卒業論文

知識構築のためのOPML変換自動化プログラムの開発

関西学院大学 理工学部 情報科学科
2656 伊達 万祥

2006年3月

指導教員　西谷　滋人　教授

概 要

知識を構築する過程には,「情報の収集」「情報の加工」「情報の発信」の３つの分野がある．それぞれの分野で現在さまざまなソフトウェアやシステムが開発されている．例えば,「情報の収集」では google などの情報検索サイト,「情報の加工」ではアウトラインプロセッサ,「情報の発信」では「Wiki」やブログなどのツールがある．知識を構築する過程では「情報の発信」と「情報の加工」の部分が非常に重要である．

Wiki，e-Learnig，ブログなどの情報発信ツールについて調べ，研究室での利用に適しているか検討した．その中でも特に Wiki に注目したが，マークアップ記法の習得やテキスト要素の直接入力などの問題から研究室での利用を断念した．

そこで,「情報の加工」の分野に着目し,「加工から発信まで」の過程の作業を円滑に行うことで知識構築の効率化が可能になるのではないかと考えた．本研究室で利用されていた，TAO という階層的に文章を記述できるアウトラインプロセッサとその TAO が提供する OPML フォーマットに注目した．

gnuplot ページのように階層構造で表示される HTML ページを作成し，その作業を自動化することを目的にした．まず，gnuplot ページと OPML データのソースコードから，タグの仕組みを読み解き，手動でタグの付け替えを行った．さらにその作業を自動化するためのプログラムを作成することを試みた．Python の XML::Parser を用いて，解析を行ったが，日本語の問題からテキスト要素の中の文字列を抽出できなかった．

Perl の XML::Parser を用いて，コマンドラインから出力したいページの数値と保存先を入力すれば，自動的にタグの付け替えを行い，TAO から書き出した OPML データを階層構造を持った HTML ページに変換出力してくれるプログラム MakeALL を作成した．

目 次

第 1 章　研究目的 **3**

第 2 章　開発目標と背景 **4**
2.1 知識構築モデル 4
2.1.1 研究と共同作業モデル 4
2.1.2 静的コンテンツと動的コンテンツ 4
2.2 ブログ 4
2.3 e-Learning 5
2.4 Wiki 6
2.4.1 Wiki の特徴 7
2.4.2 Wiki の効果 7
2.4.3 まとめ 7
2.5 gnuplot 8
2.5.1 右コンテンツ部 8
2.5.2 左メニュー部 8
2.5.3 考察 10
2.6 アウトラインプロセッサ 11
2.6.1 アウトラインプロセッサとは？ 11
2.6.2 アウトラインプロセッサの成り立ち 11
2.6.3 アウトラインプロセッサの利点 12
2.7 TAO 13
2.7.1 TAO とは？ 13
2.7.2 HTML 出力ファイル 13
2.7.3 OPML 出力ファイル 16
2.8 OPML 16
2.8.1 OPML とは？ 16
2.8.2 OPML の可能性 17
2.8.3 OPML フォーマット 17

第 3 章　開発ツール（XML ライブラリ） **21**
3.1 Python 21
3.1.1 Python における XML ソリューション 21

3.1.2 まとめ 25
3.2 Perl 26
3.2.1 Perl とは 27
3.2.2 XMLParser のインストール 27
3.2.3 XML::Parser の使用 27
3.2.4 XML::Parser の落とし穴 29
3.2.5 XML の生成 30
3.2.6 まとめ 30

第 4 章 開発プログラム **32**
4.1 MakeMenu.pl 32
4.2 MakeContent.pl 35
4.3 シェルプログラム 37
4.4 出力結果 38

第 5 章 まとめ **40**

第1章　研究目的

知識を構築する過程には,「情報の収集」「情報の加工」「情報の発信」の３つの分野がある．それぞれの分野で現在さまざまなソフトウェアやシステムが開発されている．例えば,「情報の収集」では google などの情報検索サイト,「情報の加工」ではアウトラインプロセッサ,「情報の発信」では[1]「Wiki」やブログなどのツールがある．知識を構築する過程では「情報の発信」と「情報の加工」の部分が非常に重要である．

本研究では，情報発信の元となる知識構築を効率的に行うため「情報の加工」の分野に注目した．本研究室では文章作成のときに「TAO」というアウトラインプロセッサを主に使っている．TAO は階層的に文章を記述でき，文書作成に優れたソフトウェアである．この TAO が提供している出力フォーマットのひとつである OPML (Outline Processor Markup Language) をソースに,「情報の発信」が容易となる HTML に変換するツールの開発を目的とした．

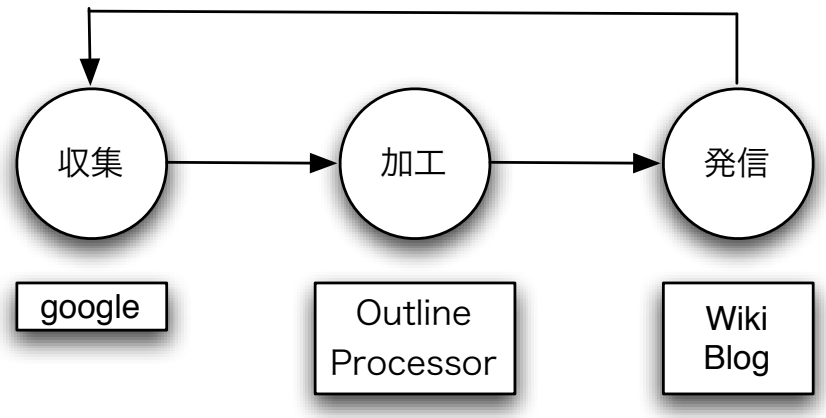


図 1.1: 知識構築の流れとそれぞれの分野のツール

第2章　開発目標と背景

2.1　知識構築モデル

2.1.1　研究と共同作業モデル

大学の研究室において，研究員の知識構築の効率的に行うためにどんな知識構築モデル，どんなアプリケーションが適しているのかを考えた．同じ研究室では，研究員の研究テーマが，似通っていることも多くある．その場合，研究を行うために必要な基礎知識が同じだったり，情報交換や共同作業を行うことは普通である．そこで，学生たちの毎日の作業公開の場，共同作業を行う場として利用できるツールを実装すれば，知識構築の促進を行えると考えた．ネットワークを介した共同作業ツールのモデルとして，以下の３つが知られている．

- 電子メールのやり取り（メーリングリストを含む）
- 共有フォルダ／ファイルへのアクセス
- 双方向のコンテンツ更新／アクセス

2.1.2　静的コンテンツと動的コンテンツ

静的コンテンツとして，たとえば，書籍と雑誌の２つが挙げられる．書籍は掲載する情報の正確性を，雑誌は即時性を重視しているが，どちらも記述された情報・知識を蓄積し，留めておくためのコンテンツである．

現在，さまざまな動的コンテンツが登場してきている．例えば，ブログ，e-Learning，Wikiなどが挙げられる．下記にブログ，e-Learning，Wikiの特徴を挙げて比較した．

2.2　ブログ

作者の個人的な体験や日記，特定のトピックに関する話題を時系列で比較的に頻繁に記録される情報についてのWebサイト全般を含む．このようなWebサイトの作成機能を提供するソフトウェアやサービスなどを指して呼ぶ場合もある

特徴

- それぞれの項目にタイトルがつけられて，時間軸やカテゴリで投稿を整理、分類する構造となっている．
- 主に管理者が記事を投稿する私的ニュースサイトの一面もある．
- 双方向性：管理者が書いたコメントに対して，他のユーザが新たにコメントを記入することができる．そのコメントを削除するかは，管理者が選択することができる．
- ブログは投稿する特定の方法に限定されないが，ブログ向けのソフトウェアがある．

2.3 e-Learning

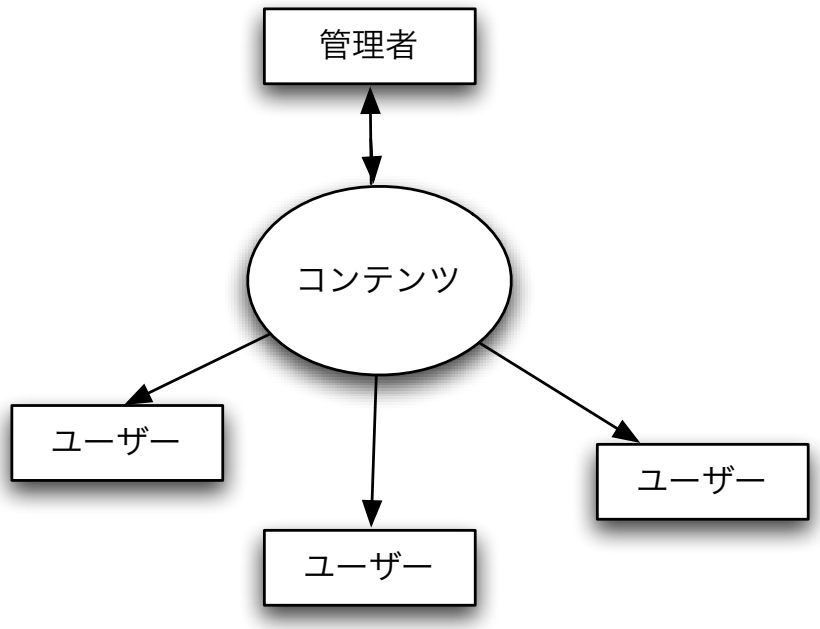


図 2.1: 一方向的な情報発信モデル (e-Learning)

パソコンやコンピュータネットワークなどを利用して教育を行うなうことである．最近では，各企業の個人研修や個人の資格取得の学習システムとして，幅広く用いられている．

特徴

- 教室で学習を行う場合と比べて，遠隔地にも教育を提供できる．
- コンピュータならではの教材が利用できる．
- 長年の経験を活かしたプロのテクニックがオンラインで学べる．
- 質疑応答，定期テスト採点などがよりスムーズに行われ，学習効率が上がる．

2.4 Wiki

WikiWikiWeb サーバーコンセプトは，Ward Cunningham が発案したもので，大抵の場合単に「Wiki」と呼ばれる．Wiki は，自由に拡張可能な，連結されたウェブ「ページ」の集積で，情報を格納し，修正するためのハイパーリンクシステムである．入力形態としてウェブブラウザ・クライアントを使うことで，各ページが誰にでも簡単に編集可能になるデータベースである．[2]

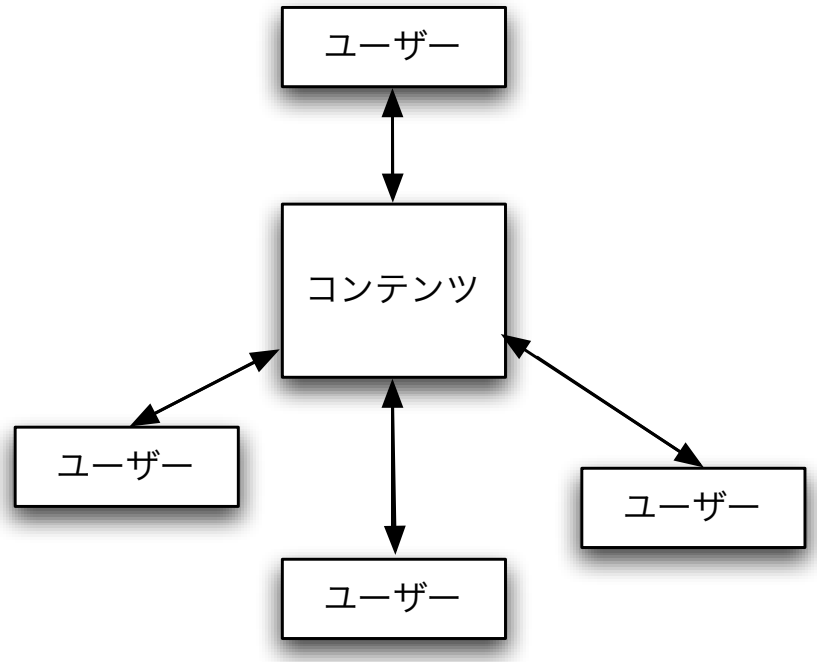


図 2.2: 双方向的な情報発信モデル (Wiki)

2.4.1 Wikiの特徴

- Wikiは，手早く相互リンクを行う手法に加え，単純なナビゲーションモデルを採用し,「１クリックで行える」というコンセプトの実現を促進する.（その手法のおかげで，少なくともWikiページ間でリンク切れがないことも保証されている.）
- ページ内容の編集は，必要なときに単純な「マークアップ」を用い，ウェブブラウザから文章を入力してやれば,「１クリックで行える」.
- 世界中の誰でも，何でも変更可能である（またそれを元に戻すこともできる）.
- Wikiは，高速な検索機能を内蔵している．ページタイトルがリンクになっているからである.

2.4.2 Wikiの効果

- 変更履歴リストを含む，たくさんの変更可能な入り口
- 新しい関係性を反映するページの相互リンクによる柔軟な再構築
- マルチスレッド化された，非線形の議論
- 書きやすさ，共同作業のしやすさ

2.4.3 まとめ

「研究と共同作業モデル」の節で書いたように大学での研究は，共同作業や情報交換を行う場が重要になってくる．e-Learningは，有能な管理者側の情報・知識を多数のユーザーに提供し，個々のユーザーがそのシステムに応じてある分野ごとの学習を行うには非常に優れたツールである．だから，ネットワーク上からなら，いつでもどこでも誰もが編集閲覧可能で，共同作業の場を提供してくれるWikiが研究室での利用に適しているのではないかと考えた.

しかし，Wikiにも研究室での知識構築促進のための利用に適しているかどうか，疑問が残る部分もあった．Wikiに文書を掲載するには，Wiki特有のマークアップ記法を覚える必要がある．それに文章をそのタグの中に一つずつ直接入力していかなければいけない．もし，日々の研究の成果や卒業論文などの長い文書を記述する場合なら，これは非常に面倒な方法である．文章内容や文章全体の構成を考えることにあまり集中できないからである.

では，文章全体の構成を考えながら，文章を作成することに集中でき，簡単に自分の作成した文書を情報発信してくれるツールを開発すれば，研究員の知識構築の促進に繋がるのではないかと考えた.

2.5 gnuplot

この「gnuplot」のページは，階層構造で記述されたHTMLページで，左メニュー部と右コンテンツ部から構成されている．このページのように階層構造で文章を表示できれば，あとで他の人が閲覧したときに非常に理解しやすいと考えた．知識データの提示例として，gnuplotから図2.3,2.4,2.5を引用した．[3]

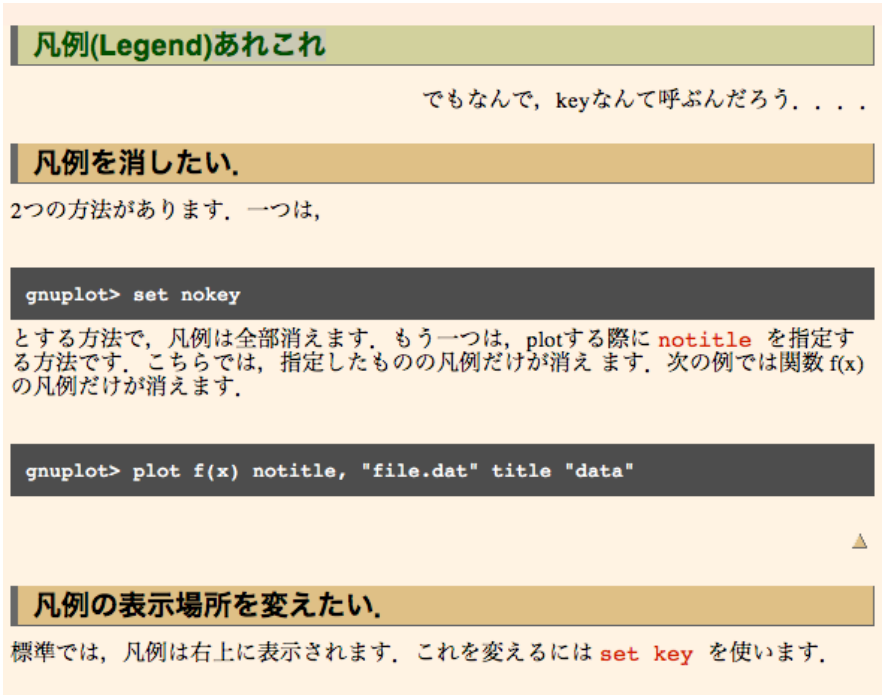
図 2.3: gnuplot/legend

2.5.1 右コンテンツ部

gnuglopt/legendページの右コンテンツ部には，レベル１トピック「凡例（legend)」のレベル２トピックごとに文章が記述されている．

2.5.2 左メニュー部

gnuplot/legendページの左メニュー部 not so FAQには，レベル別の階層構造で文章のタイトルが記述されている．



凡例(Legend)あれこれ

でもなんで，keyなんて呼ぶんだろう．．．．

凡例を消したい．

2つの方法があります．一つは，

```
gnuplot> set nokey
```

とする方法で，凡例は全部消えます．もう一つは，plotする際に `notitle` を指定する方法です．こちらでは，指定したものの凡例だけが消えます．次の例では関数 f(x) の凡例だけが消えます．

```
gnuplot> plot f(x) notitle, "file.dat" title "data"
```

凡例の表示場所を変えたい．

標準では，凡例は右上に表示されます．これを変えるには `set key` を使います．

図 2.4: gnuplot/legend の右コンテンツ部

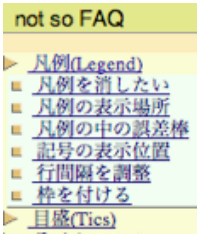


図 2.5: gnuplot/legend の左メニュー部

2.5.3 考察

メニュー部とコンテンツ部で構成される gnuplot は，階層構造を持つ HTML ページである．階層構造でタイトルが記述されているメニュー部と文章内容が記述されているコンテンツ部がある．まず，gnuplot のページから，無駄なタグを削除し，必要な部分だけを残した HTML ページを作成した．HTML ページとそのソースコードを以下に示しておく．このページのタグを読み解いて，OPML で書き出したファイルからタグの付け替えを行う．さらに，その変換を自動化できれば，TAO の階層構造を持ったページを容易に作成できると考えた．

図 2.6: gnuplot/legend の最小版

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML lang=ja><HEAD><TITLE>gnuplot / legend</TITLE>
<META http-equiv=Content-Type content="text/html; charset=utf-8"><LINK
href="gnuplot - legend.files/style-new.css" type=text/css rel=stylesheet>
<META content="MSHTML 6.00.2900.2769" name=GENERATOR></HEAD>
<BODY>

<TABLE cellSpacing=0 cellPadding=0 width="100%" border=0>
<COL width="30%">
<COL width="70%">
<TBODY>
<TR>
<TD id=menu>
<P></P>
<UL>
<LI>状態図とは
```

```
<LI>状態図の定義
<UL>
<LI>『物質と材料」の違い
</UL>
</UL>
<BR>
</TD>
<TD id=content>
<H1>状態図とは</H1>
<H2>状態図の定義</H2>
<P>成分濃度と温度を軸にとり、その合金に用いられる成分がどのような状態変化（固相、液相）をとるかを目で捉えることのできる図。</P>
</TD></TR></TBODY></TABLE>
</BODY></HTML>
```

2.6 アウトラインプロセッサ

ここではアウトラインプロセッサの必要性について，記述しておく．[4]

2.6.1 アウトラインプロセッサとは？

アウトラインというのは骨子，輪郭という意味で，文章作成の話のときに使うときは，まさに文章の基本となる骨組みのことである．アウトラインプロセッサとは，この骨組みの扱いに長けたソフトのことである．アウトラインプロセッサは，レベル別の階層構造によって文章を作成，表示することができる．

2.6.2 アウトラインプロセッサの成り立ち

おそらく，最初のアウトライン･ソフトウェアは，1960年代にAugmentシステムの一部としてDougEngelbartによって開発されたものである．Living Videotext(1981-87)は、PC用のポピュラーなアウトライン・ソフトウェアをいくつか開発した．それらは，現在Userlandのウェブサイトoutliners.coに保管されている．1992年に創業したFrontierはアウトライン・ソフトウェア周りの開発を行っている．Frontierのテキストエディタ・メニューエディタ・スクリプトエディタは，アウトライン・ソフトウェアであると共に，オブジェクト・データベース・ブラウザでもある．

2.6.3　アウトラインプロセッサの利点

全体を見渡しながら長文執筆ができる．

アウトラインプロセッサの利点の一つは，長文の作成が容易になることである．長い文章を書くときには、どのような章立てにするかを考えることが大変重要であるが，書いている最中には，極めて狭い部分に意識が集中してしまい，全体の構成から文章の内容が外れたことを書いてしまいがちである．アウトラインプロセッサでは，段落の先頭にあるグレーの三角をクリックすることで，トピックを一時的に隠すことができるので，全体の構成を見渡すことができる．

エディタモードにして，文章を入力し，アウトラインメニューから「子トピックを作成」を選んでみると，一段下がった位置に新しい三角が現れる．この行は，ひとつ上のトピックの子供という扱いで，上のトピックをドラッグして移動すれば，子トピックもいっしょに移動することになる．

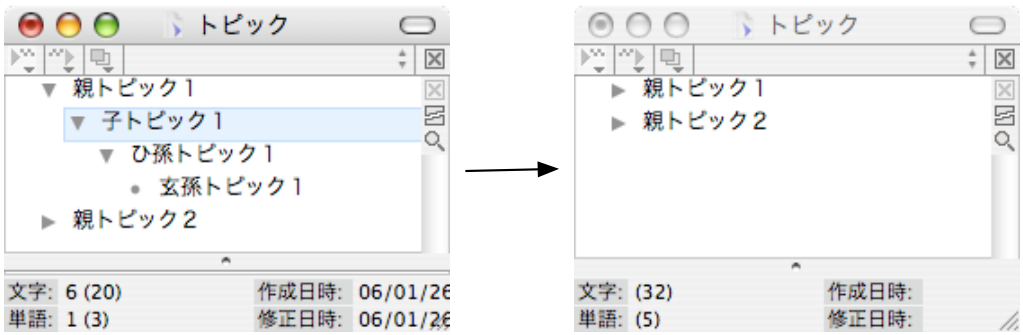


図 2.7: 階層レベルごとのトピック

ここで，親トピックの三角をクリックすると，子トピックが見えなくなってしまう．でも，子トピックが削除されてしまったわけではなく，一時的に隠されているだけである．これを利用して，今，執筆するところだけを開いて，その他のトピックを閉じてしまえば，常に全体の構成を見渡しながら執筆を進めることができる．

書き進めながら，全体の構成を変えることができる．

アウトラインプロセッサの二つ目の利点は，執筆を進めながら，全体の構成を変更したくなったとき，トピックを移動させることができることである．

移動させたいトピックをクリックして，ツールバーの「上へ移動」「下へ移動」のボタンをクリックすると，トピックが移動する．このとき，子トピックもいっしょに移動されるので，章を入れ替えるということが実に簡単にできるようになっている．

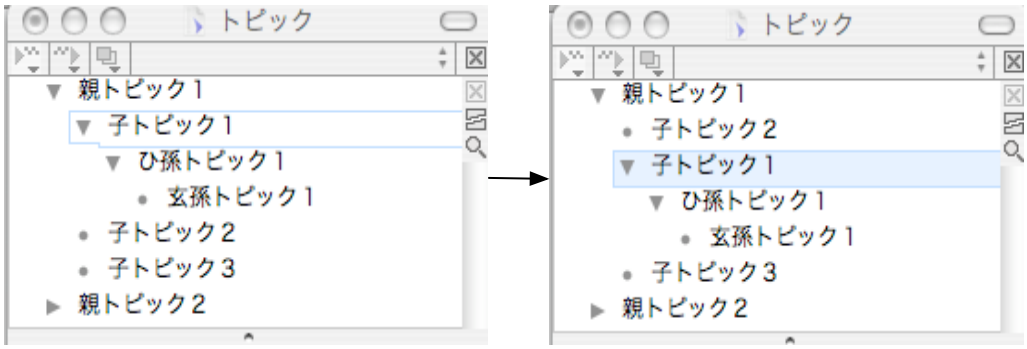


図 2.8: 章の入れ替え

2.7 TAO

2.7.1 TAO とは？

アウトラインプロセッサの一つで，本研究室で使われている．階層的に文章を記述でき，文書作成することに非常に優れたテキストエディタである．図 2.9 は，状態図に関して，伊達が TAO で作成した文書の一部である．

2.7.2 HTML 出力ファイル

他の研究員が TAO で作成した文書をいつでもどこでも閲覧できるようにするために，ウェブ上に掲載できる HTML 形式でファイルを出力保存することを試みた．

手順

- TAO で文章を作成保存する．
- 書き出しで HTML ファイルを出力する．

図 2.9 は，TAO ファイルを書き出した HTML ページである．図 2.9 は，文章が階層的に表示されている．しかし，すべての階層レベルの文章が展開されてしまっている．アウトラインプロセッサの不必要な文章を隠せるという機能がなくなってしまっている．短い文書の場合は，不具合は出ない．しかし，図 2.10 のように文書が長い場合，全体の構成が掴みにくく，文章内容を理解することが難しい．やはり，TAO から書き出した HTML ページでは無理がある．図 2.11 は，HTML ページのソースコードである．

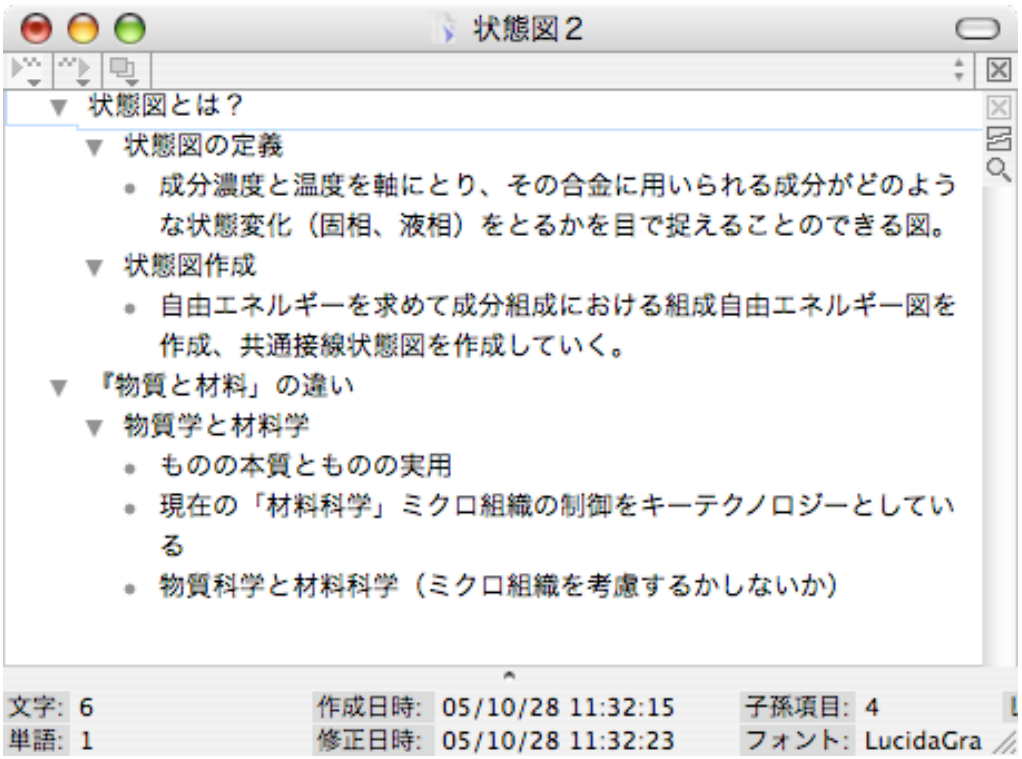


図 2.9: 「状態図」に関するファイル（TAO）

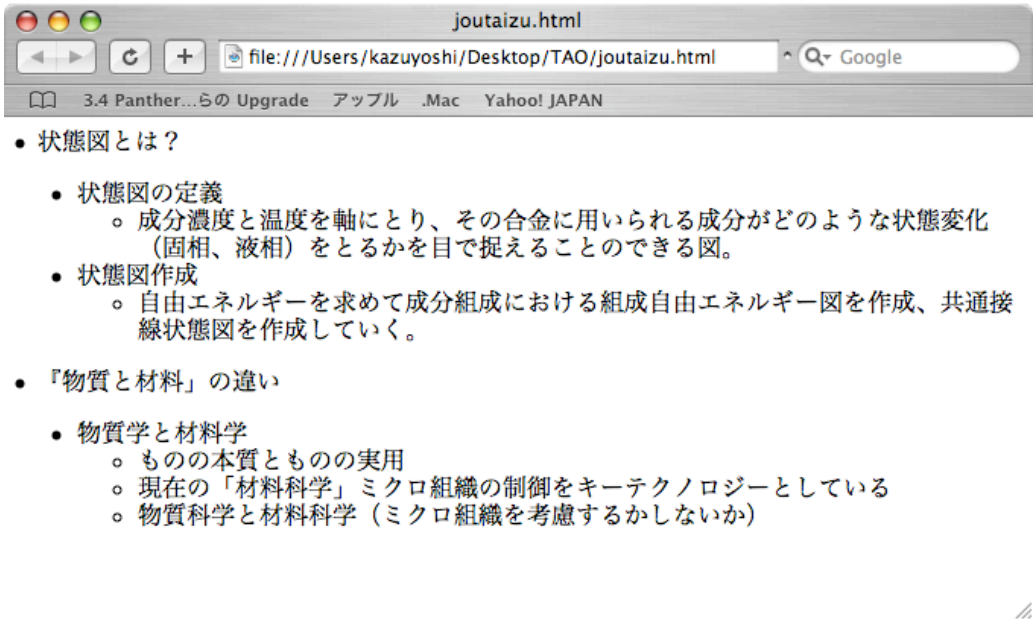


図 2.10: 状態図のファイル１（HTML 出力）

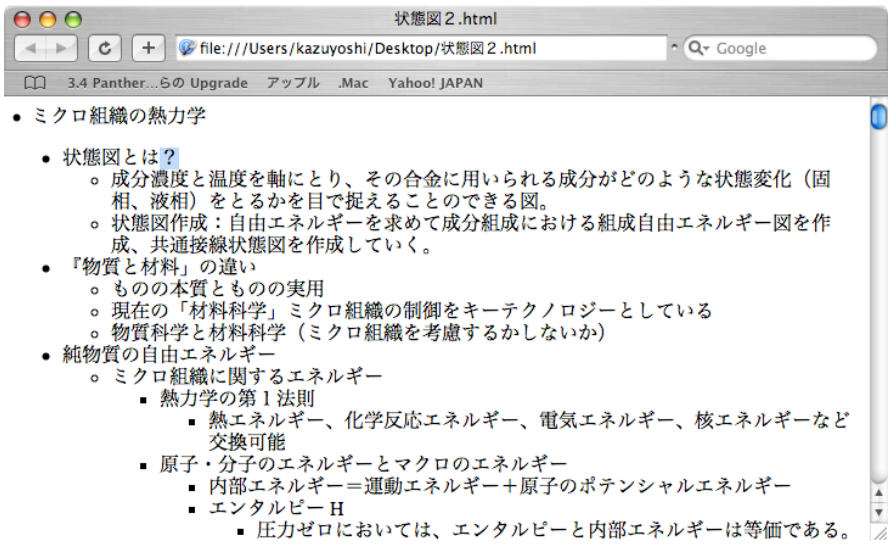


図 2.11: 状態図の HTML ファイル２

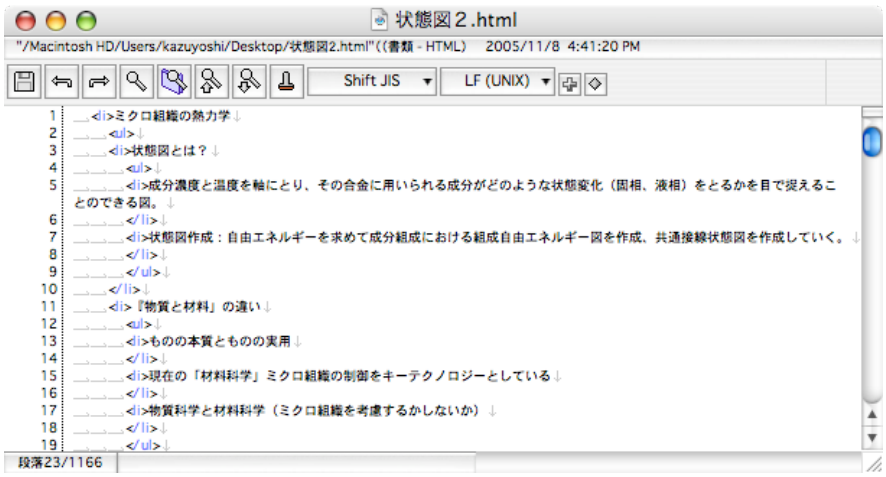


図 2.12: 状態図のファイル３（HTML ソース）

2.7.3 OPML 出力ファイル

TAO のファイルを OPML で書き出し，出力保存した．図 2.13 が OPML ファイルである．

図 2.13: 「状態図」に関するファイルの簡易版（OPML）

2.8 OPML

TAO が提供する出力フォーマットの 1 つである OPML について記述しておく．[5]

2.8.1 OPML とは？

OPML とは，Outline Processior Markup Language の略で，文書のアウトライン構造の情報を異なる OS や異なる環境で交換できることを可能とするための XML 規格である．[1]RSS アグリゲーターなどがサイトの一覧なんかを表現する場合によく用いられている．このフォーマットの目的は，アウトライン・ソフトウェアでアクセスできるインターネット上のサービスとアウトライン・ソフトウェアとの間で情報をやり取りする方法を提供することである．

例えば，SharpReader や NewsGlu などでは OPML に対応している．これで巡回リストのインポート/エクスポートが可能になっている．例えば，SharpReader から NewsGlue に乗り換えたときでも，SharpReader が吐き出した OPML を NewsGlue に読み込ませることで，サイトの一覧を簡単に移行できる．以下の URL のホームページから参照した．

XML とは？

ウェブページから情報を抽出する際の問題点の１つは，必要とするデータがページのどこにあるか知ることが難しかった．それをするためには，HTML ファイルを丹念に調査し，必要なデータを囲むタグが何か見つけ出すことが必要だった．つまり，ページのデザインが変われば，毎回プログラムを書き直す必要があった．

こうした問題を解決するために XML が設計された．XML は EXtensible Markup Language の略である．実際には，XML はマークアップ言語ではなく，特定のタスクによりよくフィットするマークアップ言語を定義する方法である．動作には，Document Type Definitions（DTD）のシンタクスを定義する必要がある．DTD は文書内に出現するエレメントのセットとともに，アトリビュートや相互関係を定義する．各エレメントが必須か省略可能か，定義された出現順があるか，各エレメントが他のエレメントを内包することができるか，などを定義する．

2.8.2 OPML の可能性

OPML フォーマットの設計において目指すものは，様々なデータを容易に閲覧・編集できるように見通しの良いシンプルさ，ドキュメントの内蔵化，拡張性を備え人間に読めるフォーマットであることである．これは公開されたフォーマットであるということが意味するのは，他のアウトライン・ソフトウェア販売会社やサービス会社が Radio Userland との互換性のためやその他の目的でこのフォーマットを自由に使うことが出来るということである．

2.8.3 OPML フォーマット

OPML とは

<opml>は，version という属性のみが必要な XML 要素である．その配下には head 要素と body 要素の両方が必要で，version 属性は”x.y” という形のバージョン番号を示す文字列である． x と y のどちらも数値である．以下に OPML フォーマットを記す．

[1]RSS:ウェブページのメタデータ形式のひとつで，日付と 1 行のヘッドラインからなる，ニュース，日記，メールなどサイトの概略を表現する新しい XML 規格で，米国を中心に急速な広がりを見せている．

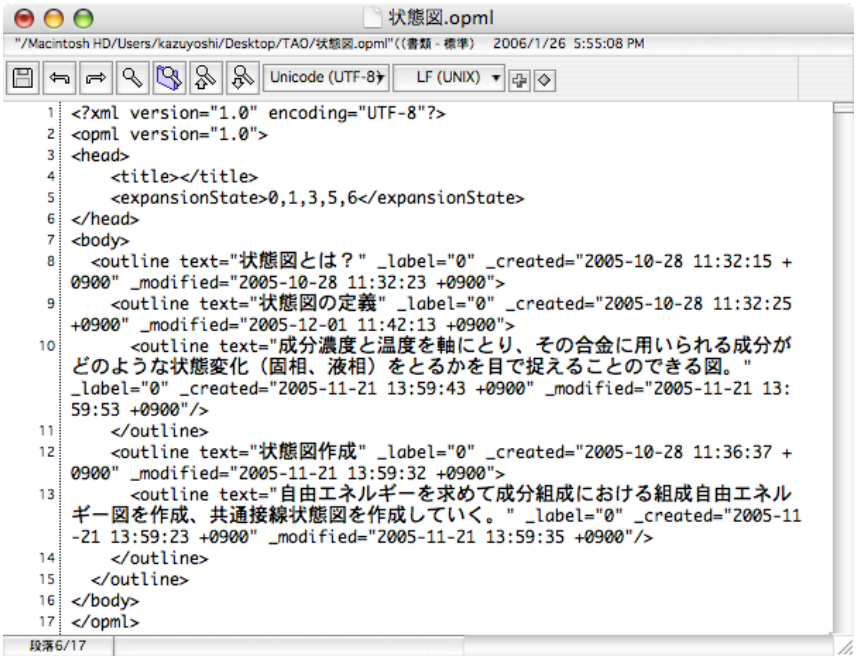


図 2.14: 「状態図」に関するファイル（OPML）

head とは

- <head>は，以下に示される 0 以上の必須ではない要素から成る．
- <title>は，その文書のタイトルである．
- <dateCreated>は，その文書の作成された日付け及び時刻である．
- <dateModified>は，その文書の更新された日付け及び時刻である．
- <ownerName>は，その文書のオーナーの名前である．
- <ownerEmail>は，その文書のオーナーの電子メール・アドレスである．
- <expansionState>は，展開する行番号のカンマ区切りのリストである．この行番号はどのヘッドラインが展開されるかを示す．順序は重要で，リストのそれぞれの構成要素”X”は、一番最初のもの (一番最初の<outline>) から始まり，X 回フラットに辿って (単純に上から数えてということ) 展開する．リストのそれぞれの構成要素ごとに繰り返される．
- <vertScrollState>は，数値でどの<outline>をウィンドウの一番上に表示するかを示すものである．expansionState がすでに適用されているものとして、計算されます．

- <windowTop>は，数値でウィンドウの上端の座標を示す.
- <windowLeft>は，数値でウィンドウの下端の座標を示す.
- <windowBottom>は，数値でウィンドウの左端の座標を示す.
- <windowRight>は，数値でウィンドウの右端の座標を示す.

head における注意点

- すべての<head>サブ要素は，アウトラインプロセッサに無視される可能性がある．もしアウトラインが他のアウトライン内で開かれた場合，アウトラインプロセッサは windowXxx 要素を無視する必要がある．これらの要素は，アウトラインがそれ自体のウィンドウで開かれた場合に位置とサイズを制御するものだからである.
- すべての日付け及び時刻は RFC822 の Date and Time Specification に従って記述する.
- OPML クライアント・ソフトウェアで OPML 文書を読み込んだとき，expansionState 要素を尊重するかどうか選択できる．expansionState 要素は，ある状況で必要になることがあるので存在する.

body とは

<body>は一つ以上の outline 要素を含む.

outline とは

<outline>は，一つ以上の属性を持つこととサブ要素として任意の数の outline を含む.

一般的な属性

- text は，アウトライン閲覧時や編集時に表示される文字列である．仕様において，text 属性の長さに制限はない.
- type は，文字列で<outline>のほかの属性をどう解釈するべきかを示す.
- isComment は，”true”または”false”という文字列で，そのアウトラインがコメントかどうかを示す．アウトラインがコメントである場合，すべてのその配下のアウトラインを同じようにコメントと見なす．省略された場合は”false”とみなす.

- isBreakpoint は，”true”または”false”という文字列で，アウトラインにブレークポイントが設定されているかどうかを示す．この属性は，主に実行スクリプトの編集に使用されるアウトラインのためのものである．省略された場合は”false”と見なす．

互換性

1.0 のフォーマット以前では，最上位の要素は outlineDocument とされる．Radio UserLand はこれからもこのような文書も読むことが出来ると考えられる．

制限事項

含まれる outline 要素の数や含むことの出来る outline 要素の数に，文書上の制限はない．

注意事項

OPML は，文書フォーマットでプロトコルではない．HTML 文書内のリンクをクリックしても，サーバー上にある文書に何らかの変更が加わることはない．OPML もそれと同じ様に扱われる．

一般的に HTTP を通じて，OPML 文書にアクセスした場合の mimetype は text/xml である．このことはウェブブラウザが XML の表示形式に従って OPML 文書を表示してかまわないということである．Radio UserLand に内蔵されている HTTP サーバは，HTTP リクエストの Accept ヘッダを参照して，どうやって OPML 文書を送るかを決定する．もし，Accept ヘッダに text/x-opml を解釈できることが示されていた場合，Radio UserLand はそのままの XML テキストを返す，そうでない場合は，Radio UserLand は mimetype を text/html として，アウトラインテキストを返す．

第3章　開発ツール（XMLライブラリ）

実際にプログラミングを行っていく際に利用したXMLパーサについて報告する．

3.1　Python

3.1.1　Pythonにおける XMLソリューション

まず，TAOから書き出し，OPMLファイルからXML::Parserを持つモジュールを用いて，文書を解析し，文書の開始タグと終了タグとその中の要素を抽出する．さらにタグの付け替えを行うようにプログラムを作成することを目的とした．そこで，Pythonが提供しているXML::Parserを用いることにした．

Pythonの XMLサポートは数あるソリューションの中でももっとも複雑な部類に入る．その主な理由は，各種XMLパーサの開発の経緯が複雑になってしまったことにある．Python1.5.2で提供された最初のXML解析システムはxmllibであり，すべてのPythonディストリビューションに標準装備されている．このxmllibは，SGML解析ツール群を提供するsgmllibモジュールと開発基盤を共有する．

xmllibパーサはシンプルな検証用パーサの機能とイベント駆動型のデータパーサの機能を兼ね備えており，XML文書の解析に使用できる基本的なメソッド群を持っている．このクラスには，開始タグ，終了タグ，データセクション，エンティティなど取り出すために必要なメソッドが用意されている．

Python2.0では，XMLベースの開発に使用するモジュールとパッケージのまったく新しい階層が導入された．基本的なxmlパッケージには，DOMを使って処理するためのxml.dom，イベント駆動型のパーサを提供するxml.sax，他の多くの言語で使用する用のExpatパーサに対するインターフェースを提供するxml.parsers.expatが含まれている．xmllibモジュールも標準Pyhtonライブラリの一部として残っているが，上位のxml.saxパッケージによって置き換えられるため，使用とサポートは非推奨扱いになっている．[6]

xmllib モジュール

XML 規格全体をサポートするという思想に基づき，文書を読み取るときに文書構造の基本的なチェックを実行するようになっている．たとえば，開始タグと終了タグがきちんとペアになっているか，文章に１つのトップレベル要素があるか，といった基本的なチェックである．

xmllib は，XML 文書を解析するための標準的なイベント駆動型の xml.sax の影に隠れてしまったが，xmllib が全く役に立たないわけではない．もっとも，xmllib 今後の Python リリースで更新されないのは事実なので，自動システムでの使用は避けた方がいいだろう．

xmllib はもはや理想的なモジュールとは言えなくなったが，旧バージョンの Python インタープリンタを確実にサポートしたい場合にはこれが唯一のソリューションになる．Expat，SAX，DOM の各ソリューション，これらはいずれも基本的に同じ仕組みになっている．シンプルな xmllib を見ておけば，その他のシステムの仕組みも理解しやすい．以下のプログラムは，最もシンプルな XML::Parser である．

シンプルな XML::Parser

```
import xmllib,sys


class MyParser(xmllib.XMLParser):

def __init__(self,filename=None):
xmllib.XMLParser.__init__(self)
if filename:
self.Loadfile(filename)

def Loadfile(self,filename):
xmlfile = open(filename)
while 1:
data = xmlfile.read(1024)
if not data:
break
self.feed(data)
self.close()

def unknown_starttag(self,tag,attrs):
print "START: ",tag,attrs
```

```
def unknown_endtag(self,tag):
print "End: ",tag

def handle_data(self,data):
print "Data: ",data


try:
filename = sys.argv[1]
except IndexError:
print "You must supply a filename"
sys.exit(1)

try:
parser = MyParser(sys.argv[1])
except EOFError:
pass
except xmllib.Error,data:
print "There was an error in the XML:",data
sys.exit(1)
except:
print "Something went wrong"
sys.exit(1)

print "Everything seems fine"
```

Expat を使った解析

Expat は，James Clark 氏が C で作成した非検証型の XML パーサである．xmllib と同じく（SAX とも同じように）イベント駆動型で，XML の個々の内容を解析し，コールバックを使用して個々の開始タグ，終了タグやデータ部分を処理する．

Python で Expat を使用するには，xml.parsers.expat モジュールをインポートする必要がある．xmllib の場合と出力が少し違っている点は，Expat パーサが ASCII 文字列ではなく Unicode 文字列を処理することである．

順次処理の XML 解析を極めた最高のソリューションは Expat パーサである．このパーサは，Python のバージョン 2 以降のディストリビューションには標準装備されており，さまざまな言語でサポートされているおなじみのイベント駆動型インターフェースが用意されている．

SAX を使った解析

SAX(Simple API for XML）インターフェースはもともと Java で開発されたが，現在ではほとんどの言語でインターフェースが用意されている．Python 2 は SAX バージョン 2（略して SAX2）をサポートしており，さまざまなインターフェースの機能が提供されている．Python では，SAX パーサに対する基本インターフェースのほかに，例外処理システム，SAX ハンドラを作成するための一連の基本クラス，独自の下位レベル SAX パーサを作成するための下位レベルインターフェースも用意されている．

SAX の働きからすれば，文章中を進みながら特定の要素を読み取っていくような状況には SAX がとくに適している．たとえば，特定のタグやデータセクションを DOM テクニックよりもシンプルな方法で検出するためのトリガをインストールできる．

また，文書を順次処理して別のフォーマットに変換するときにも SAX は適している．元の XML ソースから各要素を抽出した時点で各要素を処理していくという流れになるからである．

Python の SAX モジュールは Expat システムや xmllib システムと基本的に同じ働きをするので，SAX への移行は難しくない．SAX のメリットは何と言っても XML の規格制定団体による標準規格だという点だろう．したがって，Python のインターフェースにも他の言語のインターフェースにも情報やイベントを渡すことができる．

DOM を使った解析

DOM（Document Object Model）では XML 文書をツリー構造として表現できる．実際，文書全体が一連のオブジェクトとしてまとめられているので，ツリーのブランチ（枝）をたどることによって文書全体にアクセスできる．DOM では XML 文書が 1 つのかたまりとして表現されるので，既存の文書の解析にも新しい文書の作成にも DOM を利用できる．

DOM の唯一の問題は文書全体をメモリ内に格納するということである．小さな文書では問題にならないが，512K 倍との文書を DOM オブジェクトとして内部に格納するとその 5 倍のメモリを必要とする場合もあるので，問題である．もちろん，Python ではメモリの割り振りを心配する必要がないが，逆に言えば，自分でも気づかないうちに大量のメモリを消費するおそれがある．

Python の DOM インターフェースは W3C がリリースした仕様の IDL 版をベースにしている．標準の Python2.x ディストリビューションには，基本的な DOM 解析システム（minidom）と，さらに複雑なシステム（pulldom）が組み込まれている．pulldom の場合は，DOM ツリーから個々の要素を抽出するときに XML 文書全体をメモリ内に読み込む必要がないという特長がある．

Python のオブジェクトシステムは柔軟性が高いので，XML 文書を Python オブジェクト内にそっくり映り出したツリー構造の等価物を作成することは非常に簡単である．それに加えて，使いやすいオブジェクト処理機能を活用すれば，XML 文書を処理するための優れたプラットフォームが手に入ることになる．

柔軟性が高い選択肢は DOM システムを使う kotodearu.Dom オブジェクトを作成するための基本パーサとして SAX が使われているが，いったん XML 文書を DOM フォーマットに変換してしまえばその中のタグやデータにはさまざまな方法でアクセスできるようになる．文書内の要素に名前でアクセスすることもでき，必要なら，テキストを操作しないで文書の一部を置き換えたり作成し直したりすることもできる

3.1.2 まとめ

Python による XML 処理は比較的簡単である．処理の型さえ決まれば，あとは対応するモジュールをインポートし（場合によっては対応するモジュールからクラスを継承し），XMl 文書のデータを取り込み，使用するクラスの入力としてそのデータを渡すだけのことである．

上で挙げた Python の４つのモジュールで XML 解析を試みたが，大きな問題があった．OPML ファイルの中からタグはうまく抽出することができた．しかし，Python の XMLParser では日本語の文字列を取り扱うことができなかったため，outline タグの中の text 要素の文字列をうまく抽出できなかった．そのため，解析には Perl の XML::Parser を用いることにした．

下記に Python プログラムの実行結果を載せておく．TAO から書き出した OPML データから XML::Parser によって抽出したタグと text 要素が記述されている．

実行結果

```
Data:

START:  opml {'version': '1.0'}
Data:

START:  head {}
Data:

START:  title {}
End:  title
Data:
```

```
START:  expansionState {}
Data:  0,3,4
End:  expansionState
Data:

End:  head
Data:

START:  body {}
Data:

START:  outline {'text': '\x8f\xf3\x91\xd4\x90}\x82\xc6\x82\xcd\x81H'}
Data:

START:  outline {'text': '\x8f\xf3\x91\xd4\x90}'}
Data:

End:  outline
Data:

START:  outline {'text': '\x8f\xf3\x91\xd4\x90}\x8d\xec\x90\xac'}
Data:

End:  outline
Data:

End:  outline
Data:

End:  body
Data:

End:  opml
Everything seems fine
```

3.2 Perl

Pyhton では，日本語の文字列の取り扱いに問題があったため，Perl でプログラムを書くことを試みた．ここでは，Perl について記述しておく．

3.2.1 Perl とは

Perl とは Practical Extarction and Report Language（実用データ取得レポート作成言語）の略で，Pathologically Ecletic Rubbish Lister（病的主義のがらくた出力機）の略でもある．[6]

3.2.2 XMLParser のインストール

http://sourceforge.net/projects/expat/より，1.95.8 をダウンロードする．
Perl モジュール「XML::Parser」のインストール (要管理者権限、ファイルは自動取得)

```
$ sudo perl -MCPAN -e shell
cpan> install XML::Parser
cpan> exit
```

その他の情報

http://pcweb.mycom.co.jp/column/osx/120/の fink からインストールする．perllibCPAN の Config.pm を削除し、Config.pm を Config.pm としてから、中身の 1 を 0 に変更すると、再設定可能である．

```
fink updata-all
fink install expat
fink install xml-parser-pm

[BobsNewPBG4:~/.cpan/build/XML-Parser-2.34]
bob% perl Makefile.PL EXPATLIBPATH=/sw/lib EXPATINCPATH=/sw/include
Checking if your kit is complete...
Looks good
Writing Makefile for XML::Parser::Expat
Writing Makefile for XML::Parser
```

3.2.3 XML::Parser の使用

XML::Parser は，Jame Clark 氏が開発した Expat という XML 処理ライブラリの上に構築されている．もっとも一般的なものは CPAN モジュールの XML::Parser である．このモジュールは Expat という XML パーサをベースにしている．Expat は nonvalidating パーサであり，よって Perl では，おもに文書の well-formed 性のみに興味を持つことになる．XML::Parser は，Perl で XML を処理するための不可

欠なコンポーネントである．Perl の他のほとんどのモジュールは，XML::Parser が提供している機能を利用してそれぞれもモジュールに固有の処理をサポートしているからである．

XML::Parser そのものは Expat ライブラリを利用したイベントベースのパーサなので，XML 文書が整形式かどうかチェックする簡単な検証機能も提供している．ただし，文書を DTD に対して検証することはできない．

このパーサに対するインターフェースは簡単である．新しい XML::Parser オブジェクトと，パーサが XML 文書の要素開始，要素終了，およびデータ部分を判別したときに呼び出される一式の関数を作成する．たとえば，下のコードは，文書内の開始タグと終了タグを出力する，非常にシンプルな XML パーサを構築している．

XML::Parser の動作は HTML::Parser に似ているが，XML は HTML より複雑であるため，XML::Parser も HTML::Parser より複雑になっている．[6]

シンプルな XML パーサ

```
use XML::Parser;

$parse->setHandlers{
Start => \&handler_start,
End => \&handler_end,);


$parse->parsefile($ARGV[0]);


sub handler_start{
my ($parser, $element,%attr) =@_;
print "Start: $element";
}

sub handler_end{
my ($parser, $telement) = @_;
   print "End: $element";
}
```

このコードをシンプルな XML 文書に対して実行すると，次のような出力が得られる．

```
& perl exxmlp.pl simple.xml
```

```
Start: simple
Start: paragraph
End: paragrah
End: simple
```

この結果を見るとわかるとおり，開始タグと終了タグのリストが出力される．異なる要素がきつかったときに呼び出される関数を ”登録する ”ので，関数には任意の名前を付けることができる．

関数には，見つかったタグの名前とそのタグの属性リストが提供されることに注目してほしい．解析プロセスの中でこの情報を利用すれば，伝える情報をより詳細なものにできる．

XML::Parser モジュールはイベントパーサなので，要素を抽出したり，要素を別の形式に変換したりする状況で利用するのに最適である．１つのよい例は，XML 文書を画面上に表示する HTML フォーマットに変換するコードである．

3.2.4 XML::Parser の落とし穴

XML::Parser が基礎にしている Expat ライブラリには小さな落とし穴がいくつかある．XML::Parser は Perl の他の多くのモジュールで使用されているため，話を先へ進める前にそれらの問題点に触れておくことにする．[6]

- 文書にエラーがあると例外が発生する：Expat は妥当性を検証しないパーサであるが，それでも，文書の基本的なレイアウトだけはチェックして，整形式であることを確認する．やっかいなことであるが，これは，文書の基本的な構造に何かのエラーがあると例外が発生することを意味する．これをトラップする唯一の方法は，eval() 内にパーサへの呼び出しを組み込むことである．幸い，パーサをさらに呼び出すと，直前のエラー一の直後から解析を続行できる．

- Expat はすべてのデータを提供する：XML 文書にあるすべてのデータが，XML::Parser に対してプログラマが定義したトリガ関数を介して提供される．したがって，文字データを処理するために使用する関数では，通常のテキスト以外の文字をどのように扱うかを判断しなければいけない．Expat は改行／復帰文字や空白文字など，XML 文書を人間にとって読みやすくするために入れられている文字を返す．

- データは UTF-8 で返される：Expat は厳密には Unicode パーサではないが，XML::Parser は常に UTF-8 文字列を返す．このことは，ほとんどの英語の文書では問題にならない．UTF-8 文字セットと Latin-1 文字セットは最初の 256 文字が同じだからである．その他の Unicode 文字列，とくに Latin-1 セットでサポートされていない外国語では，Unicode::String を使用できる．

- データ部分はブロックとして提供される：Expat はデータをブロックとして扱うため，データハンドラ関数に渡されるデータ部分が不完全であるケースがあるかもしれない．データ部分を独自の方法で処理したい場合は，情報をキャッチに入れて，完全なデータ部分を実際に処理するための別個もハンドラを開始する必要がある．

これらの小さな問題を除けば，XML::Parser はプログラマが予期するとおりの動作をする．

3.2.5 XML の生成

Perl 内で XML 情報を生成するための最も簡単な方法は print を使用することである．そして多くの場合，here 文書と組み合わせるとプロセスがより簡単になる．

print を使用するのは美しいソリューションではない．とりわけ，生成するコードにエラーや不整合が入り込むことがほぼ確実であり，出力をデバックすることがまったくの悪夢になり得るからである．

それよりはるかによいソリューションは，XML タグを名前ごとに構造形式で出力する方法である．ちょうど XML ツリーを自分で作成するかのように出力するわけである．これを実行するには，DOM 解析をサポートしているモジュールの１つを使用する．DOM では，XML 文書をブランチごとに，そしてリーフごとに作成できるからである．

とはいえ，XML::Generator モジュールなどのツールを使用するほうがさらによい方法である．XML タグやオブジェクトや構造を自分で構築しなくても，XML::Generator を利用すれば関数を使ってタグを定義できる．その関数への引数によって，追加のブランチ，リーフ，および属性が作成される．[6]

3.2.6 まとめ

- Perl におけるほとんどの問題点については，その適切な解決策を CPAN アーカイブから見つけられる．その点では XML 処理も例外ではない．そこには XML の処理や解析に関する問題を Perl を使って解決するためのたくさんのモジュールが収録されている．

- Perl による基本的な XML 処理のためには XML::Parser モジュールがある．このモジュールは，XML 文書内で異なる要素が識別されたときに特定の関数を呼び出すための順次的な方式を提供する．XML::Parser は，XML 文書の内容全体を，HTML などの別のフォーマットに変換する場合の理想的なソリューションである．

- Perl での XML パースは XMl::Parser やさまざまなサブクラスを使えば，とても簡単になる．
- XML::Parser にはいくつかの異なるスタイルがあり，特定のパースタスクを解くのに使うことができる．標準のスタイルが要件に合わなければ，ハンドラを使って，パーサの動作をより制御できる．

第4章　開発プログラム

Perl で MakeMenu.pl と MakeContent.pl というプログラムを作成した．タグの付け替えには，Perl 内で XML 情報を生成するための最も簡単な方法である print を使用した．

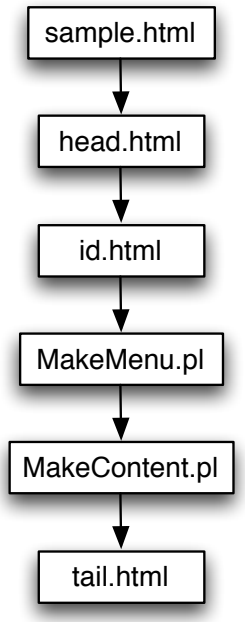


図 4.1: プログラムの流れ

4.1　MakeMenu.pl

MakeMenu.pl は，OPML ファイルから指定したページの右メニュー部を作成するプログラムである．このプログラムは，タグとその中のテキスト要素を読み取り，タグの付け替えを行って，階層レベル１，２の文章を HTML ページの右タイトルメニューとして出力する．図 4.1 は MakeMenu.pl のソースコードである．

```
use strict;
```

```
use XML::Parser;
binmode STDOUT,":utf8";

my $parse = XML::Parser->new(Handlers => {
Init => \&handler_init,
Start => \&handler_start,
Final => \&handler_final,
Attlist => \&handler_attr,
Char => \&handler_char,
End => \&handler_end,

});

my $level;
my $text;
my $count;
my $number;

#数値を受け取る。
$number = <STDIN>;


$parse->parsefile($ARGV[0]);

sub handler_init{
$level=0;
$text='';
$count=0;
print "<UL>\n";
}

sub handler_start{
my ($parser, $tag) = (shift, shift);
my %attr=@_ if @_;
if ($tag eq 'outline'){
$level++;
if ($level == 1){
print "<LI>",$attr{"text"},"\n";
}
```

```
if ($count == ($number-1) ){
if ($level == 1){
print "<UL>\n";
}
if ($level == 2){
print "<LI>",$attr{"text"},"\n";
}
}
}
}

sub handler_end{
my ($parser, $tag) = @_;
if ($tag eq 'outline'){
$level--;
$text ='';
if ($level == 0){
$count += 1;
if ($count == $number){
print "</UL>\n";
}
}
}
}

sub handler_char{
my ($parser, $str) = (shift, shift);
return unless $str =~ /\S/;

$str =~ s/^\s+//;
$str =~ s/\s+$//;

$text .= $str;
}

sub handler_final{
print "</UL>\n";
}
```

4.2 MakeContent.pl

MakeContent.pl は，OPML ファイルから指定したページの左コンテンツ部を作成するプログラムである．このプログラムは，階層レベル 1，2，3 の文章のタグとテキスト要素を読み取り，タグの付け替えを行って，HTML ページの左コンテンツ部として階層構造の文章を出力する．図 4.2 は MakeContent.pl のソースコードである．

```
use strict;
use XML::Parser;
binmode STDOUT,":utf8";

my $parse = XML::Parser->new(Handlers => {
Init => \&handler_init,
Start => \&handler_start,
Attlist => \&handler_attr,
Char => \&handler_char,
End => \&handler_end,

});

my $level;
my $text;
my $count;
my $number;

$number = <STDIN>;


$parse->parsefile($ARGV[0]);

sub handler_init{
$level=0;
$text='';
$count=0;
}

sub handler_start{
my ($parser, $tag) = (shift, shift);
my %attr=@_ if @_;
```

```
if ($tag eq 'outline'){
$level++;
if ($level == 1){
$count++;
}
if ($count == $number){
if ($level == 1){
print "<H1>",$attr{"text"},"</H1>","\n";
}
if ($level == 2){
print "<H2>",$attr{"text"},"</H2>","\n";
}
if ($level > 2){
print "<P>",$attr{"text"},"</P>","\n";
}
}
}
}

sub handler_end{
my ($parser, $tag) = @_;
if ($tag eq 'outline'){
$level--;
$text ='';
}
if ( ($count == $number) && ($level == 0) ){
exit;
}
}

sub handler_char{
my ($parser, $str) = (shift, shift);
return unless $str =~ /\S/;

$str =~ s/^\s+//;
$str =~ s/\s+$//;

$text .= $str;
}
```

4.3 シェルプログラム

まず，指定した HTML ページに head.html を書き，さらに id.html を上書きする．さらに MakeMenu.pl と MakeContent.pl で右メニュー部と左コンテンツ部を作成する．最後に tail.html を書き加えた．このシェルコマンドをまとめたものが，下のシェルスクリプト MakeAll である．ここでは，保存した topic.opml のタグの付け替えと文章の抽出を行い，sample6.html という HTML ページを作成している．さらに head.html と id.html と tail.html のソースコードを記述しておく．

```
cat head.html > sample6.html
perl MakeMenu.pl topic.opml >> sample6.html
cat id.html >> sample6.html
perl MakeContent.pl topic.opml >> sample6.html
cat tail.html >> sample6.html
```

head.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML lang=ja><HEAD><TITLE>gnuplot / legend</TITLE>
<META http-equiv=Content-Type content="text/html; charset=utf-8"><LINK
href="gnuplot - legend.files/style-new.css" type=text/css rel=stylesheet>
<META content="MSHTML 6.00.2900.2769" name=GENERATOR></HEAD>
<BODY>

<TABLE cellSpacing=0 cellPadding=0 width="100%" border=0>
<COL width="30%">
<COL width="70%">
<TBODY>
<TR>
<TD id=menu>
<P></P>
```

id.html

```
<BR>
</TD>
<TD id=content>
```

tail.html

```
</TD></TR></TBODY></TABLE>
</BODY></HTML>
```

4.4 出力結果

展開したい階層レベル１のタイトル番号をコマンドラインから入力する．すると，右メニューには入力した数字のタイトルメニューだけ展開される．他のレベル１のタイトルは展開されずに閉じた状態で表示される．以下に各ページの出力結果とそのソースコードを示しておく．

出力した HTML ページのソースコード

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML lang=ja><HEAD><TITLE>gnuplot / legend</TITLE>
<META http-equiv=Content-Type content="text/html; charset=utf-8"><LINK
href="gnuplot - legend.files/style-new.css" type=text/css rel=stylesheet>
<META content="MSHTML 6.00.2900.2769" name=GENERATOR></HEAD>
<BODY>

<TABLE cellSpacing=0 cellPadding=0 width="100%" border=0>
<COL width="30%">
<COL width="70%">
<TBODY>
<TR>
<TD id=menu>
<P></P>
<UL>
<LI>親トピック１
<UL>
<LI>子トピック１
</UL>
<LI>親トピック２
</UL>
<BR>
</TD>
<TD id=content>
<H1>親トピック１</H1>
```

```
<H2>子トピック１</H2>
<P>ひ孫トピック１</P>
</TD></TR></TBODY></TABLE>
</BODY></HTML>
```

コマンドラインから数値１を入力したものが以下の HTML ページである．

図 4.2: ページ１の HTML ファイル

コマンドラインから数値２を入力したものが以下の HTML ページである．

図 4.3: ページ２の HTML ファイル

第5章　まとめ

この研究では,「情報の加工」の分野に着目し，加工から発信までの過程の作業を円滑に行うことで知識構築の効率化が可能になるのではないかと考えた．TAO という階層的に文章を記述できるアウトラインプロセッサと TAO が提供する OPML フォーマットに注目した．アウトラインプロセッサの特長である階層構造をもった HTML ページを作成し，その作業を自動化することを目的にした．

そして，TAO で書いた文書をその階層構造を持った HTML ページで出力することのできるプログラム MakeALL を Perl で作成した．コマンドラインから出力したいページの数値と保存先を入力すれば，自動的にタグの付け替えを行い，TAO で書いた文書が HTML ページに変換出力してくれる．

このプログラムを使えば，研究員は，ややこしい HTML や OPML タグについて学習しなくても，TAO から容易に HTML ページを出力保存することができる．「情報の加工から発信」の過程の自動化によって，研究員は文書の作成にだけ集中することができるようになる．さらに保存した HTML ページを他の研究員が閲覧すると内容を理解しやすくなっていると予測する．これらのことから，知識構築の効率化が促進されることを期待する．

付録

XML::Paser は Perl のモジュールのうちもっとも複雑なもの１つである．よく使用されるメソッドの簡単なリファレンスである．[7]

表 5.1: XML::Parser スタイル

Style 名	結果
Debug	ドキュメントのアウトラインを出力する．
Subs	XML の開始タグを見付けると，パーサはタグ名と同名の関数を呼び出す．XML 終了タグを見付けると，タグ名にアンダースコアを先頭に付加した関数を呼び出す．これらの関数は Pkg パラメータで指定した名前空間に存在すると想定される．これらの関数パラメータは，Start や End ハンドラに渡るものと同じものである．
Tree	parse メソッドはドキュメントを表現するパースツリーを返す．各ノードは２要素の配列リファレンスで表現される．第１要素はタグ名，もしくはテキストの場合は０となる．第２要素はタグの中身である．中身または，別の配列のレファレンスである．この配列の第１要素は，アトリビュート／値のペアのハッシュへのリファレンスである．配列の残りの要素は，含まれるノードのタイプと中身を表すペアからなる．
Objects	Parse メソッドはオブジェクトを表すパースツリーを返す．ツリーの各ノードは，オブジェクトに bless されたハッシュである．オブジェクトのタイプ名は pkg パラメータに::を付け，タグのタイプ値を付加したものである．テキストノードは::Characters クラスに bless される．各ノードは kids アトリビュートを持ち，ノードの子供への配列リファレンスを所持している．
Stream	このスタイルは Sub スタイルと同様に動作する．パーサが特定の XML オブジェクトを見つける度，様々なサブルーチンを呼び出す．これらのサブルーチンは Pkg パラメータで指定した名前空間に存在すると相定される．サブルーチンの名前は Start Document, StartTag, EndTag, Text, PI, EndDocyument である．これらのうち，いつ呼ばれるのか明らかでないのは，ＰＩです．このサブルーチンは，パーサがドキュメントに処理命令を見つけた際に呼ばれる．

```
$Paser =  XML::Paser ->new(Style = $style,
```

```
Handlets => ＼%handlers,
Pkg => $package)
```

XML::Paser オブジェクトを生成する．いくつかの省略可能な名前つき引数をとる．Style パラメータはパーススタイルのうち、どれを使用するか決定する．下の表は使用可能なスタイルとそのスタイルを選んだ結果のリストです。

表 5.2: XML::Parser Handlers

ハンドラ	いつ呼ばれるか	サブルーチンのパラメータ
Init	パーサがドキュメント処理を開始する前	Expat オブジェクトのリファレンス
Final	パーサがドキュメント処理を終了した後	Expat オブジェクトのリファレンス
Char	パーサが文字データを見つけた時	Expat オブジェクトのリファレンス，文字列
Proc	パーサが処理命令を見つけた時	Expat オブジェクトのリファレンス，PI ターゲット名，PI データ
Comment	パーサがコメントを見つけた時	Expat オブジェクトのリファレンス，コメントデータ
CdataStart	パーサが CDATA セクションの開始を見つけた時	Expat オブジェクトのリファレンス
CdataEnd	パーサが CDATA セクションの終了を見つけた時	Expat オブジェクトのリファレンス
Default	パーサがハンドラが割り当てらていないデータを見つけた時	Expat オブジェクトのリファレンス，データ文字列
Unparsed	パーサがパースされていないエンティティ宣言を見つけた時	Expat オブジェクトのリファレンス，エンティティ名，アドレスを解決する際に使用するベース URL，システム ID，パブリック ID
Notation	パーサが notation 宣言を見つけた時	Expat オブジェクトのリファレンス，notation 名，アドレスを解決する際に使用するベース URL，システム ID，パブリック ID
ExternEnt	パーサが外部エンティティ宣言を見つけた時	Expat オブジェクトのリファレンス，エンティティ名，アドレスを解決する際に使用するベース URL，システム ID，パブリック ID

Handlers パラメータはハッシュリファレンスである．キーはドキュメントをパースしながらトリガするイベント名で，値はその際に呼ばれるサブルーチンのリファレンスである．サブルーチンは pkg パラメータによって指定した名前空間に存在

すると想定される．表A・４は様々なタイプのハンドラのリストである．これらハンドラの第１パラメータは，パージングを処理する Expart オブジェクトへのリファレンスである．このオブジェクトは，パージングプロセスをより正確にコントロールするためのメソッドを備えている．

表 5.3: XML::Parser Handlers

Entity	パーサがエンティティ宣言を見つけた時	Expat オブジェクトのリファレンス，エンティティ名，エンティティの値，システム ID，パブリック ID，エンティティの notation
Element	パーサがエレメント宣言を見つけた時	Expat オブジェクトのリファレンス，エレメント名，コンテント名
Attlist	パーサがアトリビュート宣言を見つけた時	Expat オブジェクトのリファレンス，エレメント名，アトリビュート名，アトリビュートタイプ，デフォルト値，アトリビュートが固定かどうかを示す文字列
Doctype	パーサが Doctype 宣言を見つけた時	Expat オブジェクトのリファレンス，ドキュメントタイプ名，システム ID，パブリック ID，インターナショナルサブセット
XMLDecl	パーサが XML 宣言を見つけた時	Expat オブジェクトのリファレンス，XML バージョン，ドキュメントのエンコーディング，DTD がスタンドアロンかどうかを示す文字列

Pkg はパッケージ名である．全てのハンドラはこのパッケージにあると想定され，ユーザ定義のサブルーチンに依存するスタイルは全てこのパッケージを検索する．このパラメータが指定されない場合，デフォルトパッケージは main である．

このメソッドはいくつかのオプションパラメータを取り，全て Expart オブジェクトにそのまま渡される．

```
$paser ->parse($source)
```

ドキュメントをパースする．source パラメータはドキュメント全体を含むスカラ変数か，IO::Handle オブジェクトへのリファレンスである必要がある．返り値は，選択したスタイルによって変わる．

```
$paser -> parse_file ($filename)
```

指定したファイルを開いて，中身をパースする．返り値は，選択したスタイルによって変わる．

```
$paser -> setHandlers(%handlers)
```

ハンドラのセットを新たなセットでオーバーライドする．パラメータは new に渡すものと同じフォーマットのハッシュとして解釈される．空文字列や undef を含めることによって，ハンドラを off にすることが出来る．

参考文献

[1] 大澤幸生,「知識マネジメント」(オーム, 東京,2004)

[2] Bo Leuf, Ward Cunningham,「Wiki Way」(ソフトバンクパブリッシング, 東京,2000)

[3] gnuplot
http://t16web.lanl.gov/Kawano/gnuplot/legend.html

[4] アウトラインプロセッサ　解説
http://www.ergo.co.jp/support/egtips_10.html

[5] OPML 解説
http://hail2u.net/documents/opml_spec.html

[6] Martin Brown,「XML プロセッシング」(日本ユニテック, 東京,2003)

[7] David Cross,「Perl データマンジング」(ピアソンエデュケーション, 東京,2003)

[8] Randal L.Scbwatz, Tom Pboenix,「初めての Perl」(オライリージャパン, 東京,2003)