o 用語知識ベースブラウザーの機能拡張

－ XMLデータベース化、入出力の高速化、編集機能の強化 －

後藤 智範

神奈川大学 理学部 情報科学科

1. はじめに

科学技術分野の知識体系としてのシソーラス、および国語辞典、英和辞典などの各種用語辞典について、知識構造に基づく 用語 - 概念 の探索を可能とするブラウザーの開発を行ってきた[1] [2] [3] [4]。1997 年度からマルチプラットフォームで実現できる Java 言語[5] [6]で記述したブラウザーを試作してきた。

本年度は、下記の項目について改善を行った。

用語知識ベース
- (1) 用語知識ベースの XMLDB 化
- (2) 入出力高速化のためのオブジェクト分割

用語木表示
- (3) 用語木表示高速化のための描画アルゴリズム

概念関係編集機能
- (4) 概念関係の編集機能

用語木編集機能
- (5) 用語木編集機能の強化

その他
- (6) 用語木ウィンドウの連続表示機能

次章以降に、上記の項目についての改善点を報告する。

2. 用語知識ベースの XMLDB 化

従来、当該用語知識ベース[7]は、タグ付けされてはいたが、それぞれ個別のレコード書式および論理構造を採用していた。用語知識ベースのデータ構造に対して、このような知識ベース固有のデータ構造では、対象知識ベースが増大すると、入出力モジュールなど、プログラムコードを個別に作成する必要があり、ソフトウエアシステムとして複雑になり、また開発コストを要することになる。

この問題を解決するために、汎用的な用語知識ベース構造を意図した XML DTD を設計し、これにしたがって、対象知識ベースを XML データベース化した。DTD の要素型、言い換えれば用語知識ベースの属性は下記の 4 種類の項目に分類した。

(a) 見出し語、　(b) 用語属性、　(c) 意味関係、　(d) 階層関係

これらの項目は、DTD において、OR とされているため、通常の辞書には含まれない(c), (d)のような項目をもたなくとも、1 つの DTD で扱うことができる。
下記にそれぞれの項目内容を説明する。

(a) 見出し語

「見出し語」を構成する属性であり、4 つの属性からなる。ＥＤＲ、INSPEC シソーラス、JICST シソーラスについては見出し語について「日本語の見出し語」と「英語の見出し語」の２種類が存在

するため、それらの見出し語についてはブラウザーから見出し語データファイルを読み込んだ際に(ＯＲ)で繋ぎ表示させている。読みは、見出し語が日本語で漢字が含まれている場合、その見出し語の頭文字を平仮名で表現できない語が出てしまう。これを避けるために平仮名、またはカタカナのみの読みという要素を追加し、現在の見出し語の頭文字の識別ができるようにしている。

木番号は、論理木構造のある用語知識データに関して、倫理木ごとにバイナリデータ化しているため、その見出し語がどの論理木の用語なのかを区別するための情報である。

(1) 見出し語(entry)
(2) 見出し語ＩＤ(term-id)
(3) 読み(reading)
(4) 木番号

(b) 用語属性

用語が持つ様々な属性がこの項目に含まれる。これらを表１にに列挙する。

表１ 用語属性を構成する項目群

見出し語ＩＤ(term-id)	品詞(part-of-speech)
一般語(general)	読み、日本語訳(reading)
略語(abbreviation)	漢字の当て方(other)
見出し語区切り(leave-off)	訓読み(phonetic)
発音(pronunciation)	音読み(japanese)
専門語、別表記(technical-term)	用例(ex)
外来語(loanword)	分野、位相(fields)
画数(strokes)	意味(means)
類義語(synonym)	参考、関連語(refs)
対義語(antonym)	参照(cfs)
比較語(conpare)	

(c) 意味関係

意味関係データとはある用語知識データの意味関係情報を表すものであるが、当然その情報を持データベースのみが持つデータであるため、現在ＥＤＲ，ＩＮＳＰＥＣ，ＪＩＣＳＴ，角川類義語辞典のみが必要とするデータである。

(d) 階層関係

用語間の階層関係を明示的に規定し、表現するための項目からなる。

表２ 階層関係を構成する項目群

最上位語識別記号(topterm-yn)	木レベル(level)
木番号(tree-no)	非ディスクリプタ(nondesterm)
非ディスクリプタ個数(no-of-nondes)	関連語(relatedterm)
最上位語個数(no-of-topterm)	分類語(code)
上位語個数(no-of-bterm)	関係する識別子(relationterm)
下位語個数(no-of-nterm)	関係子(relation)
関係子の数(no-of-relation)	記述区分(des-division)
分類コード個数(no-of-category)	

現時点でXMLデータベース化されている各種辞書を下記に挙げる。上述した同一の用語知識ベースDTDに基づいてXMLデータベース化され、いずれも当該ブラウザーで内容にアクセスすることができる。

■ 国語辞典
(1) 広辞苑、　　(2) 三省堂国語辞典、　　(3) 角川類語辞典
■ 英和／和英辞典
(1) リーダーズ英和辞典、　　(2) センチュリー英和辞典
■ 専門用語辞典
(1) コンピュータ用語辞典、　　(2) EB 科学技術用語辞典
(3) 科学技術用語辞典、　　(4) 25 万語医学用語辞典
■ シソーラス
(1) INSPEC シソーラス 1988 年度版 、　(2) JICST シソーラス 1988 年度版
■ 電子辞書
(1) EDR[8] 電子辞書、　　(2) NTT 日本語語彙体系[9]

3 ファイル読込みの高速化

従来のブラウザーでは当該の用語知識ベース全てを主記憶に読み込み、用語リストと論理木の作成を行ってきた。広辞苑、EDR、NTT 日本語語彙体系などでは、見出し語総数が 20 万語を超え、この方法では、用語リスト表示に数秒を要する。前章で説明した当該知識ベースの XML データベース化によっても、この問題は改善されない。問題は、起動時に当該データベースの内容を全て読み込むことに起因する。この問題を解決するために、上述の 4 種類の項目毎にファイルとして独立させ、さらにそれぞれの利用目的に最適化したファイル構造を採用した。具体的には、

(b) 用語属性ファイル ・・・・・ ランダム・アクセス・ファイル
(c) 意味関係ファイル ・・・・・
(d) 階層関係ファイル ・・・・・ ランダム・アクセス・ファイル

3.1 見出し語ファイル

見出し語ファイルは[dat / dic / ファイル名 /]に「用語ファイル[頭文字]」の形式でテキストファイル(フラットファイル)として保存されている。見出し語ファイルはリスト作成時に呼び出される。頭文字ごとにファイルが分割されていることにより、リスト作成までの時間が短縮された。

表 3 はブラウザーを立ち上げ、メニューから「A」を選択した時点から、頭文字Aの用語について(すなわち、「用語ファイルA」に含まれる全ての語)リストを作成し表示するまでにかかる時間を計測し、従来のものと比較したものである。

表 3 用語リスト表示までの処理速度

辞書名	JIDST	INSPEC	EDR
従来：処理速度(ms)	3562	844	75610
今年：処理速度(ms)	62	79	1718

この表から語彙数(見出し語数)が最も多い EDR では、400 倍以上高速化されていることがわかる。

3.2 用語属性ファイル

見出し語ファイル同様、頭文字ごとにファイルが分割されている。さらに、それぞれがランダム・アクセス・ファイル化されている。このファイルはリスト作成後、図 1 のポップアップメニューの内容出力画面を選択後、呼び出される。テキストファイルの場合、選択語の情報を用語属性ファイルから得たい場合、ファイルの先頭から読み込み、目的の情報を得ることとなるが、これでは効率が悪い。そのため、ファイルをランダムアクセス化することにより、目的の情報をファイルの先頭から探索することなく得ることができる。ランダム・アクセス・ファイルの生成は JAVA の API を利用した。

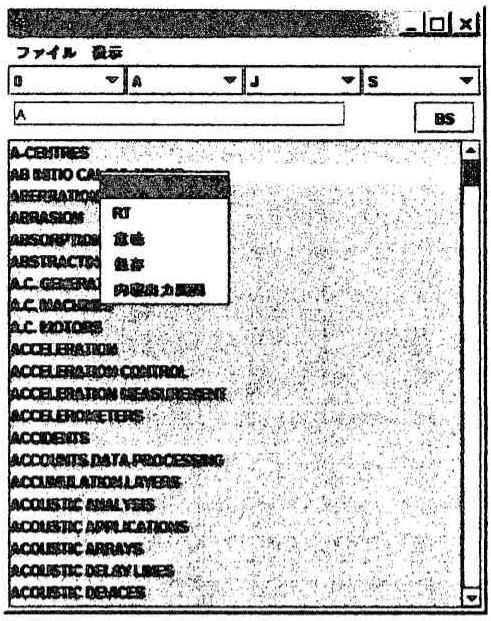

図1 リストからポップアップメニューの表示

3.3 意味関係ファイル

このファイルは概念ブラウザー表示の際に必要なファイルである(図1　GR選択時)。用語属性ファイルと同様、頭文字ごとに分割され、それぞれのファイルはランダム・アクセス・ファイル File 化されている。

3.4 階層関係ファイル

従来のブラウザーではリストから見出し語を選択した辞典で木を作成し表示させていた。リスト作成時に用語知識データ全ての用語を主記憶上に読み込み、その読み込んだデータに基づき木を作成していたため、巨大な用語知識データの場合、論理木作成にかなりの時間を要した。しかし、今回前もって論理木を作成しておきそれをバイナリファイル化することにより、そのファイルを読み込むだけで論理木が作成できるようになった。これによりストレスなく表示ができる。図1のNTを選択すると、リスト作成時に読み込まれていた最上位語情報を取得する。論理木はそれぞれ、最上位語をファイル名として分割されているため、その最上位語のファイルを読み込めば、論理木を瞬時に読み込み、表示ができるというわけである。バイナリデータは[dat / dic / ファイル名 / 階層 /]にある。

4. 用語木表示

4.1 用語木表示の高速化

従来のブラウザーは、用語数の非常に多い木(以降、巨大用語木と呼ぶ)を表示させる場合、広大な表示領域を必要とするため、膨大なメモリを消費してしまう問題があった。その原因として用語木を生成するときに、木構造全体を描画していたことが挙げられる。この方法は、1度描画してしまえば編集を行わない限り再描画することはないので、用語木上の処理は高速に行うことができる。しかし利用方法から考えると、巨大用語木の表示領域は全体のほんの僅かにすぎず、そのとき表示されていない領域が無駄になってしまう。また最悪の場合、用語木そのものがメモリ不足により表示できない。そこで、表示領域およびその周辺のみを描画し、スクロール等の操作を行った際に再描画を行う方法に改めた。これにより、従来表示できなかった巨大用語木が表示できるようになった。
よって本年度は、以下のような描画方法を用いた。ただし、この方法を用いるのはサイズが 1M(100万)ピクセルを超える巨大用語木のみとする。

・描画領域を、1000×900 を基本とする領域の9倍とする。ただし、縦または横のサイズが

描画領域のサイズに満たない場合はそれを適用する。
・表示領域を描画領域の中心として、描画を行う。
・リストの用語の選択や、用語木のスクロールを行った場合は再描画する。

この結果、巨大用語木を表示する際に消費するメモリ使用量は激減した。表4にその結果を示す。

表4 メモリ使用量

用語	"ZOOLOGY"	「思い」	「地位」
辞書データ	INSPEC	EDR	EDR
高さ	6	5	4
用語総数	333	869	1533
改善前の使用量(MB)	138	339	552
改善後の使用量(MB)	28.0	20.8	19.4

この表より、全体を表示するとメモリ使用量が500MBを超えるような巨大用語木であっても、メモリ使用量が20～30MB程度に抑えられていることがわかる。

4.2 再描画速度の高速化手法

前節の改善の結果、巨大用語木の表示に成功した。しかし再描画に要する時間が長く、場合によっては動作が重く感じられることがあった。そのため、以下の改良を行った。

・描画領域の確保を、必要最小限にとどめるようにした。
・用語木を走査するとき、必要な個所のみを走査するようにした。

これにより、再描画に要する時間は激減し、利用者にストレスを与えない高速な処理が可能となった。

4.2.1 描画領域確保の方法の改善

巨大用語木問題の解決直後は、再描画を行う度に描画領域の確保を行っていた。しかし、このことが再描画に時間を要する原因の1つであることがわかった。そのため、描画領域の確保は描画領域を新たに計算しなければいけないとき(新しい用語木を表示するときや、編集を行った後など)にとどめ、スクロール等を行った後は描画領域の確保を行わないようにした。

その結果を表5に示す。このとき使用したマシンは研究室のraphael3(cpu：AthlonXP1800+、メモリ：1GB)である。また描画速度の値は、再描画を10回行い、それに要した時間の平均を用いた。

表5 描画速度の変化

用語	"ZOOLOGY"	「思い」	「地位」
辞書データ	INSPEC	EDR	EDR
高さ	6	5	4
用語総数	333	869	1533
改善前の描画速度(ミリ秒)	433	386	397
改善後の描画速度(ミリ秒)	72	73	83

この表より、改善後の描画速度が数倍速くなっていることは明らかである。

4.2.2 用語木の走査方法の改善

用語総数が1000～2000程度の用語木であれば、前節の改善のみで十分である。なぜなら、描画を行う際に用語木全体を走査したとしても、さして時間はかからないからである。しかし、用語総数が1万を超えるような用語木の場合は、全体を走査すると非常に時間がかかってしまう。しかも、実際に走査しなければならないノードは表示領域とその周辺の一部に過ぎない。そのため、巨大用語木の

中でも特に巨大な用語木は、走査を行う際に必要な個所のみを走査するように改めた。
例として、横型表示の用語木の走査方法を図2に示す。

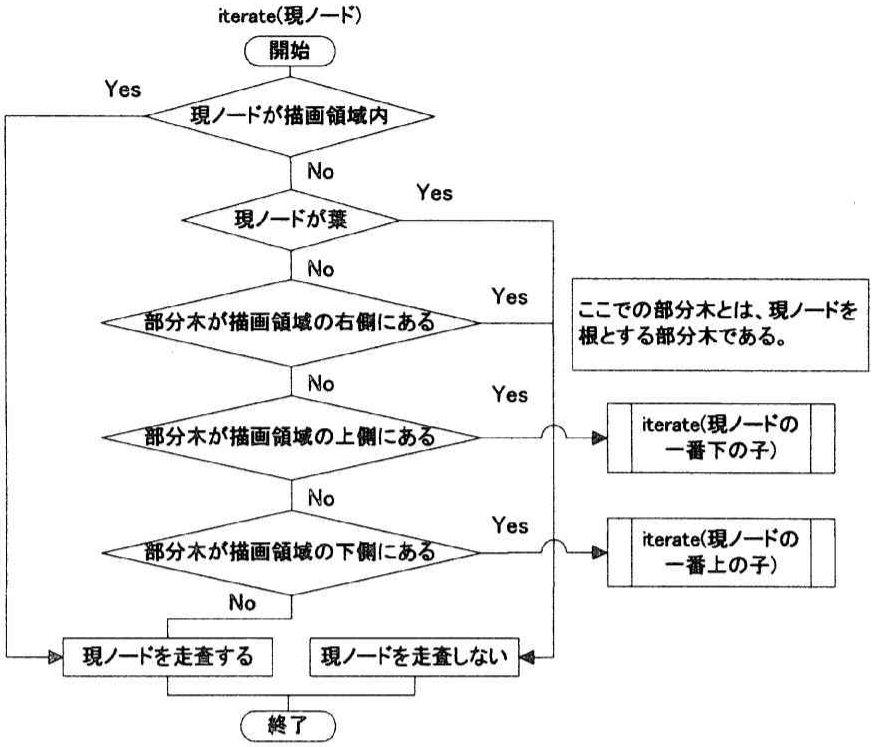

図2 新しい走査方法(横型表示の用語木の場合)

改善の結果を表3に示す。使用したマシンは表4、表5で使用されたもの同等である。また、ここでの「改善前」とは、前節の「改善後」を示す。

表6 描画速度の変化

用語	「地位」	「ありか」*	「ありか」*
辞書データ	EDR	EDR	EDR
高さ	4	3	9
用語総数	1533		
改善前の描画速度(ミリ秒)	83	97	548
改善後の描画速度(ミリ秒)	58	38	85

*左側の「ありか」は、右側の「ありか」の部分木である。

この表よりわかることは、用語総数が1万を超える用語木に関しては再描画に要する時間が激減したが、1000～2000程度の用語木に対しては効果が薄いことである。そのため、新しい走査方法を用いるのは、用語木のサイズが200万ピクセルを超える用語木のみとした。

4.3 用語リストに関する新機能

用語総数が増えるに従い、用語リストから目的の用語を探すのは困難となる。そこで本年度は、用語のレベル表示の付加、およびソート機能の実装を行った。これにより、用語総数が増えても目的の用語が見つけやすくなった。

4.3.1 用語のレベル表示の付加

従来の用語リストは、レベル情報の表示がないため、目的の用語がどの深さにあるのかが判断できない。そこで、用語のレベル表示の付加を行った(図2)。これにより、目的の用語の深さが一目でわかるようになった。

4.3.2 ソート機能

昨年度までの用語の並びは、用語木を先行順に走査した順番になっている。しかし、これは辞書順ではないため、利用者にとって使いやすい並び順ではない。そこで、用語リストのソート機能の追加を行った。可能となったソートは以下の 2 つである(ソートのアルゴリズムはクイックソートを使用した)。

- レベル順のソート(昇順・降順、同レベルの用語はそのレベル内で用語順となる)
- 用語順のソート(昇順・降順)

例として、レベル順のソートを挙げる(図 3)。

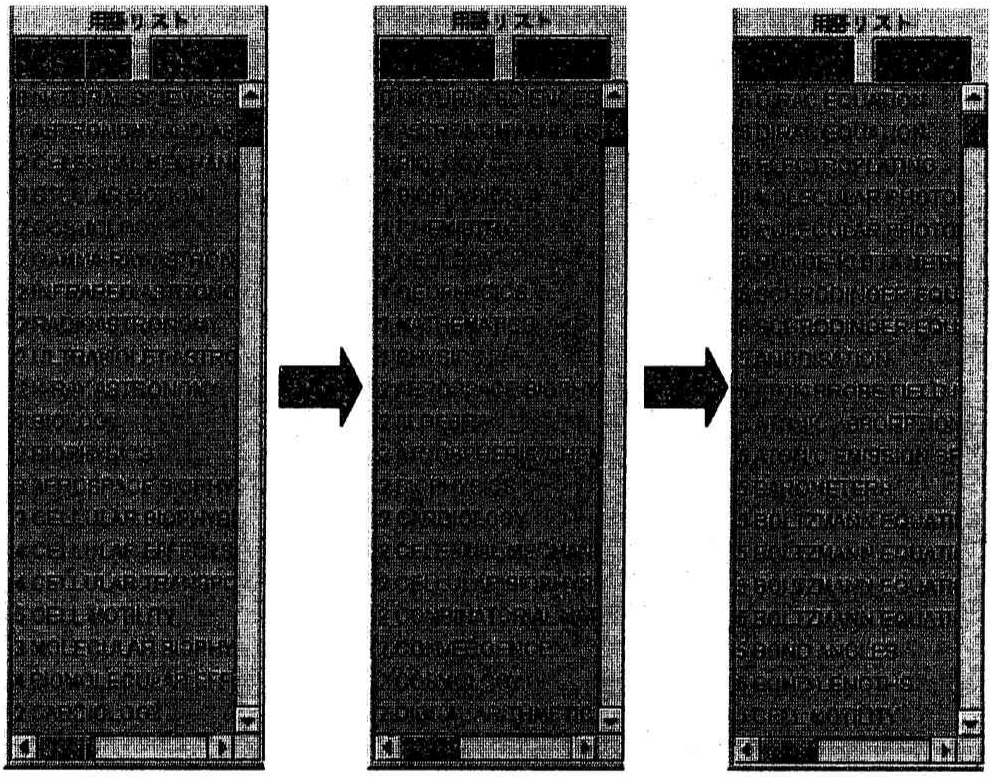

図 3 用語リストのソート機能(ソート前、レベル順・昇順、レベル順・降順)

ソート前(図 2 左)は用語の順番がばらばらで、目的の用語が探しづらい。そこで、用語リスト上部のレベルボタンをクリックする。すると、用語がレベル順(昇順)にソートされ、目的の用語が探しやすくなった(図 3 中)。もう一度用語ボタンをクリックすると降順にソートされる(図 3 右)。用語順にソートしたい場合は用語ボタンをクリックすれば良い。

5. 概念関係ブラウザーの編集機能

昨年度実装された概念関係ブラウザーに対し、本年度は編集機能を追加した。このことにより、利用者は関係子と概念の関係を視覚的に把握した状態で概念関係を編集することができる。実装した機能は以下のとおりである。

- 関係子の編集
- 概念の追加・削除

上記の編集はブラウザー下部の追加・削除・編集の各ボタンが押されたときに実行される(図 4)。

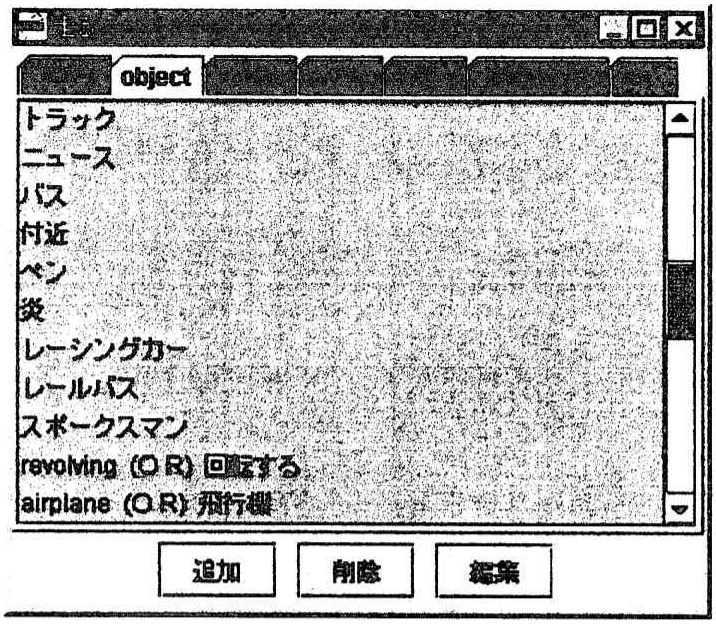

図4 概念関係ブラウザー

5.1 関係子の編集

関係子の編集機能として、以下の機能を実装した。

・関係子の追加(新しい関係子およびそれに属する概念を追加する)
・関係子の削除
・関係子名の変更

これらの編集は編集ダイアログ(図5)で行われ、実行ボタンを押されたときに、「追加」が選択されていれば新しい関係子およびそれに属する概念が追加され、「削除」または「変更」が選択されていればダイアログ上部のコンボボックスにおいて選択されている関係子に対して編集が行われる。

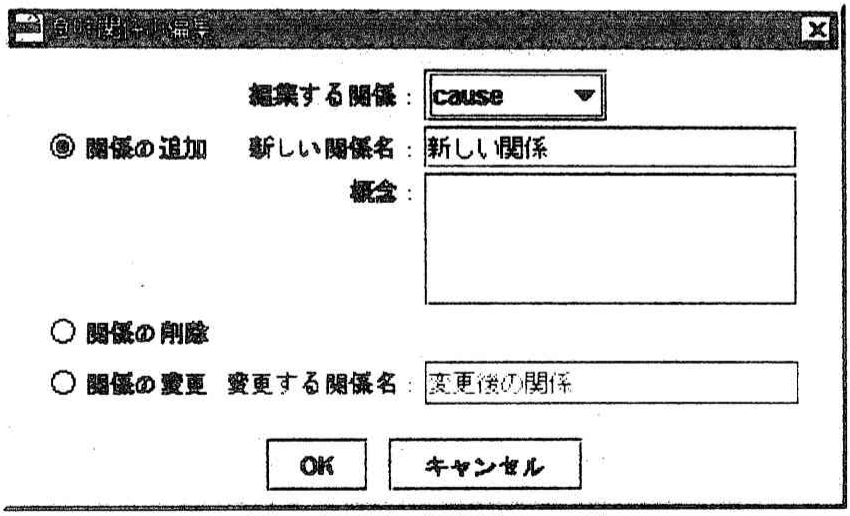

図5：関係子の編集ダイアログ

5.2 概念の追加・削除

概念を追加したい場合は、関係子を選択し追加ボタンを押す。するとダイアログボックス(図 6)が表示されるので、そこに追加したい概念を入力(複数入力可)し、OK ボタンをおすと概念が追加される。

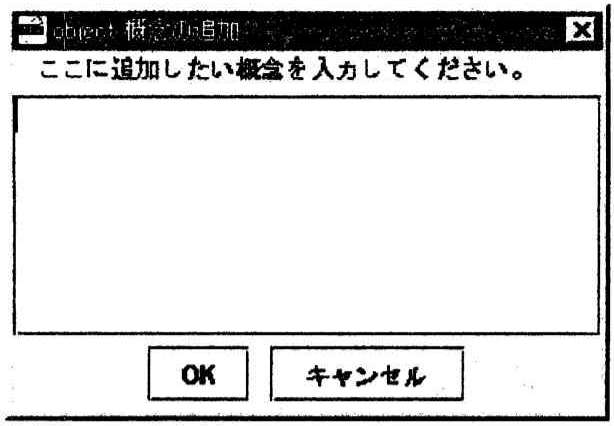

図6 概念の追加ダイアログ

概念を削除したい場合は、関係子を選択しその中の削除したい概念を選択する。そして削除ボタンを押すと、選択された概念が削除される。

6. 用語木編集に関する新機能

昨年度実装された用語木の編集機能に対し、本年度は以下の機能の追加を行った。

- ノードの追加(選択されたノードの内部節に新たなノードを生成する)
- 用語木の分割機能

6.1 ノードの追加機能(内部節)

昨年度の段階で、ノードの追加機能は既に実装されているが、選択されたノードの葉にしか追加することができなかった。そこで、本年度はノードを内部節に追加する機能を追加した。また、追加する際に新しいノードの下位語を選択することができる。具体的な手順を以下に示す。

- 編集対象とするノードを選択し、編集ダイアログ(図7左)を呼び出す。
- ダイアログボックスの、「ノードの追加」と「内部節に追加」を選択する。
- 新しく追加するノードの名前を入力する。
- 「下位語の選択」ボタンを押す。
- すると選択されたノードの下位語の一覧が表示される(図7右)ので、新しいノードの下位語を選択して、okボタンを押す(下位語は複数選択可能)。
- 下位語をコンボボックスで確認したあと、実行ボタンを押す。すると、新しいノードが内部節に追加される。

例を図8に示す。ここでは、JICSTシソーラスの「加工蛋白質」というノードの内部節に「new name」というノードを追加した(「new name」の下位語には、「組織状蛋白質」と「蛋白質濃縮物」を選んだ)。

6.2 用語木の分割機能

用語木の分割を行うには、分割を行いたいノード(部分木の根)の上でポップアップメニューを表示させ、分割コマンドを選択する。すると、選択されたノードを根とする部分木が切り取られ、別のWindowに表示される。なお、分割コマンドは用語木の根および葉では実行できないようになっている。

分割の例を図9、図10に示す。ここでは、JICSTシソーラスの「東山」というノードを根とする部分木を切り取り、他のWindowに表示させた。

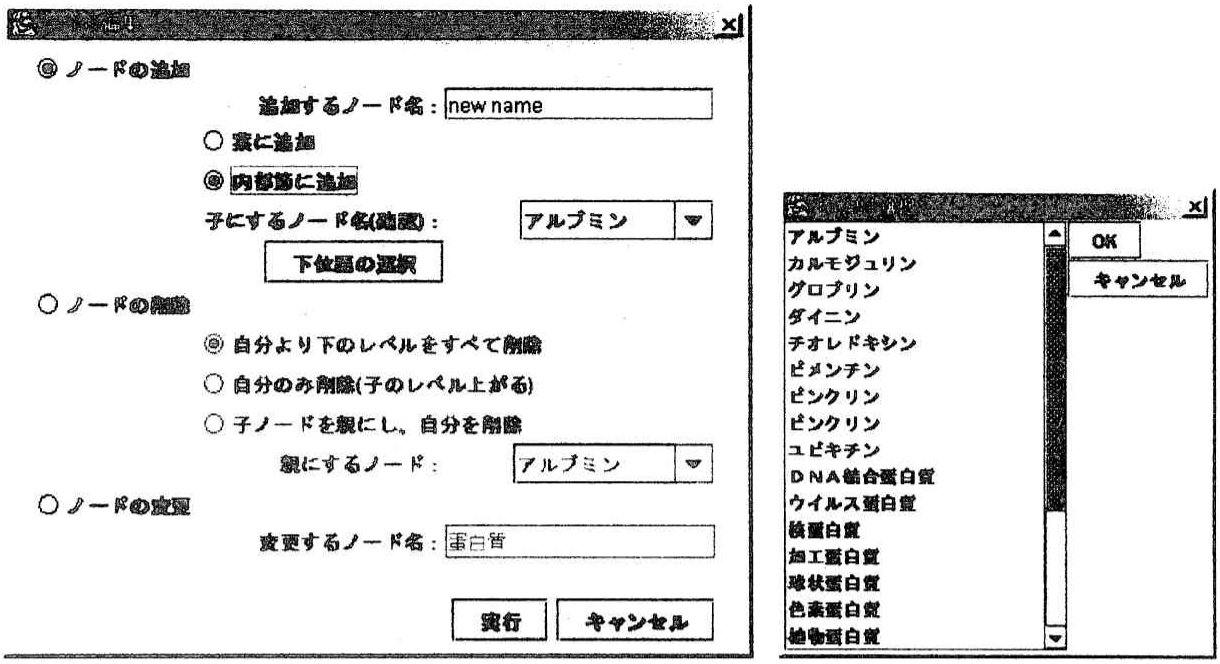

図7 編集ダイアログ・下位語の選択ダイアログ

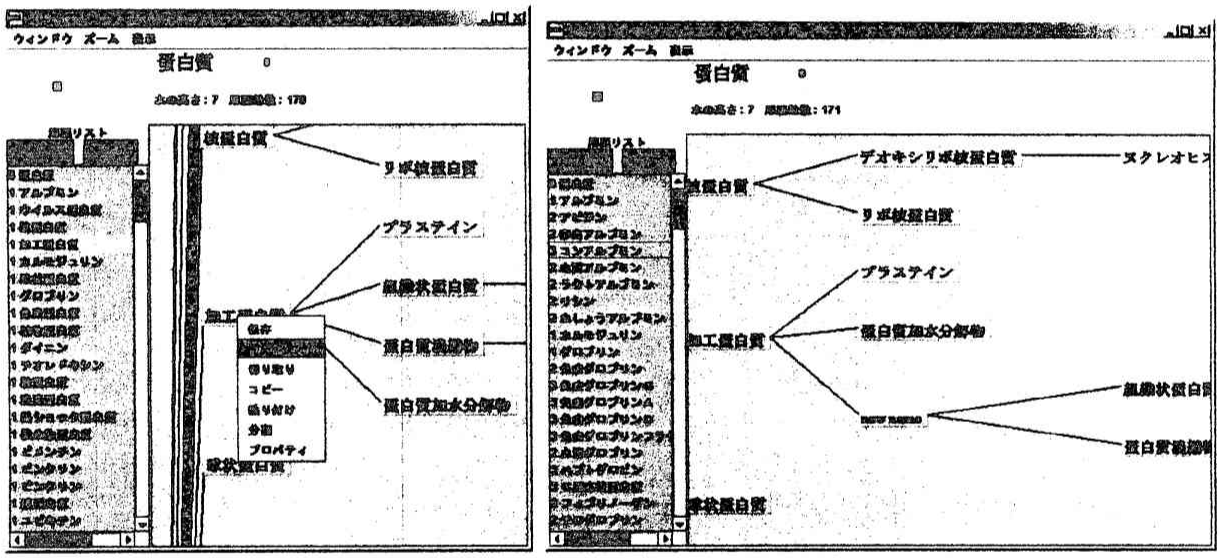

図8 ノードの追加例(追加前、追加後)

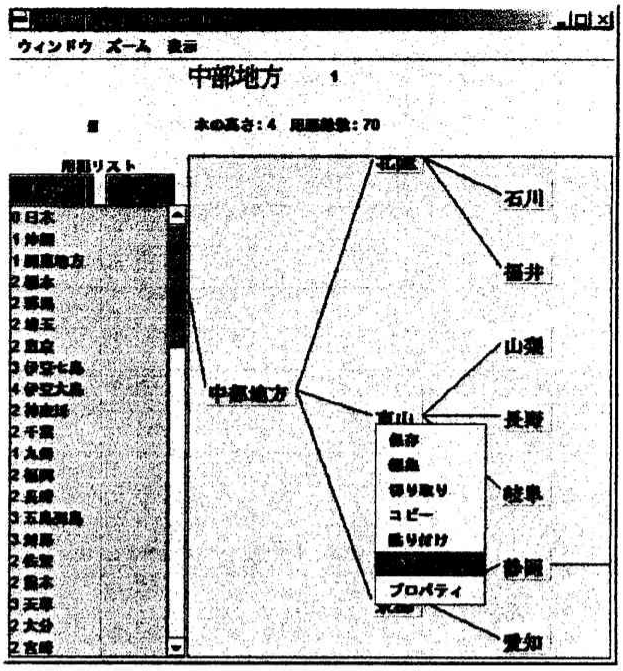

図 9 分割前の用語木

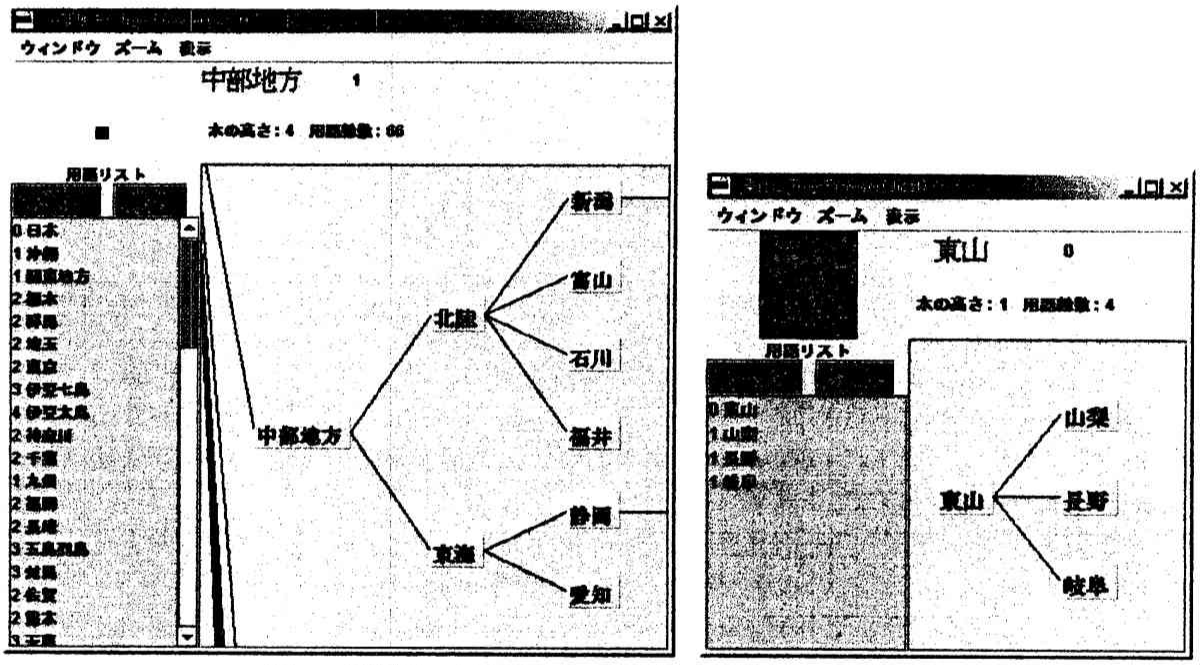

図 10 分割後の用語木(元の用語木、分割された用語木)

7. 用語木の自動表示機能

用語木の自動表示機能とは、あらかじめ保存しておいた用語が属する用語木を、一度にすべて表示する機能である。これにより、複数の用語木を少ない操作で表示することが可能になった。

自動表示機能を使用するためには、まず状態保存 Window のメニューバーの「オプション」を選択し、その中の「Tree の自動表示」という項目のチェックボックスを on にする(図 11)。

次に、「ファイル」メニューの「開く」を選択し、既に保存されている.tcd 形式のファイルを開く。すると、ファイルに保存されていた用語の属する用語木が次々と表示される。

例を図 12 に示す。このとき開いたファイルには、INSPEC の「ALLOYS」と「ZOOLOGY」、JICST の「中部地方」、「甲殻類」、および EDR の「走る」が保存されているが、「走る」には木構造が存在しないので用語木は表示されない。

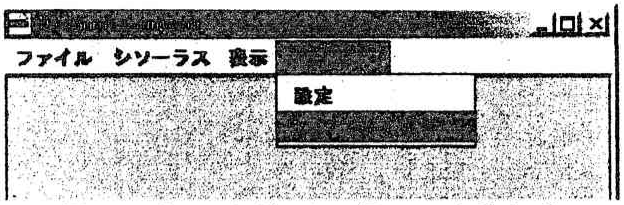

図 11：チェックボックス(Tree の自動表示)

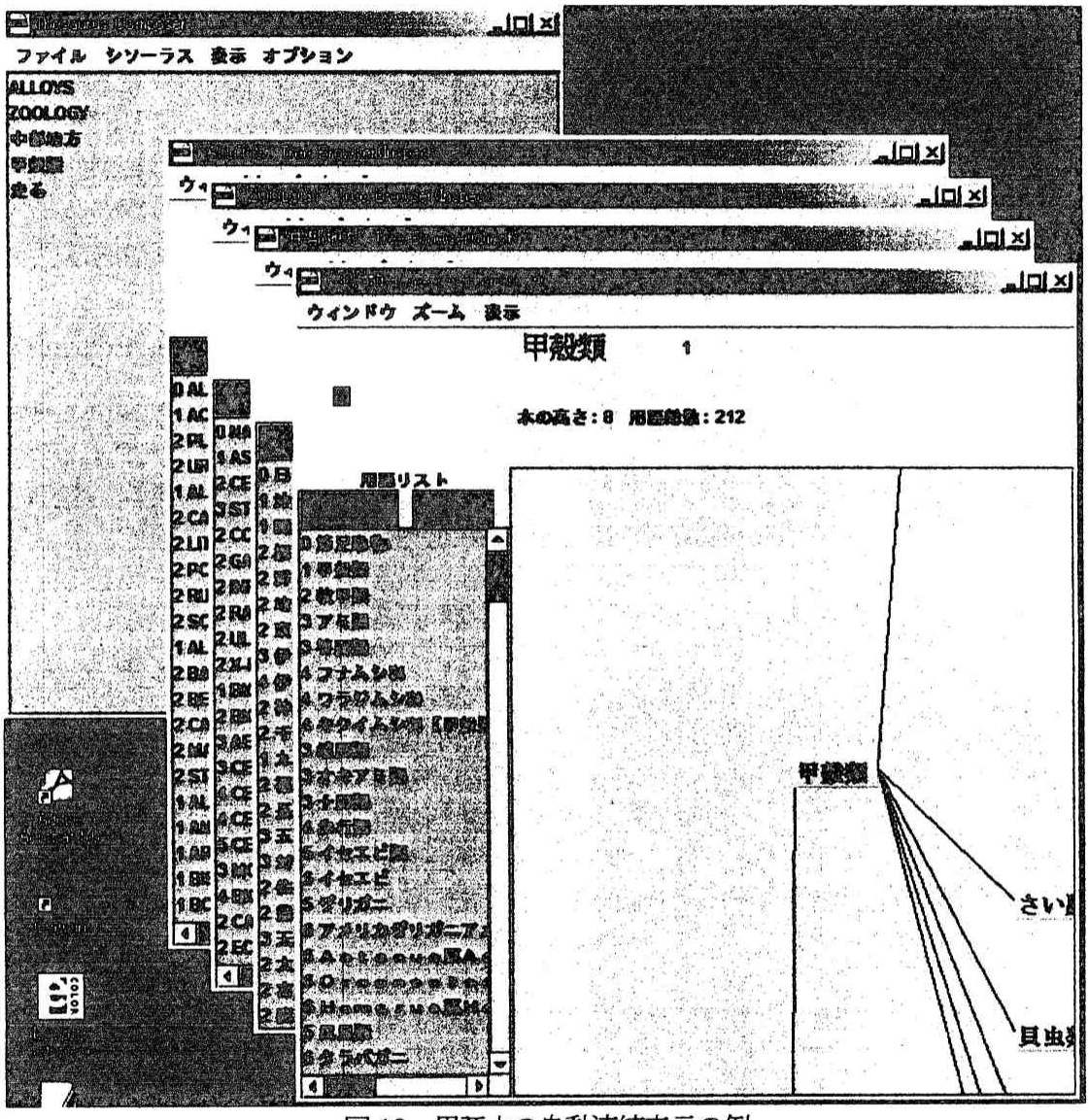

図 12：用語木の自動連続表示の例

8. 考察

8.1 用語知識ベースの XML DB 化による効率化

従来のブラウザーでは各辞書ファイルがそれぞれ異なる形式で記述されていたため、見出し語、見出し語 ID 等を区別するためにそれぞれの用語知識データごとに解析クラスを必要としていた。しかし、今回用語知識データを全て同一のタグ名で XML 化し、統一することにより、あるひとつのクラスのみで全ての用語知識データを解析することが可能となった。

また、今回、選択語の内容出力画面はＸＭＬデータの特性を生かし、XSL で HTML に変換したの

ちに表示している。従来の Swing で表示させる場合よりも、デザイン変更の際に手間がかからなくなった等の利点ができた。XSL ファイル等は[dat / dic / xml]にある。

8.2. ファイル入出力

今回のファイルの分割化により、リストの表示や、論理木の表示の際、処理速度が向上し、良い結果が出たが、反面分割化による問題も生まれた。それは概念関係ブラウザー表示の際(図1 GR選択時)、の関係子で結ばれている見出し語の探索方法である。階層関係データファイルにある関係する概念識別子の情報はすべて見出し語 ID で示されているため、その ID の見出し語を探索する場合、まず頭文字が何であるかが問題となる。ID だけではその情報は得られないため、現在 ID とその見出し語の頭文字情報を持つファイルを一つ作成し、そのファイルから頭文字情報を得た後、目的の用語ファイルから見出し語を探索するという方法をとっている。しかし、この方法では関係子を多く持つ用語などは見出し語の探索に時間がかかり、現時点では概念関係ブラウザーを表示できないものが多い。また、図 13 のように表示した論理木から概念関係ブラウザーを表示しようとした場合も同様の問題が発生する。

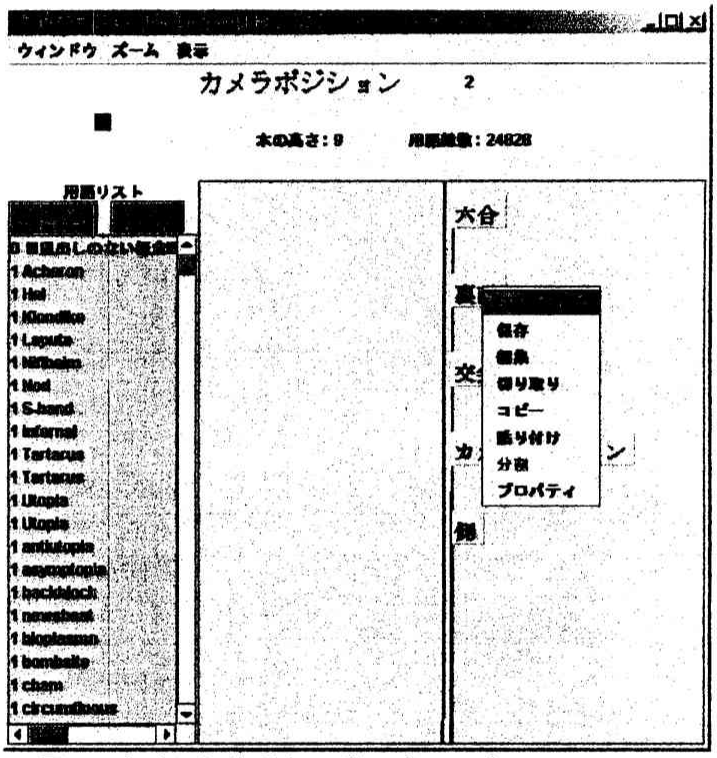

図 13 表示した論理木のポップアップメニュー

現在、辞書によっては、見出し語に漢字が含まれているもの、含まれていないものがあり、頭文字を指定する要素が見出し語であったり、読みであったりする場合がある。よって今後読みの無い辞書についても見出し語と同じ情報となるが「読み」情報を付加する必要があると思われる。

8.3. 用語木表示の高速化

本年度の改善により、巨大用語木を表示する際のメモリ使用量を大幅に軽減することに成功した。再描画を行う方法をとったこともあり、実行時間に影響が出ることも考えられたが、描画領域確保の方法の改善および用語木の走査方法の改善を行った結果、再描画の高速化に成功し、利用者に与えるストレスをより軽減することができた。特に前者の改善は、再描画にかかる時間の短縮のみならず、cpu およびメモリの使用量を軽減することにもつながり、非常に有用であることがいえる。それに対し、後者の改善は動作が安定せず、場合によっては従来と同じか、それ以上の時間がかかってしまうことがある。その原因の 1 つとして、走査方法のアルゴリズムが適切ではないことが考えられる。EDR シソーラスの用語木に多い、1 つのノードに対して数多くの下位語が存在するような部分木のある用語木に対しては今回のアルゴリズムは相性が良くない。なぜなら、部分木が縦に(縦型表示の

場合は横に)非常に長くなってしまうので、その一部が描画領域にかかる可能性が高くなり、結局のところ数多い下位語をすべて走査しなければならなくなってしまうからである。これを解決する方法として、部分木において描画領域内の下位語のみを走査していくアルゴリズムが考えられる。

用語リストに関しては、ソート機能が付加されたことにより目的の用語を検索しやすくはなったが、用語総数が 1 万を超えるような巨大用語木を考えるとまだ不十分である。これを解決する方法の 1 つとして、目的の用語(または、その見出しの一部)を直接用語リストから検索する方法が考えられる。

最後に、編集機能に関して概念関係ブラウザーと CPC VIEW システム[10]の比較を行った。概念関係ブラウザーは本年度の改善により関係子の編集および概念の追加・削除は可能となったが、追加できるのはあくまでも新規の概念で、既存の概念を追加することができない。これに対し CPCVIEW システムは逆で、既存の概念は cut & paste 等を用いることで追加できるが、関係子の編集および新規概念の追加は元のデータベースを編集するしかない。よって、編集機能に関しては概念関係ブラウザーのほうが良いと考えられるが、既存の概念の追加が出来ないことは問題である。

9. 終わりに

多方面での応用を考慮すると、用語知識ベース XML DTD はさらなる詳細化が必要とされよう。また、用語知識ブラウザーに必要な未実装の機能としては下記が挙げられる。

- 新規用語木の生成
- 木構造以外の多様な表示形式(エクスプローラー)への対応
- 概念関係ブラウザーのグラフィカル表示機能

参考文献

[1] 下村 央人 他. 汎用シソーラスブラウザーの試作. .第 33 回情報科学技術研究集会発表論文集,pp.99-105(1997).

[2] 下村 央人 他.『JAVA による索引支援ブラウザーの試作 2』.第 35 回情報科学技術研究集会予稿集,pp.133-137(1998).

[3] 後藤貴信、鈴木祐介、後藤智範. 階層構造をもつ用語データのためのBrowsing Tool. 情報知識学会第7回(1999年)研究報告会講演論文集. pp.57-60(1999).

[4] 渡辺基広、宮崎林太郎、後藤智範. 用語知識ベースのための編集機能の試作. 情報知識学会第10回(2002年)研究報告会講演論文集. pp.43-46(2002).

[5] Cay S.Horstman、Gary Cornell 共著.:Core JAVA Volume I, II, (株)アスキー

[6] David M.Geary 著. Graphical JAVA Volume I, II, (株)アスキー

[7] ここでは、「用語知識ベース」とは、CD-ROM などの電子媒体で提供されている、国語辞典、英和／和英辞典、専門用語辞典、シソーラス、また EDR、NTT 日本語語彙体系などの自然言語処理等コンピュータでの利用を目的とする用語データファイルを指している。

[8] EDR 電子化辞書. 日本電子化辞書研究所. 1995 年.

[9] NTT コミュニケーション研究所 監修. 日本語語彙体系. 岩波書店. 1999 年.

[10] CPCVIEW 説明書, http://www.jsa.co.jp/soft/seihin/CPCVIEW/doc/TreeView_doc.htm