自然言語処理 125－4
（1998. 5. 29）

特定のプログラムに依存しない日本語形態素解析APIの設計

神林 隆, 風間 一洋
NTT 光ネットワークシステム研究所
東京都武蔵野市緑町 3-9-11

自然言語処理を行なうプログラムを複数のプラットフォームで同時に開発する場合、プラットフォームごとに異なる日本語形態素解析プログラムを使い分ける必要があり、独立性の高いプログラムを作成することは困難である。本稿では、まず、既存の日本語形態素解析プログラムに対して、プラットフォーム独立性を確保する時に問題となる点を調べるため、その機能を分析した。次に、これらの課題を解決すべく、特定の日本語形態素解析プログラムやプラットフォームに依存しない日本語形態素解析APIを設計した。さらに、設計方針に基づきJavaによる実装を行なった。

Design of Japanese Morphological Analysis API Independent upon a Particular Program

Takashi KAMBAYASHI, Kazuhiro KAZAMA
NTT Optical Network Systems Laboratories
3-9-11 Midori-cho Musashino-shi Tokyo

It is difficult to develop natural language processing programs on multiple platforms at the same time because different Japanese morphological analysis programs are required on different platforms. In this paper, we at first investigate functions of exsiting Japanese morphological analysis programs, in order to clarify issues on platform-independency. Next, we describe a design of a Japanese Morphological Analysis API that is independent of Japanese morphological analysis programs and platforms. Finally, we define several Java classes based on the design.

1 はじめに

計算機の発達・普及により、さまざまな分野で自然言語処理の要求が高まっている。計算機で自然言語を処理する場合、まず最初に形態素解析を行なって、文を構成する語を認定しなければならない。日本語のように、語と語の間に空白を置かずに記述する言語では、品詞や活用形を判定して、文を語に分割することになるが、通常、分割方法は複数考えられるために容易ではない。そこで、日本語の形態素解析に関するさまざまな研究やプログラムの開発が盛んに行なわれてきた。

我々は、インターネット上の情報探索を支援する分散検索システムを研究しており、今後システムを再設計するにあたり、マルチスレッド化による性能向上とプラットフォーム独立性の確保を狙って、新たにJava言語で全面的に書き直す予定である。しかし、現時点では、Java言語で記述された自然言語処理システムはいくつか開発中である[1][2]程度で、一般に利用可能なJavaで記述された日本語形態素解析プログラムはまだ公開されておらず、既存の日本語形態素解析プログラムをそのまま使用せざるをえない。さらに、動作するプラットフォームごとに別々のプログラムを使用しなくてはならない。

そこで、本稿では、日本語形態素解析を行なうユーザプログラムを書き直さなくてもさまざまな環境で動作するような、特定の日本語形態素解析プログラムやプラットフォームに依存しない日本語形態素解析APIを設計した。第2節では、従来の日本語形態素解析プログラムが持つ機能を分析し、その違いを吸収するために解決すべき課題を挙げる。第3節では、これらの課題を解決するために取った設計アプローチ、および、設計方針に基づいたJavaによる実装について述べる。

2 解決すべき課題

2.1 日本語形態素解析プログラム

形態素解析の目的は、文を構成する形態素を認定することである。形態素とは、正確には言語学的に意味のある最小言語単位のことであるが、形態素解析プログラムにおいては、辞書に登録されている単語のことを指す[3]。形態素解析では、文章を辞書に登録されている単語に分割する際に、ただ単に辞書に登録されている単語と比較するだけではなく、日本語の用言など活用変化があるものに対しては、辞書に登録されていない活用形から登録されている原形への復元を行ないながら解析を進める。

表1: 日本語形態素解析プログラム

プログラム名	開発元
茶筌 1.5 [4]	奈良先端大学 松本研
juman 3.4 [5]	京都大学 長尾研
ANIMA 1.0 [6]	日立 基礎研究所
Breakfast 4.0.4f [7]	富士通研究所
LAX [8]	ICOT

現在、入手可能な日本語形態素解析プログラムには、表1のものがある。本節の以降では、これらの日本語形態素解析プログラムの機能の違いと、そこから生じる課題について論じる。

2.2 利用形態の違いから来る課題

Javaプログラムから日本語形態素解析プログラムを利用する形態には、次の4種類がある。

- コマンドラインからの実行
- ライブラリ関数の利用
- 形態素解析サーバへのアクセス
- Java言語で実装したメソッドの呼び出し

このような利用形態の違いにより、以下のような課題が存在する。

2.2.1 データの受け渡し

ライブラリ関数を利用したり、Java言語で実装したメソッドを呼び出す場合には、日本語形態素解析プログラムとユーザプログラムの間で、構造を持つデータの受け渡しが可能である。これに対して、コマンドラインで実行した場合には、

データが文字列のようなフラットな構造で授受されるので、さらにそれを解析する必要がある。

2.2.2 マルチスレッド対応

本稿のAPIは、マルチスレッド化されたユーザプログラムから使用されることを考慮しているために、最大プロセス数や並行性などの制御が必要となる。サーバにアクセスする場合は、最大プロセス数はサーバ側で制御されるべきだが、コマンドラインから実行する場合に毎回プロセスを起動するとしたら、API内でプロセス数が一定値を越えないように制御する必要がある。また、マルチスレッド対応していないライブラリや、一つの形態素解析プロセスに複数の解析要求を逐次化して渡す場合には、API内部で並行性を制御する必要がある。

2.3 アルゴリズムの違いから来る課題

日本語文の形態素解析を行なうアルゴリズムのうち、主なものを以下に示す。

右方向最長一致法 文を左から右に見て、最も長い形態素を優先させて分割する方法。

文節数最少法 文を文節に分けた時、全体の文節の個数が最少になるものを優先する方法。

コスト最小法 隣接する形態素の接続妥当性(コスト)を接続表から得て、接続コストの最も小さいものを優先する方法。

上の二つは、経験則を用いた方法であり、処理が簡単で計算コストがかからないが、処理結果に問題が生じやすい。そこで、解析精度を上げるために、表1に示した日本語形態素解析プログラムすべてが、全解探索を行なえるコスト最小法を採用している。

アルゴリズムの違いにより、以下のような課題が生じる。

2.3.1 解析結果の利用

コスト最小法を採用している日本語形態素解析プログラムでは、コスト幅を指定することにより、最適解のコストとの差が指定したコスト幅以内の解を複数個まとめて解析結果として得ることが可能である。このようにして得られた複数解を表示する場合、得られた複数の解を各パスごとに順番に表示する「パスごとの表示」、および、パスの形を取らずに、複数解に含まれるすべての形態素を出現位置順に表示する「形態素ごとの表示」の二種類の表示方法がある。図1に、同じ文章を茶筌を用いて形態素解析した場合のそれぞれの表示例を示す。

(A) パスごとの表示

```
アルプス 固有名詞
のやま   普通名詞
は       副助詞
美しい   形容詞
。       句点
EOP
アルプス 固有名詞
の       名詞接続助詞
やま     普通名詞
は       副助詞
美しい   形容詞
。       句点
EOP
アルプス 固有名詞
の       格助詞
やま     普通名詞
は       副助詞
美しい   形容詞
。       句点
EOP
EOS
```

(B) 形態素ごとの表示

```
アルプス 固有名詞
の       格助詞
の       名詞接続助詞
のやま   普通名詞
やま     普通名詞
は       副助詞
美しい   形容詞
。       句点
EOS
```

図 1: 二種類の表示方法の例

結果をパスごとに表示すれば、各パスは文章を構成する形態素の列となっており、構文解析などに利用しやすい。形態素ごとに表示すれば、文章内に出現する辞書登録語がすべて形態素として得られるので、検索エンジンの索引作成に利用可能である。

2.3.2 複数パスの表示順序

コスト幅の指定により、最適解とのコスト差がコスト幅以内の解が得られるが、結果をパスごとに表示する場合に、表示されるパスの順番については、最適解が最初に得られることが保証されて

表 2: 日本語形態素プログラムの標準付属文法

プログラム	増岡・田窪 文法 [9]	森岡の 形態素論 [10]
茶筌	○	×
juman	○	×
ANIMA	○	×
Breakfast	○	○ [11]
LAX	×	○

いるだけで、それ以降の解に関しては、必ずしもコストの小さい順に並んでいるとは限らない。たとえば、茶筌ではコスト順ではなく、解析木を深さ優先で検索した順番に表示される。

2.4 辞書体系の違いから来る課題

2.4.1 文法辞書

文法辞書とは、日本語形態素文法を定義するものであり、形態品詞分類や連結規則などから構成される。表 2に、日本語形態素解析プログラムに標準で付属している文法辞書が準拠している文法規則を示す。

どの形態素解析プログラムにおいても、ユーザは自由に文法辞書を変更することができるが、解析文章のコスト計算は文法辞書に強く依存するので、形態素解析プログラムが同じでも、文法辞書を変更すると得られる結果も異なる。

2.4.2 単語辞書

単語辞書とは、個々の形態素についての情報を記述するものである。文法辞書と同じように、どの日本語形態素解析プログラムもユーザが自由に単語辞書を設定／拡張可能である。表 3に、茶筌に付属している辞書、RWC テキストデータベース、および、EDR 日本語単語辞書の 3 種類の単語辞書で使用されているトップレベルの品詞分類を示す。このように、単語辞書ごとに品詞分類が異なっており、使用する単語辞書の変更は解析結果に影響を与える。

表 3: 単語辞書のトップレベル品詞分類

品詞	茶筌	RWC	EDR
名詞	○	○	○
動詞	○	○	○
形容詞	○	○	○
形容動詞			○
判定詞	○		
助動詞	○	○	
指示詞	○		
副詞	○	○	○
助詞	○	○	
連体詞	○	○	○
接続詞	○	○	○
感動詞	○	○	
接頭辞	○		○
接尾辞	○		○
特殊	○	○	
語尾			○
構文要素			○
その他		○	○

2.5 エンコーディングの相違点

日本で使われている文字集合として、JIS X 0201、JIS X 0208、JIS X 0212 や ISO/ITC 10646 の JIS 規格である JIS X 0221 (Unicode[12]) がある。

JIS、日本語 EUC やシフト JIS の間では、基本的に相互変換が可能であるが、サポートしている文字に微妙に違いがあることがある。たとえば、JIS X 0201 のカナや JIS X 0212 がサポートされていないことがある。また、ベンダが独自に定義した文字は、互いに互換性があるとは限らない。また、ユーザ定義文字の使用が許されているとは限らない。

Java プログラム中で文字列は Unicode で扱われる。しかし、既存システムではほとんど Unicode は使われていないので、Java プログラム以外と文字列をやりとりする際にはエンコーディン

グ変換が行なわれる。Javaでは、文字エンコーディングの多様性に対して、コンバータを追加する方針で対処している。たとえば、Microsoft, IBM、Apple Computerなどの一部のベンダに関しては専用のコンバータが用意されており、ベンダが独自に定義した文字もUnicodeにマッピングすることができる。しかし、これらのコンバータはユーザによる変更ができないために、コンバータがサポートしていない文字集合やユーザ定義文字を扱うことはできない。また、既存のエンコーディングからUnicodeへのマッピングがベンダ依存であるために、別のUnicode文字シーケンスに変換されてしまうことがある。

3 APIの設計と実装

3.1 設計方針

今回は、次の方針に基づいて日本語形態素解析APIを設計した。

- 使用する日本語形態素解析プログラム、および、辞書の動的な変更が可能
- 日本語形態素解析プログラムや辞書を変更した場合でも、ユーザプログラムの変更は不要
- 使用する日本語形態素解析プログラムや辞書に合わせてカスタマイズ可能
- マルチロケール対応
- マルチスレッド対応
- Java APIのプログラミングスタイルを踏襲

3.2 デザインパターン

現在、オブジェクト指向プログラミングの分野においては、プログラムの設計を分析し、何度も使用される有用な設計パターンをカタログ化し、再利用を容易にするための作業が積極的に行なわれている[13]。これらのオブジェクト指向プログラミングにおける設計パターンを、デザインパターンと呼ぶ。

Javaのコア APIでは、これらのデザインパターンが積極的に活用されており、本APIも、これらのデザインパターンを踏まえて設計した。

3.3 仕様と実装の分離

特定の形態素解析プログラムや辞書に依存しないようにするためには、プログラムの仕様を定義する部分と実装を提供する部分を分離する方法が適切である。本APIでは、このためにBridgeパターンを使用した。形態素解析APIのBridgeパターンをOMT (Object Modeling Technique)を使って、図2に示す。

org.ingrid.text.MorphologicalAnalyzerクラスは、形態素解析の仕様を定義するためのクラスである。実際に形態素解析を行なう時に必要な処理は、org.ingrid.text.MorphologicalAnalyzerImpl抽象クラスのサブクラスとして、使用する形態素解析プログラムに応じて実装する。

本稿では、一つの実例として、茶筌サーバにアクセスするクラスを実装した。

3.3.1 パッケージ

APIはorg.ingrid.textパッケージに定義され、実際に形態素解析を行なうのに必要な処理は、別パッケージで定義される。本設計では、一つの形態素解析プログラムに対して、一つのJavaのパッケージを割り当てる。今回作成した茶筌サーバを使用して形態素解析を行なうクラスのパッケージ名はorg.ingrid.text.chasenである。同じようにjumanを使用するクラスを追加するには、org.ingrid.text.jumanパッケージに定義する。

パッケージ名のプリフィックスがorg.ingrid.textである理由は、Java言語仕様のパッケージの命名規則がドメイン名をベースにしているためである。別のパッケージを使用する場合には、パッケージ名のプリフィックスをシステムプロパティに追加する。プリフィックスは複数指定できる。

3.3.2 実装クラスの呼び出し

各実装クラスの呼び出しは、与えられた文字列からクラスを検索して返すjava.lang.Classクラスの forName() メソッドを使って実装されている。形態素解析オブジェクトを生成する時に、実際に

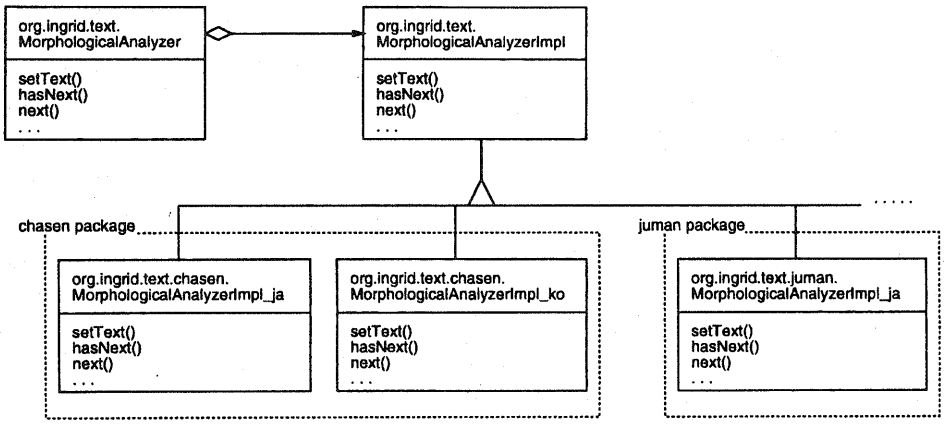

図 2: 仕様と実装の分離

どの実装クラスのインスタンスが生成されるかは、指定した形態素解析エンジン名、設定名、ロケールによって決定される。

Javaのロケールは、言語コード、国コード、バリアントコードの3つの要素から構成される。言語コードはISO 639で定義されている二文字の小文字アルファベットを使用し、たとえば日本語はjaで表される。国コードはISO 3166で定義されている二文字の大文字アルファベットを使用し、日本はJPで表される。バリアントコードは、ベンダやブラウザに依存するコードを示す。

追加パッケージが指定されていない場合には、次の順序で検索される。

1. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語_国_バリアント_設定
2. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語_国_設定
3. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語_設定
4. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語_国_バリアント
5. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語_国
6. org.ingrid.text.エンジン.MorphologicalAnalyzerImpl_言語

複数のパッケージが指定されている時には、すべてのパッケージを検索して最初に見つかった実装クラスを返す。これは、Javaのロケール検索手法を継承し、該当するロケールが見つからない場合に、できるかぎり近いロケールを見つけるためである。見つからなかった場合は、そのロケールがサポートされていないことを示す例外UnsupportedLocaleExceptionを発生する。

今回の茶筌サーバに対する実装では、日本語だけを処理するのでorg.ingrid.text.chasenパッケージのMorphologicalAnalyzerImpl_jaクラスとして実装した。茶筌の韓国語版がリリースされた場合には、同じパッケージにMorphologicalAnalyzerImpl_koクラスを追加すればよい。

形態素解析エンジン名が省略された場合には、デフォルト値としてchasenが使用される。ロケールが省略された場合には、デフォルトロケールに従う。

3.4 プログラムに依存する処理の分離

一般的に、APIのプラットフォーム独立性と高機能を両立させることは難しい。その最も大きな理由として、プラットフォーム独立なAPIを設計した場合には、APIが提供する仕様は、すべての機能の和集合ではなく、積集合に近い仕様になるからである。特に形態素解析特有の問題として、使用する辞書や文法に依存するコードがユー

ザプログラム中に分散する可能性が高いことが挙げられる。

そこで、次のようにコーディングすることで、APIの独立性を維持する。

1. 形態素解析全般に共通する処理や一般化できる処理は、org.ingrid.textパッケージのMorphologicalAnalyzerImplクラスに、個々の形態素に共通するデータや一般化できるデータは、同パッケージのMorphemeクラスに定義する。
2. 実際の処理やデータを記述するクラスは、プログラムや辞書ごとにサブクラスとして定義する。たとえば、org.ingrid.textパッケージのMorphemeクラスに対する、茶筌の標準付属辞書に関するデータを記述するサブクラスはorg.ingrid.text.chasenパッケージのMorpheme_jaクラスであり、茶筌の形態素データと同等の情報を持つオブジェクトとして定義される。
3. 辞書や文法などの設定の変更や機能拡張は、さらにサブクラスを作成し、そこで定義する。このサブクラスは設定名で指定する。

3.5 解析結果の取得

Javaのコア APIでは、文を単語に分割する以下のようなクラスが存在する。

- java.util.StringTokenizer …デリミタベースの分割処理。
- java.text.BreakIterator …ロケール依存の単語分割処理。ただし、辞書ベースの処理ではなく、日本語の場合、文字種で分割される。

本APIでは、同様に、個々の形態素を得る場合、Iteratorパターンを使用している。setText()メソッドで解析する文章を指定し、hasNext()メソッドで次の形態素が存在するを調べ、next()メソッドで次の形態素を値として得ることが可能である。

さらに、形態素をパスを構成する順番で取り出すことができる。この場合、Iteratorで取り出すパスの順番は、最適解とのコスト差がコスト幅以内の解がコストの小さい順に並ぶことを保証する。コスト幅は、setCostLimit()メソッドで指定する。また、辞書に登録されている形態素を出現位置順に取り出すことができる。

3.6 品詞定義

本APIでは、自立語を判定できるメソッドのみorg.ingrid.textパッケージのMorphemeクラスに追加した。ここでいう自律語とは、名詞，動詞，形容詞，形容動詞，副詞，連体詞，接続詞，感動詞の8つである。

3.7 マルチスレッドへの対応

マルチスレッド処理に対しては、次の2種類の配慮が必要になる。

- APIで提供するメソッドのThread Safe化
- 並行実行される時の並列度制御

本APIでは、使用する日本語形態素解析プログラムの利用形態とは無関係に、処理ごとに形態素解析オブジェクトを作成する。前者に関しては、メソッドを排他制御するためにsynchronized宣言するとともに、共有される部分の排他制御を実装側で行なう。後者に関しても、実装側で並列度制御を行なう。

今回の茶筌サーバに対する実装では、形態素解析オブジェクトごとに独立してサーバにアクセスするように記述できた。並列度の制御に関しては、本来、サーバ側で行なう必要があるので、今回は特に実装側で配慮しなかった。

3.8 エンコーディングの違いへの対応

本APIでは、形態素解析プログラムごとに専用のエンコーディングコンバータを作成し、サポート文字の判定や、サポートしていない文字が検出された時のUnsupportedCharacterException例外の発生を検討している。

また、プラットフォームごとの文字のマッピングの違いは、本APIとは別に、文字マッピングを

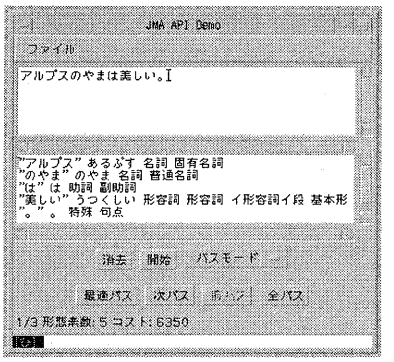

図 3: デモプログラムの実行例

```
MorphologicalAnalyzer ma =
        new MorphologicalAnalyzer();
Morpheme m;
ma.setText("アルプスのやまは美しい。");
while (ma.hasNext()) {
    m = ma.next();
    System.out.println(m.toString());
}
```

図 4: ユーザプログラムのサンプル

正規化するラッパークラスを作成して使用している。今後、Java の実装が改善された際には、このラッパークラスは不要になると思われる。

3.9 プログラム例

今回実装した日本語形態素解析 API を使用したデモプログラムを作成したので、その実行例を図 3に示す。この例では、得られた解析結果の第一候補 (最適解) を出力しており、これは図 4のようなプログラムで実行される。

4 おわりに

既存の日本語形態素解析プログラムの機能の違いを分析し、これらの違いを吸収する日本語形態素解析 API を定義した。さらに、Java を用いて、実際に茶筌サーバにアクセスして形態素解析を行なうクラスを作成した。

現在、次のような課題が残されている。

1. メモリ使用効率の向上
2. クラス構成の再検討
3. Iterator による解の取得方法の再検討
4. コスト幅の抽象的な値へのマッピングの検討
5. API で定義する品詞の妥当性の検討
6. サーバ以外の利用形態で実装する時の並列度制御
7. JDK 1.2 から追加されるクラス java.util.AttributedString の利用

謝辞 本稿を書くにあたり、アドバイスをいただいた沖電気工業 (株) 関西総合研究所の村田稔樹氏に感謝いたします。

参考文献

[1] 村田稔樹: "すべて Java で記述された機械翻訳システム PENSEE", 1998, <http://www.oki.co.jp/OKI/RDG/JIS/java/pensee/>.
[2] 今昭記: "Canna for Java 発表", 1997, <http://www.nec.co.jp/japanese/product/computer/soft/canna/1997/07/java.html>.
[3] 田中穂積: "自然言語解析の基礎", 産業図書, 1989.
[4] 松本裕治, 北内啓, 山下達雄, 平野善隆, 今一修, 今村友明: "日本語形態素解析システム『茶筌』 version 1.5 使用説明書", NAIST Technical Report, NAIST-IS-TR97007, 1997.
[5] 黒橋禎夫, 長尾真: "日本語形態素解析システム JUMAN 使用説明書 version 3.4", 1997.
[6] 櫻井博文, 久光徹: "日本語形態素プログラム ANIMA", 1997, <http://www.harl.hitachi.co.jp/~hirofumi/anima/index-j.htm>.
[7] 颯々野学: "日本語形態素解析システム Breakfast", 1997, <http://www.fujitsu.co.jp/hypertext/free/breakfast/>.
[8] 久保幸弘: "形態素意味解析システム LAX", 1992, <http://www.icot.or.jp/AITEC/IFS/IFS-abst/037-J.html>.
[9] 益岡隆志, 田窪行則: "基礎日本語文法 — 改定版 —", くろしお出版, 1992.
[10] 森岡健二: "語彙の構成", 現代語研究シリーズ 1, 明治書院.
[11] 佐野洋, 川田亮一, 永井桃子, 福本文代: "汎用日本語形態素解析規則 1.5 版", 1997, <http://www.fujitsu.co.jp/hypertext/free/breakfast/wjpmhp15.taz>.
[12] The Unicode Consortium: "The Unicode Standard, Version 2.0", Addison Wesley, 1996.
[13] 本位田真一, 吉田和樹: "オブジェクト指向における再利用のためのデザインパターン", ソフトバンク, 1995.