音声認識エンジン Julius/Julian の API 実装

住吉 貴志　李 晃伸†　河原 達也
京都大学 情報学研究科
〒 606-8501 京都市 左京区 吉田本町
† 奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県 生駒市 高山町 8916-5
e-mail: sumiyosi@kuis.kyoto-u.ac.jp

あらまし　我々が開発した音声認識エンジン Julius/Julian をアプリケーション開発者にとって利用しやすいものにするため、音声認識における API の仕様を考察し、SAPI 5.0 の実装を行った。その結果、認識エンジンの入出力が標準化され、各種モデルの切り換えが容易に行えるようになり、SAPI のアプリケーションにおいて Julius/Julian の統計的言語モデルと記述文法による音声認識機能が利用できるようになった。これらの成果物は連続音声認識コンソーシアム (CSRC) [1] において配布している。また Julius/Julian を統合し、複数の文法を処理する試みも行った。

Implementing Applications Programming Interface (API) for Speech Recognition Engines Julius/Julian

Takashi SUMIYOSHI　Akinobu LEE†　Tatsuya KAWAHARA
School of Informatics, Kyoto University, Kyoto 606-8501, Japan
†Nara Institute of Science and Technology, Ikoma 630-0101, Japan
e-mail: sumiyosi@kuis.kyoto-u.ac.jp

Abstract　We have developed speech recognition engine Julius/Julian. For easy development of application programs, desirable specification of API is examined, and then our ASR engines are ported to support SAPI 5.0. As a result, their interfaces are standardized, and their functions of speech recognition get available from SAPI applications. The software is distributed via CSRC (Continuous Speech Recognition Consortium). In addition, the extention to handle multiple grammars is explored.

[1] http://www.lang.astem.or.jp/CSRC/

1 緒論

我々が開発した大語彙連続音声認識エンジン Julius は、そのソースコードが公開されており、目的に応じて柔軟に改造が可能なため、ディクテーションシステム、音声対話システム、CALL システム、音声認識の基礎研究などで幅広く利用されている。

一方、近年の音声認識技術はある条件の下でなら既に十分実用の域に達しており、エンドユーザ向けの音声認識エンジンや音声アプリケーションも市場に出回っている。Julius をこのようなエンドユーザ向けのアプリケーションに適用させるためには、アプリケーションからエンジンを利用する手段が体系的である必要がある。アプリケーション開発者の立場から見ると、音声認識エンジンの機能が 1 つのモジュールとしてまとめられた形で提供されるのが望ましい。しかし Julius にはそのような整ったインターフェイスがなかった。

そこでこの問題を解決するために、エンジン開発とアプリケーション開発の双方を考慮した音声認識 API についての考察、Julius/Julian への Speech API (以下 SAPI) の実装、そして多くの音声アプリケーションにおいて必要となる複数文法処理機能の拡張を行った。

2 Julius/Julian の概要

Julius [1] と Julian [2] はともに京都大学音声メディア研究室で開発された大語彙連続音声認識エンジンである。図 1 に、これらのエンジンによる音声認識処理の流れを示す。

両者の違いは使用する言語モデル (制約文法) であり、Julius は統計的言語モデル (SLM) を、Julian は BNF による記述文法を用いるが、どちらも同じコンセプトの 2 パス探索アルゴリズムを採用している。

第一パスでは、単語 2-gram 確率 (Julian では文法カテゴリ対制約) を用いた left-to-right の時間同期ビーム探索を行う。入力音声の先頭から逐次、言語制約と音響尤度に基づき仮説をビームで絞りながら、トレリスの単語終端の評価値を格納する。トレリス形式では単語境界を非決定的に表現することができる。第二パスでは、単語 3-gram 確率 (Julian では記述文法を変換した FSA) と第一パスで求めたトレリス上の候補を用いた right-to-left のスタックデコーディングを行う。第一パスの結果をヒューリスティックとした A*探索によって効率的に最適解を求める。

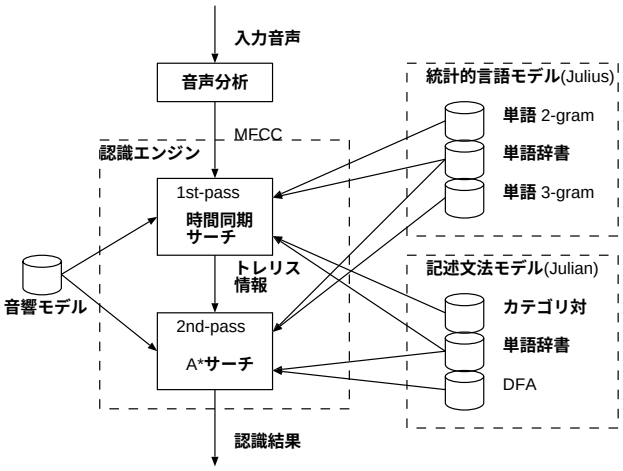

図 1: Julius/Julian による音声認識のブロック図

Julius は IPA 「日本語ディクテーション基本ソフトウェア」における認識デコーダとして、ディクテーション用の単語辞書、統計的言語モデル、音響モデルなどとともに配布されている [3][4]。

3 アプリケーションと音声認識エンジンの関係

一般的なコンピュータ環境において音声認識エンジンを用いたアプリケーションの操作を行う場合、2 種類の方法が考えられる。

1 つは、音声認識エンジンの認識結果を各アプリケーションが完全に受動的に利用する方法である。アプリケーションは音声入力を想定して作成されたものである必要はなく、認識エンジンが音声入力をキーボード入力やマウス入力などのイベントに変換してアプリケーションに渡すというものである。この方式は、従来のアプリケーションをそのまま使用できる利点がある一方、アプリケーションからエンジンを制御すること (文法、語彙を変更するなど) が原理的に不可能である。

そこで本研究ではより効率的で高い精度も得られ、また利便性の高い、音声入力を想定して作成されたアプリケーションがエンジンを能動的に利用するという方式を考える。

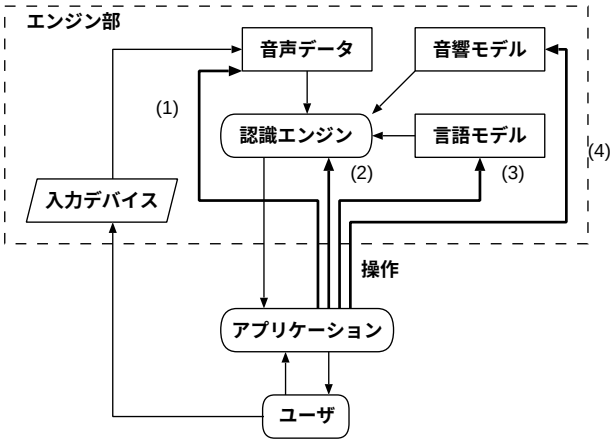

図 2: 音声アプリケーションからの認識エンジンの制御

4 音声認識エンジンの API

音声認識のみならず、一般的に API の仕様策定の段階で重要なのは、どれだけの機能を仕様に取り入れるか、という点である。もちろんそれはアプリケーション開発に十分な機能を満足していなければならないが、あまり必要のない機能を仕様に組み込むと認識エンジンの処理が非効率的になるおそれがある。

音声認識システムにおけるエンジンとアプリケーションのブロック図を図 2 に示す。図のアプリケーションから認識エンジンへの 4 つの操作について、どの操作を API として定めるのが適切かを検討する。

(1) 音声の入力

一般的なタスクではマイク入力などのオーディオデバイスによる音声入力を想定するので、音声入力に関してはアプリケーションから隠蔽するとよい。さらに、標準のライブラリを用意することにより、各認識エンジンが低水準な入力デバイスを操作する必要もなくなる。

今回の実装においても、アプリケーションやエンジンが低水準なデバイスを管理する必要がなくなった。

(2) 認識エンジンの設定 (精度、速度など) の指定

大語彙連続音声認識は比較的多くの CPU 資源とメモリ資源を消費する。

計算資源の少ない環境では、各種モデルの簡易版を利用したり、高速な探索アルゴリズムを使うなどして、多少認識の精度を落としてでも実時間に近い処理速度で動作させるべきである。すなわち、認識エンジンを完全にアプリケーションから隠蔽してしまうのは不都合であり、これらの設定をユーザが行える機能が要求される。

今回の実装では Unix 版 Julius/Julian で用いられてきた設定ファイル (jconf ファイル) を利用できるようにした。

(3) 認識に使用する言語モデルの指定

音声認識に使われる言語モデル (=文法) には大きく分けて統計的言語モデルと記述文法がある。

統計的言語モデルは、ディクテーションや記述文法で表しにくい一般的な言語の制約として使用される。現在において、統計的言語モデルを用いるのは汎用的なディクテーションが主流であり、単一の言語モデルと認識エンジンが一体化しているのが実状であるので、フォーマットについても標準的な仕様は必要ないと考えられる。

記述文法は、アプリケーションに対するコマンドなど、語彙や文法が限定された定型的な文を認識するのに使われる。タスクやプロンプトによって受理すべき単語や文は異なるので、アプリケーションの指定により単語や文法を切り替えられる機能が必要である。この場合、アプリケーションが用意する文法はその API を実装するあらゆる認識エンジンで利用できなければいけないので、標準の文法フォーマットを定める必要がある。文法の解析や単語の発音情報 (読み) の付与、また各文法の状態の管理など、認識エンジン一般で利用する機能はライブラリが用意されるのが望ましい。

今回の実装では、2 万語彙、6 万語彙の言語モデル (それぞれ非圧縮、圧縮 3-gram) を、認識エンジンが用意するダイアログボックスによって切り換えられるようになっている。

(4) 認識に使用する音響モデルの指定

音声認識では、汎用的な音響モデルを使うよりも、使用するユーザ (性別、年齢など) や入力環境 (オフィス、車内、電話など) に応じて適切な音響モデルを使用する方が認識精度の点で望ましい。したがって、

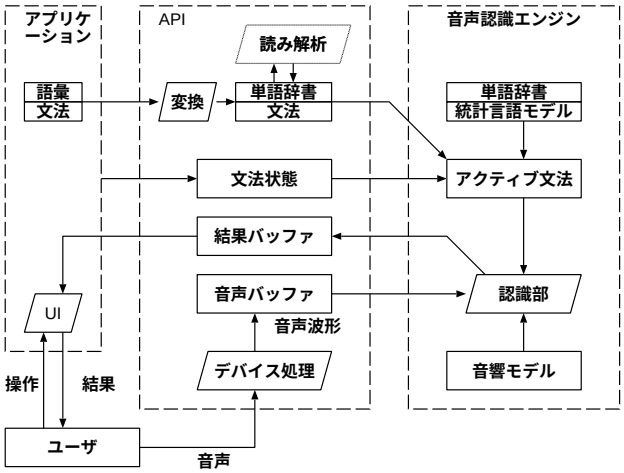

図 3: API を利用した音声認識のブロック図

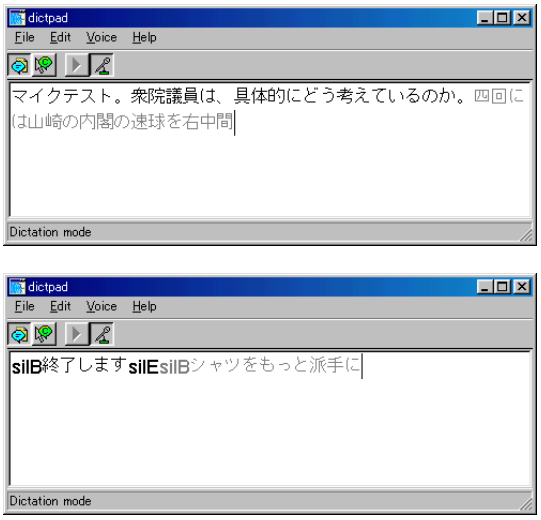

図 4: Dictation Pad における Julius for SAPI(上) と Julian for SAPI(下) の動作画面

音響モデルを変更できるようにすべきであるが、認識エンジンが使用する音響モデルの種類を統一することは難しいので、ユーザや環境の抽象的な区分 (男性/女性、静かな部屋/電話 など) を決定しておくことが望ましい。ただし、パソコンでの使用など、ユーザが限定されるときは、プロファイルで指定するのが一般的である。

今回の実装では、精度 (モノフォン/トライフォン/PTM)、ユーザカテゴリ (男性/女性/性別非依存、高齢者)、使用環境 (電話帯域) 毎に用意された音響モデルを、認識エンジンのダイアログボックスにおいて切り換えることができる。また、SAPI ではユーザプロファイル情報を管理できるようになっているので、ユーザごとに指定できる実装も可能である。

以上をまとめた、API を利用した音声認識のブロック図を図 3 に示す。

5 Julius for SAPI

5.1 SAPI について

Microsoft® 社は Windows® Speech API と呼ばれる標準規格を提唱し、2000 年に Speech API 5.0 [5] と SDK [1] をリリースした。SAPI はアプリケーションが音声認識と音声合成を統合的に扱えるように規格化されている。音声認識では、認識エンジンの種々の機能を API として規格化し、アプリケーション開発者、エンジン開発者に提供するものである。また SAPI は、ウィンドウシステムにおけるマルチタスクアプリケーションのインターフェースとしての音声の利用を考慮した設計となっており、今後、 Windows における音声アプリケーションの標準的な基盤となることが予想される。

今回、Julius と Julian をそれぞれ別のモジュール “Julius for SAPI”、“Julian for SAPI” として、Windows に移植するとともに、SAPI への対応を行った。

5.2 実装と動作実験

実装は、これまで開発された Unix 版の Julius と Julian (rev.3.2) のソースを移植する形で行った。Microsoft Speech SDK に付属のサンプルアプリケーション Dictation Pad 上で動作している (図 4)。

さらに、認識性能をオリジナルの Unix 版と比べるための実験を行った。言語モデルは 2 万語彙 (圧縮 3-gram) を、音響モデルは性別非依存の PTM を、評価用の音声サンプルには IPA-98-TestSet のうち 20 発話 (計約 113 秒、16kHz,16bit,Mono,wav 形式) を用い、実行開始から初期化終了までの時間と、20 発話を連続で認識させるのに要した時間を測定した。実行環境と測定結果を表 5 に示す。

キャッシュ・ヒット等の誤差のため、この結果を正

[1] Software Development Kit : 開発ツール群

	Julius/Julian	Julius/Julian for SAPI
CPU	PentiumIII 600MHz	
メインメモリ	320MB	
OS	Linux 2.2.14	Windows 98 SE
コンパイラ	gcc 2.95.2	Visual C++ 6.0
最適化	-O2	実行時間最適化
初期化時間	8 秒	18 秒
認識時間	178 秒	177 秒

図 5: 認識速度の比較

確に比較することは難しいが、Linux OS で動くオリジナルの Julius/Julian と同程度の認識速度で動作することが確認された。

5.3 課題

現段階の実装は、SAPI の要求仕様を完全には満たしておらず、完全にコンプライアンスなものにするには、以下の 2 つの課題を解決する必要がある。

- SAPI 文法 (XML) の処理

  SAPI の仕様では XML 形式で書かれた SAPI の文法を認識エンジンが処理する必要がある。SAPI の想定する文法は文脈自由文法である。これは Julian のサポートする正規文法のクラスよりも大きいため、そのまま移植することはできない。

  今回のバージョンでは、6 章で述べる Julian の文法フォーマットをサポートするという形となった。

- マルチコンテクスト・マルチインスタンス処理

  SAPI ではコンテクスト (アプリケーションの処理単位) やインスタンス (アプリケーションから見えるエンジンそのもの) をエンジンが複数管理できなければならない。

  この項目に関しては 7 章で考察と対処を述べる。

6 Julian の文法フォーマット

Julian は、2 章で述べたように、記述文法を言語制約として利用する音声認識エンジンである。タスク (サブタスク) 毎に文法ファイルと辞書ファイルを用意する。

文法ファイル (.grammar)

```
S : NS_B NAME NS_E
S : NS_B NAME FILL_E NS_E
S : NS_B FILL_B NAME FILL_E NS_E
S : NS_B FILL_B NOISE NAME FILL_E NS_E
S : NS_B ANSWER NS_E
```

辞書ファイル (.voca)

```
% NAME
小泉    k o i z u m i
森山    m o r i y a m a
田中    t a n a k a

% FILL_B
えーと  e: t o
あのー  a n o:

% FILL_E
さん    s a N
さんお願い  s a N o n e g a i
```

図 6: Julian の文法、辞書ファイルの例

文法は正規文法 (有限状態文法) に対応しており、BNF 形式で記述する。この終端記号は 1 つの単語カテゴリを表わし、辞書ファイルにおいて各カテゴリに 1 つ以上の単語とその発音を登録する。

サンプルを図 6 に示す。

7 複数文法処理

音声対話システムにおいては、対話の進行に伴って予測されるユーザ発話の語彙や文法が変化する。例えばフォームフィリングタスクでは、日付をたずねたり行き先地をたずねたりするので、プロンプト毎に受理する語彙が変わる。また、WWW ブラウザではページごとに語彙が変化する。このように、アプリケーションの実行中に受理する文法や語彙を変更する場合、毎回認識エンジンを再起動させると初期化に時間を要する。

また、ウィンドウシステムなどのマルチタスク環境においては、同時に複数の音声アプリケーションを動作させたい場合があり、単一の認識エンジンでこれらを扱えることが望ましい。

これらの問題は、アプリケーションが起動するごとに複数の文法を読み込み、文法の変更を動的に行うことで対処が可能である。入力音声に複数の文法を適用し、認識結果を得るようにする。ディクテー

ション時にアプリケーションの定める特定のコマンドを受理することも可能になる。この複数文法への対応を試みた。

1 つのディクテーション文法 (統計的言語モデル) と 2 つのコマンド文法 (記述言語モデル) を用いて動作実験を行った。認識ごとにある文法をアクティブにし、それ以外の文法を非アクティブにする。初期化時に全ての文法を読み込むため、初期化時間と動作中の使用メモリ量が増大したが、切り換え時、認識時の動作時間は従来とほとんど変わらないことが確認された。

8 おわりに

本稿では、音声認識の API について考察し、認識エンジン Julius/Julian に API を実装する上での問題点に対する検討と解決法を提示した。その上で、SAPI5.0 の仕様に沿って Julius/Julian の音声認識機能を持ったエンジンの実装を行い、SAPI のアプリケーションにおいて利用できるようにした。また、エンドユーザ用の音声アプリケーションで必要と考えられる複数文法処理機能の Julius/Julian での実現を行った。

今回実装した Julius for SAPI と Julian for SAPI は SAPI の要求仕様をまだ完全に満たしていないが、今後行う作業を列挙しておく。

- SAPI 標準形式の記述文法への対応 (ワイルドカード遷移、ディクテーション遷移、テキストバッファ遷移などの特殊な遷移を含む)
- 複数文法処理の組み込み
- プログラムのオブジェクト化

参考文献

[1] 李晃伸, 河原達也, 堂下修司: 文法カテゴリ対制約を用いた A*探索に基づく大語彙連続音声認識パーザ, 情報処理学会論文誌, Vol. 40, No. 4, pp. 1374–1382 (1999).

[2] 李晃伸, 河原達也, 堂下修司: 単語トレリスインデックスを用いた段階的探索による大語彙連続音声認識, 電子情報通信学会論文誌, Vol. J82-DII, No. 1, pp. 1–9 (1999).
http://winnie.kuis.kyoto-u.ac.jp/pub/julius/

[3] 河原達也, 李晃伸, 小林哲則, 武田一哉, 峯松信明, 嵯峨山茂樹, 伊藤克亘, 伊藤彰則, 山本幹雄, 山田篤, 宇津呂武仁, 鹿野清宏: 日本語ディクテーション基本ソフトウェア (99 年度版) の性能評価, 情報処理学会研究報告, 2000-SLP-31-2 (2000).
http://winnie.kuis.kyoto-u.ac.jp/dictation/

[4] 鹿野清宏, 伊藤克亘, 河原達也, 武田一哉, 山本幹雄 編著: 音声認識システム, オーム社 (2001).

[5] Microsoft: *Microsoft Speech SDK version 5.0* (2000).
http://www.microsoft.com/speech/