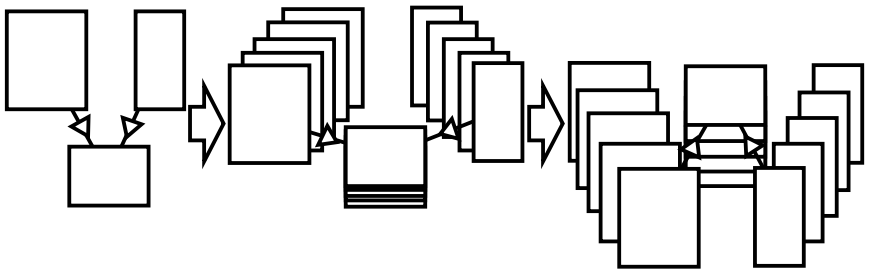

図 5.4: アニメーション

このようなシステムでは扱える次元が増えたために、ノードの配置も 2 次元のシステム以上にユーザへの負担になる。よってこのような 3 次元システムへの自動レイアウトは 2 次元のシステム以上に有用である。よって本研究ではこのようなシステムにも通信のデータフォーマットをあわせるだけで本レイアウトシステムが利用できるように、 Layout Server を 3 次元のグラフに対してもレイアウトが行えるように改良した。Layout Server はもともと 2 次元グラフシステムのためのレイアウトサーバであるため通信用のデータフォーマット、レイアウトアルゴリズムが全て 2 次元のグラフフォーマットに対応していた。本システムでは通信用のデータフォーマットを 3 次元にも拡張し、 2 次元、 3 次元両方のデータフォーマットのレイアウト要求に対応できるように改良した。そのため 2 次元のデータに対しては今まで同様に 2 次元のレイアウト結果を、 3 次元のデータに対しては 3 次元のレイアウト結果を自動的にクライアントに返送することが可能である。 3 次元グラフに対する通信データ形式を図 5.5に示す。

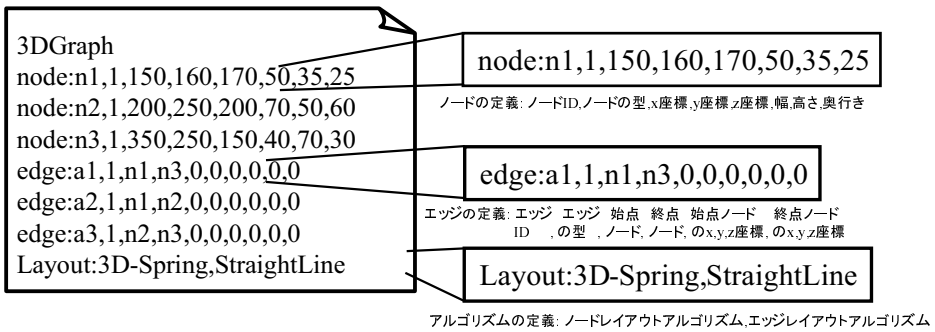

図 5.5: 3 次元グラフの通信データ形式

また 3 次元グラフ用のアルゴリズムとしてスプリングモデル [24] を実装した。スプリングモデルはマグネティックスプリングモデルの基となるアルゴリズムであり、マグネティックスプリングの力のうち、スプリングによる力と非隣接ノード間の斥力の

二つの力でレイアウトを行なう。2 次元でのスプリングモデルでは、X 方向 Y 方向に対してノード間の距離を

$$d = \sqrt{x^2 + y^2}$$

としてノード間に働く力を計算している。そこで 3 次元スプリングモデルではノード間距離は、次元を一つ追加し X 方向 Y 方向 Z 方向に対してノード間距離を

$$d = \sqrt{x^2 + y^2 + z^2}$$

とすることで 3 次元のレイアウトを可能にする。

適用例として 3 次元ビジュアルプログラミングシステムである 3D-PP [36] に本 LayoutServer との通信機能を実装し、レイアウトを行った。このシステムは並列論理型言語を対象としたビジュアルプログラミングシステムである。ユーザは 3 次元アイコンを使用して 3 次元空間において図形を組み合わせることによって実用的なプログラムを記述していく。このシステムの特徴はプログラムの記述を 2 次元ではなく、3 次元の空間上に記述することで、大規模なプログラムにも適用できる点、図形や、図形を結ぶ線の交差や重なりを減少できる点である。実際に本研究のレイアウトシステムを用いて 3D-PP 上でレイアウトした結果を図 5.6に示す。レイアウト前のグラフは床に平面的にノードを配置したものである。グラフにはエッジの交差があり、3 次元を有効に利用できていない。レイアウト後はノードを 3 次元空間上に正四面体に配置されており、エッジの交差は解消されており、3 次元空間を有効に利用している。3D-PP を実際にユーザが使用すると 3 次元上に手動でノードを配置することは 2 次元のグラフを描画する以上に労力のいる作業であり、このように自動的に 3 次元空間上にノードを配置するレイアウトシステムは大変有効である。

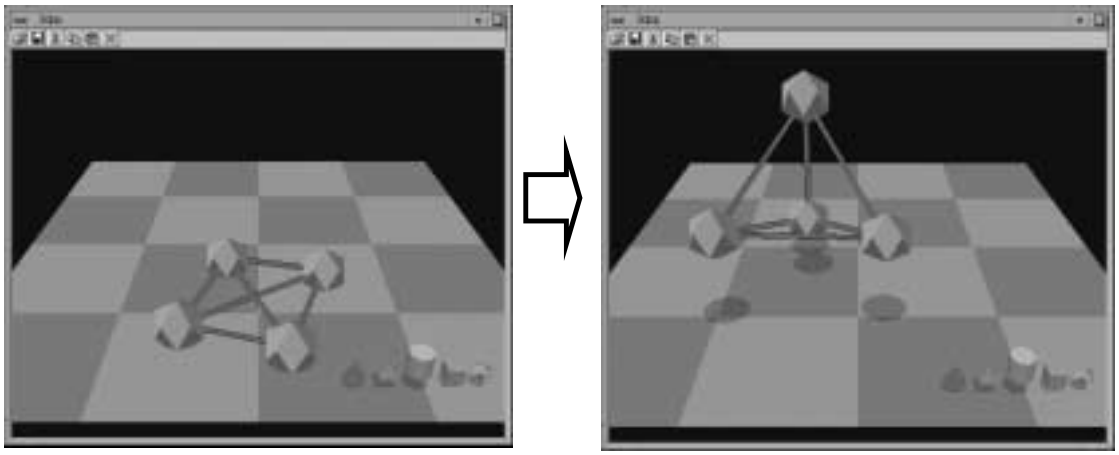

図 5.6: 3D-PP での LayoutServer によるレイアウト

第 6 章

本手法の適用結果

オブジェクト図のレイアウトに対する本手法の適用結果を以下に示す。始めにノードの大きさに対する今までの手法と本手法の適用結果について比較した結果を示す。次にインタラクティブ性を考慮したレイアウトの適用結果を示す。最後に本研究で提案したマグネティックスプリングモデルのサブセットを適用したレイアウト手法を、商用の CASE ツールである Rational Rose[11] の自動レイアウト機能、本研究室の中島により提案された手法 [12, 13, 14, 15] と比較評価した結果を示す。

6.1 ノードの大きさに対する手法の適用結果

ノードの大きさによってスプリングの自然長を変化させる手法と本研究で提案するノードの端から端までをノード間の距離とする手法の適用結果の比較を図 6.1、 6.2に示す。

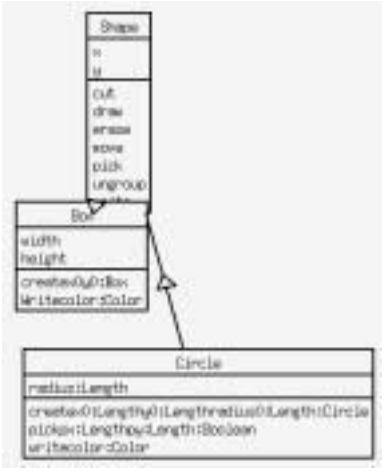

図 6.1: スプリングの自然長を変化させる手法

図 6.1はノードの大きさによってスプリングの自然長を変化させる手法の適用結果である。この手法では縦に長いノードや横に長いノードが存在する場合にはノード間の距離がノードの相対位置によって大きく変化してしまうためにノードが重なってしまう場合がある。よって縦に長い Shape クラスと横に長い Circle クラスの両方にはさまれた Box は両方のクラスと適切なノード間の距離がとれず Shape クラスと重なってしまっている。

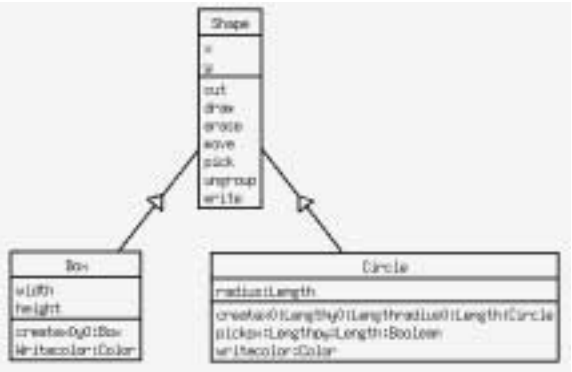

図 6.2: ノードの端から端までをノードの距離とする手法

図 6.2は本研究で提案する、ノードの端から端までをノード間の距離とする手法の適用結果である。本手法ではノード間の距離を決める要素としてノードの大きさを利用していないため Shape クラスや Circle クラスのように長いノードであっても重なること無くみやすくレイアウトが行われている。

6.2 インタラクティブ性を考慮したレイアウト手法の適用結果

図 6.3はインタラクティブ性を重視するための慣性力を適用していないアルゴリズムを用いてオブジェクト図を描画する過程を示した図である。本システムではノードが新たに描画され関係エッジによってオブジェクト図に連結されることで自動的にレイアウトが行われる。最初の図では Light,Camera,Transformal_object の 3 つのクラスが描画されているオブジェクト図をファイルからロードしている。その後 Renderer クラスを描画し、 Light クラス、 Camera クラスと集約エッジによって連結され、レイアウトが行われた結果が 2 つ目の図である。同様にそれぞれの図はノードを一つ描画し、他のノードに連結した後、レイアウトが行われた後のオブジェクト図を示す。

最後の図が完成されたオブジェクト図のレイアウト結果である。関連、継承、集約の 3 種類のエッジがそれぞれ 4.2 節で定義した磁場の方向へほぼそろってレイアウトされており、最終的には見やすいレイアウトになっている。よって慣性力を用いない

マグネティックスプリングモデルを適用することで、完成したオブジェクト図のレイアウトは可能になる。

しかし、最初の図から 2 番目の図へ変化する際に Light,Camera クラスと Transformal_object クラスの相対的な位置関係が全く変わってしまっている。よってその後新たにノードを描画し、他のノードとエッジによって連結する際に移動したノードを把握しなおさなければならない。このようにノードの位置が大きく変わるレイアウトは描画の最中に用いるには不向きである。

次に図 6.4でインタラクティブ性を重視するための慣性力を適用したアルゴリズムを用いてオブジェクト図を描画した過程を示す。

図 6.4も図 6.3と同様に最初に 3 つのクラスが描画されたファイルをロードし、 2 つ目の図からはノードを一つ描画し、他のノードに連結した後、レイアウトが行われた後のオブジェクト図を示す。

最初の図から 2 番目の図への移行において Light,Camera クラスは図 6.3の場合程大きくは移動していない。各図の間に一回ずつインタラクティブ性を重視したレイアウトが行われているためこの 2 つのクラスは徐々に Transformal_object クラスの下に移動して行き 4 番目の図においてほぼ安定した状態になっている。このようにノードの移動がレイアウトを重ねて行く間に徐々に移動することで、ユーザのメンタルマップの破壊を極力抑えることが可能になっている。

6.3 既存のアルゴリズムとの比較評価

本研究で手法において慣性力を導入する前のマグネティックスプリングモデル、中島の手法 [12, 13, 14, 15]、商用の CASE ツールである Rational Rose[11] の自動レイアウト機能を比較評価した。本研究では以下のような二種類の比較実験を行った。

比較実験 1 3 種類のレイアウト手法によるレイアウト結果の比較。

比較実験 2 オブジェクト図を描画する際に自動レイアウト機能を併用した時の、本研究での手法と中島の手法の比較。

両実験とも 7 人の被験者に対する課題として、以下のような 2 種類のオブジェクト図を用意した。

課題 1 酒倉庫問題 [39] のオブジェクト図 (ノード数: 9, エッジ数:12)

課題 2 アニメーションシステム”OSCAR” のオブジェクト図 [1] の一部 (ノード数:14, エッジ数:14)

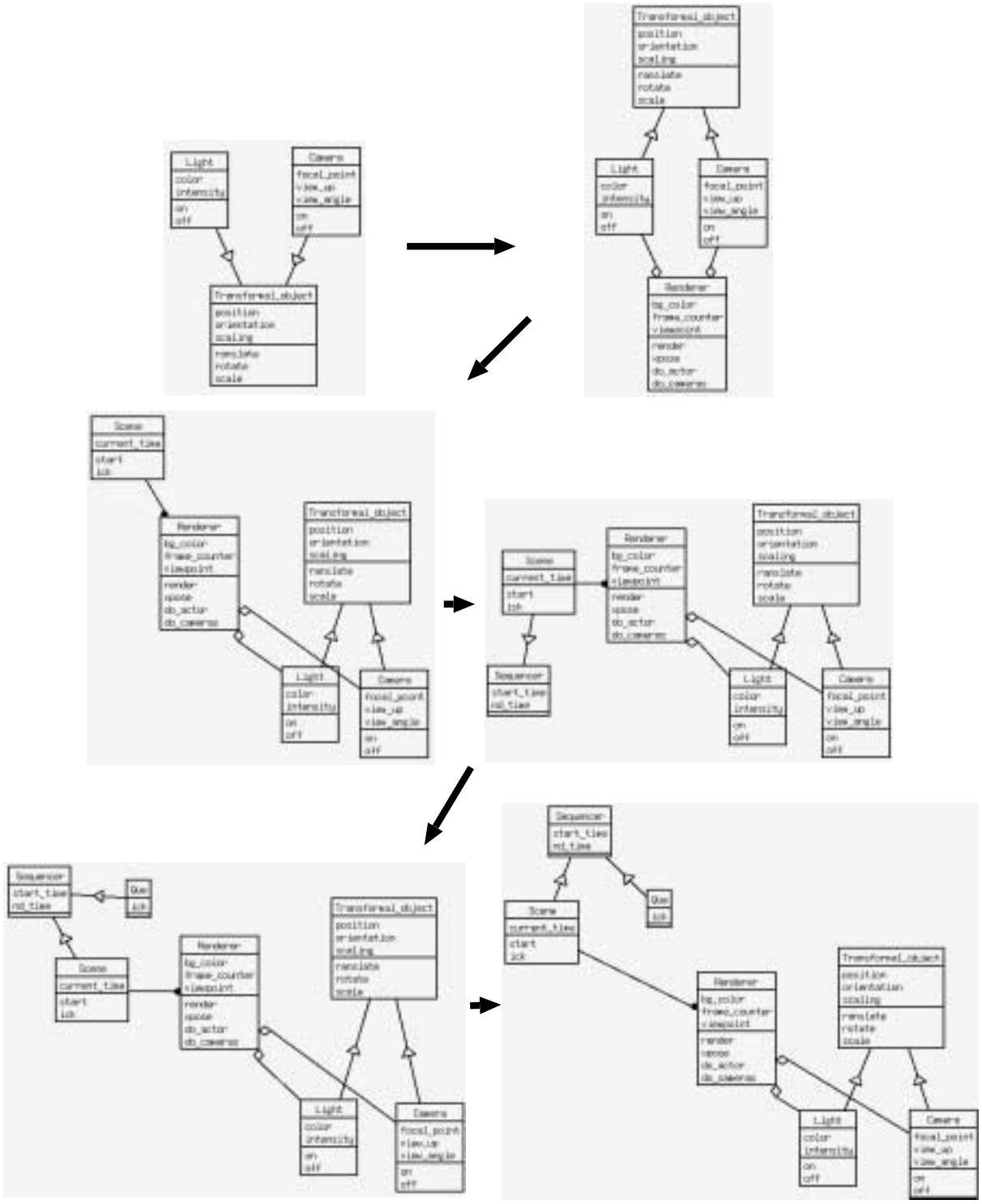

図 6.3: 慣性力のないマグネティックスプリングモデルによるオブジェクト図の描画過程

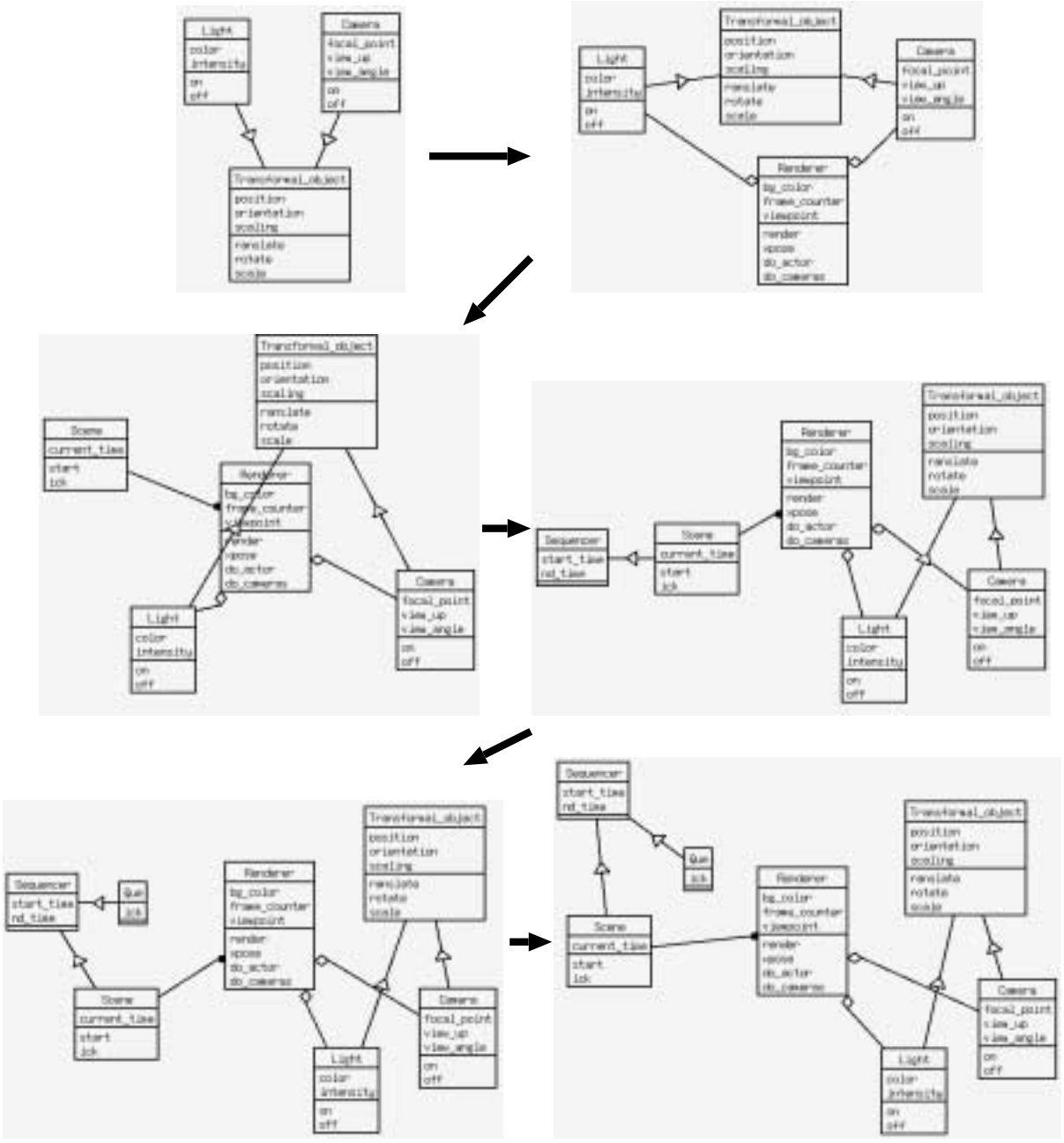

図 6.4: インタラクティブ性を重視したレイアウト手法を用いたオブジェクト図描画過程

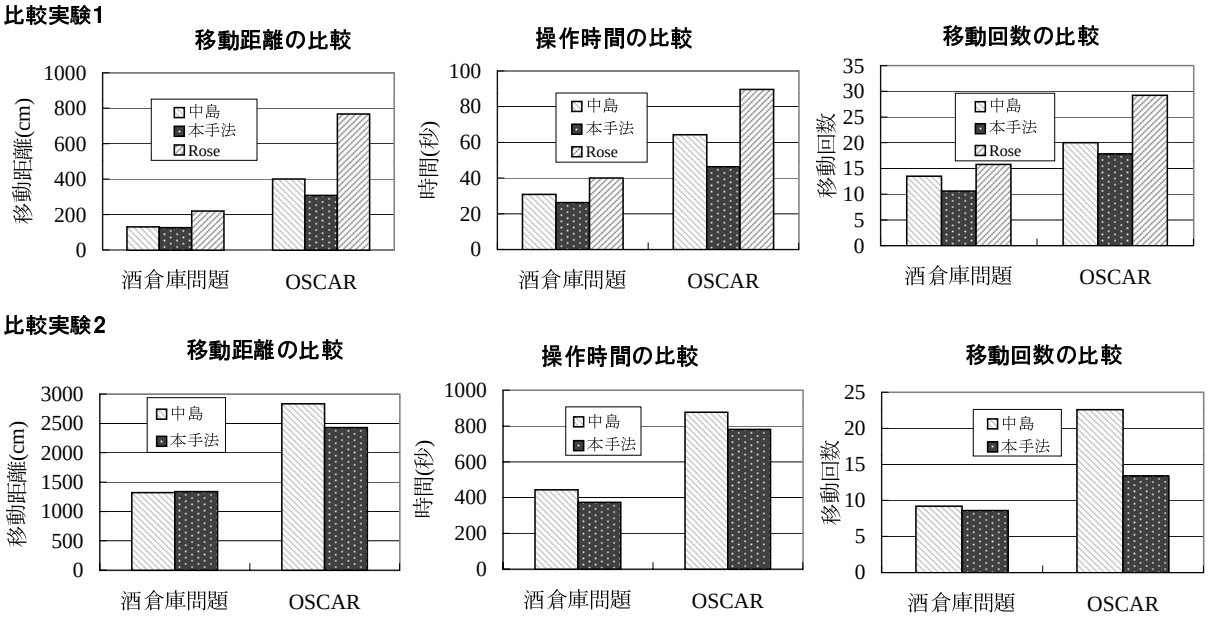

図 6.5: 評価実験結果 (平均値)

比較実験 1 では既に三種類の手法の自動レイアウト結果を与え、被験者が満足のいくレイアウトになるようにオブジェクト図を修正する。比較実験 2 ではテキスト形式のオブジェクト図を見ながら一から描くといった作業をしてもらう。その間レイアウト機能は自由に使ってもらった。両実験とも操作時間、ノードを移動した回数、ノードを移動した距離の三項目についてデータを収集した。その結果の 7 人の平均を表 6.5に示す。

比較実験 1 では Rational Rose にくらべ収集したデータすべてに対して約 1.5 倍から 2 倍の良い結果が得られた。中島の手法に対しては酒倉庫問題についてはほぼ同等の結果が出た。これは酒倉庫問題のオブジェクト図が関連関係のみからなっているためであると思われる。 OSCAR については操作時間がきわだって中島の手法よりも良かった。これはユーザの思考時間が我々の手法の場合の方が短かったためであると思われる。比較実験 2 では酒倉庫問題については比較実験 1 とほぼ同等の結果が出た。OSCAR についてはどの項目でも中島より良い結果が出たが、ノードの移動回数がきわだって少かった。これは中島のアルゴリズムに比べてより多くのノードがユーザの期待する位置に配置されていたためと思われる。

第 7 章

まとめ

グラフ描画アルゴリズムの一つであるマグネティックスプリングモデルを適用し、オブジェクト図の描画アルゴリズムを開発した。このアルゴリズムは

- オブジェクト図の 3 種類の関係に着目したレイアウトが行なえる。
- 任意の大きさに変化するノードに対応し、重なりの無いレイアウトを実現する。
- 新たな力を導入することにより、ユーザとのインタラクティブなレイアウトを実現する。

の 3 点が優れた特徴である。

またこのアルゴリズムを実装したレイアウトシステムとオブジェクト図の描画行なうエディタを製作した。レイアウトシステムでは 3 次元に拡張することで、3 次元のグラフもレイアウトすることを可能にし、3 次元のビジュアルプログラミングシステム 3D-PP へ適用した。エディタではアニメーションを実装することにより、よりユーザとのインタラクティブなレイアウトを実現した。

謝辞

本研究を進めるにあたり，指導教官の西川博昭教授には細かい点まで配慮いただいた。また，田中二郎教授には研究内容について終始親切にしていただいた。この場を借りて深く感謝の意を表します。またオーストラリア、 Southern Queensland 大学の Wei Lai 氏からは、グラフ描画アルゴリズムについての多くの貴重な意見や、プレゼンテーションに関する助言を頂きました。また田中研究室の皆さんからはさまざまな形でサポートして頂きました。ここに深く感謝の意を表します。

参考文献

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorenson(羽生田栄一 監訳): オブジェクト指向方法論 OMT ～モデル化と設計
トッパン (1992)

[2] G. Booch, J. Rumbaugh and I. Jacobson: *Unified Modeling Language Semantics and Notation Guide 1.0*
SanJose CA: Rational Software Corporation (1997)

[3] H. Eriksson and M. Penker: *UML Toolkit*
John Wiley & Sons,Inc (1998)

[4] 本位田真一: オブジェクト指向によるシステム開発の実践と課題
日本ソフトウェア科学会 ISOTAS'96 チュートリアル資料 pp71-78 (1996)

[5] G. Booch(山城明宏 訳): *Booch* 法: オブジェクト指向分析と設計
アジソン・ウェスレイ・パブリッシャーズ・ジャパン (1995)

[6] 本位田真一 山城明宏: オブジェクト指向システム開発
日経 BP 社 (1993)

[7] S. Shlaer and S.J. Mellor(本位田真一 伊藤潔 監訳) : オブジェクト指向システム分析
啓学出版 (1992)

[8] 岡部雅夫 小熊康弘 渡辺香里 羽生田栄一 皆川誠 佐藤英人: オブジェクト指向モデリング手法「MELON」 - ビジネスドメインに特化した手法 -
情報処理学会 OO'96 シンポジウム論文集 pp.1-7 (1996)

[9] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard: *Object-Oriented Software Engineering*
Addison-Wesley Publishing Company (1994)

[10] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremes: *Object-Oriented Development: The Fusion Method*
Upper Saddle River NJ: Prentice-Hall (1994)

[11] http://www.rational.com/products/rose/index.jtmpl

[12] 中島哲: オブジェクト指向方法論に基づくダイヤグラムの自動レイアウト
平成 8 年度筑波大学大学院修士課程理工学研究科修士論文 (1997)

[13] 中島哲 田中二郎: グラフ描画アルゴリズムを用いた OMT 法に基づく設計図の自動生成
情報処理学会オブジェクト指向'96 シンポジウム論文集, pp.103-110 (1996)

[14] S. Nakashima and J. Tanaka: *An Automatic Layout System for OMT-based Object Diagram*
Integrated Design and Process Technology, IDPT-Vol2, pp82-89, (1996),

[15] S. Nakashima and J. Tanaka: *Automatic Layout of Object Diagrams based on Object Oriented Methodology*
Transactions of Information Processing Society of Japan(IPSJ), Vol.39, No.12, pp3282-3293, (1988)

[16] 野口隆佳 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
日本ソフトウェア科学会第 15 回大会論文集, pp.297-300 (1998)

[17] T. Noguchi and J. Tanaka: *New Automatic Layout Method based on Magnetic Spring Model for Object Diagrams of OMT*
International Symposium on Future Software Technology 98 (ISFST98), pp.89-94 (1998)

[18] 野口隆佳 田中二郎: *CASE* ツールにおけるオブジェクト図のレイアウト手法
日本ソフトウェア科学会第 16 回大会論文集, pp.1-4 (1999)

[19] T. Noguchi and J. Tanaka: *Interactive Layout Method for Object Diagrams of OMT*
Proceedings of Asia-Pacific Software Engineering Conference(APSEC '99), pp.110-117 (1999)

[20] 原田実 澤田隆志 藤澤照忠: 構造化オブジェクトモデリング環境 SOMESOMM に基づく OOA/OOD
情報処理学会ソフトウェア工学研究会資料, Vol.94-SE-101, pp1-8 (1994)

[21] 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 (1995)

[22] G. Di Battista, P. Eades, R. Tamassia and I.G. Tollis: *Algorithms for Drawing Graphs: an Annotated Bibliography*
ftp.cs.brown/pub/papers/compgeo/gdbiblio.ps.Z (1994)

[23] P. Eades and K. Sugiyama: *How to draw a directed graph*
Journal on Information Processing, Vol.13(4), pp424-437, (1990)

[24] P. Eades: *A Heuristics for Graph Drawing*
Congressus Numeranitium, Vol.42, pp.149-160 (1984)

[25] 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)

[26] J.Q. Walker: *A node-positioning algorithm for general trees*
Software Practice and Experience 20-7, pp.685-705 (1990)

[27] J. Vauchar: *Pretty Printing of Trees*
Software-Practice and Experience 10-7, pp.553-561 (1980)

[28] C. Wetherell and A. Shannon: *Tidy Drawing of Trees*
IEEE Trans. Software Engineering, SE-5-5, pp.514-520 (1979)

[29] R. Reingold and J. Tilford
Tidier Drawing of Trees
IEEE Trans. on Software Engineering, SE-7-2, pp.223-228 (1981)

[30] K. Sugiyama, S. Tagawa and M. Toda: *Method for Visual Understanding of Hierachical System Structures*
IEEE Trans. on System, Man abd Cybernetics, SMC-11-2, pp.109-125 (1981)

[31] K. Sugiyama: *A Cognitive Approach for Graph Drawing*
Cybernerics and Systems, 18-6, pp.447-488 (1987)

[32] K. Sugiyama and K. Misue
Visualization of Structural Information : Automatic Drawing of Compound Digraphs
IEEE Trans. Systems, Man, and Cybernetics, SMC-21-4, pp.876-892 (1991)

[33] 杉山公造: リニエーションに着目した階層グラフの描画法 -SKETCH 法の改良
日本シミュレーション学会 pp.79-84 (1983)

[34] T. Fruchterman and E. Reingold: *Graphdrawing by force-directed placement*
Software Practice and Experience 12-4.pp45-55 (1994)

[35] 鈴木和彦 鎌田富久 榎本彦衛: 単純無向グラフ自動描画アルゴリズム
コンピュータソフトウェア, Vol.12, No.4, pp45-55 (1995)

[36] 宮城幸司 大芝崇 田中二郎: 3 次元ビジュアル・プログラミング・システム 3D-PP
日本ソフトウェア科学会第 15 回大会論文集, pp125-128 (1998)

[37] H. Shinozawa and Y. Matsushita: *WWW visualization giving meanings to interactive manipulations*
HCI International '97, (1997)

[38] 舘村 純一: DocSpace: 文献空間のインタラクティブ視覚化
インタラクティブシステムとソフトウェア IV: 日本ソフトウェア科学会 WISS '96, pp11-19 (1996)

[39] 山崎利治: 共通問題によるプログラム設計技法解説 (その 2)
情報処理, Vol.25, no.11, pp.1219 (1984)

付録 A

システムのソースコード

本システムのソースコードを付録として添付する.

A.1 Layout Server

Layoutserv.C

```
#include <iostream.h>
#include <sockinet.h>
#include <std.h>
#include <LEDA/graph.h>
#include <LEDA/queue.h>
#include <LEDA/string.h>
#include "parser.h"
#include "GraphManager.h"


main(int argc, char **argv)
{
   int portno;
   char buf[1024];
   string str,nm;
   sockinetbuf si (sockbuf::sock_stream);
   portno = atoi(argv[1]);

   si.bind(si.localhost(),portno);
   si.listen();

   queue<string> ReceiveQueue, SendQueue;
   MessageParser mp,*p;
   GraphManager GM(&mp);

   while(1){
     iosockinet s = si.accept();

  while(!(ReceiveQueue.empty()))ReceiveQueue.pop();
```

```
      while (s >> buf) {
        ReceiveQueue.append(buf);
cout << buf << endl;
        str = buf;
      if(str(0,5) == "Layout"){
          mp.parse(&ReceiveQueue,&GM);
          if(str != "Layout:TellMeAlgorithms"){
            GM.SendResult(s);
  } else {
            GM.SendAlgorithms(s);
  }
          break;
}
      }
   }

   si.close();

   cout << endl;
}
```

GraphManager.C

```
#include <LEDA/list.h>
#include <LEDA/point.h>
#include <LEDA/graph.h>
#include <std.h>
#include "GraphManager.h"
#include "parser.h"
#include "Node.h"
#include "Layouter.h"


void GraphManager::GenerateGraph(list<string> *l1, list<string> *l2){
  int i,w,h,type;
  double x,y;
  string s,id;
  list<int> ilist;
  node n;
  Node N;

  if(l1->empty() != 1){
     id = l1->pop();
     type = atoi(l1->pop());
     x = atoi(l1->pop());
     y = atoi(l1->pop());
     w = atoi(l1->pop());
     h = atoi(l1->pop());
```

```
      x = x+w/2;
      y = y+h/2;

      Graph[Graph.new_node()] = Node(id,type,x,y,w,h);
      N = Graph[Graph.last_node()];
    }

    if(l2->empty() != 1){
      string sn,en;
      list<point> l;

      id = l2->pop();
      type = atoi(l2->pop());
      sn = l2->pop();
      en = l2->pop();

      int i,lgt;
      node n1,n2;

      lgt = l2->length();
      for(i=0;i<lgt;i=i+2){
        x = atoi(l2->pop());
        y = atoi(l2->pop());
        l.append(point(x,y));
      }

      forall_nodes(n,Graph){
        s = Graph[n].GetNodeId();
        if(sn == s)n1=n;
        if(en == s)n2=n;
      }
      Graph[Graph.new_edge(n1,n2)] = Edge(id,type,l);

      //DisplayGraphState();

    }
  }

  void GraphManager::DisplayGraphState(){
    node n;
    edge e;
    Node N;
    Edge E;
    cout << "*******" << endl;
    cout << "NodeNum: " << Graph.number_of_nodes() << endl;
    forall_nodes(n,Graph){
      N = Graph[n];
      cout << "=============" << endl;
      cout << "NodeId: " << N.GetNodeId() << endl;
      cout << "NodeType: " << N.GetNodeType() << endl;
```

```
    cout << "X: " << N.xcoord() << endl;
    cout << "Y: " << N.ycoord() << endl;
    cout << "W: " << N.GetWidth() << endl;
    cout << "H: " << N.GetHeight() << endl;
  }

  cout << "*******" << endl;
  cout << "EdgeNum: " << Graph.number_of_edges() << endl;

  forall_edges(e,Graph){
    E = Graph[e];
    cout << "********** "<< endl;
    cout << "EdgeId: " << E.GetEdgeId() << endl;
    N = Graph[Graph.source(e)];
    cout << "StartNode: " << N.GetNodeId() << endl;
    N = Graph[Graph.target(e)];
    cout << "EndNode: " << N.GetNodeId() << endl;
    cout << "EdgeType: " << E.GetEdgeType() << endl;
    cout << "EdgeLine: " << E.GetEdgeLine() << endl;
  }
}

void GraphManager::SendResult(iosockinet s){

  node n;
  Node N;
  string str;
  int x,y,w,h;

  s << "Graph" << endl;

  str = "node:";
  forall_nodes(n,Graph){
    N = Graph[n];
    x = (int)(N.xcoord() - N.GetWidth()/2);
    y = (int)(N.ycoord() - N.GetHeight()/2);
    w = (int)N.GetWidth();
    h = (int)N.GetHeight();

    s << str << N.GetNodeId() << "," << N.GetNodeType() << ","
 << x << "," << y << "," << w
        << "," << h << endl;
  }

  str = "edge:";
  int i,l;
  edge e;
  Edge E;
  Node N1,N2;
  point p;
```

```
  list<point> lt;

  forall_edges(e,Graph){
    E = Graph[e];
    N1 = Graph[Graph.source(e)];
    N2 = Graph[Graph.target(e)];
    s << str << E.GetEdgeId() << "," << E.GetEdgeType() << ","
 << N1.GetNodeId() << "," << N2.GetNodeId();
    lt = E.GetEdgeLine();

    l = lt.length();

    for(i=0;i<l;i++){
      p = lt.pop();
      s << "," << (int)p.xcoord() << "," << (int)p.ycoord();
    }
      s << endl;
  }

  s << "Layout" << endl;
}

void GraphManager::GraphLayout(string algm){
  Layouter *layouter;

  if(algm == "Layout:Spring,StraightLine"){
    layouter = new Layouter(new SpringNodePlacer, new StraightLine, &Graph);
  }

  if(algm == "Layout:MagneticSpring,StraightLine"){
    layouter = new Layouter(new MagneticNodePlacer, new StraightLine, &Graph);
  }

  if(algm == "Layout:MNLmentalmap,StraightLine"){
    layouter = new Layouter(new MNLmentalmap, new StraightLine, &Graph);
  }
    return;
  }

  layouter->layout();
  delete layouter;

}
```

NodePlacer.C

```
#include "NodePlacer.h"
#include <math.h>
#include <LEDA/string.h>
#include <LEDA/polygon.h>
```

```
#include <LEDA/list.h>
#include <LEDA/node_matrix.h>
#include <LEDA/node_set.h>
#include "TreeLayouter.h"

#define ITERATION 100
#define NODEMAX 100
#define EDGEMAX 100
#define DC1 20.0
#define DC2 70.0
#define DC3 100.0
#define DC4 0.2
#define CanvasMinx 150
#define CanvasMiny 150
#define Space 20

node get_node_fromid(GRAPH<Node,Edge>& g, string id);

double NodePlacer::CheckOverlap(GRAPH<Node,Edge>& Graph,
 node v, node w){

 double vx1,vy1,vx2,vy2,wx1,wy1,wx2,wy2,dx,dy,l;
 list<point> pl1,pl2;

 vx1 = Graph[v].xcoord() - Graph[v].GetWidth()/2 - Space;
 vy1 = Graph[v].ycoord() - Graph[v].GetHeight()/2 - Space;
 vx2 = Graph[v].xcoord() + Graph[v].GetWidth()/2 + Space;
 vy2 = Graph[v].ycoord() + Graph[v].GetHeight()/2 + Space;

 pl1.append(point(vx1,vy1));
 pl1.append(point(vx2,vy1));
 pl1.append(point(vx2,vy2));
 pl1.append(point(vx1,vy2));

 polygon pg(pl1);

 wx1 = Graph[w].xcoord() - Graph[w].GetWidth()/2 - Space ;
 wy1 = Graph[w].ycoord() - Graph[w].GetHeight()/2 - Space;
 wx2 = Graph[w].xcoord() + Graph[w].GetWidth()/2 + Space;
 wy2 = Graph[w].ycoord() + Graph[w].GetHeight()/2 + Space ;

 pl2.append(point(wx1,wy1));
 pl2.append(point(wx2,wy1));
 pl2.append(point(wx2,wy2));
 pl2.append(point(wx1,wy2));

 polygon pg2(pl2);

 double r=0,olp1,olp2,olp3,olp4;
 point t,*co;
```

```
 if(pg.inside(point(wx1,wy1))==1){
   t = point(vx2,vy2);
   r = Graph[v].distance(t);
   return r;
 }
 if(pg.inside(point(wx2,wy1))==1){
   t = point(vx1,vy2);
   r = Graph[v].distance(t);
   return r;
 }
 if(pg.inside(point(wx2,wy2))==1){
   t = point(vx1,vy1);
   r = Graph[v].distance(t);
   return r;
 }
 if(pg.inside(point(wx1,wy2))==1){
   t = point(vx2,vy1);
   r = Graph[v].distance(t);
   return r;
 }
 if(pg2.inside(point(vx1,vy1))==1){
   t = point(wx2,wy2);
   r = Graph[w].distance(t);
   return r;
 }
 if(pg2.inside(point(vx2,vy1))==1){
   t = point(wx1,wy2);
   r = Graph[w].distance(t);
   return r;
 }
 if(pg2.inside(point(vx2,vy2))==1){
   t = point(wx1,wy1);
   r = Graph[w].distance(t);
   return r;
 }
 if(pg2.inside(point(vx1,vy2))==1){
   t = point(wx2,wy1);
   r = Graph[w].distance(t);
return r;
 }
 if(r!=0){
   return r;
  }

  return 0;
}

void NodePlacer::ShiftGraph(GRAPH<Node,Edge>& Graph){
  double minx,miny;
```

```
  node n1;

  minx = Graph[Graph.first_node()].xcoord() - Graph[Graph.first_node()].GetWidth()/2;
  miny = Graph[Graph.first_node()].ycoord() - Graph[Graph.first_node()].GetWidth()/2;

  forall_nodes(n1,Graph){
    if(minx > Graph[n1].xcoord() - Graph[n1].GetWidth()/2)
       minx = Graph[n1].xcoord() - Graph[n1].GetWidth()/2;
    if(miny > Graph[n1].ycoord() - Graph[n1].GetHeight()/2)
       miny = Graph[n1].ycoord() - Graph[n1].GetHeight()/2;
  }
  double x, y;
  if(minx != CanvasMinx || miny != CanvasMiny){
    forall_nodes(n1,Graph){
      x = Graph[n1].xcoord();
      y = Graph[n1].ycoord();
      Graph[n1] = Node(Graph[n1].GetNodeId(), Graph[n1].GetNodeType(),
      x-(minx - CanvasMinx), y-(miny - CanvasMiny),
                        Graph[n1].GetWidth(), Graph[n1].GetHeight());
    }
  }
}

point NodePlacer::GetCorner(GRAPH<Node,Edge>& Graph, int p){
  double minx,miny,maxx,maxy;
  node n;

  minx = Graph[Graph.first_node()].xcoord();
  miny = Graph[Graph.first_node()].ycoord();
  maxx = minx;
  maxy = miny;

  forall_nodes(n,Graph){
    if(minx > Graph[n].xcoord())minx = Graph[n].xcoord();
    if(miny > Graph[n].ycoord())miny = Graph[n].ycoord();
    if(maxx < Graph[n].xcoord())maxx = Graph[n].xcoord();
    if(maxy < Graph[n].ycoord())maxy = Graph[n].ycoord();
  }
  switch(p){
  case 0: return point(minx,miny);
  case 1: return point(maxx,miny);
  case 2: return point(maxx,maxy);
  case 3: return point(minx,maxy);
  }
}

int NodePlacer::EliminateOverlap(GRAPH<Node,Edge>& Graph, int TreeMove){
  node nd,n1,n2;
  edge e;
  int N=0;
```

```
point p1,p2;
double x,y,ovr,l,deltaX,deltaY,d1,d2,S,W=1000,H=1000;
list<node> NodeList,OverlapList;
node_set NS(Graph);

forall_nodes(nd,Graph){
  forall_inout_edges(e,nd){
    if(Graph[e].GetEdgeType() == 3 && TreeMove == 0){
      NS.insert(nd);

    }
  }
}
TreeLayouter *TL;

forall_nodes(nd,Graph){
  NodeList.append(nd);
  if((l = Graph[nd].GetWidth()) < W)W=l;
  if((l = Graph[nd].GetHeight()) < H)H=l;
}

NodeList.pop();
W = W/2;
H = H/2;

p1 = GetCorner(Graph,0);
p2 = GetCorner(Graph,2);

x = (p1.xcoord() + p2.xcoord())/2;
y = (p1.ycoord() + p2.ycoord())/2;

point cm(x,y);

forall_nodes(n1,Graph){
  forall(n2,NodeList){
    if((ovr = CheckOverlap(Graph,n1,n2)) != 0){
      if(Graph[n1].distance(cm) > Graph[n2].distance(cm))nd = n1;
        else nd =n2;
      S = Graph[nd].distance(cm);
      d1 = (Graph[nd].xcoord() - cm.xcoord())/S;
      d2 = (Graph[nd].ycoord() - cm.ycoord())/S;

      if(W < ovr)deltaX = d1*W;
        else deltaX = d1*ovr;
      if(H < ovr)deltaY = d2*H;
        else deltaY = d2*ovr;

    x = Graph[nd].xcoord();
    y = Graph[nd].ycoord();
```

```
      //   if(!(NS.member(nd))){
  // if(nd == n1)nd = n2; else nd = n1;

        Graph[nd] = Node(Graph[nd].GetNodeId(), Graph[nd].GetNodeType(),
         x+deltaX, y+deltaY, Graph[nd].GetWidth(), Graph[nd].GetHeight());

         OverlapList.append(nd);
 // }
      }
    }
    if(NodeList.empty() != -1){
      NodeList.pop();
    } else break;
  }

  NodeList.clear();
  forall_nodes(nd,Graph){
    NodeList.append(nd);
  }

  NodeList.pop();

  forall(n1,OverlapList){
    forall_nodes(n2,Graph){
      if(n1 != n2){
        if((ovr = CheckOverlap(Graph,n1,n2)) != 0)N++;
      }
    }
  }

  return N;
}

node get_node_fromid(GRAPH<Node,Edge>& g, string id){
  node n;
  forall_nodes(n,g){
    if(id == g[n].GetNodeId())return n;
  }
  return nil;
}
```

MNLayouter.C

```
#include "NodePlacer.h"
#include <math.h>
#include <LEDA/node_matrix.h>

#define ITERATION 1000
#define PI M_PI      // π
#define DC1 30.0     // スプリング定数
```

```
#define DC2 100.0    // スプリングの長さ
#define DC3 20000.0  // 非隣接頂点間定数 (使ってない)
#define DC4 0.05     // 微小移動量定数
#define DC5 0.01     // 関連エッジの磁力定数
#define DC6 0.01     // 集約エッジの磁力定数
#define DC7 0.01     // 継承エッジの磁力定数
#define DC8 10.0     // 磁場の強さ
#define DC9 1        // 辺の長さの回転力への影響
#define DC10 1       // 角度の回転力への影響
#define DC11 0       // 関連エッジの角度
#define DC12 PI/4.0  // 集約エッジの角度
#define DC13 PI/2.0  // 継承エッジの角度
#define DC14 0.1     //InertailForce 係数

extern node get_node_fromid(GRAPH<Node,Edge>& g, string id);

void MagneticNodePlacer2::NodeLayout(GRAPH<Node,Edge>& Graph){

  node n1,n2, n3;
  edge e;
  int i;
  double x,y,w,h,d1,d2;
  double nx=0, ny=0;

  node_array<double> forcex(Graph),forcey(Graph);

  node_array<double> C4(Graph);
  node_array<double> C5(Graph);
  node_array<double> CX(Graph);
  node_array<double> CY(Graph);
  node_matrix<double> C1(Graph);
  node_matrix<double> C2(Graph);
  node_matrix<double> C3(Graph);

  // 定数の決定
  forall_nodes(n1,Graph){
    forall_nodes(n2,Graph){
      d1 = sqrt(pow((Graph[n1].GetWidth()/2),2)+pow((Graph[n1].GetHeight()/2),2));
      d2 = sqrt(pow((Graph[n2].GetWidth()/2),2)+pow((Graph[n2].GetHeight()/2),2));

      // スプリング定数
      C1(n1,n2) = DC1;

      // スプリングの自然長
      C2(n1,n2) = DC2;

      // 非隣接頂点間定数
      C3(n1,n2) = DC3;
    }
    // 微小移動量定数
```

```
    C4[n1] = DC4;
    C5[n1] = DC14;
    CX[n1] = Graph[n1].xcoord();
    CY[n1] = Graph[n1].ycoord();
  }

  for(i=0;i<ITERATION;i++){
    forall_nodes(n1,Graph){
      forcex[n1] = 0;
      forcey[n1] = 0;

      forall_nodes(n2,Graph){
int flag = 0;

if (n1 !=n2){
  forall_inout_edges(e,n1){
    if (Graph.opposite(n1,e) == n2 && Graph.opposite(n2,e) == n1){

      //n1 と n2 が辺 e で隣接していた時スプリングによる力を計算
      forcex[n1] = forcex[n1] + AttractiveForce(Graph,n1,n2,0,C1[n1][n2],C2[n1][n2]);
      forcey[n1] = forcey[n1] + AttractiveForce(Graph,n1,n2,1,C1[n1][n2],C2[n1][n2]);

      //n1 と n2 が関連関係の時磁力を計算
      if (Graph[e].GetEdgeType() ==1){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,0,1,DC5,DC8,DC9,DC10,DC11);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,0,2,DC5,DC8,DC9,DC10,DC11);
      }

      //n1 が集約関係の親の時磁力を計算
      if (Graph[e].GetEdgeType() ==2 && n1 == Graph.source(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,1,1,DC6,DC8,DC9,DC10,DC12);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,1,2,DC6,DC8,DC9,DC10,DC12);
      }
      //n1 が集約関係の子の時磁力を計算
      if (Graph[e].GetEdgeType() ==2 && n1 == Graph.target(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,2,1,DC6,DC8,DC9,DC10,DC12);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,2,2,DC6,DC8,DC9,DC10,DC12);
      }

      //n1 が継承関係の親の時磁力を計算
      if (Graph[e].GetEdgeType() ==3 && n1 == Graph.source(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,1,1,DC7,DC8,DC9,DC10,DC13);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,1,2,DC7,DC8,DC9,DC10,DC13);
      }
      //n1 が継承関係の子の時磁力を計算
      if (Graph[e].GetEdgeType() ==3 && n1 == Graph.target(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,2,1,DC7,DC8,DC9,DC10,DC13);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,2,2,DC7,DC8,DC9,DC10,DC13);
      }
```

```
      flag = 1;
    }
  }
  // 隣接していない時斥力を計算
  if (flag == 0){
    forcex[n1] = forcex[n1] + RepulsiveForce(Graph,n1,n2,0,C3[n1][n2]);
    forcey[n1] = forcey[n1] + RepulsiveForce(Graph,n1,n2,1,C3[n1][n2]);
  }
}
      }
    }

    // ノード n1 を移動する
    forall_nodes(n1,Graph){
      x = Graph[n1].xcoord();
      y = Graph[n1].ycoord();
      x = x + C4[n1]*forcex[n1];
      y = y + C4[n1]*forcey[n1];
       Graph[n1] = Node(Graph[n1].GetNodeId(), Graph[n1].GetNodeType(),
                        x, y, Graph[n1].GetWidth(), Graph[n1].GetHeight());
    }
  }

  cout << "┌──── 調整前の座標と大きさ ────┐ " << endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
 << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight()<< ")" << endl;
  }
  cout << "└──────────────────────────────┘ "<< endl;
  // 全体の位置の調整
  ShiftGraph(Graph);

  cout << "┌──── 調整後の座標と大きさ ────┐ "<< endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight()<< ")" << endl;
  }
  cout << "└──────────────────────────────┘ "<< endl;


  cout << "magneticspring-end" << endl;
}
```

```
double MagneticNodePlacer2::AttractiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c, double d0){

  double d,dc,f,dx,dy;
  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();

  d = GetDistance(Graph, v, w);

  dc = d/d0;

  f = -c*log(dc);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

double MagneticNodePlacer2::RepulsiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c){

  double d,dc,f,fx,fy,dx,dy;

  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();

  d = GetDistance(Graph, v, w);

  f = c/(d*d);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

int intersect2(GRAPH<Node,Edge>& Graph, node v, node w){

  if ((Graph[v].xcoord()-Graph[v].GetWidth()/2) <
```

```
                    (Graph[w].xcoord()+Graph[w].GetWidth()/2)&&
      (Graph[w].xcoord()-Graph[w].GetWidth()/2) <
                    (Graph[v].xcoord()+Graph[v].GetWidth()/2)&&
      (Graph[v].ycoord()-Graph[v].GetHeight()/2)<
                    (Graph[w].ycoord()+Graph[w].GetHeight()/2)&&
      (Graph[w].ycoord()-Graph[w].GetHeight()/2)<
                    (Graph[v].ycoord()+Graph[v].GetHeight()/2))
    return 1;
  else
    return 0;
}

double MagneticNodePlacer2::GetDistance(GRAPH<Node,Edge>& Graph, node v, node w){

  double dLv, dRv, dLw, dRw, a, x1, x2, y1, y2;
  double vx, vy, wx, wy;

  // 重なっていた場合
  if (intersect2(Graph, v, w)){
    return 10;
  }

  // ノードに重なりがなかった場合
  else{
    x1 = Graph[v].xcoord();
    y1 = Graph[v].ycoord();
    x2 = Graph[w].xcoord();
    y2 = Graph[w].ycoord();

    // 二つのノードが縦に並んでいた時
    if (x1 == x2)
      if (y1 > y2){
return (Graph[v].ycoord()-Graph[v].GetHeight()/2)-
                    (Graph[w].ycoord()+Graph[w].GetHeight()/2);
      }
      else{
return (Graph[w].ycoord()-Graph[w].GetHeight()/2)-
                    (Graph[v].ycoord()+Graph[v].GetHeight()/2);
      }
    // 二つのノードが横に並んでいた時
    if (y1 == y2)
      if (x1 > x2){
return (Graph[v].xcoord()-Graph[v].GetWidth()/2)-
                    (Graph[w].xcoord()+Graph[w].GetWidth()/2);
      }
      else{
return (Graph[w].xcoord()-Graph[w].GetWidth()/2)-
                    (Graph[v].xcoord()+Graph[v].GetWidth()/2);
      }
    // ノードが重なってもいなく真横、真縦にも並んでいない場合
```

```
    a = (y1-y2)/(x1-x2);
    dRv = -Graph[v].GetHeight()/Graph[v].GetWidth();
    dLv = Graph[v].GetHeight()/Graph[v].GetWidth();

    dRw = -Graph[w].GetHeight()/Graph[w].GetWidth();
    dLw = Graph[w].GetHeight()/Graph[w].GetWidth();

    //cout << "dRv:" << dRv << ", dLv:" << dLv << " ,a:" <<a << endl;

    //v は第１象現の上
    if (x1<x2 && y1>y2 && dRv>a){
      vy = Graph[v].ycoord()-Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第３象現の上
      if (dRw<a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第３象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第１象現の下
    else if (x1<x2 && y1>y2 && dRv<a){
      vx = Graph[v].xcoord()+Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第３象現の上
      if (dRw<a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第３象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第２象現の上
    else if (x1<x2 && y2>y1 && dLv>a){
      vx = Graph[v].xcoord()+Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第４象現の下
      if (dLw>a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第４象現の上
      else{
```

```
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第２象現の下
    else if (x1<x2 && y2>y1 && dLv<a){
      vy = Graph[v].ycoord()+Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第４象現の下
      if (dLw>a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第４象現の上
      else{
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第３象現の下
    else if (x2<x1 && y2>y1 && dRv>a){
      vy = Graph[v].ycoord()+Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第１象現の上
      if (dRw>a){
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
      //w は第１象現の下
      else{
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
    }
    //v は第３象現の上
    else if (x2<x1 && y2>y1 && dRv<a){
      vx = Graph[v].xcoord()-Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第１象現の上
      if (dRw>a){
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
      //w は第１象現の下
      else{
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
    }
    //v は第４象現の下
```

```
    else if (x2<x1 && y1>y2 && dLv>a){
      vx = Graph[v].xcoord()-Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第２象現の上
      if (dLw>a){
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第２象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第４象現の上
    else if (x2<x1 && y1>y2 && dLv<a){
      vy = Graph[v].ycoord()-Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第２象現の上
      if (dLw>a){
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第２象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    return sqrt((pow((vx-wx),2))+(pow((vy-wy),2)));
  }
}

double MagneticNodePlacer2::MagneticForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axisST, int axisXY, double c,
                double b, int alpha, int beta, double K){

  double d,dx,dy,dk,t,f,k;
  d = Graph[v].distance(Graph[w]);
  dx = (Graph[v].xcoord() - Graph[w].xcoord());
  dy = (Graph[v].ycoord() - Graph[w].ycoord());
  dk = atan2(dy,dx);

  //source のノードについて求める時は磁界の向きを逆転させる
  if (axisST == 1){
    if (K > 0)
      k = K - PI;
    else
      k = PI + K;
  }
```

```
  //target の時はそのまま
  else if (axisST == 2)
    k = K;

  // 無向磁針の時は角度の小さい方を磁界の向きにする
  else if (axisST ==0){
    if (fabs(K-dk) > 3.0/2.0*PI || PI/2.0 >= fabs(K-dk))
      k = K;
    else if (3.0/2.0*PI >= fabs(K-dk) || fabs(K-dk) > PI/2.0)
      if (K > 0)
k = K - PI;
      else
k = PI + K;
  }

  // 磁界と自分のなす角を求める
  if (fabs(k - dk) > PI)
    t = fabs(k - dk - 2.0*PI);
  else
    t = fabs(k - dk);

  // なす角が 0, または距離が 0 の時は磁力も 0
  if (d == 0 ||t == 0)
    return 0;

  f = c*b*pow(d,alpha)*pow(t,beta);

  if (k < 0){
    if ((dx < 0 && dy < 0) || (dx >=0 && dy >=0)){
      if (k < -PI/2.0 && ((PI/2.0 >= dk && dk >PI+k) || (k < dk && dk < -PI/2.0))){
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
    }
    else if ((dx < 0 && dy >= 0) || (dx >=0 && dy <0)){
      if (k > -PI/2.0 && ((-PI/2.0 < dk && dk < k) || (PI/2.0 < dk && dk < PI+k))){
if (axisXY == 1){
  return f*(fabs(dy)/d);
```

```
}
else{
  return f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
    }
  }

  if (k >= 0){
    if ((dx < 0 && dy < 0) || (dx >=0 && dy >=0)){
      if (k < PI/2.0 && ((PI/2.0 > dk && dk >k) || (k-PI < dk && dk < -PI/2.0))){
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
    }
    else if ((dx < 0 && dy >= 0) || (dx >=0 && dy <0)){
      if (k > PI/2.0 && ((-PI/2.0 < dk && dk < k-PI) || (PI/2.0 < dk && dk < k))){
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
```

```
}
      }
    }
  }
}

// ノードを円周上に初期配置する
int MagneticNodePlacer2::init(GRAPH<Node,Edge>& Graph){

  int NodeNum;
  int ox = 300, oy = 300, r;
  double angle, da;
  node nd;
  node_array<double> xp(Graph), yp(Graph);

  NodeNum = Graph.number_of_nodes(); // 全ノード数
  angle = 6.28 / NodeNum; // 角度差
  r = NodeNum * 5 + 40;   // 半径
  da = 0.0;               // 角度

  forall_nodes(nd,Graph){
    xp[nd] = r * cos(da) + ox;
    yp[nd] = r * sin(da) + oy;

    Graph[nd] = Node(Graph[nd].GetNodeId(), Graph[nd].GetNodeType(),
     xp[nd], yp[nd], Graph[nd].GetWidth(), Graph[nd].GetHeight());
    da += angle;
  }
}
```

MNLwithMentalmap.C

```
#include "NodePlacer.h"
#include <math.h>
#include <LEDA/node_matrix.h>

#define ITERATION 1000
#define PI M_PI       // π
#define DC1 30.0      // スプリング定数
#define DC2 100.0      // スプリングの長さ
#define DC3 20000.0      // 非隣接頂点間定数 (使ってない)
#define DC4 0.05       // 微小移動量定数
#define DC5 0.01      // 関連エッジの磁力定数
#define DC6 0.01      // 集約エッジの磁力定数
#define DC7 0.01      // 継承エッジの磁力定数
#define DC8 10.0       // 磁場の強さ
#define DC9 1         // 辺の長さの回転力への影響
#define DC10 1        // 角度の回転力への影響
#define DC11 0        // 関連エッジの角度
#define DC12 PI/4.0   // 集約エッジの角度
```

```
#define DC13 PI/2.0  //継承エッジの角度
#define DC14 0.1       //InertailForce係数

extern node get_node_fromid(GRAPH<Node,Edge>& g, string id);

void MNLmentalmap::NodeLayout(GRAPH<Node,Edge>& Graph){

  node n1,n2, n3;
  edge e;
  int i;
  double x,y,w,h,d1,d2;
  double nx=0, ny=0;

  node_array<double> forcex(Graph),forcey(Graph);

  //init(Graph);

  node_array<double> C4(Graph);
  node_array<double> C5(Graph);
  node_array<double> CX(Graph);
  node_array<double> CY(Graph);
  node_matrix<double> C1(Graph);
  node_matrix<double> C2(Graph);
  node_matrix<double> C3(Graph);

  //定数の決定
  forall_nodes(n1,Graph){
    forall_nodes(n2,Graph){
      d1 = sqrt(pow((Graph[n1].GetWidth()/2),2)+pow((Graph[n1].GetHeight()/2),2));
      d2 = sqrt(pow((Graph[n2].GetWidth()/2),2)+pow((Graph[n2].GetHeight()/2),2));

      //スプリング定数
      C1(n1,n2) = DC1;

      //スプリングの自然長
      C2(n1,n2) = DC2;

      //非隣接頂点間定数
      C3(n1,n2) = DC3;
    }
    //微小移動量定数
    C4[n1] = DC4;
    C5[n1] = DC14;
    CX[n1] = Graph[n1].xcoord();
    CY[n1] = Graph[n1].ycoord();
  }

  for(i=0;i<ITERATION;i++){
    forall_nodes(n1,Graph){
      forcex[n1] = 0;
```

```
      forcey[n1] = 0;

      forall_nodes(n2,Graph){
int flag = 0;

if (n1 !=n2){
  forall_inout_edges(e,n1){
    if (Graph.opposite(n1,e) == n2 && Graph.opposite(n2,e) == n1){

      //n1 と n2 が辺 e で隣接していた時スプリングによる力を計算
      forcex[n1] = forcex[n1] + AttractiveForce(Graph,n1,n2,0,C1[n1][n2],C2[n1][n2]);
      forcey[n1] = forcey[n1] + AttractiveForce(Graph,n1,n2,1,C1[n1][n2],C2[n1][n2]);

      //n1 と n2 が関連関係の時磁力を計算
      if (Graph[e].GetEdgeType() ==1){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,0,1,DC5,DC8,DC9,DC10,DC11);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,0,2,DC5,DC8,DC9,DC10,DC11);
      }

      //n1 が集約関係の親の時磁力を計算
      if (Graph[e].GetEdgeType() ==2 && n1 == Graph.source(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,1,1,DC6,DC8,DC9,DC10,DC12);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,1,2,DC6,DC8,DC9,DC10,DC12);
      }
      //n1 が集約関係の子の時磁力を計算
      if (Graph[e].GetEdgeType() ==2 && n1 == Graph.target(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,2,1,DC6,DC8,DC9,DC10,DC12);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,2,2,DC6,DC8,DC9,DC10,DC12);
      }

      //n1 が継承関係の親の時磁力を計算
      if (Graph[e].GetEdgeType() ==3 && n1 == Graph.source(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,1,1,DC7,DC8,DC9,DC10,DC13);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,1,2,DC7,DC8,DC9,DC10,DC13);
      }
      //n1 が継承関係の子の時磁力を計算
      if (Graph[e].GetEdgeType() ==3 && n1 == Graph.target(e)){
forcex[n1] = forcex[n1] + MagneticForce(Graph,n1,n2,2,1,DC7,DC8,DC9,DC10,DC13);
forcey[n1] = forcey[n1] + MagneticForce(Graph,n1,n2,2,2,DC7,DC8,DC9,DC10,DC13);
      }


      flag = 1;
    }
  }
  // 隣接していない時斥力を計算
  if (flag == 0){
    forcex[n1] = forcex[n1] + RepulsiveForce(Graph,n1,n2,0,C3[n1][n2]);
    forcey[n1] = forcey[n1] + RepulsiveForce(Graph,n1,n2,1,C3[n1][n2]);
  }
```

```
}
      }
      forcex[n1] = forcex[n1] +InertialForce(Graph,n1,0,C5[n1],CX[n1],CY[n1]);
      forcey[n1] = forcey[n1] +InertialForce(Graph,n1,1,C5[n1],CX[n1],CY[n1]);
    }

    //ノードn1を移動する
    forall_nodes(n1,Graph){
      x = Graph[n1].xcoord();
      y = Graph[n1].ycoord();
      x = x + C4[n1]*forcex[n1];
      y = y + C4[n1]*forcey[n1];
       Graph[n1] = Node(Graph[n1].GetNodeId(), Graph[n1].GetNodeType(),
                        x, y, Graph[n1].GetWidth(), Graph[n1].GetHeight());
    }
  }

  cout << "┌──── 調整前の座標と大きさ ────┐ " << endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight()<< ")" << endl;
  }
  cout << "└──────────────────────────────┘ "<< endl;
  //全体の位置の調整
  ShiftGraph(Graph);

  cout << "┌──── 調整後の座標と大きさ ────┐ "<< endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight()<< ")" << endl;
  }
  cout << "└──────────────────────────────┘ "<< endl;


  cout << "magneticspring-end" << endl;
}

double MNLmentalmap::AttractiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c, double d0){

  double d,dc,f,dx,dy;
  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();
```

```
  d = GetDistance(Graph, v, w);

  dc = d/d0;

  f = -c*log(dc);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

double MNLmentalmap::RepulsiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c){

  double d,dc,f,fx,fy,dx,dy;

  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();

  d = GetDistance(Graph, v, w);

  f = c/(d*d);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  }
}

double MNLmentalmap::InertialForce(GRAPH<Node,Edge>& Graph,
 node v, int axis, double c, double dkx, double dky){

  double dk,s,dx,dy,f;
  dx = Graph[v].xcoord() - dkx;
  dy = Graph[v].ycoord() - dky;
  dk = sqrt(pow((dx),2)+pow((dy),2));
  s = sqrt(pow((Graph[v].GetWidth()/2),2)+pow((Graph[v].GetHeight()/2),2));

  if (dk > s){
```

```
        f = -c*(pow((dk-s),2));

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

    switch(axis){
    case 0: return f*(dx/dk);
    case 1: return f*(dy/dk);
    }
  }
  else{
    return 0;
  }
}

int intersect(GRAPH<Node,Edge>& Graph, node v, node w){

  if ((Graph[v].xcoord()-Graph[v].GetWidth()/2) <
                  (Graph[w].xcoord()+Graph[w].GetWidth()/2)&&
      (Graph[w].xcoord()-Graph[w].GetWidth()/2) <
                  (Graph[v].xcoord()+Graph[v].GetWidth()/2)&&
      (Graph[v].ycoord()-Graph[v].GetHeight()/2)<
                  (Graph[w].ycoord()+Graph[w].GetHeight()/2)&&
      (Graph[w].ycoord()-Graph[w].GetHeight()/2)<
                  (Graph[v].ycoord()+Graph[v].GetHeight()/2))
    return 1;
  else
    return 0;
}

double MNLmentalmap::GetDistance(GRAPH<Node,Edge>& Graph, node v, node w){

  //dL: ノードの左上がりの対角線の傾き, dR: ノードの右上がりの対角線の傾き
  //a: ノードの中心間を結んだ線の傾き, (x1,y1): ノード v の中心, (x2,y2): ノード w の
中心
  double dLv, dRv, dLw, dRw, a, x1, x2, y1, y2;
  //(vx,vy),(wx,wy): ノード v,w から出ている線とノード v,w の枠との接点
  double vx, vy, wx, wy;

  // 重なっていた場合
  if (intersect(Graph, v, w)){
    return 10;
  }

  // ノードに重なりがなかった場合
  else{
    x1 = Graph[v].xcoord();
    y1 = Graph[v].ycoord();
```

```
    x2 = Graph[w].xcoord();
    y2 = Graph[w].ycoord();

    // 二つのノードが縦に並んでいた時
    if (x1 == x2)
      if (y1 > y2){
return (Graph[v].ycoord()-Graph[v].GetHeight()/2)-
                        (Graph[w].ycoord()+Graph[w].GetHeight()/2);
      }
      else{
return (Graph[w].ycoord()-Graph[w].GetHeight()/2)-
                        (Graph[v].ycoord()+Graph[v].GetHeight()/2);
      }
    // 二つのノードが横に並んでいた時
    if (y1 == y2)
      if (x1 > x2){
return (Graph[v].xcoord()-Graph[v].GetWidth()/2)-
                        (Graph[w].xcoord()+Graph[w].GetWidth()/2);
      }
      else{
return (Graph[w].xcoord()-Graph[w].GetWidth()/2)-
                        (Graph[v].xcoord()+Graph[v].GetWidth()/2);
      }
    // ノードが重なってもいなく真横、真縦にも並んでいない場合
    a = (y1-y2)/(x1-x2);
    dRv = -Graph[v].GetHeight()/Graph[v].GetWidth();
    dLv = Graph[v].GetHeight()/Graph[v].GetWidth();

    dRw = -Graph[w].GetHeight()/Graph[w].GetWidth();
    dLw = Graph[w].GetHeight()/Graph[w].GetWidth();

    //v は第 1 象現の上
    if (x1<x2 && y1>y2 && dRv>a){
      vy = Graph[v].ycoord()-Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第 3 象現の上
      if (dRw<a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第 3 象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第 1 象現の下
    else if (x1<x2 && y1>y2 && dRv<a){
      vx = Graph[v].xcoord()+Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
```

```
      //w は第３象現の上
      if (dRw<a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第３象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第２象現の上
    else if (x1<x2 && y2>y1 && dLv>a){
      vx = Graph[v].xcoord()+Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第４象現の下
      if (dLw>a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第４象現の上
      else{
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第２象現の下
    else if (x1<x2 && y2>y1 && dLv<a){
      vy = Graph[v].ycoord()+Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第４象現の下
      if (dLw>a){
wx = Graph[w].xcoord()-Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第４象現の上
      else{
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第３象現の下
    else if (x2<x1 && y2>y1 && dRv>a){
      vy = Graph[v].ycoord()+Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第１象現の上
      if (dRw>a){
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
```

```
      //w は第１象現の下
      else{
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
    }
    //v は第３象現の上
    else if (x2<x1 && y2>y1 && dRv<a){
      vx = Graph[v].xcoord()-Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第１象現の上
      if (dRw>a){
wy = Graph[w].ycoord()-Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
      //w は第１象現の下
      else{
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
    }
    //v は第４象現の下
    else if (x2<x1 && y1>y2 && dLv>a){
      vx = Graph[v].xcoord()-Graph[v].GetWidth()/2;
      vy = a*vx + y1 - a*x1;
      //w は第２象現の上
      if (dLw>a){
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第２象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
    }
    //v は第４象現の上
    else if (x2<x1 && y1>y2 && dLv<a){
      vy = Graph[v].ycoord()-Graph[v].GetHeight()/2;
      vx = (vy - y1 + a*x1)/a;
      //w は第２象現の上
      if (dLw>a){
wx = Graph[w].xcoord()+Graph[w].GetWidth()/2;
wy = a*wx + y1 - a*x1;
      }
      //w は第２象現の下
      else{
wy = Graph[w].ycoord()+Graph[w].GetHeight()/2;
wx = (wy - y1 + a*x1)/a;
      }
```

```
    }
    return sqrt((pow((vx-wx),2))+(pow((vy-wy),2)));
  }
}

double MNLmentalmap::MagneticForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axisST, int axisXY, double c, double b,
                            int alpha, int beta, double K){

  double d,dx,dy,dk,t,f,k;
  d = Graph[v].distance(Graph[w]);
  dx = (Graph[v].xcoord() - Graph[w].xcoord());
  dy = (Graph[v].ycoord() - Graph[w].ycoord());
  dk = atan2(dy,dx);

  //source のノードについて求める時は磁界の向きを逆転させる
  if (axisST == 1){
    if (K > 0)
      k = K - PI;
    else
      k = PI + K;
  }
  //target の時はそのまま
  else if (axisST == 2)
    k = K;

  // 無向磁針の時は角度の小さい方を磁界の向きにする
  else if (axisST ==0){
    if (fabs(K-dk) > 3.0/2.0*PI || PI/2.0 >= fabs(K-dk))
      k = K;
    else if (3.0/2.0*PI >= fabs(K-dk) || fabs(K-dk) > PI/2.0)
      if (K > 0)
k = K - PI;
      else
k = PI + K;
  }

  // 磁界と自分のなす角を求める
  if (fabs(k - dk) > PI)
    t = fabs(k - dk - 2.0*PI);
  else
    t = fabs(k - dk);

  // なす角が 0, または距離が 0 の時は磁力も 0
  if (d == 0 ||t == 0)
    return 0;

  f = c*b*pow(d,alpha)*pow(t,beta);

  if (k < 0){
```

```
    if ((dx < 0 && dy < 0) || (dx >=0 && dy >=0)){
      if (k < -PI/2.0 && ((PI/2.0 >= dk && dk >PI+k) || (k < dk && dk < -PI/2.0))){
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
    }
    else if ((dx < 0 && dy >= 0) || (dx >=0 && dy <0)){
      if (k > -PI/2.0 && ((-PI/2.0 < dk && dk < k) || (PI/2.0 < dk && dk < PI+k))){
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
    }
  }

  if (k >= 0){
    if ((dx < 0 && dy < 0) || (dx >=0 && dy >=0)){
      if (k < PI/2.0 && ((PI/2.0 > dk && dk >k) || (k-PI < dk && dk < -PI/2.0))){
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
```

```
  return -f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
    }
    else if ((dx < 0 && dy >= 0) || (dx >=0 && dy <0)){
      if (k > PI/2.0 && ((-PI/2.0 < dk && dk < k-PI) || (PI/2.0 < dk && dk < k))){
if (axisXY == 1){
  return -f*(fabs(dy)/d);
}
else{
  return -f*(fabs(dx)/d);
}
      }
      else{
if (axisXY == 1){
  return f*(fabs(dy)/d);
}
else{
  return f*(fabs(dx)/d);
}
      }
    }
  }
}

//ノードを円周上に初期配置する
int MNLmentalmap::init(GRAPH<Node,Edge>& Graph){

  int NodeNum;
  int ox = 300, oy = 300, r;
  double angle, da;
  node nd;
  node_array<double> xp(Graph), yp(Graph);

  NodeNum = Graph.number_of_nodes(); //全ノード数
  angle = 6.28 / NodeNum; //角度差
  r = NodeNum * 5 + 40;   //半径
  da = 0.0;               //角度

  forall_nodes(nd,Graph){
    xp[nd] = r * cos(da) + ox;
    yp[nd] = r * sin(da) + oy;

    Graph[nd] = Node(Graph[nd].GetNodeId(), Graph[nd].GetNodeType(),
     xp[nd], yp[nd], Graph[nd].GetWidth(), Graph[nd].GetHeight());
    da += angle;
  }
```

```
}
```

StraightLine.C

```
#include "EdgeRouter.h"
#include <math.h>
#include <LEDA/line.h>

void StraightLine::EdgeLayout(GRAPH<Node,Edge>& Graph){
  edge e;
  Node N1,N2;
  Edge E;
  int i;
  point sp1,sp2,sp3,sp4,ep1,ep2,ep3,ep4,sp,ep;
  polygon *spg, *epg;
  segment *el1, smt;
  list<point> pl, Ispl1, Ispl2, eline;

  forall_edges(e,Graph){
    N1 = Graph[Graph.source(e)];
    N2 = Graph[Graph.target(e)];
    E = Graph[e];

    sp1 = point(N1.xcoord()-N1.GetWidth()/2, N1.ycoord()-N1.GetHeight()/2);
    sp2 = point(N1.xcoord()+N1.GetWidth()/2, N1.ycoord()-N1.GetHeight()/2);
    sp3 = point(N1.xcoord()+N1.GetWidth()/2, N1.ycoord()+N1.GetHeight()/2);
    sp4 = point(N1.xcoord()-N1.GetWidth()/2, N1.ycoord()+N1.GetHeight()/2);

    ep1 = point(N2.xcoord()-N2.GetWidth()/2, N2.ycoord()-N2.GetHeight()/2);
    ep2 = point(N2.xcoord()+N2.GetWidth()/2, N2.ycoord()-N2.GetHeight()/2);
    ep3 = point(N2.xcoord()+N2.GetWidth()/2, N2.ycoord()+N2.GetHeight()/2);
    ep4 = point(N2.xcoord()-N2.GetWidth()/2, N2.ycoord()+N2.GetHeight()/2);

    pl.append(sp1);
    pl.append(sp2);
    pl.append(sp3);
    pl.append(sp4);

    spg = new polygon(pl);
    pl.clear();

    pl.append(ep1);
    pl.append(ep2);
    pl.append(ep3);
    pl.append(ep4);

    epg = new polygon(pl);
    pl.clear();

    Ispl1 = spg->intersection(segment(N1,N2));
```

```
    if(Ispl1.size() == 0){
      cout << "Ispll.size = 0" << endl;
      Ispl1.push(point(N1.xcoord(),N1.ycoord()));
    }
    eline.append(Ispl1.pop());

    Ispl2 = epg->intersection(segment(N1,N2));
    if(Ispl2.size() == 0){
      cout << "Ispl2.size = 0" << endl;
      Ispl2.push(point(N2.xcoord(),N2.ycoord()));
    }
    eline.append(Ispl2.pop());

    Graph[e].SetEdgeLine(eline);
    eline.clear();
  }
}
```

3DspringLayouter.C

```
#include "NodePlacer.h"
#include <math.h>
#include <LEDA/node_matrix.h>

#define ITERATION 5000
#define PI M_PI      // π
#define DC1 30.0     //スプリング定数
#define DC2 600.0     //スプリングの長さ
#define DC3 20000.0     //非隣接頂点間定数 (使ってない)
#define DC4 5      //微小移動量定数
#define DC5 0.01     //関連エッジの磁力定数
#define DC6 0.01     //集約エッジの磁力定数
#define DC7 0.01     //継承エッジの磁力定数
#define DC8 10.0      //磁場の強さ
#define DC9 1        //辺の長さの回転力への影響
#define DC10 1       //角度の回転力への影響
#define DC11 0       //関連エッジの角度
#define DC12 PI/4.0  //集約エッジの角度
#define DC13 PI/2.0  //継承エッジの角度
#define DC14 0.1       //InertailForce 係数

extern node get_node_fromid(GRAPH<Node,Edge>& g, string id);

void SpringNodePlacer::NodeLayout(GRAPH<Node,Edge>& Graph){

  node n1,n2, n3;
  edge e;
  int i;
  double x,y,z,w,h,d,d1,d2,d3;
```

```
double nx=0, ny=0, nz=0;

node_array<double> forcex(Graph),forcey(Graph),forcez(Graph);

//init(Graph);


node_array<double> C4(Graph);
node_array<double> C5(Graph);
node_array<double> CX(Graph);
node_array<double> CY(Graph);
node_array<double> CZ(Graph);
node_matrix<double> C1(Graph);
node_matrix<double> C2(Graph);
node_matrix<double> C3(Graph);

// 定数の決定
forall_nodes(n1,Graph){
  forall_nodes(n2,Graph){
    d1 = sqrt(pow((Graph[n1].GetWidth()/2),2)+pow((Graph[n1].GetHeight()/2),2));
    d2 = sqrt(pow((Graph[n2].GetWidth()/2),2)+pow((Graph[n2].GetHeight()/2),2));
    d3 = sqrt(pow((Graph[n2].GetDepth()/2),2)+pow((Graph[n2].GetDepth()/2),2));

    // スプリング定数
    C1(n1,n2) = DC1;

    // スプリングの自然長
    C2(n1,n2) = DC2;

    // 非隣接頂点間定数
    C3(n1,n2) = DC3;
  }
  // 微小移動量定数
  C4[n1] = DC4;
  C5[n1] = DC14;
  CX[n1] = Graph[n1].xcoord();
  CY[n1] = Graph[n1].ycoord();
  CZ[n1] = Graph[n1].zcoord();
}

  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2
         << Graph[n1].zcoord()-Graph[n1].GetDepth()/2 << ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight()<< ","
         << Graph[n1].GetDepth() << ")" << endl;
  }
```

```
  for(i=0;i<ITERATION;i++){
    forall_nodes(n1,Graph){
      forcex[n1] = 0;
      forcey[n1] = 0;
      forcez[n1] = 0;

      forall_nodes(n2,Graph){
int flag = 0;

if (n1 !=n2){
  forall_inout_edges(e,n1){
    if (Graph.opposite(n1,e) == n2 && Graph.opposite(n2,e) == n1){

      //n1 と n2 が辺 e で隣接していた時スプリングによる力を計算
      forcex[n1] = forcex[n1] + AttractiveForce(Graph,n1,n2,0,C1[n1][n2],C2[n1][n2]);
      forcey[n1] = forcey[n1] + AttractiveForce(Graph,n1,n2,1,C1[n1][n2],C2[n1][n2]);
      forcez[n1] = forcez[n1] + AttractiveForce(Graph,n1,n2,2,C1[n1][n2],C2[n1][n2]);

      flag = 1;
    }
  }
  // 隣接していない時斥力を計算
  if (flag == 0){
    forcex[n1] = forcex[n1] + RepulsiveForce(Graph,n1,n2,0,C3[n1][n2]);
    forcey[n1] = forcey[n1] + RepulsiveForce(Graph,n1,n2,1,C3[n1][n2]);
    forcez[n1] = forcez[n1] + RepulsiveForce(Graph,n1,n2,2,C3[n1][n2]);
  }
}
      }
    }

    // ノード n1 を移動する
    forall_nodes(n1,Graph){
      x = Graph[n1].xcoord();
      y = Graph[n1].ycoord();
      z = Graph[n1].zcoord();
      x = x + C4[n1]*forcex[n1];
      y = y + C4[n1]*forcey[n1];
      z = z + C4[n1]*forcez[n1];
       Graph[n1] = Node(Graph[n1].GetNodeId(), Graph[n1].GetNodeType(),x, y, z,
            Graph[n1].GetWidth(), Graph[n1].GetHeight(), Graph[n1].GetDepth());
    }
  }

  cout << " ┌──── 調整前の座標と大きさ ────┐ " << endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ","
         << Graph[n1].zcoord()-Graph[n1].GetDepth()/2<< ")"
```

```
        << ":(" << Graph[n1].GetWidth() << ","
        << Graph[n1].GetHeight() << ","
        << Graph[n1].GetDepth() << ")" << endl;
  }
  cout << " └──────────────────────────────────┘ "<< endl;
  // 全体の位置の調整
    ShiftGraph(Graph);

  cout << " ┌──── 調整後の座標と大きさ ────┐ "<< endl;
  forall_nodes(n1,Graph){
    cout << Graph[n1].GetNodeId()
         << ":(" << Graph[n1].xcoord()-Graph[n1].GetWidth()/2 << ","
         << Graph[n1].ycoord()-Graph[n1].GetHeight()/2<< ","
         << Graph[n1].zcoord()-Graph[n1].GetDepth()/2<< ")"
         << ":(" << Graph[n1].GetWidth() << ","
         << Graph[n1].GetHeight() << ","
         << Graph[n1].GetDepth() << ")" << endl;
  }
  cout << " └──────────────────────────────────┘ "<< endl;


  cout << "SpringLayout-end" << endl;
}

double SpringNodePlacer::AttractiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c, double d0){

  double d,dc,f,dx,dy,dz;
  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();
  dz = Graph[v].zcoord() - Graph[w].zcoord();

  d = Graph[v].distance(Graph[w]);

  dc = d/d0;

  f = -c*log(dc);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  case 2: return f*(dz/d);
  }
}
```

```
double SpringNodePlacer::RepulsiveForce(GRAPH<Node,Edge>& Graph,
 node v, node w, int axis, double c){

  double d,dc,f,dx,dy,dz;

  dx = Graph[v].xcoord() - Graph[w].xcoord();
  dy = Graph[v].ycoord() - Graph[w].ycoord();
  dz = Graph[v].zcoord() - Graph[w].zcoord();

  d = Graph[v].distance(Graph[w]);

  f = c/(d*d);

  if (f >500)
    f = 500;
  if (f < -500)
    f = -500;

  switch(axis){
  case 0: return f*(dx/d);
  case 1: return f*(dy/d);
  case 2: return f*(dz/d);
  }
}
```

A.2 OMT Editor

OMT Editor に関しては、エディタ本体、アニメーションを行う部分についてのソースコードを添付する。

omteditor

```
#! /opt/bin/wish -f

# ファイル（*.xbm）のあるディレクトリを設定
set xbmpath "~nogu/OMT/ALS/editor3/xbm"
# ファイル（omte,*.tcl,）のあるディレクトリを設定
set path "~nogu/OMT/ALS/editor3"
# ファイル（sendmes,als）のあるディレクトリを設定
set servpath "~nogu/OMT/ALS/server2"

if ![file exists $path/omte] {
  tk_dialog .d "error" "Files not found" error 0 OK
  exit 0
}

# オートロード機構の初期化
lappend auto_path $path
```

```
# プロシージャのオートロード
#auto_mkindex $path *.tcl

#global 変数の初期化
set Project ""
set DSLfile ""
set ItemList {}
set TagList {}
set EditType NONE
set CurStd NONE
set FromTo {}
set NewArcType NONE
set shadowbox 0
set AttNo 1
set EvtNo 1
set ALSERVER ""
set PORT ""

set OdNodesList {}
set OdArcsList {}
set EtdNodesList {}
set EtdArcsList {}

# 対象とする図の種類
set DiagramType OD

# タイトルの表示
wm title . "OMT Editor"

# ウィンドウの最大サイズ
wm maxsize . 750 750

# キャンバスウィジェットの設定
set cvs .mf2.canvas

if {$argc == 2} {
  set ALSERVER [lindex $argv 0]
  set PORT [lindex $argv 1]
}

frame .menu -relief raised -bd 2
menubutton .menu.file -text "ファイル" -menu .menu.file.m
menubutton .menu.edit -text "編集" -menu .menu.edit.m
menubutton .menu.layout -text "レイアウト" -menu .menu.layout.m

frame .button -relief raised -bd 2
button .button.b1 -bitmap @$xbmpath/od.xbm -command {set_defaultMode; \
  change_diagram OD}
button .button.b2 -bitmap @$xbmpath/ed.xbm -command {set_defaultMode; \
  change_diagram ETD}
```

```
button .button.b3 -bitmap @$xbmpath/std.xbm -command {set_defaultMode; \
  change_diagram STD}
message .button.ms -text "Object Diagram" \
-relief sunken -bd 2 -aspect 1000

menu .menu.file.m
.menu.file.m add command -label "新規作成 " \
   -command {set_defaultMode; clear_all}
.menu.file.m add command -label "開く " \
   -command {set_defaultMode; make_filebox load}
.menu.file.m add command -label "保存 " \
-command {savefile}
.menu.file.m add command -label "名前を付けて保存 " \
   -command {make_filebox save}
.menu.file.m add command -label "終了 " -command exit

menu .menu.edit.m
.menu.edit.m add command -label "クラスの作成 " \
    -command {set_defaultMode; \
    make_entrybox new_od_class "クラス名を入力して下さい。 "}

.menu.edit.m add command -label "関連の作成 " \
    -command {set_defaultMode; add_Arc assoc ""; display_message link1}

.menu.edit.m add command -label "集約の作成 " \
    -command {set_defaultMode; add_Arc aggr "" ; display_message agg1}

.menu.edit.m add command -label "継承の作成 " \
    -command {set_defaultMode; add_Arc inh "" ; display_message inh1}

.menu.edit.m add separator

.menu.edit.m add command -label "削除 " \
-command {set_defaultMode; set_delete_mode}

menu .menu.layout.m
.menu.layout.m add command -label "アルゴリズム " \
    -command mk_lbox
.menu.layout.m add command -label "レイアウト " \
    -command re_layout_all

frame .mf1 -relief raised -bd 1
frame .mf2
frame .mf1.fn -relief flat -bd 3
frame .mf1.fn.pro -relief groove -bd 2
frame .mf1.fn.dsl -relief groove -bd 2

button .mf1.fn.pro.name -text Project: -relief raised -bd 1 -pady 3 \
-command {set_defaultMode; \
         make_entrybox project "プロジェクト名を入力して下さい。 "}
```

```
label .mf1.fn.pro.ent -width 25 -relief sunken -bd 2 -text $Project
label .mf1.fn.dsl.name -text "DSL file:" -relief flat -pady 3
label .mf1.fn.dsl.ent -width 25 -relief sunken -bd 2 -text $DSLfile

message .mf1.msg -text {} -relief sunken -bd 2 -anchor w -width 1200 \
  -justify center

scrollbar .mf2.sbx -relief sunken -orient horizontal \
 -command {.mf2.canvas xview}
scrollbar .mf2.sby -relief sunken -orient vertical -command {.mf2.canvas yview}

canvas $cvs -xscrollcommand {.mf2.sbx set} \
  -yscrollcommand {.mf2.sby set} -width 1600 -height 1500 -bg oldlace \
  -scrollregion "100 100 1700 1600" -cursor top_left_arrow

pack .mf1.fn.pro.ent .mf1.fn.pro.name -side right -padx 10 -pady 3
pack .mf1.fn.dsl.ent .mf1.fn.dsl.name  -side right -padx 10 -pady 2
pack .mf1.fn.pro .mf1.fn.dsl -side top -fill x

pack .menu.file .menu.edit .menu.layout -side left -padx 2m -pady 1m
pack .menu -side top -fill x

pack .button.b1 .button.b2 .button.b3 -side left -padx 2m -pady 2m
pack .button.ms  -side right -padx 2m -pady 2m

pack .mf1.msg -side left -padx 5 -pady 3  -fill both -expand 1
pack .mf1.fn -side right

pack .mf1 -fill x

pack .mf2.sby -side right -fill y
pack .mf2.sbx -side bottom -fill x
pack .mf2.canvas -side top
pack .button -side top -fill x -side top

pack .mf2 -ipadx 5

bind $cvs <1> "CanvasB1Press %x %y name"
bind $cvs <2> "CanvasB23Press %x %y"
bind $cvs <3> "CanvasB23Press %x %y"

#trace variable EditType w {puts @@@@}

set Algorithms(AddNode) MagneticSpring2,StraightLine
set Algorithms(AddLine) MNLmentalmap,StraightLine

proc re_layout {LayoutType} {
  global Algorithms
```

```
  send_graph $Algorithms($LayoutType)
}

proc re_layout_all {} {
    global Algorithms

    if {[array size Algorithms]==0} {
        send_graph MagneticSpring,ExtLineSearch
    } {
     send_graph $Algorithms(AddNode)
    }
}
```

aces2serv.tcl

```
proc send_graph {LayoutType} {
    global servpath OdNodesList OdNodeLocation OdNodeSize OdArcsList OdArc \
    OdArcType ALSERVER PORT NodeName OdPreAnimLocation

    set MessageList {Graph}

    puts "┌──── ノードリスト ────┐ "
    foreach id $OdNodesList {
set x [lindex $OdNodeLocation($id) 0]
set y [lindex $OdNodeLocation($id) 1]
set w [lindex $OdNodeSize($id) 0]
set h [lindex $OdNodeSize($id) 1]
set mes node:$id,1,$x,$y,$w,$h
lappend MessageList $mes

set OdPreAnimLocation($id) $OdNodeLocation($id)

puts "$id $NodeName($id)"
    }
    puts "└──────────────────────┘ "
    puts "┌──── エッジリスト ────┐ "
    foreach id $OdArcsList {
set sn [lindex $OdArc($id) 0]
set en [lindex $OdArc($id) 1]
set x1 0
set y1 0
set x2 0
set y2 0
switch $OdArcType($id) {
    assoc {
set t 1
    }
    aggr {
set t 2
    }
```

```
    inh {
set t 3
    }
}
set mes edge:$id,$t,$sn,$en,$x1,$y1,$x2,$y2
lappend MessageList $mes
puts "$id $NodeName($sn) $NodeName($en)"
    }
    puts " └──────────────────────────┘ "

    lappend MessageList Layout:$LayoutType

    puts " ┌────── レイアウト前 ──────┐ "
    foreach l $MessageList {
puts $l
    }
    puts " └──────────────────────────┘ "
    set fp [open "|$servpath/sendmes $ALSERVER $PORT $MessageList"]

    puts " ┌────── レイアウト後 ──────┐ "
    while {![eof $fp]} {
set mes [gets $fp]
puts $mes
if {[llength $mes] > 0} {
    parse_message $mes
}
    }
    close $fp
    puts " └──────────────────────────┘ "

}

proc parse_message {message} {
    global OdNodeLocation OdNodesList OdNodeSize OdAssocName OdArc \
    OdArcMul OdArcType OdArcCoord OdArcsList Algm

    set Algm {}

    set message [split $message ,]
    set head [lindex $message 0]

    switch -regexp $head {
Graph {
}
node {
    set id [lindex [split $head :] 1]
    set x [lindex $message 2]
    set y [lindex $message 3]

    set OdNodeLocation($id) [list $x $y]
```

```
}
edge {
    set id [lindex [split $head :] 1]
    set etype [lindex $message 1]
    set sn [lindex $message 2]
    set en [lindex $message 3]
    set OdArc($id) [list $sn $en]
    set OdArcCoord($id) [lrange $message 4 end]
}
Layout {
    if {$message == "Layout:TellMeAlgorithms"} {
#  return;
    }
    clear_canvas

    animation

    foreach node $OdNodesList {
draw_OdNode $node
    }
    foreach arc $OdArcsList {

draw_OdArc $arc

set x1 [lindex $OdArcCoord($arc) 0]
set y1 [lindex $OdArcCoord($arc) 1]
set x2 [lindex $OdArcCoord($arc) 2]
set y2 [lindex $OdArcCoord($arc) 3]
set xe [lindex $OdArcCoord($arc) [expr [llength $OdArcCoord($arc)] - 2]]
set ye [lindex $OdArcCoord($arc) end]
set xe2 [lindex $OdArcCoord($arc) [expr [llength $OdArcCoord($arc)] - 4]]
set ye2 [lindex $OdArcCoord($arc) [expr [llength $OdArcCoord($arc)] - 3]]
set line [list $x1 $y1 $x2 $y2]

set snodeid [lindex $OdArc($arc) 0]
set sw [lindex $OdNodeSize($snodeid) 0]
set sh [lindex $OdNodeSize($snodeid) 1]
set sx1 [lindex $OdNodeLocation($snodeid) 0]
set sy1 [lindex $OdNodeLocation($snodeid) 1]
set sx2 [expr $sx1 + $sw]
set sy2 [expr $sy1 + $sh]

set enodeid [lindex $OdArc($arc) 1]
set ew [lindex $OdNodeSize($enodeid) 0]
set eh [lindex $OdNodeSize($enodeid) 1]
set ex1 [lindex $OdNodeLocation($enodeid) 0]
set ey1 [lindex $OdNodeLocation($enodeid) 1]
set ex2 [expr $ex1 + $ew]
set ey2 [expr $ey1 + $eh]
```

```
set smul [lindex $OdArcMul($arc) 0]
set emul [lindex $OdArcMul($arc) 1]

if {$y1 == $sy1} {set sside TOP}
if {$y1 == $sy2} {set sside BOTTOM}
if {$x1 == $sx1 && $x2 < $sx1} {set sside LEFT}
if {$x1 == $sx2 && $x2 > $sx2} {set sside RIGHT}

if {$ye == $ey1} {set eside TOP}
if {$ye == $ey2} {set eside BOTTOM}
if {$xe == $ex1 && $xe2 < $ex1} {set eside LEFT}
if {$xe == $ex2 && $xe2 > $ex2} {set eside RIGHT}

draw_rol_mul $arc $line

set flag 0
foreach id [array names OdAssocName] {
    if {$arc == $id} {set flag 1}
}
if {$flag == 1} {
    draw_assName $line $arc
}
if {$OdArcType($arc) == "inh"} {
    draw_inhMark $line $arc
}
if {$OdArcType($arc) == "aggr"} {
    draw_aggrMark $x1 $y1 $sside $arc
}
if {$OdArcType($arc) == "aggr" || $OdArcType($arc) == "assoc"} {
    if {$smul == "*" || $smul == "0,1"} {
draw_mulMark $x1 $y1 $sside $arc $smul
    }
    if {$emul == "*" || $emul == "0,1"} {
draw_mulMark $xe $ye $eside $arc $emul
    }
}
    }
}
NodePlacer {
    if {[lsearch $Algm $message] == -1} {
lappend Algm $message
    }
}
EdgeRouter {
    if {[lsearch $Algm $message] == -1} {
lappend Algm $message
    }
}
    }
}
```

```
proc animation {} {
    global OdNodeLocation OdNodesList OdNodeSize OdPreAnimLocation \
    OdAnimLocation cvs

    set frame 60

    foreach node $OdNodesList {
set x1 [lindex $OdPreAnimLocation($node) 0]
set y1 [lindex $OdPreAnimLocation($node) 1]
set width [lindex $OdNodeSize($node) 0]
set height [lindex $OdNodeSize($node) 1]
set x2 [expr $x1 + $width]
set y2 [expr $y1 + $height]
$cvs create rectangle $x1 $y1 $x2 $y2 \
-width 2 -fill whitesmoke -tags NN$node
    }

    for {set f 1} {$f <= $frame} {incr f} {
foreach i $OdNodesList {
    set nx [lindex $OdNodeLocation($i) 0]
    set px [lindex $OdPreAnimLocation($i) 0]
    set ny [lindex $OdNodeLocation($i) 1]
    set py [lindex $OdPreAnimLocation($i) 1]
    set nwX [expr (($nx - $px) * $f / $frame) + $px]
    set nwY [expr (($ny - $py) * $f / $frame) + $py]
    set seX [expr $nwX + [lindex $OdNodeSize($i) 0]]
    set seY [expr $nwY + [lindex $OdNodeSize($i) 1]]
    .mf2.canvas coords NN$i $nwX $nwY $seX $seY
}
update
    }
    $cvs delete all
}
```

A.3 3D-PP

3D-PP に関して、 Layout Server と通信をし、レイアウトを行っている部分のソースコードを添付する。

layout.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <strings.h>
#include <netdb.h>
#include "layout.h"

/*!
  Layout Class
*/
void GLBox::NodeLayout() {
  GUINodeCore* guinode;
  GUINodeCore* guinodenext;
  GUINodeCore* guiedgestart;
  GUINodeCore* guiedgeend;
  GUINodeCore* guinodefind;

  double d,dc,dk,f,fx,fy,fz,dx,dy,dz,d1x,d1y,d1z,d2x,d2y,d2z,fx1,fy1,fz1,fx2,fy2,fz2;
  int i,ID,ID1,ID2,ID3,Flag, sk, nbyte;
  struct sockaddr_in serv_addr;
  struct hostent *host;
  char mes[100];
  char* cut;
  char MessageList[10000] ="";
  char ReceiveList[10000] ="";

    for(guinode = gui_list_start->NodeNext; // 初期化ループ
guinode != NULL;
guinode = guinode->NodeNext){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if( ID1 == 0 && ID2 == 0)
{
  // 初期化
  guinode->Fx = 0.0;
  guinode->Fy = 0.0;
  guinode->Fz = 0.0;
  // グループ ID 初期化
  // ノードであればグループ ID を入れる
  guinode->GrpID = guinode->ID;
}
    }

    for(guinode = gui_list_start->NodeNext; // グループ ID 再付加処理 (リスト先頭
より開始)
guinode != NULL;
guinode = guinode->NodeNext){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if(guinode->GrpID != 0){
// ノード ID であればグループ ID の値を入れる
ID3 = guinode->GrpID;
```

```
      }
      // エッジがあった場合の処理
      if( ID1 != 0 && ID2 != 0)
{
  for(guiedgestart = gui_list_start->NodeNext; //Edge 始点レイアウトノード ID 検
索
      guiedgestart->ID < ID1 ;
      guiedgestart = guiedgestart->NodeNext){
  }
  for(guiedgeend = gui_list_start->NodeNext; //Edge 終点レイアウトノード ID 検索
      guiedgeend->ID < ID2 ;
      guiedgeend = guiedgeend->NodeNext){
  }
  // グループ ID 再付加
  // 既に継っているがグループ ID が異なる場合の再付加処理
  if(guiedgestart->GrpID != guiedgeend->GrpID)
    {
      if(guiedgestart->GrpID > guiedgeend->GrpID)
{
  guiedgeend->GrpID = guiedgestart->GrpID;
}
      if(guiedgestart->GrpID < guiedgeend->GrpID)
{
  guiedgestart->GrpID = guiedgeend->GrpID;
}
    }
}
    }
    for(guinode = gui_list_end->NodePrev; // グループ ID 再付加処理 (リスト後尾よ
り開始)
guinode != gui_list_start;
guinode = guinode->NodePrev){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if(guinode->GrpID != 0){
// ノード ID であればグループ ID の値を入れる
ID3 = guinode->GrpID;
      }
      // エッジがあった場合の処理
      if( ID1 != 0 && ID2 != 0)
{
  for(guiedgestart = gui_list_start->NodeNext; //Edge 始点レイアウトノード ID 検
索
      guiedgestart->ID < ID1 ;
      guiedgestart = guiedgestart->NodeNext){
  }
  for(guiedgeend = gui_list_start->NodeNext; //Edge 終点レイアウトノード ID 検索
      guiedgeend->ID < ID2 ;
      guiedgeend = guiedgeend->NodeNext){
  }
```

```
  // グループ ID 再付加
  // 既に継っているがグループ ID が異なる場合の再付加処理
  if(guiedgestart->GrpID != guiedgeend->GrpID)
    {
      if(guiedgestart->GrpID > guiedgeend->GrpID)
{
  guiedgeend->GrpID = guiedgestart->GrpID;
}
      if(guiedgestart->GrpID < guiedgeend->GrpID)
      {
  guiedgestart->GrpID = guiedgeend->GrpID;
}
    }
}
    }

  strcat(MessageList, "Graph\n");

  for(guinode = gui_list_start->NodeNext; // レイアウト処理
      guinode != NULL;
      guinode = guinode->NodeNext){

    if(6 < guinode->ID){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if( ID1 == 0 && ID2 == 0)
{
  // ノードの処理
  ID = guinode->ID;
  d1x = guinode->shape.vector.x;
  d1y = guinode->shape.vector.y;
  d1z = guinode->shape.vector.z;
  sprintf(mes,"node:%d,1,%f,%f,%f,350,350,350\n",ID,d1x,d1y,d1z);
}
      else
{
  // エッジの処理
  ID = guinode->ID;
  sprintf(mes,"edge:%d,1,%d,%d,1,2,3,4,5,6\n",ID,ID1,ID2);
}
      // 全体へ付け加える
      strcat(MessageList, mes);
    }
  }
  strcat(MessageList, "Layout:Spring,None\n");

  // サーバへ送る;
  bzero((char *) &serv_addr, sizeof(serv_addr));

  host = gethostbyname("hiwind.softlab.is.tsukuba.ac.jp");
```

```
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(5678);
  bcopy(host->h_addr, &serv_addr.sin_addr, host->h_length);

  if ((sk = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    perror("socket error!!");
    exit(1);
  }

  if (host == NULL){
    fprintf(stderr, "hiwind.softlab.is.tsukuba.ac.jp: Unknown host\n");
    exit(1);
  }

  if (::connect(sk, &serv_addr, sizeof(serv_addr)) == -1){
    perror("connect error!!");
    exit(1);
  }

  if ((nbyte = write(sk, MessageList, strlen(MessageList)+1)) == -1){
    perror("write error!!");
    exit(1);
  }
  printf("%s\n", MessageList);

  do {
    if ((nbyte = read(sk, ReceiveList, 10000)) == -1){
      perror("read error!!");
      exit(1);
    }

    if (nbyte > 0)
      printf("%s", ReceiveList);
  } while (nbyte > 0);

  printf("\n");

/*  if ((nbyte = close(sk)) !=0) {
    perror("close:");
    printf("%d\n", nbyte);
    exit(1);
  }
*/
  //返してもらった値を入れ直す
  cut = strtok(ReceiveList,":");

  do{
    //ノードの処理
    if(strcmp(cut, "node") == 0){
      cut = strtok(NULL, ",");
```

```
      printf("NODE: %2d: ", atoi(cut));
      ID = atoi(cut);

      for(guinode = gui_list_start->NodeNext;
  guinode->ID < ID ;
  guinode = guinode->NodeNext){
      }

      cut = strtok(NULL, ",");
      cut = strtok(NULL, ",");
      printf("(%9.4f", atof(cut));
      guinode->shape.vector.x = atof(cut);

      cut = strtok(NULL, ",");
      printf(",%9.4f", atof(cut));
      guinode->shape.vector.y = atof(cut);

      cut = strtok(NULL, ",");
      printf(",%9.4f)\n", atof(cut));
      guinode->shape.vector.z = atof(cut);

      cut = strtok(NULL, "\n");
    }

    else if(strcmp(cut, "edge") == 0){
      cut = strtok(NULL, ",");
      printf("EDGE: %2d:", atoi(cut));

      cut = strtok(NULL, ",");
      cut = strtok(NULL, ",");
      printf(" (%s - ", cut);
      cut = strtok(NULL, ",");
      printf("%s)\n", cut);
      cut = strtok(NULL, ",");
      cut = strtok(NULL, ",");
      cut = strtok(NULL, ",");
      cut = strtok(NULL, "\n");
    }
    cut = strtok(NULL, ":");

  }while(!(strcmp(cut,"Layout\n") == 0));



  updateGL();
  printf("-------------------------------------------------------------\n");
}


void GLBox::Force1( int ID1, int ID2 ) {
```

```
  GUINodeCore* guinode;
  GUINodeCore* guinodenext;

  double d,dc,dk,f,fx1,fy1,fz1,dx,dy,dz,d1x,d1y,d1z,d2x,d2y,d2z;
      if( ID1 != 0 && ID2 != 0)
        {
          for(guinode = gui_list_start->NodeNext; //Edge 始点レイアウトノード
              guinode->ID < ID1 ;
              guinode = guinode->NodeNext){
          }
          d1x = guinode->shape.vector.x;
          d1y = guinode->shape.vector.y;
          d1z = guinode->shape.vector.z;

          //  printf("LayoutTest1!!\n");
          for(guinodenext = gui_list_start->NodeNext; //Edge 終点レイアウトノー
ド
              guinodenext->ID < ID2 ;
              guinodenext = guinodenext->NodeNext){
          }
          //printf("LayoutTest2!!\n");
          d2x = guinodenext->shape.vector.x;
          d2y = guinodenext->shape.vector.y;
          d2z = guinodenext->shape.vector.z;

          dx = d1x-d2x;
          dy = d1y-d2y;
          dz = d1z-d2z;

          dk = sqrt((dx*dx)+(dy*dy)+(dz*dz));

          /*  バネの力を計算  */
          f = DC1*log(dk/DC2);
          fx1 = f*(dx/dk);
          fy1 = f*(dy/dk);
          fz1 = f*(dz/dk);

          //  ノードの移動

          if(guinodenext->ID>6)
            {
              guinodenext->shape.vector.x += LAYOUT_MOVE*fx1;
              guinodenext->shape.vector.y += LAYOUT_MOVE*fy1;
              guinodenext->shape.vector.z += LAYOUT_MOVE*fz1;
            }
          updateGL();
        }
}
  void GLBox::Force2() {
    printf("TEST_Force2\n");
```

```
  }

void GLBox::NodeFind() {
  // 論理変数リストから相手のノード ID を参照する
  GUINodeCore* guinode;
  GUINodeCore* guinodenext;
  //printf("NodeFinder!!\n");
}
```