平成 12 年度

筑波大学第三学群情報学類

卒業研究論文

題目：ビジュアルシステム恵比寿における制約解消の高速化

主専攻	情報科学
著者名	土屋 洋夢
指導教員	電子・情報工学系 田中 二郎

要旨

図形間に「制約」を与えることで、図に意味を持たせその図を利用することができるビジュアルシステム恵比寿が開発されている。しかし現在のシステムでは、図形の数や図形間の制約数が多くなると、解析に時間がかかり実時間で扱えなくなってくる。そこで本論文では、現在恵比寿で使用している制約を解く制約解消系 SkyBlue を高速な制約解消が行える制約解消系 HiRise と、不等式を扱う Cassowary に変更することで、ビジュアルシステム恵比寿における制約解消の効率化を試みた。

目次

図一覧

第 1 章

序論

近年、さまざまな分野において、ビジュアル言語が用いられている。ビジュアルプログラミング言語やER ダイアグラム、OMT のオブジェクト図、回路図、状態遷移図、家系図といったビジュアル言語はある構造をもった、いわば図形を用いた言語である。つまり、空間的な文法を有しているといえる。そのためこの図形言語を解析する Spatial Parser を実装する必要がある。しかし、個々のビジュアルプログラミングシステム毎に Spatial Parser を実装するのは、困難で時間のかかる作業であった。

そこでビジュアル言語の文法を定義することで、この Spatial Parser を生成し、さまざまな図形言語を解析し、さらに実行することができるシステム「恵比寿」 [1][2][3][4] が開発されている。しかし現在の恵比寿は、図形のもつ制約の数が多くなると実時間で扱えないという欠点があった。そのため大規模なビジュアルシステムとして利用できなかった。本論文では、恵比寿の文法定義における「制約」を解く「制約解消系」について考察し、制約数の増大によって実時間で扱えなくなるという恵比寿の問題点の解決方法を提案する。

本論文の構成は次の通りである。第 2 章で既存の恵比寿システムの概要とその問題点について述べる。第 3 章で制約、制約解消系について述べ、第 4 章で制約解消系とのインタフェース、第 5 章で実装方法、第 6 章でまとめる。

第 2 章

恵比寿の概要

本章では、ビジュアルシステム恵比寿の概要とその問題点について述べる。

2.1 恵比寿

恵比寿はビジュアル言語を利用したビジュアルシステムを生成することができるシステムである [1][2][3][4]。恵比寿は Spatial Parsing Generator と図形間の関係を表している制約を解く制約解消系を持つことで、ある文法に基づいて図形を描き、その描かれた図形の解析を行い、そしてその結果を利用して処理を行うことができる。（図 2.1）
つまり恵比寿の機能としては以下の 3 点が上げられる。

1. 図形間の制約を定義できる
2. 定義した制約を用いて図形を解析することができる
3. 解析した図形を利用して何らかの処理を行うことができる

2.1.1 図形エディタ

図 2.1の画面の上半分を定義ウィンドウと呼び、ビジュアル言語の作成者はここで文法の定義を行う。下半分を実行ウィンドウと呼び、ユーザーはここで実際に解析、実行したい図形を入力する。定義ウィンドウ、実行ウィンドウにおける図形の入力では、通常のドローツールにあるようなコピー、削除など操作を行うことができる。扱える図形の種類は、楕円、長方形、直線、テキスト、円弧、 GIF イメージがある。これらは、色、フォントなどの属性を指定できる。

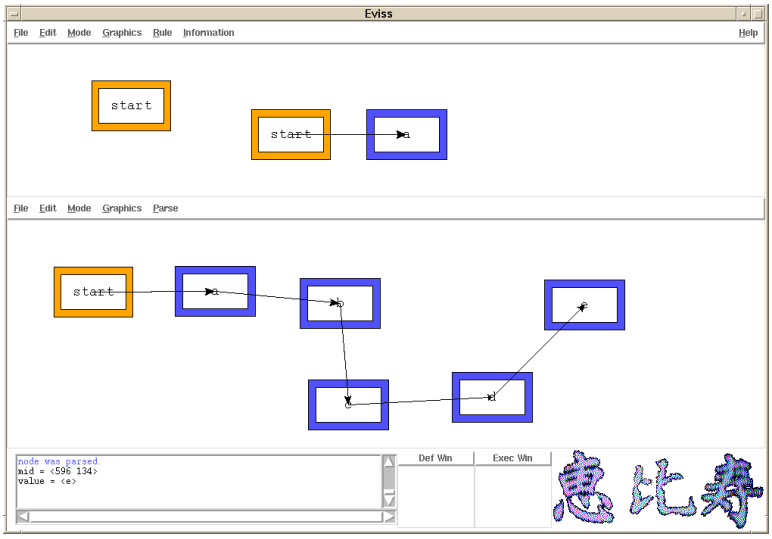


図 2.1: 恵比寿の実行画面 1

2.1.2 ビジュアル言語の定義方法

恵比寿ではビジュアル言語の定義に拡張 CMG[1] を用いる。拡張 CMG[1] とは図形間の関係を記述している CMG （Constraint Multiset Grammars） [13] に、 action の機能を備えたものである。 action の機能を持たせることで図形文法に従った図形解析だけでなく、属性値（色や文字列の値、線の太さなど）の変更や図形の移動、新たな図形の生成、削除、書き換えといったように、解析結果を利用してなんらかの処理を行わせることも可能になっている。

2.1.3 CMG 入力ウィンドウ

図形間の制約を入力する CMG 入力部（図 2.4）は上から順に名前（name）、属性（attributes）、アクション（action）、構成要素間の制約（constraints）、構成要素の数と種類を書く欄にわかれている。構成要素の数と種類を書く欄はさらに左から順に normal、 exist、 not_exist、 all にわかれている。 normal 欄には定義する図形単語の部品となる構成要素を書く。つまりここに書かれたものすべての存在が前提となり、うち一つでも欠けると図形単語として認識されない。 exist の欄には、定義する図形単語を認識するために、図全体の中の何処かに存在する必要があるものを書く。これも一つでも欠けると図形単語としては認識されない。 not_exist の欄には、定義する図形単語を認識するために、その図全体の中のどこかに存在してはならないものを書く。ここに書かれたもののうちどれか一つでも存在したら図形単語としては認識されない。 all の欄に書くと、制約を満たすもの全てを集める。インクリメンタルに all の要素として認識する機能と、 all の欄に書かれたものを使った制

約を書く機能は、まだサポートされていない。

2.1.4 恵比寿での制約の与え方

恵比寿システムでは以下のようにして、ビジュアル言語を定義していく。まず、図 2.2のように定義ウィンドウに図形を描き、大まかな文法と構成要素を描く。そしてその図を利用して恵比寿が制約を自動的に生成し、 CMG 入力ウィンドウにテキストとして書き出す。（図 2.3, 図 2.4）そのあと、ユーザーは CMG 入力ウィンドウ（図 2.4）で生成された文法をみながら制約の追加、削除や action の定義を書き、目的の制約を完成させ図形に意味を持たせる。

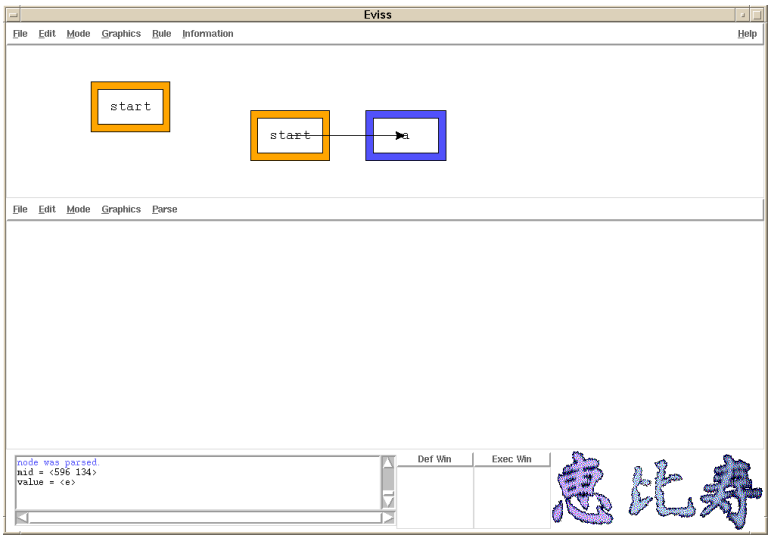


図 2.2: 恵比寿の実行画面 2

2.2 問題点

恵比寿では図形間の関係を「制約」を用いて表現しているため、図形の数や図形間に課せられた制約の数が多くなると、その制約を保つための計算に時間を要し、実時間で扱えなくなってくる。そのため既存の恵比寿では非常に小規模なビジュアルシステムとしてしか機能できず、恵比寿の持つ「さまざまな図形言語を解析し、さらに実行することができる」という機能を大きく妨げていた。そこで本論文では、ビジュアルシステム恵比寿が使用している、制約を保つための機構である制約解消系について考察し制約が増大した場合にも実時間で対応できる環境を提案する。

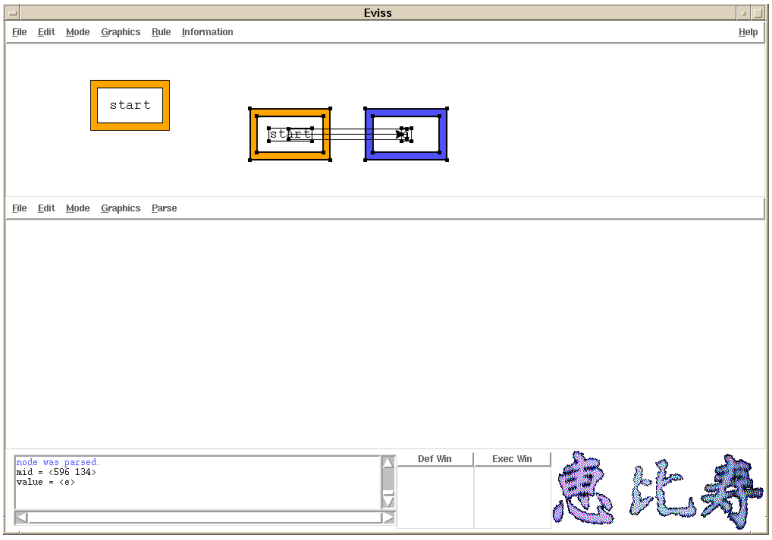


図 2.3: 恵比寿の実行画面 3

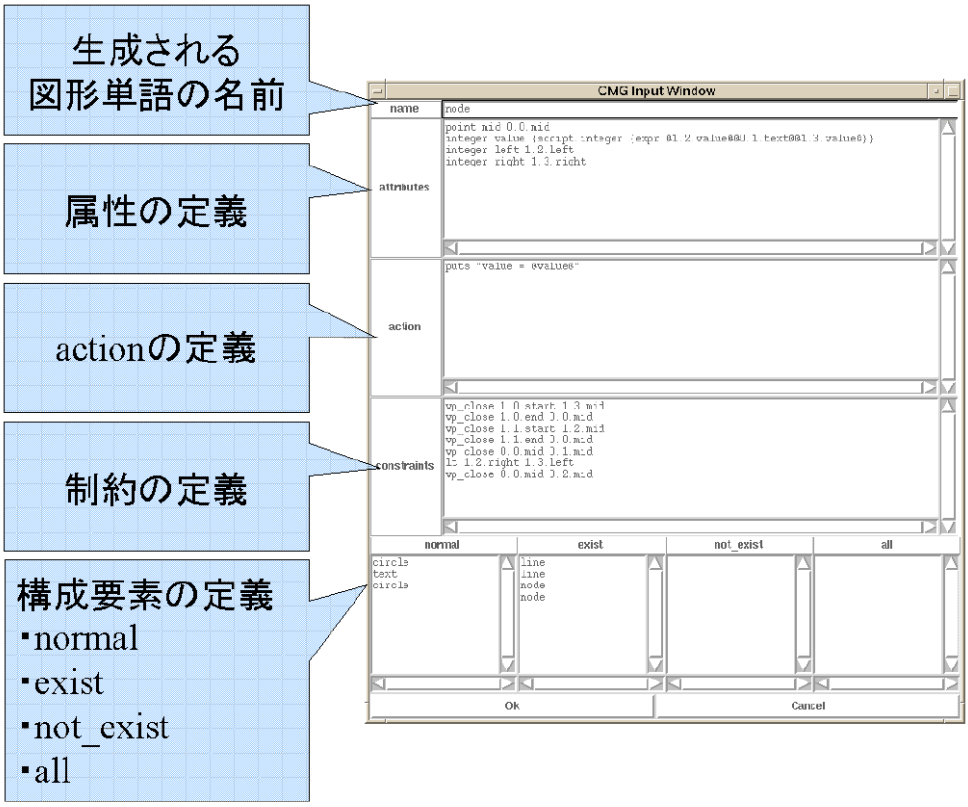


図 2.4: CMG 入力ウィンドウ

第 3 章

制約解消系と問題点の解決方法

3.1 制約解消系について

恵比寿では図形間の関係を表すのに「制約」を用いている。制約とは、いくつかの変数の間に常になり立っている関係のことである。たとえば、三つの変数 a, b, c の間に a+b=c という制約が成り立ってるとする。今、 a, b の値がそれぞれ 3, 4 だとすると、 c の値は 7 である。このとき、 a の値を 5 に変えたとする。すると、この制約を成り立たせるためには、 c の値を 9 にするか、または b の値を 2 にしなければならない。また、別な例として、三つの変数 a, b, c の間に (1) a=b (2) b=c という二つの制約が成り立っているとする。今、 a の値が 5 だとしたら、 b, c の値も 5 である。このとき、 c の値を 8 にすると、 (2) から b の値は 8 になり、その影響を受けて (1) から a の値も 8 にしなければならない。このように、複数の変数の間に成り立っている制約を常に成り立たせるための機構を「制約解消系」と呼ぶ [18]。恵比寿の内部では制約解消系として SkyBlue[10][11][12] を使用している。
図形に制約を与えた例として、オブジェクト a,b,c,d を用いて四角形を表したものを図 3.5に示す。このビジュアル言語を表現するためには、直観的にオブジェクト a, b,c,d が四角形の頂点を表しているという制約が与えられればいいというのがわかる。この制約を列挙すると、オブジェクト a,b,c,d に対しては円の中心座標とテキストの中心座標が一致するという制約、そしてそのオブジェクト a,b,c,d の中心座標が線分の端点座標と一致するという制約でこの四角形を表すビジュアル言語を表現できる。そしてこの図形をレイアウトする時、例えばオブジェクト c を移動させれば、それにともなって円の中に描かれた text、そしてオブジェクト c から、オブジェクト b、オブジェクト d につながれた line も移動されなければならない。恵比寿ではこのような処理を座標が一致という制約を利用し、制約解消系を使って自動管理している (図 3.2)。

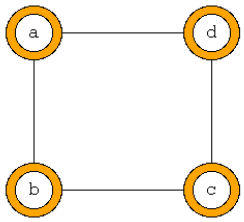


図 3.1: 制約を用いた四角形の表現

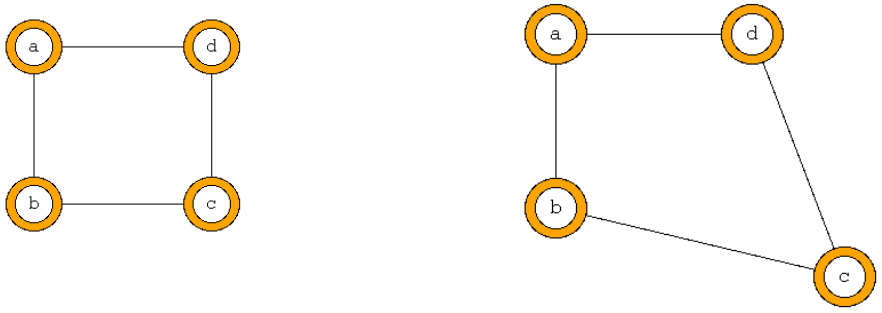


図 3.2: 頂点の移動

3.2 制約解消アルゴリズム

互いに作用し合う制約の集合を制約系と呼ぶ。ここではこの制約系を解く制約解消アルゴリズムについて説明する。制約を扱うには様々な問題がる。先ず制約過多の問題である。ユーザーが制約を作成していった場合、制約間に矛盾が生じる場合がある。矛盾が生じなければ全ての制約を充足するように、解いていけばいいわけだが制約系に矛盾が生じてしまった場合は、ただ全ての制約の充足を解の条件とする方式では解なしの状態になってしまう。しかし、恵比寿のように図形表示に制約を用いている場合、何らかの妥協的な解が必要とされる。このため制約系を可能な限り、充足し妥協的な解を求めることができる能力が必要とされる。また制約不足の問題もある。先の例で示した a+b=c （初期値では 3+4=7）の plus 制約では b の値を 5 変更した時に、この plus 制約が成立するためには、 a の値が 2 になってもよいし、 c の値が 8 になっても制約が満たされることになった。この plus 制約が与えられただけでは、ただ 1 つの解を定めることができない。つまり制約の与え方が十分でないため制約不足の状態になり、可能な解が複数個存在してしまうわけである。この問題を解決するためには、ユーザーが、ただ 1 つの解を定めれるよう必要十分な制約を指定すればよいことだが、常にそのような制約を指定することは、ユーザー

にとって大きな負担となる。これら問題の解決方法として制約に優先度を指定できるようにすることが有効であることが指摘され、制約階層 [6] と呼ばれる定式化に基づくものを中心にさまざまな研究が行われてきている。制約階層は、強さと呼ばれる有限段階の優先度を制約に与えたものである。制約に優先度がある場合、優先度の高い制約から満たすことが直観的いいと考えられる。その強さに基づき、1 つの制約階層は、階層的なレベルに分割された、制約の集合として表される。そして、制約階層の解は、できるだけ強い制約が満たされるよう決定される。制約階層のための制約解消アルゴリズムは以下の主な 3 種類と、その他に分けることができる。

3.2.1 精製法

各レベルを強い順に充足することで、制約階層を解消する。最も単純な方法。この方法を採用しているは Borning らの初期のシステム [7] や DeltaStar[8] などがある。これらは、制約階層の研究の初期のものであるので現在研究が行われている制約解消系と比較して、効率的な制約解消を行える能力は持っていない。

3.2.2 局所伝播法

制約階層の構造に基づいて制約をスケジューリングし、一意に充足可能な制約を順に選んで解いてゆくデータフロー方式。この制約解消はプランニングと実行の 2 段階からなる。まず、プランニングの段階で制約の強さを考慮し、充足すべき制約のメソッドを決定し、実行段階で制約を解いていく。この方法を採用しているのは Blue[9],DeltaBlue[9] などがある。恵比寿で使用している制約解消系 SkyBlue[10][11][12] もこの制約解消アルゴリズムを用いている。

3.2.3 最適化アプローチ

最適化アプローチは、強さを重みに対応させ、制約階層を、線形計画法などの 1 つの最適化問題に変換して解く。この解法を用いた制約解消系には、シンプレックス法を応用して解を求める Cassowary[14][15][16] と、アクティブセット法を用いて解を求める QOCA[17] がある。

3.2.4 その他の方法

以上の 3 種類に分類されない制約階層の制約解消アルゴリズムとして HiRise[19][20] がある。この解法は制約階層を階層線形系に変換して処理を行う。制約階層では、

優先度は有限段階であるが、階層線形系では、優先度は個々の制約ごとに異なり全順序になっている。 HiRise はこの階層線形系を LU 分解を利用して解く。

3.3 問題点の解決方法

制約数の増大により、実時間で扱えなくなるという恵比寿の問題点は、制約解消系の変更という方法で解決できると考えられる。なぜなら、ビジュアルシステム恵比寿の処理能力はその制約を保つための機構である制約解消系の処理能力に依存している部分があるからである。現在使用している制約解消系 SkyBlue では、増大した制約数に対応できず、恵比寿の能力を妨げていると言わざるを得ない。そのため、増大した制約数にも対応できる制約解消系、つまり高速な制約解消が行える制約解消系を使用を使用することで、今まで小規模なビジュアルシステムとしてしか機能できなかったシステムが、非常に巨大なビジュアルシステムとしても機能できる能力を得ることができる。

3.4 検討すべき制約解消系の候補とその理由

まず恵比寿に用いられている制約解消系 SkyBlue について考察する。恵比寿での SkyBlue はもともと 2 つの制約（3 つの変数 a, b, c, があった時、 a=b を表す eq 制約、 a+b=c を表す plus 制約）が実装されてあるものに、浮動小数点、文字列、点（x, y 座標の組）も扱えるようにされている。また変数には id, 値、型が与えられている。SkyBlue で操作できる機能として、変数に対しては作成、削除、値の変更、値を得る、型を得るといった操作。制約に対しては、作成、削除、制約している変数の id を得るといった操作ができる。そして、その他に「上にある」「左にある」といった図形間の位置関係も表せるよう不等式も扱えるようになっている。つまり、検討すべき候補としては以上のような操作ができるもの、または拡張することで以上のような操作ができるもの、そして制約解消の効率の向上を目的とした制約解消系が望まれる。しかしながら、不等式を扱おうとすると、どうしても制約解消の効率が悪くなり、大規模なビジュアルシステムには不向きである。その原因は「制約解消の効率の向上」と「制約の表現力の強化」の 2 点の間にはトレードオフが存在するためである。表現力の強化を行おうとすると現段階のアルゴリズムでは制約解消のスピード低下に陥ってしまう。よって、既存の恵比寿システムに制約解消の効率化を目的とした制約解消系を組み込むことで、制約系に不等式が存在すれば不等式が扱える制約解消系を、等式だけの場合は高速な制約解消が行える制約解消系を使用で

きるようにし制約解消の効率化を図る。[19] に各制約解消アルゴリズムの代表的な制約解消系を用いて実験を行った結果（図 3.6）がある。この実験に用いられている制約解消系は、局所伝播法に分類される SkyBlue と、最適化アプローチに分類される Cassowary、そして制約階層を階層線形系にして解く HiRise である。 HiRise が扱えるのは一次等式のみ、 Cassowary は一次等式、一次不等式の両方が扱える制約解消系である。実験では、 SkyBlue は原論文の著者が配布している C 版のものに Crout 法による LU 分解を用いた cycle solver を組み込んだものを使用している。 HiRise、 Cassowary については原論文者らが配布している C++ 版のものを使用している。実験内容は図 3.5の 2 分木の編集を行ったものである。この図形には、葉の間隔が一定になるようノードを配置している。制約の定義としては垂直方向には階層的に、水平方向は隣り合う部分木のバウンディングボックスが隣接するように決められている。行った実験は (a)2 分木の初期配置、 (b) ノードのドラッグによる 2 分木の移動の開始、 (c) 移動の繰返し、 (d) 移動の終了、 (e) 新しいノードの追加、 (f) 既存のノードの削除の 6 種類である。実験の結果によると、 HiRise は、他の二つに比べ高速な処理が行うことができている。例外は、 (b) の移動の繰返しにともなう計算で SkyBlue よりも遅いが、特に問題になる程の遅さではない。実験結果より HiRise を使用すれば安定したスピードで数千個の変数まで扱えるため、恵比寿で SkyBlue を使用していた時に比べて大幅な効率化が期待できる。また制約系に不等式が存在した場合には HiRise と同様の制約の与え方が行える構造を持った制約解消系 Cassowary を使用する。不等式が存在する場合に Cassowary を使用するようにすることで制約系の種類により HiRise、 Cassowary の切替が容易に行える。ビジュアル言語の大部分は等式のみで表現できるので、 HiRise を利用した高速な制約解消を行うことができる。 HiRise を用いて制約解消を行う例としてリスト構造（図 3.3）、 Cassowary を用いて制約解消を行う例として計算の木（図 3.4）をあげる。図 3.3のリスト構造は、(1) 四角形の中心にラベルが書かれたものをリストとする。 (2)1 つのリストが、ラベルが書かれた四角形に結線されているものをリストとする。といった定義で表現できる。これは「ある図形のある座標とある図形のある座標が一致する」という等式のみで表現できるので HiRise を使って制約解消を行う。図 3.4の計算の木は、 (1) 楕円の中に数字が書かれたものをノードとする。 (2) 演算子ノードは 2 つのノードに結線されている。といった定義で表現できる。これは「ある図形のある座標とある図形のある座標が一致する」という等式の他に演算子ノードが、演算子ノードにつながれている数字がかかれたノードを判別するのに「右にある、左にある」といった位置情報を不等式を用いて表現しているため Cassowary を使って制約解消を行う。

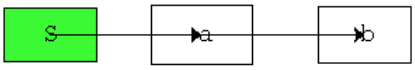


図 3.3: リスト構造

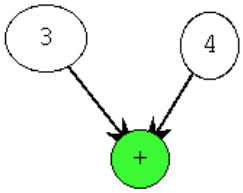


図 3.4: 計算の木

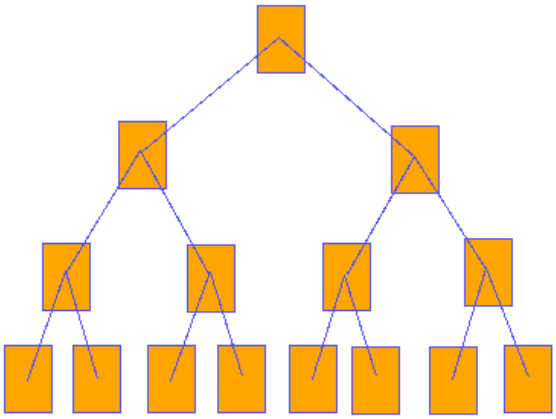
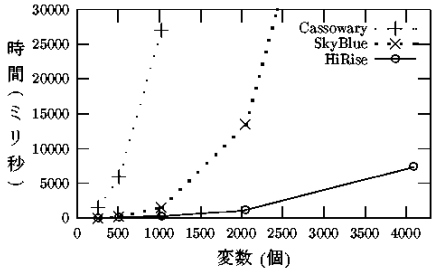
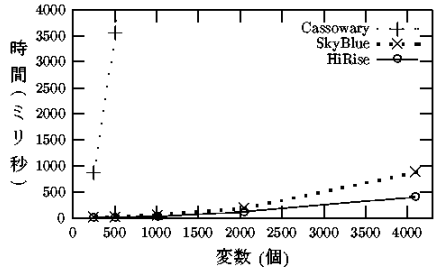


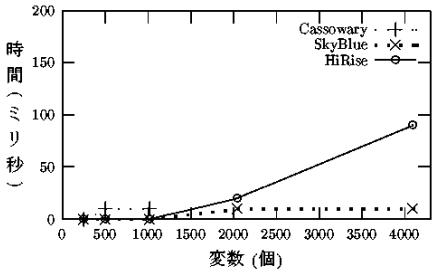
図 3.5: 2 分木の編集



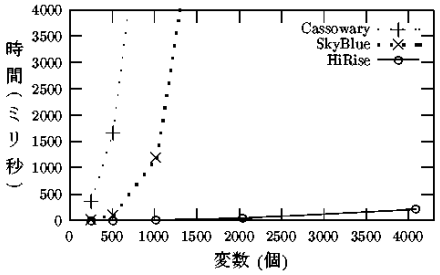
(a) 初期配置



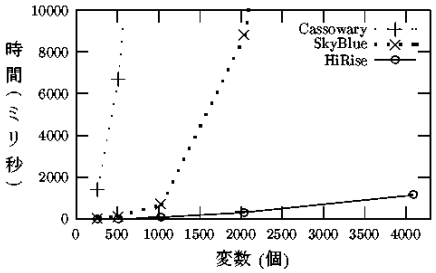
(b) 移動の開始



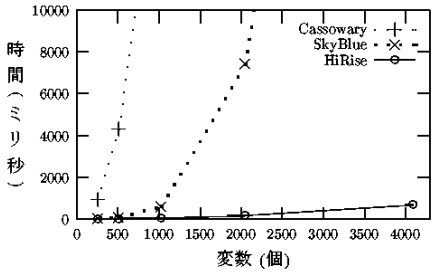
(c) 移動の繰返し



(d) 移動の終了



(e) ノードの追加



(f) ノードの削除

図 3.6: 2 分木の編集のアプリケーションにおける制約解消の時間

第 4 章

制約解消系へのインタフェース

4.1 処理の流れ

ここではリスト（図 4.1）を表すビジュアル言語を定義する例を通して恵比寿の制約作成法とその制約の制約解消系への流れを説明する。この図は長方形の中にテキストが書かれたもの（リスト）と、それらを結ぶ線（エッジ）から構成されている。このことからこのリストを定義するには、次の２つルールを定義することによって表現できる。

1. 四角形の中心にラベルが書かれたものをリストとする
2. １つのリストがラベルの書かれた四角形に結線されているものをリストとする

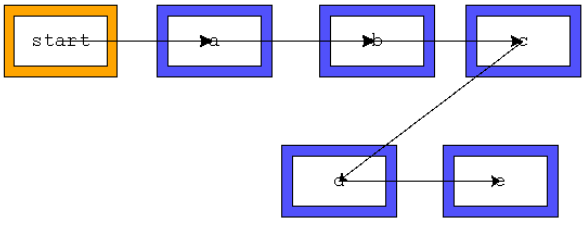


図 4.1: リスト

4.1.1 ルール１の定義方法

1. 定義ウィンドウに図 4.2の図形を描く。

2. それらの図形をまとめて選択する。

3. 選択した図形から制約が自動生成され、 CMG ウィンドウに書き出される。

4. 自動生成された制約をユーザーが修正し目的の制約を作る。（図 4.4）

4.1.2 ルール 2 の定義方法

1. 定義ウィンドウに図 4.3の図形を描く。

2. それらの図形をまとめて選択する。

3. 選択した図形から制約が自動生成され、 CMG ウィンドウに書き出される。

4. 自動生成された制約をユーザーが修正し目的の制約を作る。（図 4.5）

図 4.2: リスト 1

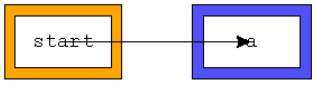


図 4.3: リスト 2

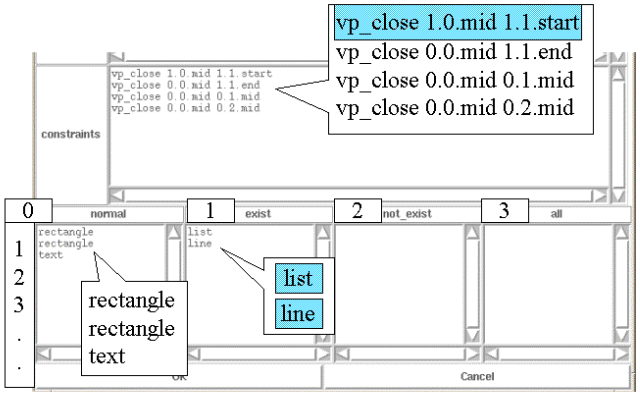


図 4.4: リスト 1 を表す制約が描かれた CMG ウィンドウ

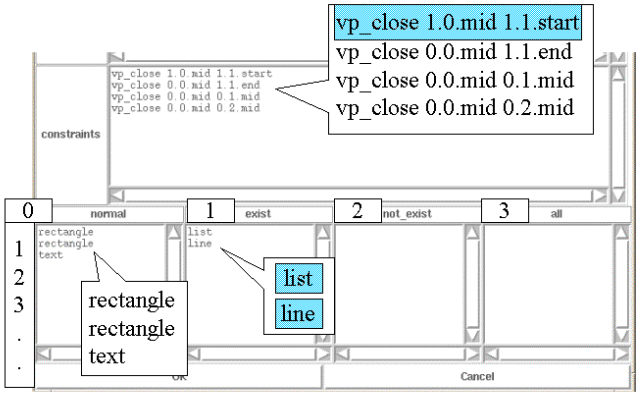


図 4.5: リスト 2 を表す制約が描かれた CMG ウィンドウ

以上の操作でリストを表したビジュアル言語にに必要な制約が作成されたのでこれをもとに目的の図形を実行ウィンドウに描く。実行ウィンドウにリストを描き（図 4.6）、その後、描かれた図形をまとめて選択し、 Spatial Parsing を行って図形に制約を与える。制約解消系に渡される制約は CMG ウィンドウで書かれた制約だけではなく、実行ウィンドウに描かれた図形の形状に関する制約も渡される。例えば Parsing を行った実行ウィンドウに長方形があれば、その長方形の左上の x 座標 (lu_x)、y 座標 (lu_y)、右下の x 座標 (rl_x)、 y 座標 (rl_y) をもとにこの 4 つの属性から、中心座標、 x 方向の長さ（幅）、 y 方向の長さ（高さ）などを求めるための計算式が制約として作られる。この四つの属性（lu_x,lu_y,rl_x,rl_y）は楕円を描いた時に決定される。長方形、直線、テキストなどに対してもこのような図形の形状に関する制約が作られ、制約解消系に渡される。

4.2 CMG 入力部の表記法

CMG ウィンドウで図形間の座標を一致させる制約は具体的に次のように記述する。

```
<eq,vp_close> 変数 1 変数 2
```

ここで eq は「等しい」、 vp_close は「ほぼ等しい」を表す。変数 1、変数 2 は図形単語の属性を示している。変数の型は次の形で表す。

```
type.id.name
```

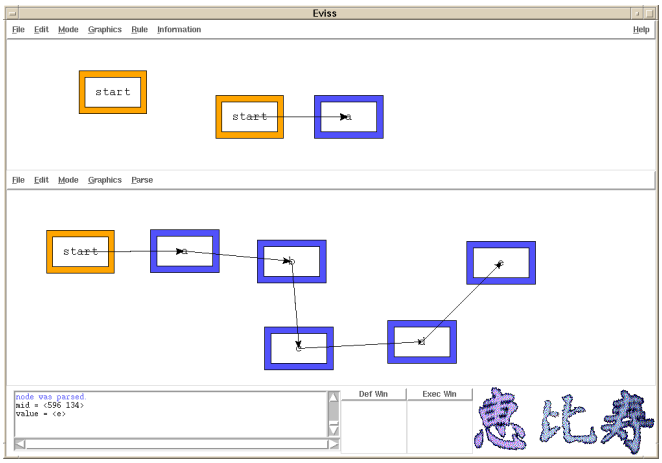


図 4.6: リストが描かれた恵比寿の実行画面

type は構成要素となる図形単語が normal 制約、 exit 制約、 not_exit 制約、 all 制約のどこで用いられてるかによって決まり、順に 0、 1、 2、 3 となる。 id は、 type で表される制約の中で何番目の構成要素であるかを表す。 name はその図形単語の属性の名前を表す。例えば normal の欄の 2 番目に書かれた構成要素の mid 属性を表すには

```
0.2.mid
```

のように書く。また「<eq,vp_close> 変数 1 変数 2」制約を利用した時の動きは変数 2 の値として変数 1 の値を代入して、変数 2 を属性として持つ図形を変わった位置に描画する。例えば、図形の構成要素として N1 と N2 があるとする。 vp_close N1.mid N2.mid は、 N1 と N2 の構成要素が解釈された後、 N2 の構成要素の属性 mid （中心の座標）の値を N1 の構成要素の属性 mid と等しくして N2 の構成要素を描画する。先に示したリスト（図 4.4の水色の部分）では vp_close 0.0.mid 0.1.mid となってるので normal （0）の欄の 0 番目の rectangle の中心座標、 normal （0）の欄の 1 番目の rectangle 中心座標がほぼ等しいという制約を表現している。また（図 4.5の水色の部分）では vp_close 1.0.mid 1.0.start となってるので exist （1）の欄の 0 番目の node の中心座標、 exist(1) の欄の 1 番目の line の始点がほぼ等しいということを表している。

4.3 インタフェース部の解析

図形エディタ、 CMG 入力部側からインタフェース部に制約を渡す時には

```
sb_constr create (-comment <comment>) <constr> 変数 1、変数 2、変数
3
                                    . . . (<strength>)
```

の形で記述する。 sb_constr は制約の操作を表している。<comment> は制約の id を持ち、<constr> で制約の種類、その後に変数を書き、<strength> で制約の強さを表す。<constr> には値を一定に保つ stay 制約, 同値を表す eq 制約, 加算を表す plus 制約, 値の大小関係を表す (gt,lt,ge,le) 制約, 乗算を表す mult 制約, そしてその他に dist 制約,scaleoffset 制約,oncircle 制約,coord 制約がある。<strength> には強い順から max, strong, medium, weak, min として指定できる。上記に示した CMG ウィンドウで作られた「vp_close 変数 1 変数 2」の形の制約は「sb_constr create eq 変数 1 変数 2」として、「lt 変数 1 変数 2」は「sb_constr create lt 変数 1 変数 2」の形で図形エディタ、 CMG 入力部側からインターフェース部分が受けとる。
変数を作るには

```
sb_var create (-comment <comment>) <type> (<value>)
```

の形で記述する。<type> には integer, double, string, point が指定でき、 value で値を与えることができる。
以上が側からインタフェース部への間、次にインタフェース部から制約解消系への間を書く。
「sb_constr create eq 変数 1 変数 2」「lt 変数 1 変数 2」という形でインタフェース部が受けとるとインタフェース部で制約解消系（SkyBlue）に制約として渡せるよう変換される。表 4.1に制約解消系 SkyBlue に値を渡す関数を呼び出す主な関数を列挙する。変数に対しては 1 から順に 1：変数を作る、 2：変数に値を与える、 3：変数の型を得る、 4：変数を削除する関数となっている。制約に対しては 5 から順に 5：一次等式制約、一次不等式制約をつくる、 6：制約解消系に制約を加える、 7：制約を削除する、 8：制約に強さを与える、 9：現在課している制約を外す関数（これは後で add_constr （制約の追加）できることが destroy_constr （制約の削除）とは違う）がある。表 4.2に制約解消系に渡される関数を示す。 1 から 1: 変数を作る、 2: 変数に値を代入する、制約に対しては 3 から 3:eq 制約、 4:plus 制約、 5:gt 制約、 6: 制約の追加、 7: 制約の強さの変更、 8: 制約をはずす、といった具合になっている。
以上の流れを eq 制約を例にまとめると CMG ウィンドウで描かれた「<eq,vp_close> 変数 1 変数 2」がインタフェース部に渡される時に「sb_constr create eq 変数 1 変数 2」の形に変換され、インタフェース部で制約の場合分けが行わる、そして今回は eq 制約なので create_equal_constraint() 関数で制約解消系に eq 制約としてわたされ

る。この恵比寿の処理の流れを図式化すると図 4.7のようになる。図形エディタ、 CMG 入力部側と制約解消系はインタフェース部のみでつながっている。

	変数操作に対する関数
1:	create_var(Tcl_Interp *interp,char *type,char *initial_value,char *comment)
2:	assign_var(Tcl_Interp *interp,char *var_id,char *new_value)
3:	get_var_type(Tcl_Interp *interp,char *var_id)
4:	destroy_var(Tcl_Interp *interp,char *var_id)
	制約操作に対する関数
5:	g_l_e_constr(Tcl_Interp *interp,int argc,char *argv[ ],int gle)
6:	add_constr(Tcl_Interp *interp,char *c_id)
7:	destroy_constr(Tcl_Interp *interp,char *c_id)
8:	constr_set_strength(Tcl_Interp *interp,char *c_id,char *strength_string)
9:	remove_constr(Tcl_Interp *interp,char *c_id)

表 4.1: インターフェース部の関数

	変数操作に対する関数
1:	create_variable(void *val,long id,int type,char *comment)
2:	set_variable(Variable var,void *val)
	制約操作に対する関数
3:	create_equal_constraint(Variable a,Variable b, Strength strength,long id,int cast)
4:	create_add_constraint(Variable a,Variable b,Variable sum, Strength strength,long id,int cast)
5:	create_gt_constraint(Variable a,Variable b, Strength strength,long id,int cast)
6:	add_constraint(cn)
7:	change_constraint_strength(Constraint cn,Strength new_strength)
8:	remove_constraint(cn)

表 4.2: 制約解消系 SkyBlue への関数

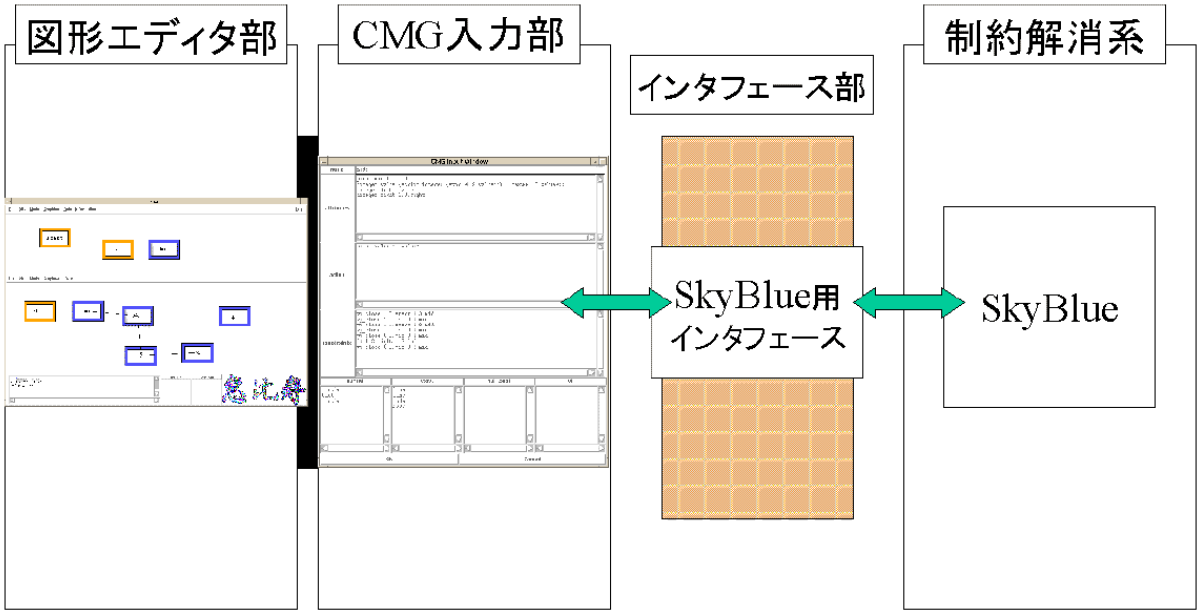


図 4.7: 制約の流れ

第 5 章

HiRise,Cassowary の恵比寿システムへの実装

5.1 実装方法

現在の制約解消系 SkyBlue から、制約解消系 HiRise,Cassowary を使用できるようにする方法を述べる。ビジュアルシステム恵比寿では図形エディタ、 CMG 入力部と制約解消系の間の制約の受渡しはインタフェース部を通しているのみである。つまり制約解消系との結合部分はインタフェース部のみということである。また恵比寿本体と制約解消系本体との値のやりとりは、インタフェース部の中で変数に付けられた id をやりとりすることで、制約の作成や値の問い合わせ、その要求に対する値の計算などを行っている。以上のことから制約解消系に対応できるインタフェースさえ作成すれば、恵比寿本体、制約解消系本体はそのままで高速な制約解消が行えるビジュアルシステム恵比寿を実装することができる。

5.2 HiRise のサンプルプログラム

制約解消系 HiRise,Cassowary の x+y=z の plus 制約を表してたサンプルプログラムを以下に示す。（原論文の著者らが配布している C++ 版の実装を利用する。）

5.2.1 HiRise のサンプルプログラム

```
1:      HRSolver solver;                //HiRise 制約解消系
2:
3:      HRVar x(3);                     // 変数（x の初期値は 3）
4:      HRVar y(4);                     // 変数（y の初期値は 4）
5:      HRVar z(7);                     // 変数（z の初期値は 7）
```

```
6:
7:      HRStay xcstay(2,x);             //x に関する強さ 2 の stay 制約
8:      solver.add(xcstay);             // 制約を追加
9:      HRStay ycstay(2,y);             //y に関する強さ 2 の stay 制約
10:     solver.add(ycstay);             // 制約を追加
11:     HRStay zcstay(2,z);             //z に関する強さ 2 の stay 制約
12:     solver.add(zcstay);             // 制約を追加
13:
14:     HREdit ycedit(1, y);            //y に関する強さ 1 の edit 制約
15:     solver.add(ycedit);             // 制約を追加
16:
17:     printf("x = %f, y = %f, z = %f\n", *x, *y, *z); //
18:
19:     HRLinear clinear(0);            //1 次等式 x+y=z 制約
20:     clinear.sum(x,y,z);
21:     solver.add(clinear);            // 制約を追加
22:
23:     solver.plan();                  // プランニング
24:
25:     ycedit.set(5);                  //edit によって y に 5 を代入
26:     solver.execute();               // 実行
27:     printf("x = %f, y = %f, z = %f\n", *x, *y, *z); //
28:
29:     ycedit.set(1);                  //edit によって y に 1 を代入
30:     solver.execute();               // 再実行
31:     printf("x = %f, y = %f, z = %f\n", *x, *y, *z); //
32:
33:     x = 3.000000, y = 4.000000, z = 7.000000
34:     x = 3.000000, y = 5.000000, z = 8.000000
35:     x = 3.000000, y = 1.000000, z = 4.000000
```

1 行目から 31 行目までは HiRise のサンプルプログラム。 33 行目以降は実行結果を表している。まず 3 行目から 5 行目で初期値 3、 4、 7 を持った変数 x、 y、 z が作られ、 stay 制約として制約解消系に加えられる。 14、 15 行目で変数 y に edit 制約が与えられる。 17 行目で変数 x、 y、 z の値を出力させると確かに x = 3、 y = 4、 z = 7 となっている (33 行目)。 19、 20 行目で強さ 0 を持った 1 次等式 x+y=z という plus 制約が作られ 21 行目で制約の追加をする。 23 行目でこの作られた制約系のプランニングをする。次に変数 y の値を変更して動作を確かめる。 25 行目で変数 y に 5 を代入する。その結果、制約 x+y=z を保つために変数 x の値が 2 になるか、z の値が 8 にならなければならない。その結果を出力させると（34 行目）変数 z の値が 7 から 8 となっていて期待通りの結果が出力されている。同様にして、 29 行目で変数 y に 1 を代入して、その結果を出力させると（35 行目）変数 z の値が 4 となり制約 x+y=z が保たれている。
制約の優先度としては強さの値が小さいほど優先される。（強さ 0 は必須制約、強さ 1、 2、 3 と順に優先度が低くなる）

5.2.2 Cassowary のサンプルプログラム

```
1:   ClSimplexSolver solver;          //Cassowary 制約解消系
2:
3:   ClVariable x(3);                 // 変数（x の初期値は 3）
4:   ClVariable y(4);                 // 変数（y の初期値は 4）
5:   ClVariable z(7);                 // 変数（z の初期値は 4）
6:
7:   cout << " x = " << x.Value()     // 値の出力
8:        << " y = " << y.Value()
9:        << " z = " << z.Value() << endl;
10:
11:  solver.AddStay(x);               //x に関する stay 制約
12:  solver.AddStay(y);               //y に関する stay 制約
13:  solver.AddStay(z);               //z に関する stay 制約
14:  ClLinearEquation plus(x + y,z+0.0); //1 次等式 x+y=z 制約
15:  solver.AddConstraint(plus);          // 制約を追加
16:
```

```
17:  solver.AddEditVar(y)
18:    .BeginEdit()
19:    .SuggestValue(y,5)            //edit によって y に 5 を代入
20:    .Resolve();                   // 実行
21:
22:  cout << " x = " << x.Value()    // 値の出力
23:       << " y = " << y.Value()
24:       << " z = " << z.Value() << endl;
25:
26:  solver
27:    .SuggestValue(y,1)            //edit によって x に 1 を代入
28:    .EndEdit();                   // 再実行
29:
30:  cout << " x = " << x.Value()    // 値の出力
31:       << " y = " << y.Value()
32:       << " z = " << z.Value() << endl;
33:
34:  x = 3 y = 4 z = 7
35:  x = 2 y = 5 z = 7
36:  x = 6 y = 1 z = 7
```

1 行目から 32 行目までは Cassowary のサンプルプログラム。 34 行目以降は実行結果を表している。まず 3 行目から 5 行目で初期値 3、 4、 7 を持った変数 x、 y、 z が作られる。 7 行目から 9 行目で変数 x、 y、 z の値を出力させると確かに x = 3、 y = 4、 z = 7 となっている。 11 行目から 13 行目でその変数 x、 y、 z が変数の値を一定にするという stay 制約として与えられる。 14 行目で 1 次等式 x+y=z の plus 制約が作られ 15 行目で制約の追加が行われる。次に変数 y の値を変更して動作を確かめる。 17 行目で変数の値を繰返し更新するための edit 制約を与えられ変数 y の値が変更できるようになった。そして、 19 行目で変数 y に 5 を代入する。その結果、制約 x+y=z を保つために変数 x の値が 2 になるか、 z の値が 8 にならなければならない。その結果を出力させると（35 行目）変数 x の値が 3 から 2 となっていて期待通りの結果が出力されている。同様にして、 26 行目から 29 行目で変数 x に 1 を代入して、その結果を出力させると（36 行目）変数 x の値がから 6 となり制約 x+y=z が

保たれていることがわかる。
この制約解消系は同じ強さであれば後に制約として与えられた方が優先される。

5.3 実装と結果

実装後の処理の流れを図に表すと図 5.1のようになる。制約の流れは制約解消系 SkyBlue を使用していた時と同様に、図形エディタ部、 CMG 入力部で生成された制約がインタフェース部を通して制約解消系に渡されている。そのためユーザーは従来の恵比寿システムと全く同じ操作方法で高速な制約解消が行えるビジュアルシステム恵比寿を利用することができる。

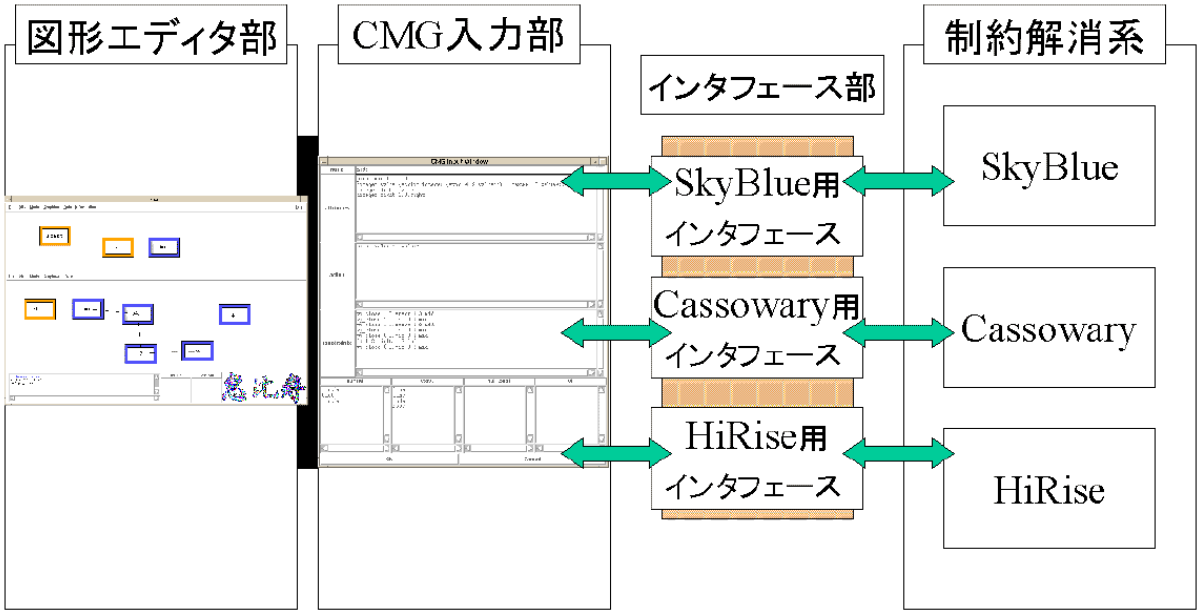


図 5.1: 実装後の制約の流れ

第 6 章

結論

既存の恵比寿では図形間の関係を表す制約が多くなってくると実時間で扱えなくなってくるという問題を制約を解く制約解消系を SkyBlue から HiRise,Cassowary に変更することで対応し、効率化を試みた。本論文では制約解消の高速化のために制約解消系の変更で対応したが、現在の恵比寿は、 CMG ウィンドウで作られた制約を制約解消系に与える時にも時間を要しているため制約の与え方も検討すればより恵比寿の操作性の向上にもなると考えられる。

謝辞

本研究を進めるにあたり終始丁寧に指導して下さいました田中二郎教授、助手である志築文太郎先生に心より感謝いたします。田中研究室のみなさんからはゼミ等を通して貴重なコメントを頂きました。また恵比寿グループのリーダーである藤山健一郎さん、飯塚 和久さんにはシステムの研究・開発にあたり有益な助言などをいただきました。ここに感謝の意を表します。

参考文献

[1] 馬場昭宏: Spatial Parser Generator を持ったビジュアルシステム. 筑波大学大学院博士課程工学研科修士論文, 1998.

[2] 馬場昭宏, 田中二郎: Spatial Parser Generator を持ったビジュアルシステム. 情報処理学会論文誌 Vol.39, No.5, 1998.

[3] 馬場昭宏, 田中二郎: Spatial Parser Generator の Tcl/Tk を用いた実装. 情報処理学会シンポジウムインタラクション '97 論文集, pages 71-78, 1997.

[4] 馬場昭宏, 田中二郎: GUI を記述するためのビジュアル言語. インタラクティブソフトウェア V, 日本ソフトウェア学会 WISS'98, pages 135-140, 近代科学社, 1997.

[5] Marriot, K: Constraint Multiset Grammars. In *Proceedings of the 1994 IEEE Symposium on Visual Languages*, pp. 245-252, 1994.

[6] Borning, A., Duisberg, B., Freeman-Benson, B., Kramer, A., and Woolf, M: Constraint Hierarchies, in *ACM OOPSLA,* Oct. 1987, pages 48-60

[7] Borning, A., Anderson, R. and Freeman-Benson, B. : Indigo: A Local Propagation Algorithm for Inequality Constraints, *Proc. ACM UIST*, 1996, pages 129-136

[8] Freeman-Benson, B., Wilson, M. and Borning, A. : DeltaStar: A General Algorithm for Incremental Satisfaction of Constraint Hierarchies, *Proc.IEEE IPCCC*, 1992, pages 561-568

[9] Maloney, J.H., Borning, A. and Freeman-Benson, B. : Constraint Technology for UserInterface Construction in ThingLab II, *Proc. ACM OOPSLA*, 1989, pages 381-388

[10] Sannella, M: The SkyBlue Constraint Solver, UW tech report 92-07-02.

[11] Sannella, M: The Skyblue Constraint Solver and Its Applicat ions, Vijay Saraswat and Pascalvan Hentenryck, editors, Proceedings of the 1993 Workshop on Principles and Practice of Constraint Programming, MIT Press, 1995 , pages 385-406.

[12] Sannella, M: SkyBlue : A Multi-Way Local Propagation Constrai nt Solver for User Interface Construction. In *Proceedings of the 1994 ACM Symposium on User Interface Software and Technology,* pages 137-146, 1994

[13] Marriot, K: Constraint Multiset Grammars. In *Proceedings of the 1994 IEEE Symposium on Visual Languages,* pages 118-125, 1994

[14] http://www.cs.washington.edu/research/constraints/cassowary/

[15] Badros G.J. and Borning, A : The cassowary linear arithmetic c onstraint soloving algorithm: Interface and Implementation. Tecnical Report UW- CSE-98-06-04, University of Washington, Seattle, Washington, June 1998.

[16] Badros G.J : Constraints in Interactive Graphical Applications, Ph.D. General Examination

[17] http://www.csse.monash.edu.au/projects/qoca/

[18] 細部博史: ユーザーインターフェースにおける制約解消法の研究動向, コンピュータソフトウェア, 第 17 巻, 第 6 号, 日本ソフトウェア科学会, 岩波書店, 2000 年 11 月, 73-85 頁.

[19] 細部博史, 松岡 聡, 米澤明憲: HiRise:GUI 構築のためのインクリメンタルな制約解消系, コンピュータソフトウェア, 第 16 巻, 第 6 号, 日本ソフトウェア科学会, 岩波書店, 1999 年 11 月, 33-45 頁.

[20] 松岡 聡, 細部博史: 階層連立 1 次方程式のための効率的解消系の開発, 委託研究, 日本情報処理開発協会先端情報技術研究所, 1997.