SlothLib: Web研究のためのプログラミングライブラリ

大島　裕明†　　中村　聡史†　　田中　克己†

† 京都大学大学院情報学研究科社会情報学専攻
〒 606–8501 京都市左京区吉田本町
E-mail: †{ohshima,oyama,tanaka}@dl.kuis.kyoto-u.ac.jp

あらまし　本稿では，Web 研究におけるソフトウェアの試作開発時のコストを軽減するためのライブラリ SlothLib の提案を行う．研究のプロトタイプシステムを作成する際には，メインのアイデアの実装以外に，Web 検索 API サービス，Web 上の情報取得とそれに伴う文字コード判別，既存のクラスタリングアルゴリズム，自然言語処理ツールなどの利用方法を習得しなければならないことがある．このような既存のツールやアルゴリズムを簡単に利用できるようすることが SlothLib の目的である．各種機能やアルゴリズムのうち，類似の機能であるものには極力共通インタフェースを持たせて実装しているため，あとで機能を入れ替えることが容易である．また，SlothLib で実装された機能を GUI 上で利用できる「ブロックプログラミング環境」のプロトタイプシステムを開発した．これは，簡単なプログラムであればビジュアルインターフェース上でブロックを接続することによってアイデアを試すことができるシステムである．

キーワード　プログラミングライブラリ, Web 検索, ビジュアルプログラミング環境

SlothLib: A Programming Library for Researches on the Web

Hiroaki OHSHIMA†, Satoshi NAKAMURA†, and Katsumi TANAKA†

† Department of Social Informatics, Graduate School of Informatics, Kyoto University
Yoshida-honmachi, Sakyo, Kyoto, 606–8501 Japan
E-mail: †{ohshima,oyama,tanaka}@dl.kuis.kyoto-u.ac.jp

Abstract　We made a programming library for researches especially on the Web. It is called SlothLib. The purpose of SlothLib is to reduce costs for implemention of ideas on researches. Usually, researcher need to study forWeb search APIs or tools for national language processing before implementing own ideas. SlothLib provides common interfaces for similar searvices, and it is easy for researcher to change some part of implementation. We also made a programming environment on a visual interface. Each small function is represented as a block on the environment, and a user connect blocks as a flow of data. It makes researchers easy to test their ideas.

Key words　Programming Library, Web Search, Visual Programming Environment

1. はじめに

研究を行うにあたって，実装は非常に重要である．いくら良いアイデアを思いついたとしても，それを実装して試してみなくては意味がない．また，良いアイデアを生み出す人が，必ずしも高い実装の能力を持っているとは限らない．最近のプログラミング言語は，昔のものよりも抽象化が進み，プログラミングはより容易になってきているとはいえ，まだ難しく思ってしまう人がいるのも事実である．

プログラミングをある程度習得していたとしても，一部に既存のツールやアルゴリズムを利用して自分のアイデアを実現する際には，作成するプログラムからのツールの利用法を勉強したり，既存のアルゴリズムの実装を行わなくてはならない．特に，同じような研究を行っている人が作成したプログラムには，同じようなコードが存在していることが多い．

我々は，研究のアイデアを実現するシステムの作成を行う際に，少しでも手間を減らすことを目的として，プログラミングライブラリ SlothLib の開発を行った．現時点では全て Windows 上で C#で実装されており，.NET 環境において利用することができる．

SlothLib が目指すのは，これまでにライブラリ化されていなかった機能を新たにライブラリ化するということよりも，むしろ，世の中に提供されている様々なツールや既存のアルゴリズムを網羅して扱えるような環境を提供することにある．

例えば，現在，Google，Yahoo!，MSN といった Web 検索エンジンはそれぞれ API を提供しているが，それぞれ独自のインターフェースを用意している．試作システムで 1 つの検索エンジンの API を利用していたとして，それを別のものに取り替えてみたいと思ったときには，これまで使っていたものとは全く別のインターフェースを利用することになり，何ヶ所もコードを書き換えなくてはならなくなる．もし，これらの API が全て同じインターフェースを持っていれば，プログラムの 1ヶ所を書き換えるだけで利用する検索エンジンを変更できる．

そこで，SlothLib では，Web 検索エンジン用の共通のインターフェースを用意し，そのインターフェースを利用するように様々な検索エンジンの API に対するラッパー的な役割を持つクラスを作成している．Web 検索エンジンだけでなく，類似した機能を持つものに対しては共通のインターフェースを持たせるようにしている．

SlothLib が特に対象とするのは，Web 検索を利用したような新たなサービスや研究である．Web の情報は増加する一方であり，blog や SNS，ソーシャルブックマークといった従来の Web ページとは性質が異なるような情報も増えてきている．そのような中，既存の Web 検索エンジンを利用したり，Web から収集した文書の解析を行ったりして，新たなサービスや研究を行う必要性は確実に増加してきている．そのようなサービスや研究における実装では，公開されている Web 検索の API を利用すること，Web 情報を収集すること，文書の特徴ベクトルを作成して類似度などの計算を行うこと，クラスタリングを行うことなどが頻繁に必要になる．それらのよく使われる機能のそれぞれに，先ほど述べた Web 検索の場合と同じように共通のインターフェースを持たせることによって，システムを部品を組み替えるような感覚で実装することができるようになる．

SlothLib を簡単に利用してもらうための文書の整備も行っている．また，ほとんどの機能について，サンプルプログラムも提供している．経験上，SlothLib を習得してもらう際には，実際の使用例のコードが書かれているサンプルプログラムが非常に役立つことが分かっている．

また，SlothLib で利用可能な機能を中心に，視覚的にプログラミングを行える環境 “BlockProGramming” のプロトタイプシステムを作成した．Web 検索や特徴ベクトル生成などの機能がビジュアルインターフェース上にブロックとして存在しており，ブロックの入力を制御することによってアイデアをより簡単にテストすることができる．

以下，**??**節で関連研究について，3. 節で SlothLib から利用できる機能について，4. 節で視覚的プログラミングの環境である BlockProGramming について，5. 説でまとめと今後の課題について述べる．

2. 関 連 研 究

2.1 基盤ライブラリと Web 検索を横断的に利用する環境の提供

プログラミングライブラリは数多いが，その多くはある特定の機能を利用してもらうためのものである．SlothLib は，むしろ，メタレベルのライブラリであり，他のライブラリを統合的に利用するためのものである．ここでは，共通基盤としてのライブラリや，数多く存在する Web 検索を横断的に利用するための環境を提供しようとする研究についていくつか紹介する．

阿部ら [1] は様々なメディアに対するアノテーション処理を統合的に扱うためのライブラリを構築している．アノテーション処理に特化しているライブラリと見ることができるが，文書要約や文書の重要語抽出など，自然言語処理の基本的な処理を行うことができ，自然言語処理を専門としないものにとっては自然言語処理のライブラリとしての利用価値も高い．

Mendelzon [2] らは，Web 検索エンジンを横断的に検索することができる WebSQL という言語を開発した．言語を提供することによって，異なるインターフェースを持つ検索エンジンを統一的に扱うことができるようにしようとしている．Kistler [3] らは，WebL という言語を開発した．これは，Web 文書の取得，サービスの統合，データの抽出，Web 文書の作成など，様々な Web における行動を記述することができ，Web 行動そのものをプログラミングしようとしている．

2.2 視覚的プログラミング環境

視覚的なプログラミング環境は，現在様々なものが開発されている．いくつかについて紹介する．

Microsoft の SQL Server 2005 には，BI Development Studio [4] という ETL 処理を視覚的なインターフェースで行う環境がある．Ito ら [5] は，Web アプリケーションに対してラッパーを作成し，それらを組み合わせたサービスを作成できるような環境を開発した．Tanaka ら [6] は，パッドというデータや機能を格納した部品を視覚環境上に配置して合成することによって新たな機能を実現するアプリケーションを開発している．松村らは 2006 年度上記の IPA 情報処理推進機構の未踏ソフトウェア創造事業において，セマンティック Web サービスの連携を視覚的に行うための PatchService [7] というサービスの開発を行っている．現在数多くあるセマンティック Web サービスのそれぞれに，ラッパーを作成して連携可能にし，視覚的な環境において組み合わせることができる．Microsoft Research ASIA の Wei-Ying Ma らは Web Studio [8] という環境を作成しており，そこでは，Web 検索機能などを利用したアプリケーションを視覚的に作成することができる．

3. SlothLib の機能

本節では，SlothLib から利用できる機能について説明する．時折，インターフェースやクラスについて，コードを示しながら説明するが，スペースの関係上断りなく一部を省略している場合がある．

3.1 Web 検索サービス

現在，様々な検索エンジンの API が公開されている．また，特に Blog 検索エンジンでは，RSS が提供されるものがあり，それらもプログラムから利用することができる．それぞれの検索エンジンは SlothLib 上でクラスとして表現されるが，それらは全て共通に ISearchEngine というインターフェースを実装している．ISearchEngine はただ 1 つ，検索を実行するための

DoSearch というメソッドを実装し，検索結果は ISearchResult というインターフェースを実装する．

```
public interface ISearchEngine
{
    ISearchResult DoSearch(string query, int num);
}
public interface ISearchResult
{
    long ResultNum { get; } // 検索結果総数
    IResultElement[] ResultElement { get; } // 得た検索結果
}
```

DoSearch メソッドの引数の query は検索語，num は最大取得結果件数である．ISearchResult の ResultNum は検索された結果の総数であり，ResultElement は実際に取得した検索結果の配列であり，URL，検索されたページのタイトルと要約を保持している．

以下は，SlothLib でこれらインターフェースを実装して作られたクラスから利用できる検索エンジンの一覧である．

- Google SOAP Search API (注1)
- Yahoo! ウェブ検索 Web サービス (注2)
- MSN Search API (注3)
- goo ブログ Search (注4)
- livedoor Blog 検索 (注5)
- Yahoo! 動画検索 Web サービス (注6)
- Youtube API (注7)
- 人力検索はてな (注8)
- 教えて！ goo (注9)
- Yahoo! 知恵袋 (注10)
- 価格.com 商品検索 API (注11)

一般的な Web ページ検索から，Blog 検索，動画検索，QA システムの検索，商品検索と，多岐にわたる検索エンジンが利用可能である．これらのいくつかでは共通インターフェースで扱う以上の情報が提供されている．例えば，Youtube API では，検索された動画に付加されているタグや，これまで何回閲覧されたかという情報がある．Youtube API の 1 件の検索結果を扱うクラスは IResultElement を実装する YoutubeResultElement というクラスであるが，このクラス自体は共通インターフェースの部分以外にもタグや閲覧回数を含む全ての情報を保持するようにしており，もしユーザがこれらの情報を利用したいと思えば利用可能であるようにしている．

いくつかの API や Web サービスには，利用時にクレジット表示を義務づけるものがある．例えば， Yahoo! を利用したプログラムにはクレジット表示が義務づけられている．そこで，Yahoo! の機能を気軽に利用するためにも，簡単にクレジット表示が行えるコントロールを検索のためのクラスとは別に提供している．

3.2 Web 情報収集

Web における研究では，多くの文書や画像データなどを Web から取得することがある．多くのファイルをダウンロードする必要があるなら，マルチスレッドを利用した方が速いことが多い．SlothLib では，URL を与えるとローカルコンピュータにファイルとして保存するクラスとして，シングルスレッドのものとマルチスレッドのものの 2 つを提供しており，下記のようにして利用することができる．

```
// シングルスレッドによる Web 情報取得
WebGet wg = new WebGet();
Result r = wg.DoFetch(url);

// 最大 15 のマルチスレッドによる Web 情報取得
WebGetMulti wgm = new WebGetMulti();
Result[] r = wgm.DoFetch(urls, 15, files, null, null)
```

Result クラスには，HTTP のヘッダ情報や，取得したファイルが保存されているファイルパスなどが格納されている．

3.3 各種文書読み込みと日本語文字コード判別

文書には，テキストファイル，PDF，MS-Word，Power Point，一太郎，HTML など，様々な形式が存在する．このような様々なフォーマットのファイルを読み込むためには専用の API やツールを利用する必要がある．

例えば，xdoc2txt (注12) は非常に多くのフォーマットの文書から，その内容のテキストだけを抽出するツールであり，Namazu や Meadow2 といったソフトウェアでテキスト形式以外のファイルを扱うために利用されている．SlothLib では xdoc2txt を利用するクラスを用意することで，様々なファイルの内容をテキストで読み込むことができるようにしている．以下が使用例である．

```
IContentReader reader =
    new Xdoc2txtReader(@"C:\usr\bin\xdoc2txt.exe");
string content = reader.Read(@"C:\usr\doc\document.pdf");
```

IContentReader は，ファイルからコンテンツを読み込むクラスが共通に実装するインターフェースで，Xdoc2txtReader クラス以外にもテキストファイルを読み込むための TextReader クラスなどが実装する．

日本語のテキストファイルを読み込む場合には，文字コードの自動判別も必要となってくる．文字列コードを判定するも

(注1)：http://code.google.com/apis/soapsearch/
現在， Google SOAP Search API のアカウント発行は中止され，Google AJAX Search API という新しいサービスが開始されている．
(注2)：http://developer.yahoo.co.jp/search/web/V1/webSearch.html
(注3)：http://search.msn.com/developer/
(注4)：http://blog.goo.ne.jp/
(注5)：http://sf.livedoor.com/
(注6)：http://developer.yahoo.co.jp/search/video/V1/videoSearch.html
(注7)：http://www.youtube.com/dev
(注8)：http://q.hatena.ne.jp/
(注9)：http://oshiete.goo.ne.jp/
(注10)：http://chiebukuro.yahoo.co.jp/
(注11)：http://apiblog.kakaku.com/KakakuItemSearchV1.0.html

(注12)：http://www31.ocn.ne.jp/ h_ishida/xdoc2txt.html

のとしては，TxtEnc (注13) や NKF (注14)等を利用することが考えられる．現時点では TxtEnc を利用した文字コードの判別のためのクラス TxtEncConverter を利用することができる．TextReader クラスでは内部から文字コード判別を行うようになっており，以下のように利用する．

```
IContentReader reader =
    new TextReader(new SlothLib.Text.TxtEncConverter());
string content = reader.Read(@"C:\usr\doc\document.txt");
```

文字列型の content にはテキストファイルの内容が格納される．

3.4 自然言語処理関連

ベクトル空間モデルでは，文書とクエリを特徴ベクトルとして表現し，ベクトルどうしの類似度を計算することによって文書検索を行う．システムとしては SMART [9], [10] が有名である．日本語の文書を特徴ベクトルで表現するためには，形態素解析によって文章を形態素に分解することがよく行われる．

また，英語の場合でも形態素解析を行ったり，形態素解析は行わないまでもステミングという語幹の抽出を行うことがある．

そのような，各種の自然言語処理のためのツールは様々なものが公開されており，SlothLib ではなるべく簡単にツールを利用できるようにしている．

3.4.1 解析ツールと結果のためのクラス

自然言語処理のツールの多くは，テキストの入力を与えるとテキストの出力によって結果を返すようになっている．そこで，まず，全ての自然言語処理ツールが実装する IAnalyzer クラスとその結果を格納する IResult クラスを作成した．

```
public interface IAnalyzer
{
    // テキストを解析して結果を返す.
    IResult DoAnalyze(string text);
    List<IFilter> FilterList { get; } // 結果の処理
}
public interface IResult : IEnumerable<string>
{
    // 結果を 1 行ごとに保持したリスト
    List<string> List { get; }
}
```

例えば，形態素解析器であれば，形態素解析器の結果を利用するために便利なメソッドを供えた結果クラスを作成することになるだろうが，その際にも IResult は必ず実装するようにしている．IResult が実装されていれば，ツールの出力そのものを取得することができる．このようにしているのは，もし，ユーザがツールの出力そのものを直接扱いたい場合に特殊な結果クラスがかえって障害になる可能性があるからである．

IAnalyzer が実装する FilterList は，解析結果の処理に関わるものであり，以降で実際にどのように用いるかを説明する．

3.4.2 日本語形態素解析器

現在，SlothLib から利用できる日本語の形態素解析器は茶筌(注15)と MeCab(注16)がある．ここでは，文書の特徴ベクトルを作成するために茶筌を利用する場合について説明する．

以下は，文書が string 型の変数 content に格納されているときに，形態素解析を行って名詞と動詞のみを取り出し，それらの語の基本形を文字列リスト result に格納するコードである．

```
ChaSenSetting setting = new ChaSenSettingOriginalPos();
IAnalyzer analyzer = new ChaSen(setting);
analyzer.FilterList.Add(new PosFilter("名詞|動詞", null));
analyzer.FilterList.Add(new RemainMorphemeOriginalFilter());

List<string> result = analyzer.DoAnalyze(content).List;
```

まず，1 行目で茶筌の設定を決めている．茶筌の設定は chasenrc というファイルで行うことができるが，SlothLib ではあらかじめいくつかの chasenrc を用意しており，ここでは，解析結果の 1 行に見出し語の基本形と形態素を出力する設定を利用している．次に，IAnalyzer を実装する ChaSen クラスのインスタンスを作成している．Windows 版の茶筌の実行ファイル名は chasen.exe であり，ChaSen クラスでは chasen.exe に対して文字列を与えてその出力を読み込む処理を行う．Windows 版の茶筌はインストール時にレジストリにデフォルトの辞書ファイルの位置が書き込んでおり，ユーザが chasen.exe の位置を指定しない場合は，その情報から chasen.exe の位置を推定する．ChaSen クラスでは，chasen.exe に文字列を渡す前に半角カナの文字を全角カナの文字に変更するなどの前処理を行っている．これは，茶筌は半角カナについては一文字ずつに分解して未知語と解析するためである．

3 行目と 4 行目では，ChaSen クラスのインスタンス analyzer にフィルタを設定している．これは，解析結果をに対する処理を行うもので，ここでは 2 つのフィルタを使用している．1 つめは PosFilter である．これは，解析結果の 1 行に品詞情報が含まれている場合にのみ利用できるフィルタであり，指定された品詞の行のみを残す．この場合，品詞情報が「名詞」または「動詞」で始まっている行だけを解析結果に残す．

```
new PosFilter("名詞|動詞|形容詞|形容動詞", "名詞-代名詞");
```

例えば，このように変更すると，品詞情報が「名詞」「動詞」「形容詞」「形容動詞」から始まる行だけを残し，「名詞-代名詞」は例外的に残さない，というフィルタになる．4 行目の RemainMorphemeOriginalFilter は，茶筌の解析結果の各行において，見出し語の基本形の部分だけ残して，他の品詞情報などは取り除くというフィルタである．

最後の行で content を解析している．例えば，文字列型の変数 content が「研究を行うにあたって，実装は非常に重要である」という文章であった場合には，上記のコード実行後に result の要素が,「研究」「行う」「実装」「非常」「重要」となる．

3.4.3 英語の単語切り出しとステミング

日本語と異なり，英語の場合は単語があらかじめ空白で切り

(注13)：G-Project: http://www.gprj.net/modules/xproject/
(注14)：https://sourceforge.jp/projects/nkf/
(注15)：http://chasen.naist.jp/hiki/ChaSen/
(注16)：http://mecab.sourceforge.jp/

分けられているので，処理としては簡単である．ただし，切り分けができても動詞や名詞の語尾変化があるため，ステミングや形態素解析を利用することを考えなくてはならない．

SlothLib では，IAnalyzer を実装する EnglishSplitter クラスを提供している．このクラスは，スペースで単語を切り分けて 1 行ごとに出力するだけである．

```
// 英単語のみを切り出しの対象とするとき
IAnalyzer analyzer = new EnglishSplitterAlpha();
```

上記は英単語のみを切り出すクラスだが，他にも，英単語，数値，記号などを切り出すためのクラスも用意している．

切り分けられた単語のステミングを行う場合は，Porter Stemmer (注17)が利用できる．例えば，ある文書に “dance” という語と “dancing” という語が含まれていたときに，何の処理も行わないとこれらは別の語とみなされる．そこで，ステミングを行い，同じ語とみなすのである．Porter Stemmer によるステミングの場合，これらの語は共に “danc” となる．単語としては正しくないが，特徴ベクトルの 1 要素として扱う分には問題はない．以下で，PorterStemmer の使用例を示す．

```
IAnalyzer analyzer = new EnglishSplitterAll();
analyzer.FilterList.Add(new PorterStemmer());
analyzer.FilterList.Add(new ToLowerFilter());
```

2 行目が Porter Stemmer であり，3 行目は全ての語を小文字にするフィルタである．これは，大文字と小文字の違いによって別の語として判断するのを避けるためである．SlothLib は他にも，大文字にするフィルタや，半角文字を全角文字にするフィルタなど，基本的な機能のフィルタを数多く供えている．

3.4.4 ストップワードリスト

文書の特徴ベクトルを作成する際には，しばしばストップワードリストが用いられる．これは，リストに含まれる語をベクトル作成の際に無視するものである．ストップワードリストとして有名なものとしては，SMART システムで使われている英語のものがある．

SlothLib では，ストップワードリストを IAnalyzer から利用するフィルタとして実装している．

```
IFilter stopWordFilter = new StopWordFilter();
stopWordFilter.LoadWordList(@"C:\usr\StopWord\word\");
stopWordFilter.LoadSymbolList(@"C:\usr\StopWord\symbol\");
```

1 行目は StopWordFilter のインスタンスを生成している．2 行目は LoadWordList メソッドを実行しており，ここでは指定したディレクトリにあるテキストファイルを全てストップワードリストとして読み込んでいる．3 行目は同様に，LoadSymbolList メソッドを実行している．LoadWordList メソッドで読み込まれるファイルには,「もの」「こと」「上」「下」「a」「an」「the」のような，いわゆるストップワードを羅列し，これらに完全に一致する場合にはストップワードとして取り除かれる．それに対して，LoadSymbolList メソッドで読み込まれるファイルには,「◆」「□」「■」「△」「)」「*」「+」といった記号を羅列し，語がこれらのいずれかを含む場合にはストップワードとして取り除かれる．

(注17)：http://www.tartarus.org/martin/PorterStemmer/

3.5 特徴ベクトル

3.5.1 特徴ベクトルの基礎

次に，特徴ベクトルについて述べる．文書の特徴ベクトルの生成手法にはいろいろなものが考えられる [11]．文書の特徴ベクトルを作成する際には一般的に，局所的重み，大局的重み，正規化の 3 つについて考える必要がある．

局所的重みとは，ある 1 つの文書においてどのような語がどの程度出現しているかということから考えられる文書の特徴のことである．ある語が文書に含まれているかいないかという 2 進重み，ある語の文書中の出現頻度であるいわゆる TF (term frequency) ，TF の対数を取ったものなどがよく使われる．

大局的重みとは，文書集合全体で考えたときに，ある語がどの程度の重要度を持っているかということである．大局的重みについて全く考えない場合もあるが，良く用いられるのは IDF (inverse document frequency) であろう．しかし，他にも大局的重みの計算手法は存在している．

正規化とは，局所的重みや大局的重みについて考慮して作成された特徴ベクトルの長さを調整するものといえる．正規化を行わないと，長い文書から作られたベクトルと，短い文書から作られたベクトルの長さが不均一になり，処理の無いようによっては不都合が起こる場合がある．正規化として良く行われるのが，長さを 1 にするコサイン正規化とよばれるものである．コサイン正規化されたベクトルどうしの内積は，2 つのベクトルが成す角のコサイン値になる．

このように，文書の特徴ベクトルを生成するのにも，様々なバリエーションがある．研究で特徴ベクトルを利用する場合にも TF-IDF ばかりでなく他の手法も試してみるのが良い．

SlothLib で特徴ベクトルを扱う基本になるインターフェースが IVector である．IVector はジェネリックであり，ベクトルの要素となる型には，文字列や整数など，場合に応じて何でも利用することができる．

```
public interface IVector<T>
{
    double this[T key] { get; set;} // 要素に対する重み
    double VectorLength { get; } // ベクトルの長さ
    T[] KeyList { get; } // 0 以外の値を持つ要素のリスト
    T[] GetSortedKeyList(); // 降順でソートされた要素のリスト
}
```

IVector を実装するクラスには様々なものがあるが，最も基本的なベクトルは，BasicVector というクラスである．他の多くのクラスはコンストラクタで与えられたベクトルに対して何らかの形で修飾することを目的とするベクトルである．それらは，デザインパターンにおけるデコレーションパターンを用いて実装されている．

3.5.2 TF を重みとする特徴ベクトルの作成

BasicVector のサブクラスの 1 つが FrequencyVector であ

り，コンストラクタで配列やリストを受け取ると，そこでの出現回数を数えて特徴ベクトルを作成する．FrequencyVector を用いて，TF を重みとする特徴ベクトルを作成するには以下のようにする．ここでは，文字列型の変数 content に文書の内容が入っているものとする．

```
IAnalyzer analyzer;
// (中略: analyzer に形態素解析器とフィルタを設定)
IVector<string> tf = new FrequencyVector<string>
                        (analyzer.DoAnalyze(content).List);
```

3.5.3 特徴ベクトルの修飾

局所的重みとして TF の対数を用いてベクトルを作成したい場合は，先ほど作成した TF によるベクトルを修飾すればよい．

```
IVector<string> logtfSa = new LogVectorSa<string>(tf);
IVector<string> logtfRe = new LogVectorRe<string>(tf);
```

上記は両方とも TF の対数によるベクトルが生成される．クラス名の最後に Sa と付くクラスと Ra と付くクラスが，様々な修飾用のベクトルに用意されている．詳しくは述べないが，Sa (Standalone の意) は，そのベクトルの要素の全ての値をコンストラクタで計算して保持するのに対し，Ra (Referential の意) は，修飾すべきベクトルの参照を保持しており，値を求められたときに毎回重みの計算を行う．

正規化もベクトルの修飾によって行われる．

```
IVector<string> costf = new CosNormVectorSa<string>(tf);
IVector<string> sumtf = new SumNormVectorSa<string>(tf);
IVector<string> maxtf = new MaxNormVectorSa<string>(tf);
IVector<string> augtf = new AugNormVectorSa<string>(tf);
```

1 行目はコサイン正規化，2 行目は要素の重みの総和を 1 にする正規化，3 行目は最大の要素の重みを 1 にする正規化，4 行目は最大の要素の重みを 1 にしながら 0 以上の値を持つ要素の重みを 0.5 以上にする正規化である．

DF (Document Frequency) を求めるためには，複数のベクトルを用意して，以下のようにする．

```
IVector<string>[] tfArray;
// (中略: tfArray に TF によるベクトルの配列が作成される)
IVector<string> df = new DFVectorSa<string>(tfArray);
IVector<string> idf = new InverseDFVectorSa<string>(df);
```

DF を求める時には，IVector の配列やリストをコンストラクタで引数として渡す．2 行目ではさらに，DF を修飾することによって IDF を求めている．

TF-IDF によるベクトルを作成するためにはこれらの重みを掛け合わせなくてはならない．

```
IVector<string> tfidf =
    new MultiplyVectorSa<string>(tf, idf);
```

これも，やはり修飾を用いて実現される．このように，ベクトルを修飾して新たなベクトルを作成するものには，他にも様々なものを用意している．

3.5.4 ベクトルの類似度と距離の計算

文書が特徴ベクトルで表されると，類似度を計算することが可能となる．類似度や距離の計算の代表的なものとしては，内積，コサイン，Jaccard 係数，Dice 係数，ユークリッド距離，マンハッタン距離などがある．現在は，これらのうち，コサインのみ SlothLib 利用できる．

```
CosineCalculator<string> clac =
                        new CosineCalculator<string>();
double cos = calc.DoCalculate(v1, v2);
```

IVector がジェネリックであることから，類似度や距離計算のクラスについても特徴ベクトルの要素の型を指定しなくてはいけなくなっており，改善を要する．上記では，IVector である v1 と v2 が成す角のコサインを計算している．

今後，他の類似度や距離についても実装を進める．

3.6 クラスタリング

アイテムがいくつか存在し，それらがベクトルで表現されているとき，クラスタリングを行うことがある．クラスタリングの手法には，大きく分けて階層型クラスタリングと非階層型クラスタリングがある．

階層型クラスタリングは，始めに全てのアイテムをそれぞれクラスタとみなし，それらを結合していくことによってクラスタ数を徐々に減らしていき，最終的には 1 つのクラスタにするような手法で行われるクラスタリング手法である．クラスタリングの結果をデンドログラムという木構造で表すことができる．非階層型クラスタリングの代表的なものには K-means 法がある．K-means 法では最終的に作成するクラスタの数をあらかじめ決めておき，得られるクラスタは階層がない平坦なものとなる．SlothLib では現在，階層型クラスタリングのうちの，最短距離法，最長距離法，群平均法を実装している．

```
double[,] distanceTable; // 上三角の距離テーブル
// (中略: distanceTable に各アイテム間の距離を計算する)
HierarchicalClusteringMethod method =
new CompleteLinkMethod(distanceTable, DistanceType.Distance);
Result result = method.DoClustering();
```

まず，全てのアイテム間の距離を計算した距離テーブルを作成する．アイテム間の距離は，コサインのような類似度やユークリッド距離のような非類似度によって計算される．ここでは最長距離法を用いているが，距離テーブルを渡すと同時に，類似度を用いるのか非類似度を用いるかを指定している．結果は専用の結果クラスで返される．ここでは終了条件を与えていないため，クラスタが 1 つになるまで階層型クラスタリングが実行される．

```
int[][] cluster = result.GetCluster(10.0);
```

結果クラスは整数型のジャグ配列で結果を返す GetCluster メソッドを持ち，終了条件を与えることでその終了条件下におけるクラスタリングの結果を取得することができる．ジャグ配列

の要素である配列には同じクラスタに属するアイテムのインデックスが納められており，これは始めに与えた距離テーブルのインデックスに対応する．

クラスタリングに関してはまだ設計が確定していない部分があり，改良していかなくてはならない．また，階層的クラスタリングには他にも重心法，メディアン法，Ward 法などがあり，それらを実装したいと考えている．非階層クラスタリングにもいくつかの手法があり，今後それらを実装するつもりである．

3.7 その他の実装中の機能

これまで説明した機能の他にも様々な機能が実装されている．いくつかについて簡単に説明する．

3.7.1 行　　列

複数のベクトルを集めて行列として処理を行いたい場合がある．例えば，TF による重みを用いて作成された文書の特徴ベクトルが複数ある時に，それらをまとめて行列として扱い，自身を転置した行列とのかけ算を行うと，語彙の共起行列を作成する，といったことを行うことができる．また，後述する特異値分解などのためにも利用できる．

3.7.2 統 計 解 析

統計解析を行うための言語に R [12] というものがある．R では様々な統計解析を行うことが可能である．ユーザも多く，多くのモジュールが公開されており，上手く利用できると非常に強力なツールである．ただ，R 自体を勉強することや，プログラムから利用するための手法を調べるにはどうしても時間がかかってしまう．

SlothLib では少しでも R を利用する障壁を減らすため，プログラムから R を利用しやすい環境を提供し，さらに，特に Web 関連の研究でも利用しそうな機能について，R を利用したクラスを作成して提供する．これには R が提供している COM を利用している．

R を利用したクラスとして，主成分分析，特異値分解，独立成分分析などが作成されている．例えば，特異値分解は，HITS アルゴリズムや，潜在的意味インデキシング (LSI: latent semantic indexing) などで利用されており，文書検索や Web の研究と非常に関わりが深い．

3.7.3 日本語構文解析，係り受け解析器

自然言語処理のツールに係り受け解析器というものがある．これは，日本語の構文解析を行い，各文節の修飾関係を得るものである．有名なツールとして CaboCha [13] というツールがあり，SlothLib から利用するためのクラスを作成している．

3.8 SlothLib を利用した実装例

図 1 に示したのは，SlothLib の様々な機能を利用して作成したアプリケーションの例である．これは，SlothLib の利用に慣れはじた人が 1 日で作成したものである．機能としては，ユーザが入力した語に対して，絞り込み検索を行うならばどのようなクエリが考えられるかを推薦するシステムである．

まず，ユーザから入力された語で Web 検索を行う．このとき利用する検索エンジンはコンボボックスから選択できるようになっている．これは，SlothLib の検索エンジンのクラスが共通のインターフェースを持っているため簡単に実現できる．次

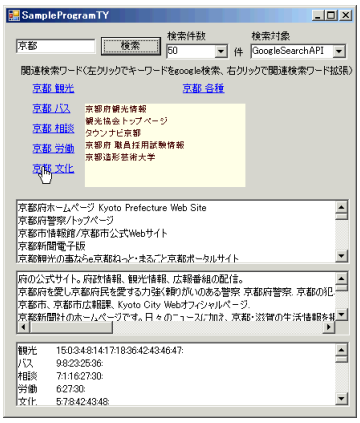

図 1　SlothLib を利用して作成したアプリケーション例

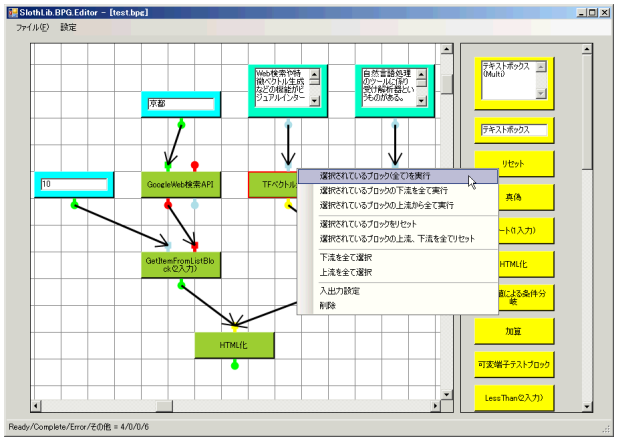

図 2　BlockProGramming の実行例

に，検索でえられた結果のタイトルとスニペットから TF を重みに使った特徴ベクトルを作成する．例えば，50 件の検索結果を取得したら 50 個の特徴ベクトルが作成される．それらのベクトルから，DF による特徴ベクトルを 1 つ作成する．これにより，Web 検索結果において，ユーザがはじめに入力した語とどのような語がよく共起するかが分かる．特徴ベクトルの要素を重みが大きい順にソートして上位 10 語を求め，各語とユーザの入力した語を合わせたものを新しいクエリとして推薦する．

コードとして書かなくてはいけないのは，ユーザインターフェースに関する部分を除けば，ほとんどアイデア部分であった．

4. BlockProGramming：ビジュアルプログラミング環境

研究でいろいろなアイデアを考えていると，大規模なプログラムを作るまでもないが，検索エンジンを使ったり特徴ベクトルを作成して実験を行いたいと思うことがある．そのような，ちょっとしたアイデアを試す環境として，視覚的にプログラミングを行える環境 BlockProGramming のプロトタイプシステムを作成した．SlothLib から利用できる機能を中心として，Web 検索や特徴ベクトル生成などの機能がビジュアルインターフェース上にブロックとして存在している．各ブロックはデータの入出力ポートを持ち，それらを接続していくことによってプログラムの流れを作成することができる．

図 2 が BlockProGramming の実行例である．右側には利用可能なブロックが並べられており，ドラッグして左側のスペースにドロップすることによって配置することができる．ブロックはデータの入出力を行うポートを持ち，各ポートはあらかじめ決められた型のデータを扱うようになっている．ある出力ポートからマウスをドラッグして別のブロックの入力ポートに矢印を接続することでデータの流れを決めることができるが，その際には接続可能な入力ポートに対して点線が表れ，どのポートに接続できるかユーザが簡単に分かるようになっている．また，一部のポートは入出力形式を設定画面上から変更できるようになっており，例えば，テキストボックスのブロックの出力は，整数，浮動小数，文字列のいずれかにできる．

ブロックの配置と入出力の接続によって作成されたプログラムは，保存することも可能になっている．

ブロックは，入力ポートに必要な情報が与えられている場合にのみ実行可能となる．実行可能のブロックはいつでも実行することができ，実行を終えると出力ポートにデータが現れる．ブロックの実行方法はいくつかあり，特定のブロックを指定して実行したり，あるブロックから下流に向かって連続的に実行したり，あるブロックを実行するために必要なブロックを上流にさかのぼって実行したりできる．

自らブロックを作成することも簡単に行うことができる．全てのブロックは IBlock というインターフェースを実装する．また，IBlock を実装する AbstractBlock という抽象クラスを継承することによって，些末な処理についてはほとんど考えることなくブロックを作成できるようになっている．矢印で表されるデータは，IData というインターフェースを実装している．ブロックの自作と同様に，データについても任意の情報を持つデータを簡単に作成できる．作成したブロックやデータは DLL 化してプラグインディレクトリに入れるだけで，自動的に BlockProGramming 環境で利用可能となる．

BlockProGramming 環境は，プログラミング初心者向けというよりもむしろ，ある程度プログラミングができる人が，プロトタイプシステムを作成するためのものである．例えば，作成しようとするシステムのある一部を BlockProGramming 環境で作成することなどが考えられる．BlockProGramming 環境では，入出力ポートにマウスカーソルを持って行くとデータの内容が表示されたりするなど，優秀なデバッグ環境でもある．

今後，BlockProGramming 環境で作成したプログラムを C#のコードに変換する機構を作成するつもりである．そうなれば，テストから実際のシステムの作成にあたって，コストをなくすことができるようになる．

5. まとめと今後の課題

本稿では，Web に関する研究のためのプログラミングライブラリ SlothLib の提案を行った．既存のツールやアルゴリズムの類似機能について共通インターフェースを提供しており，研究者はアイデアの実装により注力できるようになる．視覚的プログラミング環境 BlockProGramming も提案した．

現在，SlothLib は C#で実装されており，.NET 環境でのみ利用可能となっている．さらに多くの人が利用できるようにするために，今後の Java への展開を考えている．C#と Java は共にオブジェクト指向プログラミング言語であり，似ている言語であるため，多くの機能については簡単に移植は可能であると考えている．Java 以降の展開としては，Ruby への移植や SlothLib の機能の Web サービス化などの要望が寄せられており，考慮したいと考えている．

ブロックプログラミングについては，プロトタイプシステムという段階であり，今後，改良を行うつもりである．

謝　　辞

本研究の一部は，文部科学省 21 世紀 COE 拠点形成プログラム「知識社会基盤構築のための情報学拠 点形成」（リーダー: 田中克己，平成 14〜18 年度），文部科学省科学研究費補助金特定領域研究「情報爆発時代に向けた新しい IT 基盤技術の研究」計画研究「情報爆発に対応する新 IT 基盤研究支援プラットフォームの構築」（研究代表者: 安達淳，Y00-01，課題番号: 18049073）ならびに計画研究「情報爆発時代に対応するコンテンツ融合と操作環境融合に関する研究」(研究代表者：田中克己，A01-00-02, 課題番号 18049041），および，文部科学省研究委託事業「知的資産の電子的な保存・活用を支援するソフトウ ェア技術基盤の構築」，異メディア・アーカイブの横断的検索・統合ソフトウェア開発（研究代表者：田中克己）によるものです．ここに記して謝意を表します．また，SlothLib の開発には，稲川雅之氏，河重貴洋氏，木村塁氏，郡宏志氏，小谷彬氏，小西信次氏，近藤浩之氏，白砂健一氏，山口雅史氏，山本岳洋氏，吉田大我氏らの多大なる貢献を頂きました．ここに記して謝意を表します．

文　　献

[1] 阿部裕行, 伊藤一成, M. J.Dürst：“アノテーション処理のための基盤ライブラリの構築”, 電子情報通信学会技術研究報告，第 17 回データ工学ワークショップ (DEWS 2006), 1A-i5 (2006).
[2] A. Mendelzon, G. Mihaila and T. Milo: “Querying the world wide web”, pp. 80–91 (1996).
[3] T. Kistler and H. Marais: “Webl: A programming language for the web”, pp. 259–270 (1998).
[4] “Business Intelligence Development Studio”. http://msdn2.microsoft.com/ja-jp/library/ms173767.aspx.
[5] K. Ito and Y. Tanaka: “A visual environment for dynamic web application composition”, pp. 184–193 (2003).
[6] Y. Tanaka: “Meme media and meme market architectures for the reediting and redistribution of knowledge resources”, pp. 1–10 (1998).
[7] 松村郁生，宮浦宏暢，尾曽越雅文：“PatchService”. http://www.patchservice.net/.
[8] Wei-Ying Ma: “Building infrastructure to support web-scale data mining for search”. データベースと Web 情報システムに関するシンポジウム (DBWeb 2006) 内講演.
[9] G. Salton and M. McGill: “Introduction to Modern Information Retrieval”, McGraw-Hill (1983).
[10] G. Salton and C. Buckley: “Term-weighting approaches in automatic text retrieval”, Information Processing and Management, Vol.24, No.5, pp. 513–523 (1988).
[11] 北研二, 津田和彦, 獅々堀正幹：“情報検索アルゴリズム”, pp. 33–40, 共立出版 (2002).
[12] “The R Project for Statistical Computing”. http://www.r-project.org/.
[13] “CaboCha/南瓜: Yet Another Japanese Dependency Structure Analyzer”. http://chasen.org/ taku/software/cabocha/.