筑波大学大学院修士課程

理工学研究科修士論文

ノードを用いた Web 検索インタフェースの研究

奥 村 穂 高

平 成 1 3 年 3 月

筑波大学大学院修士課程

理工学研究科修士論文

ノードを用いた Web 検索インタフェースの研究

著者氏名　　奥村 穂高

主任指導教官　機能工学系 山下 義行

目次

要旨

ノードを操作することで対話的に条件の指定を行う Web 検索インタフェースの設計を行った。このインタフェースでは、ユーザはポインティングデバイスを利用してキーワードを表現するノードを操作しキーワードの取捨選択やクエリーの構成をおこないながら次第に Web ページを絞り込んで行くことを可能にする。検索結果は通常の検索エンジンが URL を提示するのに対して、 Web ページより抽出したキーワード群によって表示する。これにより、ユーザのキーワード選択の負担を軽減する。

また、設計に基づいて Web 検索システム “Carta” の実装を行なった。この実装では、膨大なデータをもつ外部の検索エンジンを利用して URL リストを作成する段階と、該当する Web ページよりキーワードを抽出する段階の 2 段階に検索をわけて行うことでユーザに対するキーワード群の提示を実現した。

本論文では、第 1 章で総論と背景を、第 2 章で今回設計したインタフェースについて述べる。第 3 章では第 2 章で述べたインタフェースの実装と、インタフェースと連動して検索を行うためのエンジンの実装について説明する。

第 1 章

Web 検索の現状と視覚的なキーワード検索

1.1 Web 検索エンジンの現状

ここ数年、インターネットの利用の増加に伴い、 Web ページなどの情報も急激に増加している。このような莫大な情報の中から、いち早く目的とする情報を探し出すために、検索エンジンが必要不可欠になってきている [1]。

検索エンジンとは、情報を整理分類した上、情報を検索し提供するためのインタフェースをもつシステムの総称である。 Web 検索エンジンは大きく分けてディレクトリ型とロボット型の 2 種類のサービスがあるが、一部のものを除けばほとんどがロボット型のものである [2, 3, 4, 5]。

ロボット型 Web 検索エンジンは高性能なマシンと高度なソフトウェア技術によって、世界中に広がる膨大な WWW 上の情報を高速に検索できるサービスである。このタイプの検索エンジンは、そのサービスを提供するために、プログラムを用いて情報の収集からインデックスの作成までを自動的に行い、膨大なページ情報を収集蓄積している。また、与えられたクエリーに対する検索結果をすばやくユーザに提供している。

Web 検索エンジンのインタフェースは現在、キーワードを用いたものが主流となっている。これらの Web 検索エンジンでは、 入力をテキストをキーボードやコピー&ペーストで、出力を URL とその付加情報で扱うものがほとんどである。

ロボット型検索エンジンは全ての処理をコンピュータによって行っているため、情報の網羅性、情報の新鮮さ、検索の速さなどの点で優れている。その反面、情報の質や分野を整理できないという欠点も持っている。このため、目的のページに合ったキーワードが思いつかない場合、検索は困難なものとなる。的確ではないキーワードを選定してしまうと、目的のものとは異なるページが多く引っかかってしまう。また、絞り込むための条件として更新時刻などの付加情報を利用するものもあるが、それらは操作が複雑になるため、あまり利用されていない [5]。

これらの問題により、よりすばやく正確なページを提供するために、他ページからのリンク状況や人の手によるリストとの複合的利用、インデックスの効率的な活用な

ど、 データベースからのアプローチを元とした研究が多くされている [6, 7]。また、要約を自動的に作成しユーザに提供することで、 Web ページの内容を容易に判断できるような自然言語処理からのアプローチもされている [8]。

しかし、既存のシステムのように入出力をテキストで扱う限り、１つのキーワードとしての操作が手間がかかり、また、キーワード間の関係の記述が複雑なため、複数語での検索を避ける傾向がある。

前出の研究に対し、我々は検索を対話的に行なうことにより、利用者の意図に沿った検索が可能になると考えている。データベースではなく、ユーザにとって負担の少ない、かつ、ユーザの目的に添った Web ページを絞り込むためのサポートができるようなインタフェースについて着目した。

クエリーにあたるキーワードの組み合わせをノードによって表現するインタフェースを設計し、実際に利用できるシステムとして実装した。このインタフェースでは、キーワード間の関係を視覚的でわかりやすくし、また、ポインティングデバイスによる直接操作によってキーワードであるノードの操作を直感的で容易なものにする。また、検索結果として得られた Web ページよりキーワードを抽出し、複数のページを表示するかわりに抽出したキーワードを提示することによって絞り込みの過程を対話的なものにする。

第 2 章

ノードによる対話的インタフェースの設計

我々は ノードを使った対話的 Web キーワード検索システム を設計した [9, 10]。

キーワードをノードとして扱うために、それに応じたインタフェースが必要となる。本章では、ノードを使った対話的 Web キーワード検索インタフェースの基本的原理と、その要素技術について述べる。

2.1 グラフ表現の活用

情報検索では、キーワードをクエリーに利用するのが一般的である。 and 等の関係の入力は、メニューで選択したり、演算子を用いて表記を行う。しかし、テキストによる関係表現は、認識するのに多少の時間を要する。

本インタフェースでは、クエリー内での関係表現にグラフ表現を用いた。通常のグラフ表現では、ノード間をエッジで結ぶものだが、今回、エッジの変わりに、重ねあわせ表現を用いた。これにより、ノード間を結ぶエッジが交差することによるグラフの視認性の低下を避けることができる。また、ノードを直接操作によって関係を表現できるため、直感的にクエリーが作成でき、かつ、エッジを作成するコストを削減できる。

2.2 ノードによる Web 検索インタフェース

従来の検索システムでは検索結果は URL を中心として表示されるものが主流である。しかし、 URL はそれ自体が指し示す Web ページの内容とは関係がないことが多く、また、ユーザにとっても直感的ではないため負担となる [11]。

本インタフェースでは、ユーザの入出力をテキストではなくノード化したキーワードを利用して行う。クエリーはノードを組み合わせることによって作成する。ノードを利用してクエリーを生成するシステムにすることにより、 複数のキーワードの組み合わせによるクエリーを直観的に作成できる。

インタフェースの画面構成は、左上部のクエリーの作成を行うクエリー作成部、左下部の検索結果を表示する検索結果部と右部のキーワードの履歴を掲示する履歴部、

最上部にある初期条件などのテキストでのキーワード入力をおこなう入力部の４つの部分によって構成される（図 2.1）。

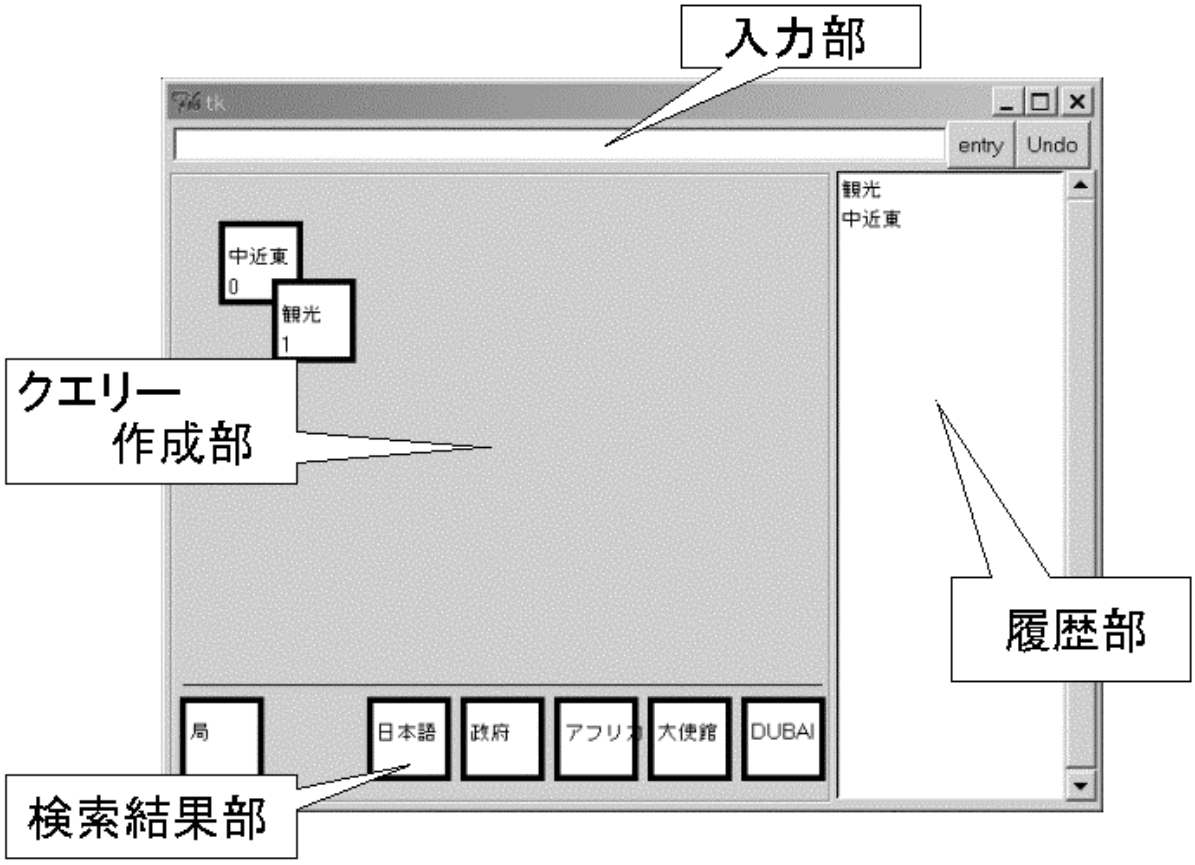

図 2.1: インタフェースの画面構成

クエリー作成部および検索結果部にあるノードはドラッグ＆ドロップにより操作する。このノードをクエリー作成部内で組み合わせることによって、検索目的となるクエリーを作成する。また、不要なノードはクエリー作成部外でドロップすることにより削除される。

検索の結果は URL ではなく目的の Web ページに関係があると思われるキーワード群で提示される。提示されたキーワード群はクエリーに利用することができる。関係のあるキーワードが提示されるため、キーワード選択にかかる負担を軽減することができる。また、検索結果内よりクエリーにもっとも適しているとされる Web ページも同時に提示される。これにより、ユーザは URL を気にすることなく対話的に検索を行うことができる。

具体的には以下に示す手順を踏むことによって検索を行う。

1. 検索初期条件となるキーワードを入力する (図 2.2)。このキーワードは入力部よ

りキーボードによってテキストで入力するか、履歴部にあるキーワードから選択する。入力後、初期条件となるキーワードがノードとしてクエリー作成部に表示される。それと同時に検索が実行され、その検索結果であるキーワード群と、 Web ページが表示される (図 2.3)。

2. 検索結果として画面に表示されるキーワード群の種類、重要なキーワードの有無、提示された Web ページの内容などによって検索を続けるかを判断する。

3. 検索を続ける場合、クエリーのキーワードの追加や削除、重ねあわせ操作による条件の変更を行う。追加は、検索結果として表示されたキーワード群から関係するキーワードをドラッグ＆ドロップによってクエリーに組み込む (図 2.4)。もしくは、初期条件と同様にテキストでの入力や履歴部からの選択でも可能である。削除の場合は不必要なキーワードをドラッグ＆ドロップにより画面外へ移動させることによって削除する。

4. 目的以外の検索結果に陥った場合、逆操作によって任意の状態まで戻り、再度検索を続ける。

以下では、本システムの特徴となる要素について説明する。

図 2.2: 検索のための操作 (a)

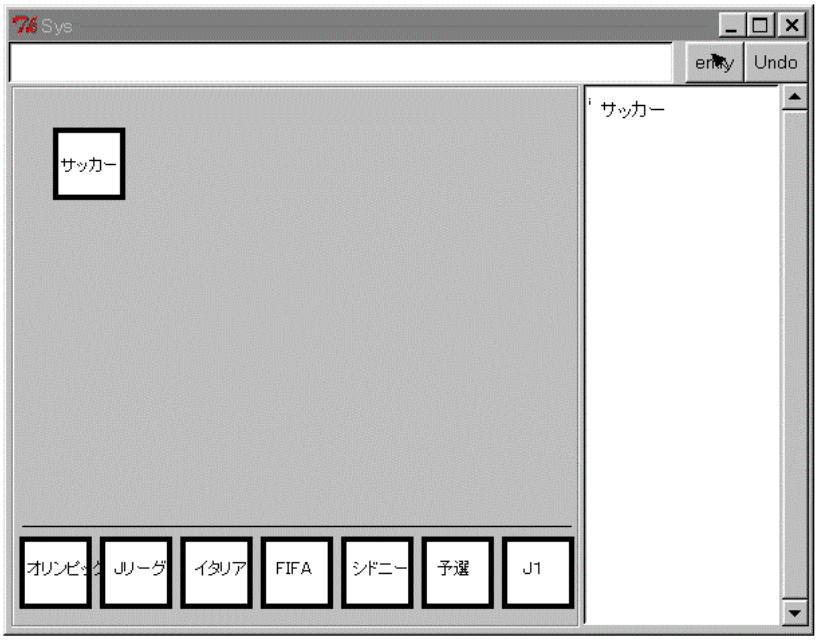

図 2.3: 検索のための操作 (b)

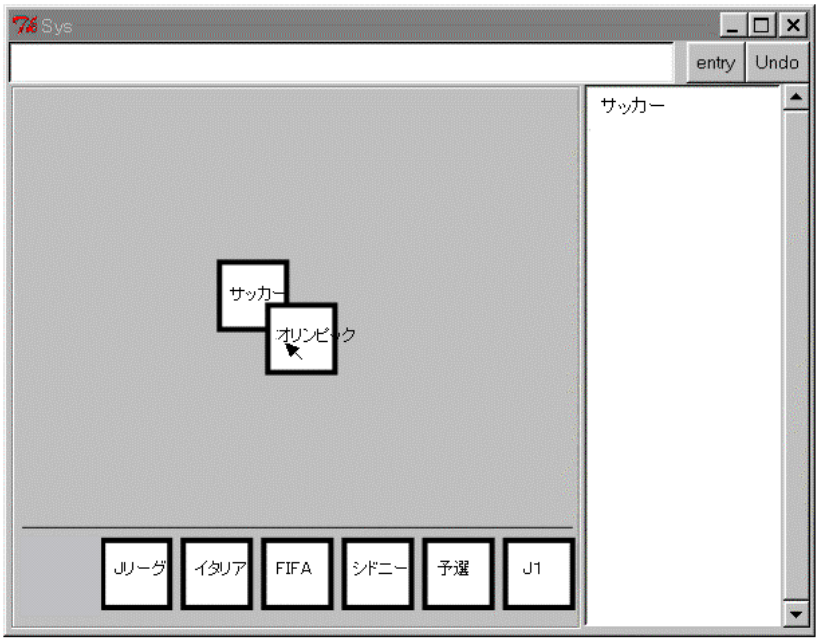

図 2.4: 検索のための操作 (c)

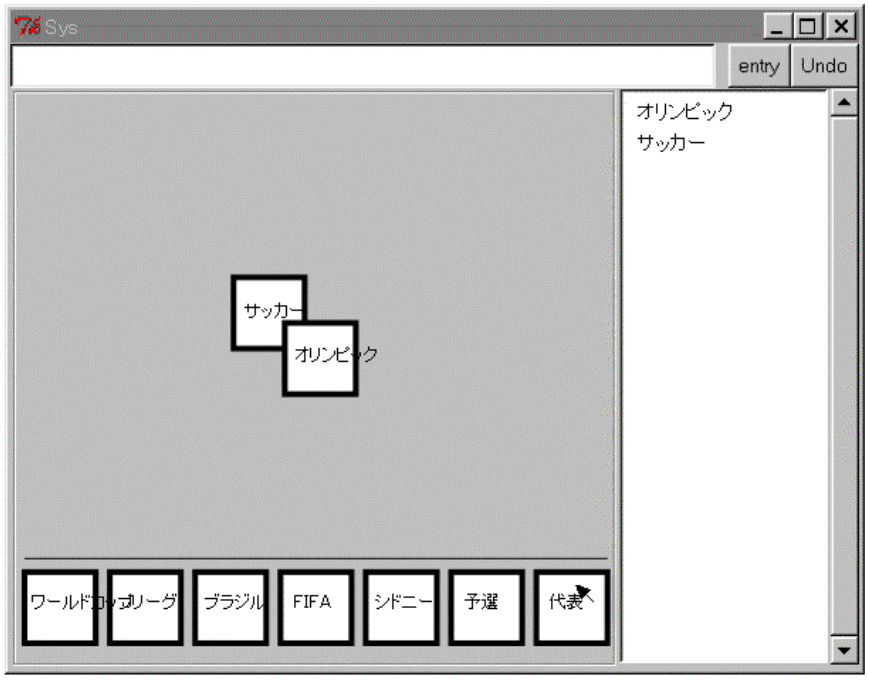

図 2.5: 検索のための操作 (d)

2.3 重ねあわせ表現

「ノードの重ねあわせ」とは、２つのノードを重ねることによって、その間に意味を持たせるものである [12]。本システムではノードをクエリーに利用するために重ねあわせ表現を用いた。ノードが重ねあわされた場合 and として、重ねあわされていない場合は or として表現する。図 2.6では（ オリンピック and サッカー ） or ワールドカップ の組み合わせを示している。

２次元的に表現されたクエリーをそのままの形で検索エンジンなどに引渡しを行なうのは困難である。このため、ビジュアルプログラミングにおける技術をもちいて、図的表現をテキストによる表現に変換して活用できるようにする [13][14]。

2.4 結果のキーワード群の利用

インタフェースの各部を操作し、ノードの追加や削除、重ねあわせの発生などクエリーの図に変化がおこれば、そのたびに検索を開始する。

検索結果は Web ページより抽出されたキーワード群として検索結果部に表示される (図 2.1下側)。表示された検索結果のキーワード群は直接操作でき、追加するキーワードとして利用できる。

また、結果のキーワード群は表示されている Web ページ以外のキーワードも含む

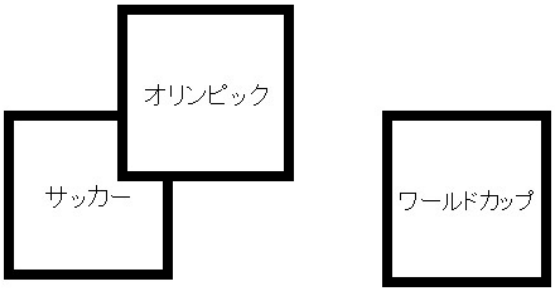

図 2.6: 重ねあわせ表現による例

ため、検索の状況を判断する材料にも利用することができる。ユーザは表示されるキーワード群の内容によって絞込みを続行するかどうかを判断する。もし問題があると判断した場合、逆操作などを用いてクエリーの修正を行う。

2.5 過去履歴の表示

履歴部では、今までクエリーに利用してきたキーワードが過去履歴として表示される。履歴にあげられているキーワードをポインティングデバイスでクリックすると、そのキーワードが新たなノードとしてクエリー作成部に表示される。表示されたノードは、現在検索中のクエリーのノードとして利用できる。

検索部のキーワードリストは、クエリー作成部でのキーワードの利用状況に基づいて並べ替えられる。新しく利用されたキーワードほど上部に記述する。

2.6 undo 機能

本インタフェースでは、逆操作 (undo) をおこなう機能を持たせている。これにより目的以外の検索結果に陥った場合、操作を遡ることによって検索の途中段階より再度検索を行なうことが可能になる [15]。

入力部右にある undo ボタンを 1 回押すことにより、クエリー作成部はドラッグ&ドロップ作業を 1 回遡ることになる。 undo ボタンを繰り返し押すことにより 1 つ目のノードを作成した状態まで遡ることが出来る。検索結果はクエリーに基づいて逐次

検索を行なうため、基本的に同様なキーワードが表示される。また、画面右側の履歴表示は変化しない。

2.7 関連研究

本システム以外にも、 Web や データベースの検索を視覚的に扱うシステムが存在する。それらの研究について述べる。

2.7.1 TITAN/V

TITAN/V[16] は検索エンジン TITAN[17] の入力から結果出力後の情報の選択までをインタラクティブかつビジュアルに画面上で行うためのインタフェースである。

入力時はテキストで、キーワードを入力することにより一度検索が行なわれ、ベン図インタフェース [18] によって 該当する Web ページのヒット数などの情報とともに、キーワードに相当する円が表示される。この円を選択し重ね合わせ、分離を繰り返し最終的にマウスをクリックすることで検索式が作成され、再度検索を開始する。

出力時は、 3 次元 GUI を用いて結果が表示される。 3 次元 GUI では、ヒット率、テキストの量、リンク数の 3 要素を元に Web ページが表示される。その中から、目的の Web ページを探し出すことになる。

このインタフェースでは、入力、出力の 2 種類のインタフェースを使い分けており、またベン図インタフェースでは基本的にテキストによるキーワード入力である。このため、単一インタフェースで初期条件以外ではキーボードを使わなくても検索可能な Carta と異なっている。

2.7.2 DualNAVI

DualNAVI[19] は ノードを用いた情報検索システムである。このシステムは特徴語グラフというもので検索対象の内容を表示する。そのために、サーバの持つデータに対して事前に各語の全体の出現頻度を算出しておき、検索結果の語の出現頻度と比較することで、特徴語を抽出している。このため、かなりサーバ依存のところがあり、サーバ自身も事前に大規模な演算を必要とする。また、他の検索エンジンのデータを利用するメタ検索エンジンとするには不向きな機構である。

2.7.3 ACCENT

ACCENT [20, 21] は文章群から抽出された単語の連想関係をノードを用いて可視化するシステムである。検索結果の表示する際に、単語間の関係度計算やレイアウトに対してスコープを制御し結果情報の分析の目的や視点を反映させた連想マップというものを利用している。これにより、検索情報の分析作業の効率化を図っている。

第 3 章

Web 検索システム Carta の実装

本章では、今回作成したノードによる Web 検索システム Carta の実装についてのべる。

3.1 システム構成

図 3.2に今回作成した ノードによる Web 検索システム Carta の概観を示す。ユーザが操作するのは、インタフェース部である Carta のみであり、ここを操作することにより、その時点で最適であると思われる Web ページと抽出されたキーワード群が表示される。本システムは ユーザインタフェースを管理するインタフェース部 Carta と キーワード抽出等を行うための検索部 Dealer の大きく 2 つの部分に分かれている（図 3.1）。

Carta では、ノードの操作や図形で書き表されたクエリーの解析、検索結果であるキーワードのユーザへの提示などを行う。

検索部である Dealer の中は 3 つの部分に分けられる。 Carta より与えられたキーワードに基づき URL リストを作成する Qpin、収集した Web ページを形態素解析器 Chasen（茶筅）[22] に渡す作業を行なう Chaseav（茶渋）、そしてそれらの作業を統括する Storm で構成している。

3.2 実装

各部は個別のプロセスとして実装し、ソケット通信によってプロセス間のデータ交換を実現している。本システムではテキスト処理やソケット通信、プロセス（スレッド）分割による並列処理を多用するため、実装言語として Ruby/Tk を用いている [23][24]。

以下に各部の動作と特徴について述べる。

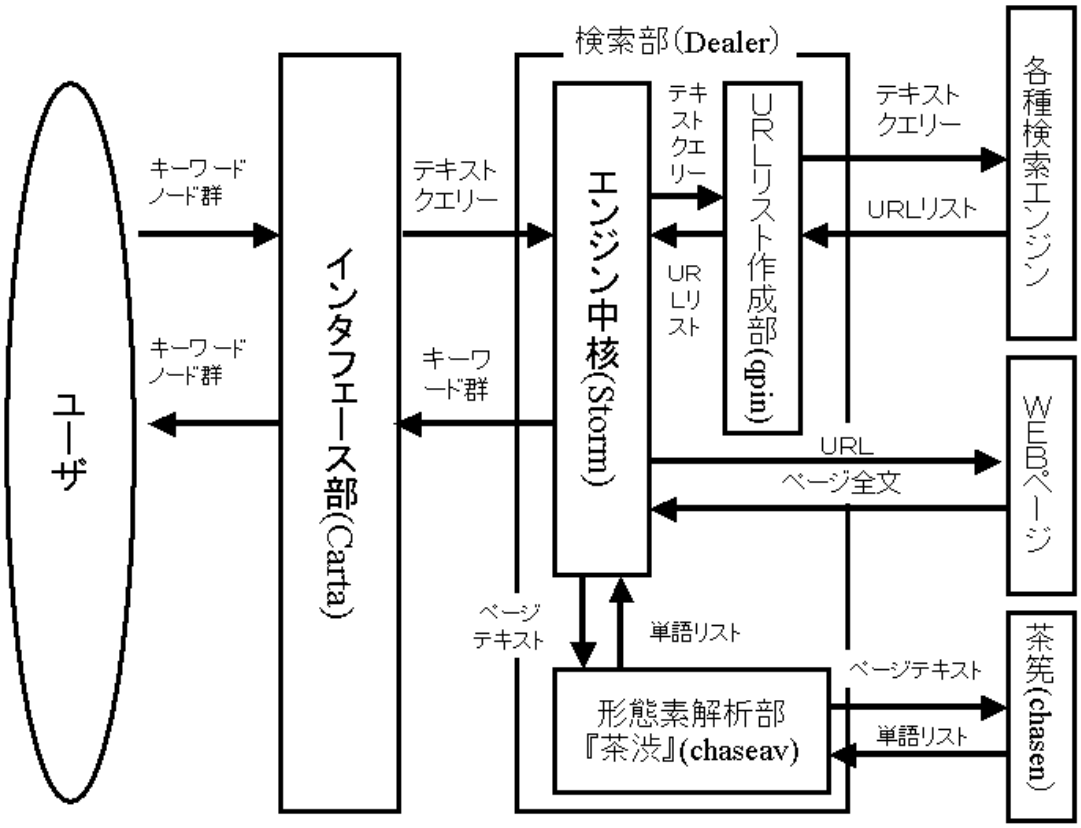

図 3.1: システムの構成

3.2.1 インターフェース部 Carta

Carta は前章で述べた、ノードの重ね合わせ表現によるクエリーの作成やキーワードの表示機能を元に設計した 検索インタフェースである。 Carta では、 GUI の制御、ノードの管理・制御、キーワードの履歴管理、検索部である Dealer との交信を行っている。ユーザはこの Carta 以外には触れることなく検索を行うことができる。

ノードの管理・制御

Carta 内部では、ノードはクエリーと結果の 2 種類の配列とドラックされているノードで管理されている。 1 つのノードについて、X／Y座標、キーワード名のデータを持っている。

クエリーノード配列では、最後に操作されたノードほど配列の先頭に配置されている。 また、後に操作を受けたノードほど前面に表示するため、配列の先頭より表示を行なっている。後に操作を受けたノードほど前面に表示することにより、重ねあわせの際、必ず最前面にドラック中のノードを配置することとなる。

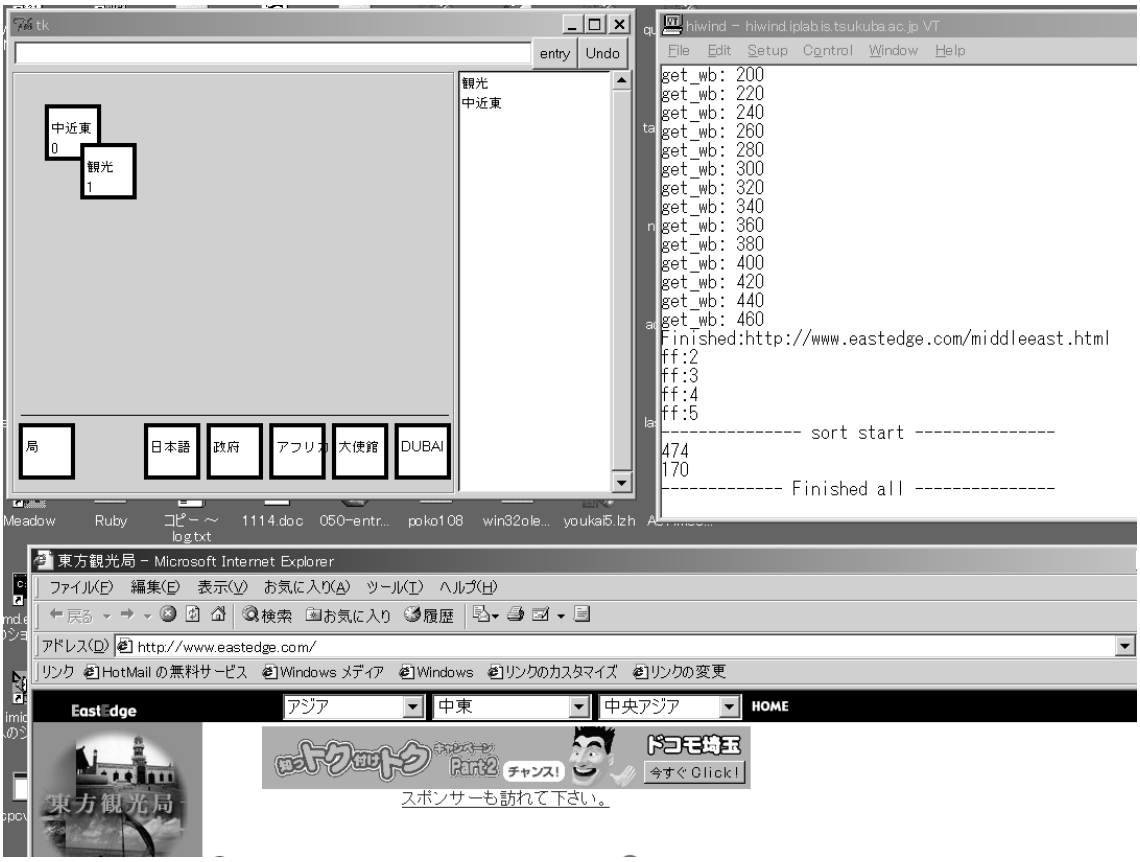

図 3.2: ノードによる Web 検索システム Carta

ノード表現の変換

ノードによって表現された論理式は、そのままでは外部の検索エンジン等にデータとして渡すことができないため検索エンジンが利用可能な形式に変換する必要がある。

本システムでは、インタフェース部である Carta において、図形で示されたクエリーをテキストに利用可能な中間表現に変換し、検索部に引き渡す。重ねあわせ図形を変換する際にはビジュアルプログラミングシステムの手法を用いている [13, 14]。

本システムでは、次の手順で図的なクエリーをテキストによる表現に変換している。

1. 最上面のノードから順に、下に重ねあわされているノードについてのハッシュを作成する。

2. 同様に最背面のノードから順に、上に重ねあわされているノードについてのハッシュを作成する。

3. 作成したハッシュに基づき、各ノードにおいて重ねあわさっているノードの集合を作る。

4. 同じ要素から構成されているの集合を取り除く。

この変換はノードの追加、削除、移動がおこなわれ、クエリーの内容に変化があるたびに行なう。

3.2.2 検索部 Dealer

本システムはメタ検索エンジンである。メタ検索エンジンとすることで、本システム自身が独自に扱うデータを小規模におさえることができる上、すでに各検索エンジンが蓄積している膨大な情報を利用することができる。

Dealer は機能の違いにより 3 つのプロセスに分かれている。それぞれのプロセスでは、同時に複数のデータ解析や外部への問い合わせを行うため、 Ruby の特徴である Thread 機能を活用することで並列処理を実現している。

現在運用されている検索エンジンのほとんどは、検索結果としてキーワードを返す機能をもたない。このため、検索部 Dealer ではキーワード群を求めるために、関係のある URL リストを検索する段階と、該当する Web ページよりキーワードを抽出する段階の 2 つに分けて検索を行う手法を取った。

以下では、各ポイントにおいてのそれぞれのプロセスの動作について述べる。

URL リストの作成

検索の 1 段階目では and、 or 表現で記述されたテキストによるクエリーより URL リストを導き出す従来型の検索を行う。その際、他の検索エンジンの結果を利用する、メタ検索エンジンとしている。メタ検索エンジンとは、すでに運用されている検索エンジンのいくつかを利用して、それらの検索エンジンに適合するようにユーザからの情報要求を変換し、各検索エンジンからの検索結果を重複などを排除してマージして出力する検索エンジンである [1]。

URL リストは Dealer 内の QPin によって作成されている。本システムでは Google[7] と infoseek[27] の 2 つの検索エンジンを活用し、それらの検索結果から抽出される URL を単純マージし、さらに、重複する URL を排除することで URL リストを作成している。

本システムが利用している検索エンジンでは CGI の形で検索サービスを提供している。そのため、提供されている GUI を使わず直接 URL を渡すことで、検索結果を得ることができる。 QPin では Storm より渡されたキーワードリストを URL の形に直し、外部の検索エンジンにわたす。たとえば、 EUC 形式で書かれた「検索」という文字列は、「%B8%A1%BA%F7」という文字列に変換し、検索を行うページの URL に付加される形で渡される。

検索エンジンからは HTML 形式の Web ページとして検索結果が返ってくる。検索結果として渡された Web ページより、ページの構造や HTML タグの配置より結果に当たる URL のみを抽出してリストを作成し、 Storm にわたしている。

Web ページの全文検索

検索の 2 段階目では 1 段階目の検索結果として返ってきた URL リストより、該当する Web ページを直接取得して個別に形態素解析をかけることによりキーワードを抽出する。形態素解析には『茶筅』[22] を用いている。

茶筅では、渡されたテキストを形態素で切り、それぞれの形態素について、品詞名や活用形、原型などの情報をリストとして返してくる。茶筅自体を Ruby より直接活用するのは困難なため、『茶渋』(chaseav) では標準入力より茶筅を利用している。

茶筅より出力されたリストはそのままの形で、 Storm に渡される。 Storm では、その中より名詞と未知語（ASCII 文字でかかれた語や茶筅の中には入っていない名詞がほとんど）を取り出して、配列を作る。複数のページから帰ってきたキーワードリストより、キーワードの出現回数を数え、それにもとずいて優先順位を決定する。優先順位の上位のキーワードは結果のキーワード群としてインタフェース部である Carta に渡される。

3.3 検索の実時間処理・ユーザへの対応

システムを実装する際、以下の処理に時間がかかるため、ユーザとの対話性を損なう恐れが出てきた。

1. 複数の Web ページを逐次処理する時
2. URL リストにあるすべての Web ページを収得する時
3. chasen での形態素解析を行う時
4. 出現頻度を数える際、日本語（2 バイト文字）同士の比較を行う時。

これらの問題を解決するため、ユーザの待ち時間を減らすためいくつかの改良を行っている。

URL リストを作成や URL リストにある Web ページよりキーワードを抽出する際、逐次処理ではネットワーク通信によるタイムロスなどで非常に時間がかかる。このため、 Ruby の Thread クラスを活用することにより、１つの Web ページの処理に対し１つのスレッドを割り当てることで、複数の Web ページを同時に収集し処理を行なえるようにした。また、検索を行う際にも、スレッドを用いてインタフェース部である Carta が GUI の反応が行えるようにした。検索を行っている間は、 Dealer へクエリーを渡す作業を停止し、検索が終了した段階でそのとき最新のクエリーを渡すようにした。これにより、ユーザは検索結果が返ってくるのを待たずに、クエリーの修正を行うことができる。

Web ページの収集の際、ネットワーク状況や対象ページのサイズ大きいことによりページの収得に非常に時間がかかる場合がある。また、 chasen では、対象の文章と内部の辞書の比較を総当りで行っているため、ページのサイズが大きい場合、形態

素解析にかかる時間が爆発的に増加する。このため、 Web ページに対する、ページの収得、タグの削除、形態素解析、キーワードの抽出を１つのスレッドとし、一定数のページの処理が終了した時点で、結果のキーワードを導き出すようにした。また、Web ページのサイズが大きい場合、テキストの長さを区切って処理することで対応した。現状では、テキストを 1500 バイトで区切り、 3 ページの処理が終わった段階で結果を導き出すようにしている。

検索結果のキーワードは、 chasen の出力結果よりキーワードリストを作成し、そのリスト内での出現頻度を数えることで優先順位を決定している。しかし、２バイト文字は単純比較は難しく、 Ruby の機能である正規表現による比較を使うと非常に時間がかかる。そこで、外部の検索エンジンへの URL を作成する際に使った手法を用いて、キーワードリストを ASCII 文字に変換した上で比較することで、パタンマッチ処理の高速化を実現した。 正規表現を利用したものと比較して、約 $\frac{1}{30}$ の時間で処理できる。

3.4 システム実行例

本システム全体では次の手順で処理を行っている。

1. 入力部や履歴部からのユーザの新規入力に反応し、 Carta が新規ノードの座標等の情報をノードリストに追加し、クエリー作成部へ表示する。同時にクエリー作成部に存在するキーワードの状況についてパーズをおこない、 Dealer へ渡せる形のキーワード群を作成する 3.4。

2. Carta が Storm へクエリーとなるキーワード群を渡す。 Storm はキーワード群をそのまま QPin にわたす。

3. QPin が渡されたキーワード群を変換し、外部の検索エンジンへ渡す 3.5。

4. QPin は外部の検索エンジンより返ってきた Web ページより URL を抽出し、URL リストとして Storm にわたす。

5. Storm が URL リストの先頭にある Web ページを Carta にわたす。このページが もっとも検索目的に近いページとしてユーザに表示される。

6. Storm が URL リストにある Web ページをすべて収集し、その中にある HTML タグをすべて削除する 3.6。

7. Storm が Web ページを加工してつくられたテキストデータ を Chaseav にわたす。 Chaseav はこのテキストデータを 茶筅に渡す。

8. Chaseav が 茶筅より受け取った結果リストを Storm にわたす 3.7。

9. Storm が茶筅の結果リストより、キーワードを選び出す。

10. Storm が選び出した中でも上位に当たるキーワード群を Carta にわたす。

11. Carta はキーワード群について、適宜つけられた座標データとともに結果ノードリストに追加し、検索結果部へ結果ノードとして提示する 3.8。

12. 入力部や履歴部からのユーザの新規入力に加え、検索結果部からのノードのドラッグ＆ドロップ、クエリー作成部内のノードの移動、削除が行わせた場合、クエリー作成部に存在するキーワードの状況についてパーズをおこない、Dealer へ渡せる形のキーワード群をわたす。以下、 2) − 12) を繰り返す。

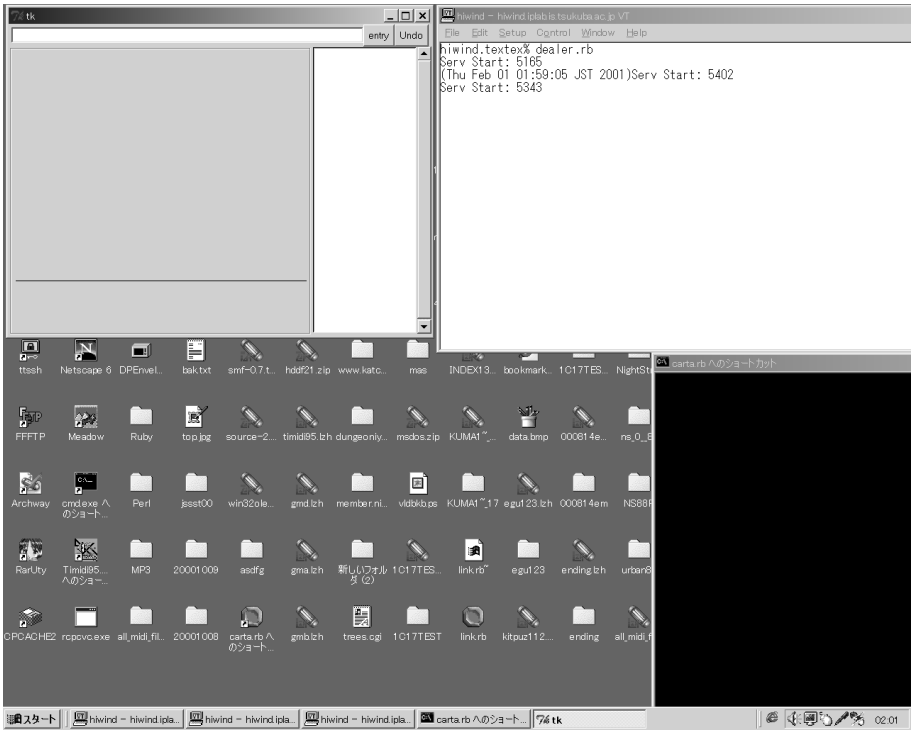

図 3.3: Carta の実行（初期状態）

3.5 関連研究

本システムはメタ検索エンジンとして実装されている。本システム以外でのメタ検索エンジンの例を述べる。

3.5.1 MetaCrawler

MetaCrawler[25] は、収集した URL が示す Web ページの有無や関連情報の収集などを行い、結果情報を加工して提示するメタ検索エンジンである。 1 つのクエリー入力より 6 件の検索エンジンから URL 情報を集め、それらの URL のページの有無や

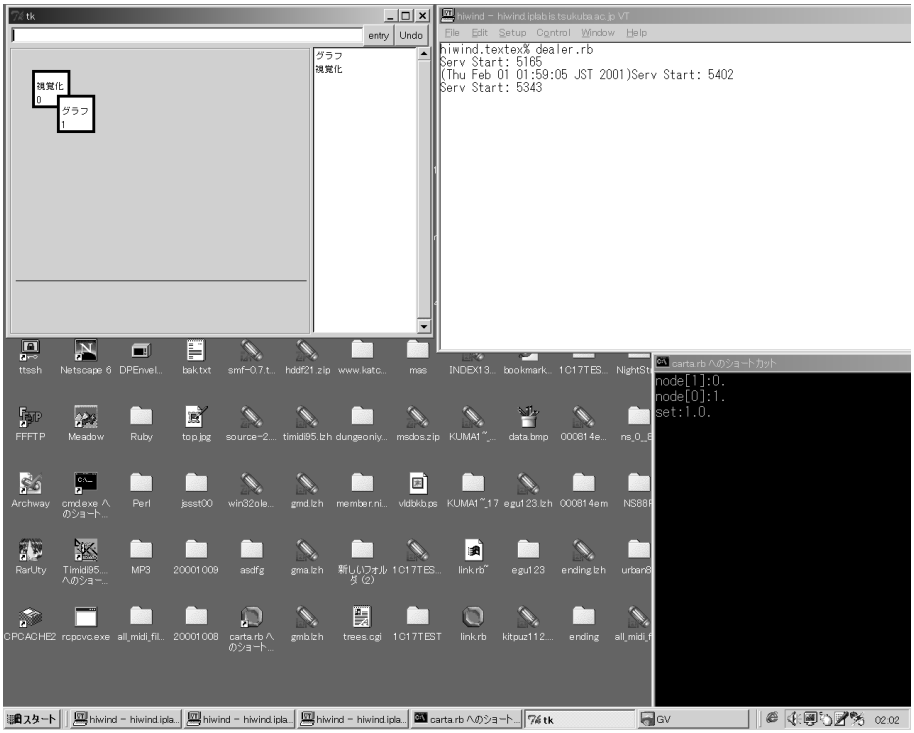

図 3.4: Carta の検索処理 (1)

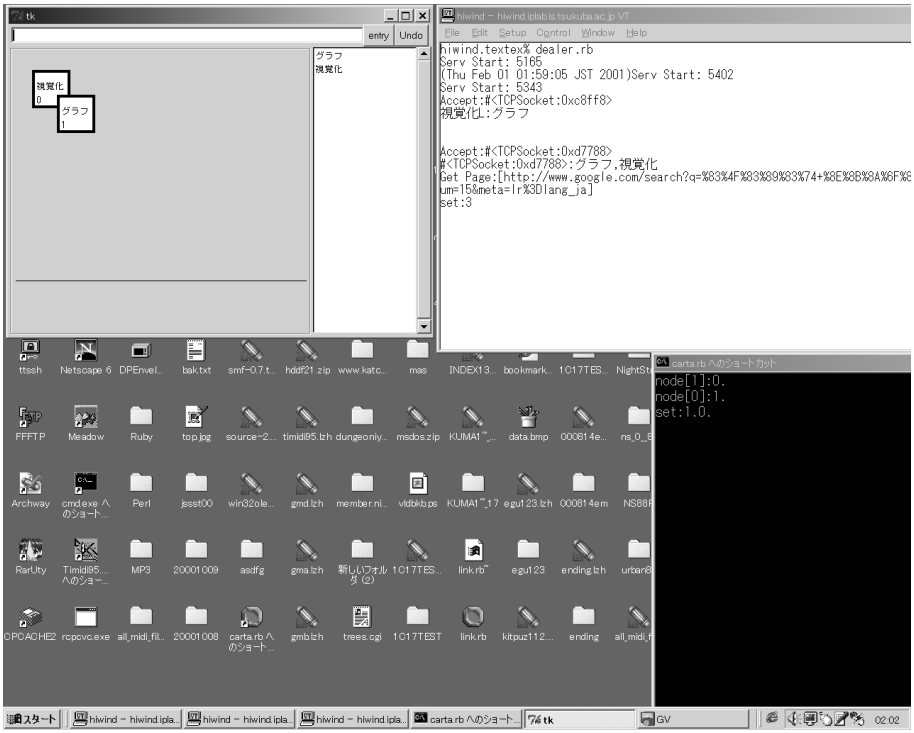

図 3.5: Carta の検索処理 (2)

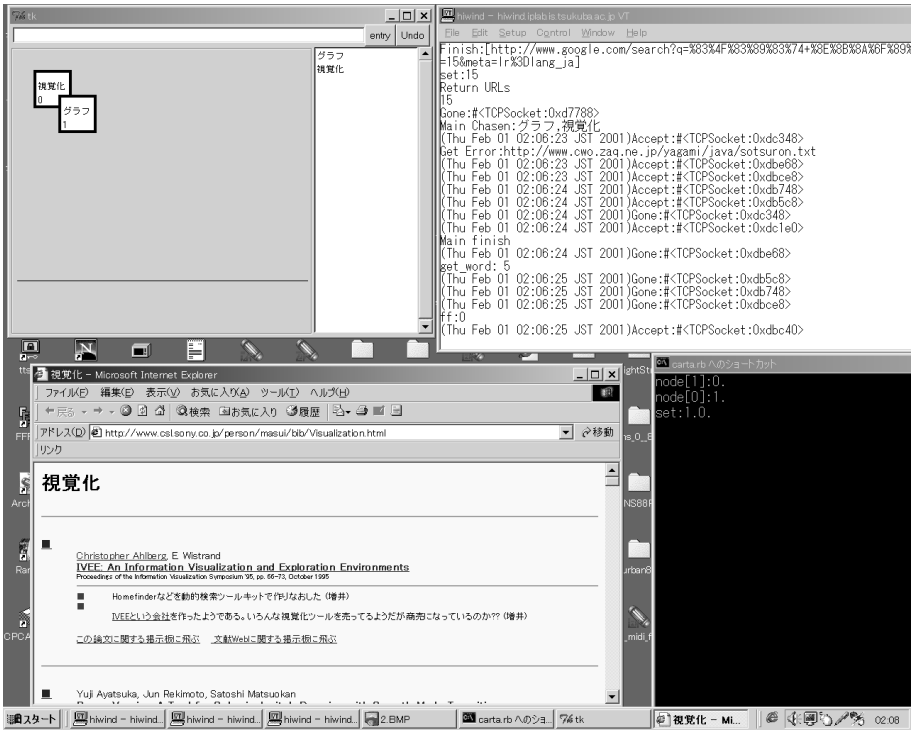

図 3.6: Carta の検索処理（3）

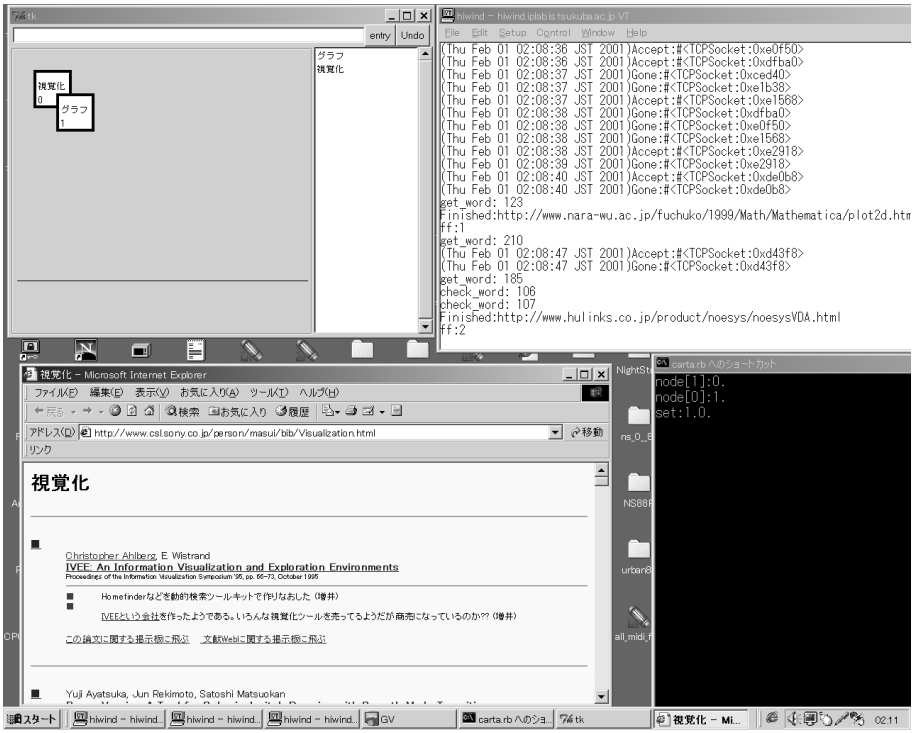

図 3.7: Carta の検索処理（4）

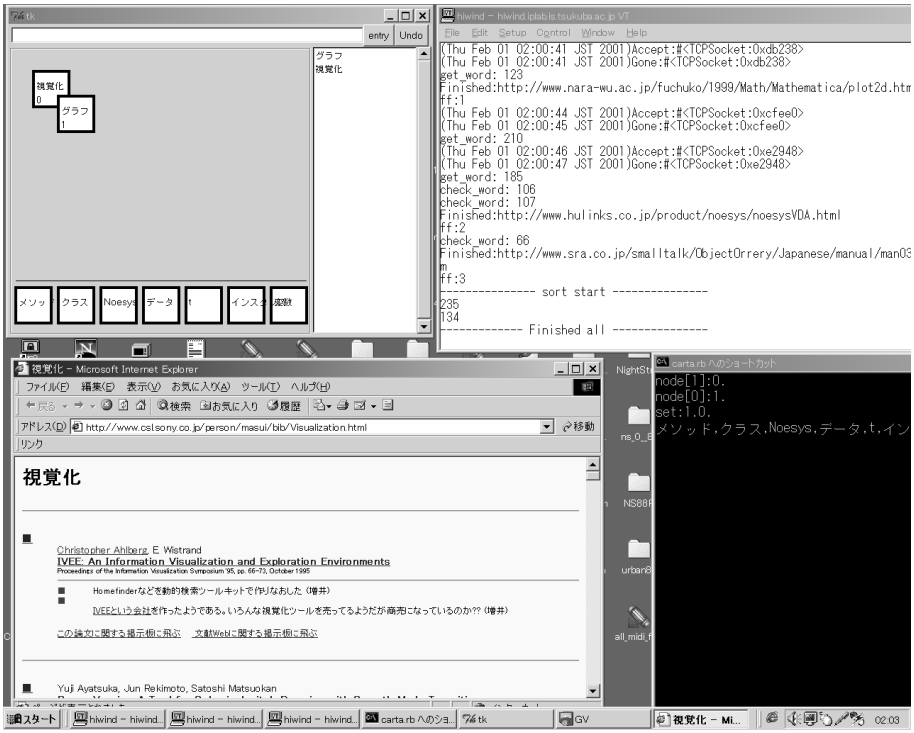

図 3.8: Carta の検索結果表示

関連の情報を収得することで、存在しないページを省いたり、検索結果の個人化を行うことで、ユーザにとっての負担が軽減している。また、マシンに対しての負担も軽いため、高度なフィルタリングやプライバシーの確保の処理も行うことができる。

3.5.2 Clever

Clever[6, 26] は対象ページのリンク関係を調べることにより、より精度の高い絞り込みを行なうメタ検索エンジンである。このシステムでは複数のページからリンクされ、また、複数のページへリンクをしている、ハブページという存在に注目している。与えられたクエリーについて ArtaVista で検索を行い、その検索結果のリンク関係を知らべ、「多くのハブページが指し示す Web ページは重要度が高い」として検索結果の優先順位つけている。

第 4 章

まとめ

4.1 まとめ

クエリーに重ねあわせによるグラフ表現を用いた、ノードによる対話的 Web キーワード検索インタフェースを設計した。また、このインタフェースに基づいて Web 検索システム Carta を実装した。 Carta では、クエリーをノードの組み合わせによって直観的に作成できる。検索結果は URL ではなくクエリーに利用可能なキーワード群で提示するため、キーワード選択の負担を軽減し Web 検索をより容易にする。

また実装では、すでに存在する検索エンジンの膨大なデータを活用するために、メタ検索エンジンとした。キーワード抽出を行うため、関係のある URL リストを検索する段階と、該当する Web ページよりキーワードを抽出する 2 つの段階に分けて検索を行う手法を取った。検索を 2 段階にわけて行うことでユーザに対するキーワード群の提示を実現した。システムを実装する際にいくつか発生した、処理に時間を要するために対話性が損なわれるという問題に対し、それぞれの問題に適応した解決法をシステムに実装した。これにより、実時間におけるキーワード抽出を可能にした。

4.2 今後の展望

今回作成したシステムでは、一切の情報を蓄積することなく検索を行っている。今後、キーワードの相関やユーザ履歴等の情報を蓄えることで、より精度の高い、ユーザに合った検索結果を提示できるよう改良する。

現在、検索結果であるキーワードの優先順位はページ内のキーワードの出現頻度に基づいて表示されている。この検索結果の精度をあげるために、タグ情報などの活用を検討している。また、現在利用している茶筅は日本語の単語のみの検索を行っているため、複合語や ASCII 文字による英単語などへの対応についても検討している。

また、インタフェースの実装に Ruby/Tk を用いたため、 Carta の実行可能な環境が限定されている。インタフェース部を JavaScript で移植し、 Web 上から利用可能にすることで、システムの利用価値を拡張を考えている。

謝辞

本研究を進めるにあたり、指導教官である山下義行先生にはさまざまな点でご配慮いただきました。また、田中二郎先生、志築文太郎先生には終始親切にしていただきました。この場を借りて感謝いたします。

また、山下研究室、田中研究室の皆様にも助言や励ましの言葉など、さまざまな支援がありました。特に 山下研究室の中家鉄雄さんには Ruby の基本的な記述法について、田中研究室の小川徹さん、飯塚和久さん、三浦元喜さんには論文作成の際のポイントについてご助言いただきました。ここに深く感謝いたします。

伊地知宏さんにはノード利用の際の問題点や Chasen の利用について、科学技術振興事業団の原田康徳さんには検索結果の Web ページ出力についてご助言をいただきました。この場を借りて感謝いたします。

参考文献

[1] 林良彦, 小林喜嗣: “WWW 上の検索サービスの技術動向”, 情報処理, Vol.39, No.9, pp. 861–865 (1998).

[2] 林良彦: “インターネットサーチエンジン”, 計測と制御, Vol.37, No.9, pp. 648–650 (1997).

[3] 高木義和: “インターネットにおける情報検索”, 情報管理, Vol.38, No.10, pp. 891–900 (1996).

[4] 山名早人: “WWW 情報検索の現状”, コンピュータソフトウエア, Vol.14, No.5, pp. 67–74 (1997).

[5] 梶谷浩一 : “サーチエンジンの最新動向” , ACM SIGMOD 日本支部 第 13 回大会講演論文集 , pp. 83–95 (1999).

[6] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins: “Mining the Web’s Link Structure”, IEEE Computer, Vol. 32, No. 8, pp. 60–67(1999).

[7] Google 日本語版
`http://www.google.co.jp/`

[8] 奥村学, 難波英嗣: “テキスト自動要約技術の現状と課題”, 北陸先端科学技術大学院大学情報科学研究科リサーチレポート, IS-RR-98-0010I(1998).

[9] 奥村穂高, 田中二郎: “重ねあわせノードによる Web のキーワード検索” 情報処理学会第 60 回 (平成 12 年前期) 全国大会講演論文集 (3), pp. 103–104 (2000).

[10] 奥村穂高, 田中二郎: “ノードによる Web 検索インタフェース”, 日本ソフトウエア科学会第 17 回大会 (2000).

[11] Dominic Stanyer and Rob Procter: “Improving Web Usability with the Link Lens.”, WWW8 / Computer Networks and ISDN Systems, Vol.31, No.11–16, pp. 1533–1544(1999).

[12] Hirakawa, M. ,Tanaka, M. and Ichikawa, T. : “An Iconic Programming System, HI–VISUAL.”,IEEE Transaction on Software Engineering, Vol.16 , No.10 ,pp. 1178–1184 (1990).

[13] 奥村穂高, 田中二郎: “重ねあわせを用いたビジュアルプログラミングの表記法”, 日本ソフトウエア科学会第 15 回大会論文集, pp. 121–124 (1998).

[14] 奥村穂高, 田中二郎: “重ねあわせを用いたシェルスクリプトプログラミング”, 日本ソフトウエア科学会第 16 回大会論文集, pp. 61–64 (1999).

[15] 馬場昭宏, 田中二郎: “ビジュアルプログラミングシステムのための UNDO 機能”, 日本ソフトウエア科学会第 13 回大会論文集, pp. 141–144 (1996).

[16] 鷲崎誠司, 菊井玄一郎, 林良彦: “WWW 上の情報検索システムにおけるインタラクティブインタフェース”, インタラクティブシステムとソフトウェアＩＶ, 近代科学社 , pp. 1–10 (1996).

[17] 林良彦, 菊井玄一郎, 鷲崎誠司, 砂場倫太郎: “WWW 情報空間における Resoure Discovery と Navigation 支援”, 電子情報通信学会 人工知能と知識処理研究会 , AI-95-31 (1995).

[18] 田中智博 他 ： “情報検索における GUI に関する一考察” , 情報処理学会第 51 回 (平成 7 年後期) 全国大会講演論文集 (6) , pp. 227–228 (1995).

[19] 西岡真吾, 丹羽芳樹, 岩山真, 高野明彦: “文献検索支援インタフェース Dual-NAVI” , インタラクティブシステムとソフトウェアＶ , 近代科学社 , pp. 43–48 (1997).

[20] 三末和男, 渡部勇: “テキストマイニングのための連想関係の可視化技術”, 情報処理学会第 55 回情報学基礎研究会 (1999).

[21] 渡部勇, 三末和男: “単語の連想によるテキストマイニング”, 情報処理学会第 55 回情報学基礎研究会 (1999).

[22] 松本裕治, 北内啓, 山下達雄, 平野義隆, 松田寛, 浅原正幸: “日本語形態素解析システム『茶筅』 version 2.0 使用説明書 第二版”, NAIST Technical Report, NAIST-T-TR99012(1999).

[23] まつもとゆきひろ, 石塚圭樹: “オブジェクト指向スクリプト言語 Ruby”, アスキー出版局 (1999).

[24] 原信一郎: “Ruby プログラミング入門”, オーム社開発局 (2000).

[25] Erik Selberg and Oren Etzioni:“Multi-Service Search and Comparison Using the MetaCrawler”, 4th International World Wide Web Conference(1995).

[26] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins: “Hypersearching the web.”, Scientific American, Vol.280, No.6, pp. 54–60(1999).

[27] インフォシーク
http://www.infoseek.co.jp/

付録 A

Carta 仕様説明書

付録Aでは、検索システム Carta の内部プロセス間で使われている通信プロトコル、各プロセス内のクラス関数、メソッドの仕様について述べる。

A.1　通信プロトコル

本システム内の各プロセスの間では、2 次元の Array をやり取りしている。発信側では、内側の Array の要素の区切りに “\r” 、外の Array の区切りに “\n” をはさみ、1 つのストリームにしている。また、終端記号として “\r\r\n” を渡すことで、送信の終端を確認している。

A.2　クラス変数

A.2.1　Carta

@node

type： Array
クエリーノードのデータを蓄積するための変数。 Array 型のノード情報が入る。 1 つのノード情報は [x 座標 (Integer), y 座標 (Integer), コマンド名 (String), ノードナンバー (String)] となっている。ノードナンバーとは、クエリーに使われたノードに与えられた通し番号である。

@rnode

type： Array
結果ノードのデータを蓄積するための変数。 Array 型のノード情報が入る。結果ノードにはノードナンバーが無いため、 1 つのノード情報は [x 座標 (Integer), y 座標 (Integer), コマンド名 (String), nil] となる。また、結果ノードをドラッグした場合、ノードナンバーに “d” が入る。

@log

type： Array

UNDO のための動作ログを蓄えるための変数。1 つの log 項目は、[x 座標 (Integer), y 座標 (Integer), コマンド名 (String), ノードナンバー (String), 状況 (String)] となっている。状況は新規ノードだと “-1”、削除されたノードだと “d”、その他クエリーとして存在するノードだと @node の先頭からの順位が入る。

@rnode_t

type： Integer
検索結果の表示回数。@call と差が生じた場合、結果ノードの描画を行った後、@call と等しい値にされる。

@call

type： Integer
検索結果の応答回数。calleng により、検索結果が得られたときに増える。

@draw_tag

type： String
現在描画中のノードの種類を示すためのタグ。“normal”、“answer”、“dragnode” の 3 種類がある。

@point

type： Array
現在のマウス座標。[x 座標 (Integer), y 座標 (Integer)] となっている。

@count

type： Integer
現在最新のノードナンバー。

@event

type： Integer
calleng の実行許可フラグ。この値が 1 のとき検索が実行される。

@drag_node

type： Array
ドラッグ中のノード情報。[x 座標 (Integer), y 座標 (Integer), コマンド名 (String)] となっている。

@dis

type： Array
ドラッグ中のノードの基準点 (左上の頂点) とマウスカーソルの基準点との差。[x 座標 (Integer), y 座標 (Integer)] となっている。

@client

type： IO
ブラウザ管理のための IO クラス。本システムでは、外部の Web ブラウザをパイププロセスとして起動させている。

@dcount

type： Integer
calleng_dammy 用のカウンタ。インタフェースのテストのとき、結果ノードのコマンド名のかわりに使われる。

@pid

type： Thread
現在検索が行われているスレッド。

@pid_b

type： Thread
次に検索を行うスレッド。

@newcom

type： TkVariable
入力部で入力されたテキストのためのテンポラリー。この変数の中身が新規ノードのキーワードになる。

@canvas

type： TkCanvas
クエリー作成部・結果表示部にあたるキャンバスウェジット。

@histry

type： TkListbox
履歴部のリストボックスウェジット。

@sber

type： TkScrollbar
履歴部のスクロールバーウェジット。履歴部のリストボックスを上下にスクロールさせる。

@entry_text

type： TkEntry
入力部のテキストウェジット。

@entry

type： TkButton
入力部のエントリーボタンウェジット。このボタンウェジットを押すことで、 entry_work メソッドが呼び出される。

@undo

type： TkButton
UNDO ボタンウェジット。このボタンウェジットを押すことで、 undo メソッドが呼び出される。

A.2.2 Storm

@gs

type： TCPServer
Carta からのソケット通信を受けるサーバソケットクラス。

@addr

type： Array
サーバソケットに来た問い合わせを蓄えるための配列。

A.2.3 QPin

@gs

type： TCPServer
Storm からのソケット通信を受けるサーバソケットクラス。

@addr

type： Array
サーバソケットに来た問い合わせを蓄えるための配列。

A.2.4 Chaseav

@gs

type： TCPServer
Storm からのソケット通信を受けるサーバソケットクラス。

@addr

type： Array
サーバソケットに来た問い合わせを蓄えるための配列。

A.3 メソッドの仕様

A.3.1 Carta

initialize

引数：なし／返り値：なし
Carta クラスの初期設定。主に、クラス変数の宣言と Tk ウェジットの宣言。

bind_start

引数：なし／返り値：なし
クエリー作成部、結果表示部、履歴部でのマウスのバインドの設定。

entry_work

引数：なし／返り値：なし
新規ノードの初期設定。

encode_word

引数： String ／返り値： String
引数として渡された String 型のデータを %xx (x は 16 進数) の形へ変換する。変換の際、漢字コードはすべて JIS にする。

select_hist

引数： Integer ／返り値：なし
履歴部のどのキーワードをクリックしたかを判別する。引数としてマウスをクリックした時点での @histry 内での y 座標を渡す。

sort_hist

引数： String ／返り値：なし
引数として渡された String が @histry が持つキーワードと一致すれば、そのキーワードを先頭に移動する。一致するものが無ければ、新規のキーワードとして @histry の先頭に追加する。

get_node

引数：なし／返り値：なし
クエリーノードのドラッグ作業を行う。

get_rnode

引数： Integer ／返り値：なし
結果ノードのドラッグ作業を行う。引数として、ドラッグされるノードの @rnode 内での位置を渡す。

move_node

引数：なし／返り値：なし
ドラッグ中のノードの移動作業を行う。

put_node

引数：なし／返り値：なし
ドラッグ中のノードのドロップ作業を行う。

search_node

引数：なし／返り値： Integer
現在のマウス位置の下にクエリーノードが存在するか調べる。存在した場合、そのノードの @node における位置を返す。もし複数存在する場合は、最前面のノードについて返す。該当するノードが無い場合は -1 を返す。

search_rnode

引数：なし／返り値： Integer
現在のマウス位置の下に結果ノードが存在するか調べる。存在した場合、そのノードの @rnode における位置を返す。該当するノードが無い場合は -1 を返す。

set_node

引数： Array ／返り値：なし
引数として渡されたノード情報を @node に追加する。

redraw_n

引数：なし／返り値：なし
@node にあるノードをクエリー作成部に描画する。

redraw_r

引数：なし／返り値：なし
@rnode にあるノードを検索結果部に描画する。

draw_node

引数： Array ／返り値：なし
引数として渡されたノード情報に基づきノード 1 つを描画する。

getlog

引数： Array ／返り値：なし
引数として渡された log 情報を @log に追加する。

undo

引数：なし／返り値：なし
@log の内容に従い、逆操作を行う。逆操作を行った際には、その作業分の log を @log の最新の個所から削除する。

calleng_dammy

引数：なし／返り値：なし
インタフェースのテストの際、 calleng の代わりに使うダミー。

calleng

引数：なし／返り値：なし
Dealer との通信作業を行う。 Web ブラウザの起動もここで行う。

parse

引数：なし／返り値： Array
@node のデータから、クエリーノードのパーズを行う。パーズ結果は 2 次元の Array として返される。

A.3.2 Storm

initialize

引数：なし／返り値：なし
Storm のメインループ。

get_urls

引数： Array ／返り値： Array
QPin より、 URL リストを収得する。引数として String の要素をもつ Array 型であるキーワードリストが渡される。結果は String 型で記述された URL を要素とした Array 型を返す。

get_webpage

引数： String ／返り値： Array
引数として渡された URL より、 Web ページを収得する。結果は改行までを 1 つの String とした Arrey を返す。

del_tag

引数： Array ／返り値： Array
渡されたページ情報から HTML タグを取り除く。

call_cha

引数： Array ／返り値： Array
渡されたページ情報を chaseav に渡し、茶筅の解析結果を得る。

get_wordlist

引数： Array ／返り値： Array
茶筅の解析結果より名詞などによるキーワードリストを得る。

sort

引数： Array ／返り値： Array
得られたキーワードリストの頻度を調べ、頻度順にキーワードを並べ替える。

encode_word

引数： String ／返り値： String
引数として渡された String 型のデータを %xx (x は 16 進数) の形へ変換する。変換の際、漢字コードはすべて JIS にする。

A.3.3 QPin

initialize

引数：なし／返り値：なし
メインループ。サーバとしての動作が記述されている。

encode_word

引数： String ／返り値： String
引数として渡された String 型のデータを %xx (x は 16 進数) の形へ変換する。変換の際、漢字コードはすべて JIS にする。

get_webpage

引数： String ／返り値： Array
引数として与えられた URL の Web ページを収得する。

get_url_google

引数： Array ／返り値： Array
検索エンジン google に対して、引数で与えられたキーワードリストについての検索を行う。帰り値として URL リストを返す。

get_url_infoseek

引数： Array ／返り値： Array
検索エンジン infoseek に対して、引数で与えられたキーワードリストについての検索を行う。帰り値として URL リストを返す。

A.3.4 Caseav

initialize

引数：なし／返り値：なし
メインループ。サーバとしての動作が記述されている。

call_chasen

引数： Array ／返り値： Array
渡された Web ページのテキスト情報を茶筅に渡し、その結果を Array として返す。茶筅を活用するために、パイプと fork を利用している。

付録 B

Carta ソースプログラム

付録Bでは、 Carta のすべてのスクリプトプログラムを添付する。内容は以下の通りである。

- インタフェース部 Carta (`carta.rb`)
- 検索部中核 Storm (`storm.rb`)
- URL リスト作成部 QPin (`qpin.rb`)
- 形態素解析部 『茶渋』 Chaseav (`chaseav.rb`)