Vol.2010-SLP-80 No.16
2010/2/13

ウェブベースの分散型音声認識アプリケーション開発プラットフォームに関する一提案

中野 鐵兵　佐々木 浩
藤江 真也　小林 哲則[†1]

複数のウェブベースのサービスの連携として構築されるウェブベースの音声認識アプリケーションの開発に利用可能な，分散型音声認識システムのためのアーキテクチャについて提案と考察を行う．今日では実利用環境で利用可能な分散型の音声サービスが提供されて来てはいるものの，それらの利用は単一のアプリケーションに閉じており，他のサービスと組み合わせて新たなアプリケーションとして提供することは困難である．そこで，ウェブサービスのおけるマッシュアップのような，連携による相乗効果を生み出す音声認識アプリケーションの開発環境を実現するために必要な枠組みとして，シンプルなプロトコルでの連携を可能にするオープンな分散型音声認識サービス用アーキテクチャの提案を行った．提案手法では，RESTful Web Services を参考に RESTful 分散型音声認識サービスを設計し，標準的なウェブ技術 (HTTP, URI, XML) を用いた連携を可能とした．リソースとしては，エンジンファクトリ，仮想音声認識エンジン，入力デバイス，出力デバイスを定義し，統一インタフェースと操作・振る舞いとの割り当てを行い，それらの実際の実装例に関する特徴を述べた．

A Proposal of the Development Platform for Web-based Distributed Speech Recognition Applications

Teppei Nakano , Hiroshi Sasaki , Shinya Fujie and Tetsunori Kobayashi[†1]

This paper presents a proposal about a software architecture for distributed speech recognition systems. This architecture are designed for the developments of web-based speech recognition applications which can be syndicated with some services. The motivation of this research is the fact that most of commercial and public distributed speech recognition systems are “closed”. This research, therefore, proposes an open software architecture. This architecture, an architecture for RESTful distributed speech recognition services, enables for services to collaborate with other services by using simple and standard protocols: HTTP, URI, and XML. Furthermore, the detail design of resources and samples of the implementation are also presented.

1. はじめに

ブロードバンドインターネット環境が普及し，ユビキタス環境の実現が近づいている今日では，音声認識に関する処理をクライアント・サーバで分散させた，分散型音声認識システム[1)2)]を利用した音声認識アプリケーションが実用的になっている．分散型音声認識システムは，音声データの取得や，特徴量の抽出と言った比較的計算負荷の低い処理のみがクライアント側で行われる．そのため，ポータブルデバイスのような，計算機のリソースが十分でなく，従来では大規模な音声認識システムの導入が困難であった環境においても，音声インターフェースの利用が可能となっている[3)]．それに対し，音声認識における後段処理が実行されるサーバ側では，クライアント側で処理されたデータをネットワーク経由で受信しながら，計算負荷の高い処理の実行が要求される．特に，実際にアプリケーションを公開するためには，複数のクライアントからの同時要求を音声認識サーバが処理する必要がある．また，インターネット経由での接続を確立し，クライアントとサーバの連携を可能にするための効果的な枠組みも必要となる[4)]．しかしながら，これらのクライアントとサーバが連携するための標準的なプロトコルやサービスも普及しておらず，誰もが容易に分散音声認識システムを利用したアプリケーション開発を行える状況にはない．

他方，今日のウェブ関連サービスの急速な発展により，従来では企業内で囲い込まれていた様々な機能や資源がウェブサービスとして公開されるようになった[5)]．その結果，アプリケーション開発者は，これらの公開されたサービスを統合することで，高品質な独自アプリケーションの開発が可能となっている．ここで開発されたアプリケーションが，さらに外部に対してサービスを提供することで，相乗効果が生まれ，アプリケーション開発を容易にしている．これらの枠組みはマッシュアップと呼ばれ，今日のウェブ技術を発展を支える重要な要素となっている[6)]．それに対して分散型音声認識アプリケーション開発においては，このような相乗効果を産み出すような環境が未だ整備されていない．すなわち，モバイル端末を中心に，実利用環境で利用可能な分散型の音声サービスが提供されて来てはいるものの，それらの利用は単一のアプリケーションに閉じており，他のサービスと組み合わせて新たなアプリケーションとして提供することは困難である．特に実利用環境で実際に利用可能な品質の音声インターフェースの開発には，既存の技術の効果的な組み合わせが重要であるにも関わらず，そのために必要な基盤技術は十分に整備されていない．ウェブサービスのおけるマッシュアップのような，連携による相乗効果を生み出す音声認識アプリケーションの開発環境を実現するためには，シンプルなプロトコルでの連携を可能にするオープンな分散型音声認識サービス構築の枠組みが必須である．そこで本研究では，広く成功している既存のウェブサービスを参考に，サービス間の連携を容易にするために必要な分散型音声認識サービスのアーキテクチャの提案とその検討を行う．

2. ウェブサービスとしての分散型音声認識サービス

音声認識サービスをウェブサービスとして他のサービスと連携させて動作させるために解決が必要な課題について述べる．

†1 早稲田大学
Waseda University

2.1 音声データ入力への対応

まず，一般のウェブサービスと，音声サービスとのもっとも大きな違いとして，サービスに対する入力データの特徴が挙げられる．通常のウェブサービスの場合，連携のために用いられる入力データが，文字列や数値のような，少量の離散値である．そのために，ウェブベースのアプリケーションやスクリプト言語での取り扱いが容易であり，ウェブサービス側が通常のウェブ API を用意するだけで，他のサービスやアプリケーションに対する埋め込みがシームレスに行われる．

それに対して，音声認識サービスで用いられる入力データは，音声データそのものや，特徴量抽出された後の大量の連続値・バイナリデータである．バイナリデータの取り扱いには，データに関するフォーマットの指定やデータの入力方法・変換方法に関する枠組みが必要であり，従来のウェブサービスとは単純に連携させることが困難である．この問題はウェブベースの音声認識アプリケーションを構築する上で対応が必須な項目でもあり，音声インタフェースをサポートするようにブラウザを直接拡張する手法[7] をはじめ，特定ブラウザのプラグインを用いる手法[1]，Java アプレットを用いてウェブページに音声入力用インタフェースを埋め込む手法[2][8] 等，様々な方式で実現されている．そのためにも，ウェブサービス側でも通常のウェブ API を用意するだけでなく，バイナリデータを適切に扱うための枠組みの提供が求められる．

2.2 ステートフルなサービスへの対応

今日のウェブサービスの多くは，全ての処理の要求が，その前後関係に依存しないステートレスなサービスとして提供されている．すなわち，処理の実行に必要な情報は全てクライアントからのリクエストに含まれており，サーバはその時に与えられた情報のみで処理を完了させることが可能である．

それに対して音声サービスでは，本質的に全てのサービスをステートレスで構築することは困難である．もっとも単純なサービスの場合は，クライアントからのリクエストに認識対象のデータを含め，レスポンスとして認識結果を返すモデルでの構築が可能である[8]．しかしながら，通常の音声認識システムで使用されているような，文法の指定や変更，話者適応技術等を用いる場合，ステートフルなサービスを可能にする枠組みが必要となる．

2.3 非同期出力への対応

一般的なウェブサービスでは，サービスに対するリクエストとサービスからのレスポンスが同期しているものとして扱われる．すなわち，ウェブサービス側ではクライアント側からの入力が全て完了してから処理を実行し，結果を出力するだけでよい．

しかしながら音声認識サービスでは，認識対象となる全てのデータを受信してから，認識処理を行い，その後結果を返すというモデルだけでは十分でない．従来の音声認識システムには VAD や早期確定の枠組みを備えたものが存在し，一度の音声入力に対して，複数の結果を返したいことがある．もし，分散音声認識システムでのレスポンスがリクエストと同期した出力しか扱うことが出来なければ，VAD や早期確定の枠組みを活用することが出来なくなる．例えば，音声対話システムを対象とした音声サービスの場合，入力音声は対話の長い文であり，VAD や早期確定を前提に対話の手続きが設計されているものがあるが，それらの使用が困難となる．また，入力データが大きいバイナリデータの場合，全てのリクエストを受信してからの認識処理を開始するのでは，実時間で認識処理の完了が困難となる問題も存在する．

3. 基本アプローチと提案手法

前節で述べた課題を解決し，他のウェブサービスとの連携を可能にするために設計した，ウェブベースの分散型音声認識アプリケーション開発プラットフォームのアーキテクチャについて，その基本的アプローチを述べる．

3.1 RESTful リソース指向アーキテクチャ

RESTful リソース指向アーキテクチャ[9] とは，REST (Representational State Transfer) のアーキテクチャスタイル[10] に従い，今日の標準的なウェブ技術 (HTTP, URI, XML) を用いて具体的に設計された，ウェブサービス用のアーキテクチャである．REST では，情報の断片としての “リソース” にそれぞれ固有の “URI” を割り当て，“統一インタフェース” を用いて “表現” を送受信することでその状態や状態変化を伝達する．特に，URI として URL を，統一インタフェースとして HTTP を，表現として，XHTML や XML，その他ウェブ技術において標準的なハイパーメディアを定義することで，ウェブサービスを構築するためのアーキテクチャとして設計された．このアーキテクチャはリソース指向アーキテクチャと呼ばれ，今日で最も成功しているウェブサー ビス用のアーキテクチャの 1 つである．

RESTful アーキテクチャでは，まずシステムの機能やデータ・状態をリソースとして表現する．各々のリソースには一意に識別可能 URI を割り当て，HTTP 経由でのアクセスを可能にする (アドレス可能性)．また，リソースへのアクセスに関しては，全ての HTTP リクエストを完全に分離させる (ステートレス性)．すなわち，一連のリソースへのアクセスに関して，クライアント側で維持されるアプリケーション状態は，全て HTTP リクエストに含めなければならない．サーバ側では，リソース状態のみを維持し，一回の HTTP リクエストに含まれる情報のみから処理が実行される．HTTP のリクエストとレスポンスで伝達される情報は，特定のリソースに関するデータであり，特定のファイル形式と特定の言語で送信される．これをリソースの表現と呼び，リソースの状態に関する有益な情報を提供する．また，表現をクライアントのリクエストに含めることで，新しい表現と一致するようにリソースを変更させる．

さらに，表現をハイパーメディアとして，他のリソースへのリンクを含めたドキュメントとする．すなわち，リソースをそれらの表現において相互のリンクを可能にする (接続性)．こうすることで，特定のリソースの表現を入手したクライアントは，その表現に含まれるリンクをたどることで，現在の状態から関連する他の状態へ移行することが可能となる．また，状態を変化させるためのリソースに対する操作を全て HTTP の基本的なものに限定する (統一インタフェース)．すなわち，分散オブジェクトやリモートプロシージャコールに代表される，従来のクライアント・サーバシステムのように，サーバ側のシステムが独自の操作用インタフェースを定義するのではなく，表 1 に示すような，HTTP の基本的なメソッドを用いることで，クライアント・サーバ間で表現を伝達し，それぞれの状態を操作する．

表 1　HTTP の基本メソッドと統一インタフェース

HTTP GET	リソースの表現を取得する
HTTP POST	新しいリソースを作成する
HTTP PUT (新しい URI に対して)	新しいリソースを作成する
HTTP PUT (既存の URI に対して)	既存のリソースを変更する
HTTP DELETE	既存のリソースを削除する
HTTP HEAD	メタデータ固有の表現を取得する
HTTP OPTIONS	特定のリソースがサポートする HTTP メソッドを調べる

3.2 RESTful 分散型音声認識サービス

RESTful リソース指向アーキテクチャに基づいて，第 2 節で述べた課題を解決するように RESTful 分散型音声認識サービスのアーキテクチャを設計した．課題解決の為の具体的なアプローチを以下に示す．

- 各々のクライアントの一連の処理要求毎に専用リソースを提供する
- 音声入力と結果の取得，及び入出力と実際の認識処理を分離する

専用リソースは，特定のクライアントからの入力と，その入力に対する認識結果を関連付けて保持する，一時的な専用仮想音声認識エンジンのリソースとして定義される．クライアントは，音声認識サービスを利用する際には，まず新規の仮想音声認識エンジンのリソースを作成し，そこに対して認識処理を依頼する．このリソースは，ライフサイクルが短く，対象とするクライアントこそ限定されるものの，通常のリソースと同様にリソース状態が保持される．すなわち，この専用リソースを用いることで，特定のクライアント用のアプリケーション状態を，リソース状態として保持することが可能となる．

次に，REST のスタイルに従い，仮想音声認識エンジンに関連付けられた別のリソースとして，認識対象となるデータの入力を受信するための入力デバイスリソースを定義する．認識処理を実行するためには，クライアントはその入力デバイスリソースに対して音声データを送信する．ここで送信された情報は，仮想音声認識エンジンリソースの状態として保持され，実際の音声認識システムによって適切なタイミングで認識処理が実行される．さらに処理の結果得られた認識結果を格納するための，出力デバイスリソースを，これも仮想音声認識エンジンに関連付ける形式で定義する．クライアントは，このリソースに対してリクエストを行うことで，認識結果を得ることが可能となる．

これらのモデルにより，文法の指定や変更，話者適応技術等に加え，入力データ，認識結果の全てがこのリソースに対する状態の変化として扱うことが可能となり，特定のクライアントからの一連の認識処理をステートフルなモデルで取り扱うことが可能となる．すなわち，あるウエブサービスでアカウントを作成すると同時に，そのアカウント用のメールボックスが生成され，そこにメールを格納することが出来るのと同様のモデルである．また，入力デバイスリソースが音声データ入力に対応する専用のリソースとして提供されることで，その他の全ての処理を一般的なリソースとして取り扱うことが可能となる．さらに，入力デバイスと出力デバイスに加え，実際の認識処理の行われるタイミングを分離することで，仮想認識エンジンは任意のタイミングで出力を生成することが可能となる．クライアントは，入力デバイスに対するデータの送信と，出力デバイスからの結果の受信を同時に行うことで，非同期な結果出力への対応が実現される．

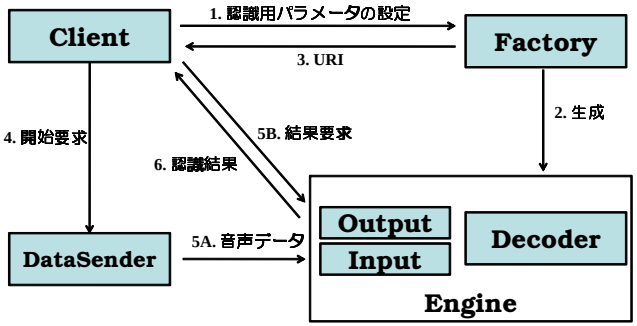

図 1　RESTful 分散音声認識システムのフロー

4. RESTful 分散型音声認識サービスのリソース定義

本節では，RESTful 分散型音声認識サービスのアーキテクチャについて，以下のリソースの詳細と各々の関連 (図 1) を述べる．

- エンジンファクトリ
- 仮想音声認識エンジン
- 入力デバイス
- 出力デバイス

前節で述べたように，提案手法では，1 つのクライアントに対して 1 つのエンジンリソースが対応するものとして専用のリソースを定義する．すなわち，クライアントは認識処理を開始する際には，常に新規のエンジンリソースが必要なモデルとして設計する (図 1(1-3))．エンジンファクトリは，その新規のエンジンリソースを作成するためのリソースとして定義される．クライアントは，その前段処理によって得られたデータを，サーバ上の入力デバイスリソースに対して送信する (図 1(5A))．サーバでは，入力デバイスに対して送信されたデータを用いて認識処理を実行し，結果を出力デバイスに設定する．クライアントは，認識結果を取得するために，出力デバイスリソースに対してアクセスを行い (図 1(5B))，データ入力とは非同期に結果を取得する (図 1(6))．

4.1 エンジンファクトリ

ファクトリリソースは，事前に公開された URI を用いてクライアントからアクセスされる．

4.1.1 統一インタフェース

GET　このリソースに関連する情報を取得する．例えば，生成されるエンジンの種類や，エンジンリソースを生成するためのパラメータ，過去に生成されたエンジンリソースの一覧を取得することが可能である．どのような情報の取得が可能かはサービスの実装が自由に定める．ただし，クエリなしでのアクセスの際には，それらの情報の取得方法が得られる．また，RESTful ウェブサービスのスタイルに従い，ここではリソースの状

態の変化を伴わない．

POST 新規のエンジンリソースを作成する．作成するエンジンのパラメータは，リクエストのエンティティボディにて設定を行う．例えば下記に示す項目をパラメータとして設定可能にする．

- 利用者の年齢・性別，言語等の属性情報
- 利用するエンジンの種類や，語彙情報等のエンジンの構成情報
- ユーザ ID 等のシステム側で利用可能な情報

新規のエンジンリソースが作成されたらば，システムはそのリソースにアクセスするための URI をクライアントに示す．具体的には，HTTP レスポンスコードの 201(Created) を用いてレスポンスヘッダーにリソースの URI を設定する．

```
POST /engines HTTP/1.1
Host: example.com

HTTP/1.1 202 Created
Location: /engines/2a383b
```

すなわち，クライアントが新たに作成されたリソースの表現を得るためには，Location への直接アクセスを求める．また，作成したリソースの URI は，要求元のクライアントの IP アドレスや日時，パラメータと合わせて保存され，後で参照可能となる．

4.1.2 表　　現

クライアントに提供する表現としては，以下の 2 種類を用意する．これらの切り替えは，Accept ヘッダーやクエリ文字列によってクライアント側から自由に指定出来るようにする．

- 開発者用（人間用）の表現
- アプリケーション用の表現

開発者用の表現としては，HTML のように，可読性の高い表現方法を用いる．例えば，自身のアプリケーションに対してどのように音声認識サービスを組み込むかに関する情報を，サンプルをつけて説明する．また，ユーザ登録が必要な場合は，そのためのリソースを用意し，そこに対するリンクを示す．

アプリケーション用の表現としては，XML のようなシステムによって取り扱いの容易な表現を用意する．例えば，利用履歴に関する統計情報を生成するためのクライアントプログラムが，XML 形式で生成されたリソースの一覧を取得出来るようにする．

4.2 音声認識エンジン

ファクトリリソースによって生成され，特定のクライアント専用の仮想音声認識エンジンを表す．このリソースは，このリソースにデータを入力するための入力デバイスリソース，このリソースからの出力を格納するための出力デバイスリソースに対するリンクを持つ．実装によってはその他の音声認識エンジンの構成要素を表すリソースが含まれる．例えば，語彙を表すリソースや，文法を表すリソースが含まれる．以下に URI の例を示す．

```
http://www.example.com/engine/2a383b/
http://www.example.com/engine/2a383b/input/
http://www.example.com/engine/2a383b/output/
http://www.example.com/engine/2a383b/grammar/
```

4.2.1 統一インタフェース

GET このリソースの状態と，関連するリソースへのリンクを取得する．状態としては，エンジンの種類やモデル，パラメータ，処理の状態・経過が得られる．また，PUT によって設定可能なパラメータの一覧が取得できる．

PUT リソースのパラメータを設定し，エンジンの処理方法等を変更する．このリソースが構成要素を表すリソースを含む場合は，その構成要素を表すリソースに対して PUT を行う必要がある．例えば，エンジンリソースが grammar リソースを含む場合，文法の変更は grammar リソースに対して行う．

DELETE このリソースを削除し，これ以上の認識処理を行わないようにする．入力データや出力データも含めた全てのデータが削除されるかどうかは，実装によって異なる．

4.2.2 表　　現

クライアントに提供する表現としては，基本的にはプログラムが利用するものとして，XML のようなシステムにとって取り扱いの容易な表現を用意する．例えば，利用履歴に関する統計情報を生成するためのクライアントプログラムが，XML 形式で生成されたリソースの一覧を取得出来るようにする．

4.3 入力デバイス

エンジンリソースは少なくとも 1 つの入力デバイスリソースと必ず関連付けられる．クライアントはこのリソースに対してデータを入力する．入力するデータの形式や，入力されたデータがどのタイミングでどのようにエンジンに渡されて，処理されるかは実装によって異なる．入力されたデータは従属リソースとして保存される．以下に URI の例を示す．

```
http://www.example.com/engine/2a383b/input/
http://www.example.com/engine/2a383b/input/0   （従属リソース）
http://www.example.com/engine/2a383b/input/1   （従属リソース）
```

4.3.1 統一インタフェース

GET このリソースにデータを入力する際のデータ形式と，既に入力された入力データリソースに対するリンクを取得する．入力データの形式は，このリソースが作成されるときに決定される．例えば，入力データとして生の音声データを受け付けるのか，MFCC 形式で特徴量が抽出されたものを受け付けるのか等，また，入力音声のサンプリング周波数や量子化ビット数等のパラメータもこのリソースから得られる．
実装によっては，従属リソースに対して GET が行われた場合，そのデータそのものが得られる，また，Accept ヘッダーやクエリ文字列によって指定された再生可能なフォーマットにデータを変換し，それが送信される．例えばクライアントは，このデータを参照することで入力されたデータを確認することが可能である．

POST このリソースに対してデータを入力する．すなわち，バイナリデータをそのまま送信可能であるようにする．データが入力されると，システムは従属リソースとして入力データリソースを作成する．一度の POST 操作で 1 つの入力データリソースが作成される．認識処理が開始されるタイミングは実装によって異なる．

4.3.2 表　　現

クライアントに提供する表現としては，クライアントの前段処理にて行う必要のある処理に関する情報を伝達する目的で，XML のようなシステムにとって取り扱いの容易な表現を用意する．

クライアントから受信する表現としては，バイナリデータの入力フォーマットとして application/octet-stream をサポートする．

4.4 出力デバイス

エンジンリソースは少なくとも 1 つの出力デバイスリソースと必ず関連付けられる．クライアントはこのリソースから認識結果を取得する．出力されるデータの形式や，結果が出力されるタイミングは実装によって異なる．認識結果を表す出力データは，従属リソースとしてシステムによって生成される．以下に URI の例を示す．

```
http://www.example.com/engine/2a383b/output
http://www.example.com/engine/2a383b/output/0    （従属リソース）
http://www.example.com/engine/2a383b/output/1    （従属リソース）
```

4.4.1 統一インタフェース

GET　このリソースから出力されるデータの形式と，出力されたデータリソースに対するリンクを取得する．出力データの形式は，このリソースが作成されるときに決定される．出力されたデータリソースに関しては，その出力のソースとなる入力データに対するリンクも併せ持つ．また，リクエストヘッダに If-Unmodified-Since を含めるか，もしくはクエリ文字列にて最終更新日時や最終更新出力番号を含めることで，新たに出力された認識結果のみを受信することが可能である．さらにタイムアウトを指定することで，コールバック形式でのデータ受信が可能となり，クライアントからサーバへのポーリングを減少させる[11]．従属リソースに対して GET が行われた場合，出力データそのものを得ることが出来る．

4.4.2 表　　現

クライアントに提供する表現としては，クライアントプログラムがが認識結果を扱うのが前提となるため，XML のようなシステムにとって取り扱いの容易な表現を用意する．

5. 実装例とその特徴

提案手法のアーキテクチャを用いた分散型音声認識システムの実装は，仮想音声認識エンジンの実装方法の違いによってによって特徴が大きく異なる．本節では，アーキテクチャの具体的な実装方法を例示し，その特徴について述べる．

5.1 仮想音声認識エンジンの実装例

サーバ側の実装としては，その構成が入出力部とデコーダ部に分けられる．入出力部は特定のクライアントからの入力データを受信するために，接続先のクライアントの数だけインスタンスが作られる．デコーダ部は，入出力部のインスタンス数に対するデコーダ部のインスタンスをどのように制御するかによって以下のように特徴が異なる．

接続先のクライアントの数と同数のデコーダのインスタンスを生成　それぞれ専任で認識処理を実行させる．その場合，クライアントからのエンジンの設定変更等の要求に応じるなど，柔軟性の高い認識サービスの運用が可能となる．反面，クライアント毎にデコーダをロードするため膨大なサーバリソースが必要となり，最大接続クライアント数はごく小さな数となる．

デコーダのインスタンスを一定数に限定　1 つのデコーダを複数のクライアントに対応させる．この実装ではサーバのリソースが固定されるために，安定したサーバ運用が可能となる．それに対し，クライアントがデータを入力してから実際に処理が行われるまでに時間差が発生する可能性が高くなる．また，クライアント毎にデコーダのインスタンスを変更することが出来ないため，デコーダの柔軟性は低下する．

例えば，プラグインによって拡張可能な音声認識システムを利用して，入出力部を実装する[12]．入出力部が HTTP のサービスを提供し，入力デバイスリソース，出力デバイスリソースとして振る舞うことで，下記のように想音声認識サービスの提供が可能となる．

- 特定の設定を持ったデコーダを複数インスタンス事前にロードする
- 入出力部に対応するインスタンスを，クライアントからの要求に応じて作成する
- 入出力部のインスタンスは，クライアントからの入力データをローカルディスク上に保存する
- 入出力部のインスタンスは，入力データを受信したタイミングで処理待ちのキューに自身を登録する
- デコーダは，キューに登録された入力データを順に処理し，入出力部のインスタンスに対して出力を書き込む

この方式では，入出力データが一度サーバのディスク上に書き込まれるため，処理の効率は若干低下するが，安定したサーバ運用が可能となる．

5.2 エンジンファクトリの実装例

クライアントからの要求を最初に受け付けるエンジンファクトリの実装によって，スケーラビリティやシステムの柔軟性が変化する．

仮想音声認識エンジンと同一プロセスでファクトリを提供　接続先のクライアントの数と同数のデコーダのインスタンスを生成するモデルの場合，クライアントの数が増大すると新規インスタンスの生成を拒否するか，生成を待機させることでシステムを安定稼働させることが可能となる．単一のサーバで稼働が可能であり，シンプルな構成となるが，スケーラビリティに欠ける．

仮想音声認識エンジンと別のプロセスでファクトリを提供　同一のデコーダをロードするプロセスを複数用意し，それぞれに対してクライアントを振り分ける．プロセス毎にロードされているデコーダの種類や構成を変化させることで，クライアントからの要求に応じたデコーダを割り当てることが可能である．また，登録ユーザや課金ユーザを，優先ユーザ用プロセスに割り当てること等も可能である．複数ホストでのサーバシステムの構成が可能であり，システムが複雑化するものの，スケーラビリティが高く，振り分け先の種類に応じた柔軟性を持つ．

例えば，以下のような仮想音声認識サービス用プロセスを用意し，クライアントからの要求を振り分ける．

- 大人男性用の音響モデルを備えたデコーダをロードしたプロセス
- 大人女性用の音響モデルを備えたデコーダをロードしたプロセス
- 子供用の音響モデルを備えたデコーダをロードしたプロセス
- 汎用の音響モデルを備えたデコーダをロードしたプロセス

仮想音声認識エンジンリソースの生成要求に個人属性（性別・年齢）が含まれていたときは，属性に応じたプロセスに対してファクトリは作成要求を出す．また，特定のプロセスが混雑しているときは，汎用のプロセスで代用を行う．

5.3 クライアント側の実装例

提案手法では，クライアント・サーバ間の通信方式が REST 方式で統一されており，サーバ側の実装によらず統一インタフェースを用いた連携が可能である．そのために，クライアント側の実装としては様々なモデルで実現が可能である．

例えば，以下の機能を備えたデーモンプロセスをクライアント側に常駐させる．

- サーバ側の入力デバイスリソースから，送信するデータのフォーマットを取得する
- サーバ側の入力デバイスリソースに対して音声データ，もしくは特徴量抽出の結果を送信する
- ローカルホストで HTTP サービスを起動し，ローカルのアプリケーションから送信・停止の要求を受ける

ローカルホストで動作するウェブアプリケーションは，このプロセスと HTTP で通信することにより，バイナリデータの送信をサーバに対して送信可能となる[13)]．この枠組みを用いて，ウェブベースのアプリケーションへの音声インタフェースの付与が可能となる．

謝辞 本研究は、独立行政法人情報通信研究機構（NICT）の委託研究『分散型音声認識アプリケーション開発プラットフォームのための基盤技術の研究開発』によるものです．

6. むすび

複数のウェブベースのサービスの連携として構築されるウェブベースの音声認識アプリケーションの開発に利用可能な，分散型音声認識システムのためのアーキテクチャについて提案と考察を行った．具体的には，既存のウェブサービスのおけるマッシュアップのような，連携による相乗効果を生み出す音声認識アプリケーションの開発環境を実現するためには，シンプルなプロトコルでの連携を可能にするオープンな分散型音声認識サービスの構築が必須であるとして，以下に示す特徴を持つ RESTful 分散型音声認識サービスを設計した．

- REST (Representational State Transfer) のアーキテクチャスタイルに従い，標準的なウェブ技術 (HTTP, URI, XML) を用いた連携が可能
- 各々のクライアントの一連の処理要求毎に専用リソースを提供し，音声入力と結果の取得，及び入出力と実際の認識処理の分離が可能
- リソースとして，エンジンファクトリ，仮想音声認識エンジン，入力デバイス，出力デバイスを定義

また，各々のリソースについて具体的な設計を行い，統一インタフェースと操作・振る舞いとの割り当てを行った．さらにそれらの実装例を検討し，それぞれの特徴について述べた．

今後の予定としては，ウェブ上に分散された様々なサービスの連携システムとして音声認識アプリケーションを構築するための枠組みとして，複数のサービスを容易に “つなぐ” 為の枠組みの検討と，それぞれの実装を行う．

参 考 文 献

1) Goddeau, D., Goldenthal, W. and Weikart, C.: Deploying speech applications over the Web, *Fifth European Conference on Speech Communication and Technology*, ISCA (1997).
2) Digalakis, V., Neumeyer, L. and Perakakis, M.: Quantization of cepstral parameters for speech recognition over the world wide web, *IEEE Journal on Selected Areas in Communications*, Vol.17, No.1, pp.82–90 (1999).
3) Schmitt, A., Zaykovskiy, D. and Minker, W.: Speech recognition for mobile devices, *International Journal of Speech Technology*, Vol.11, No.2, pp.63–72 (2008).
4) Zhang, W. Q., He, L., Chow, Y. L., Yang, R. and Su, Y.: The study on distributed speech recognition system, *IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS SPEECH AND SIGNAL PROCESSING*, Vol.3 (2000).
5) OReilly, T.: What is Web 2.0: Design patterns and business models for the next generation of software.
6) Jhingran, A.: Enterprise information mashups: integrating information, simply, *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, pp.3–4 (2006).
7) Bayer, S.: Embedding speech in web interfaces, *Fourth International Conference on Spoken Language Processing*, Citeseer (1996).
8) Nisimura, R., Miyake, J., Kawahara, H. and Irino, T.: Development of Speech Input Method for Interactive VoiceWeb Systems, *Proceedings of the 13th International Conference on Human-Computer Interaction. Part II: Novel Interaction Methods and Techniques*, Springer, p.719 (2009).
9) Richardson, L. and Ruby, S.: RESTful web services (2007).
10) Fielding, R.: Architectural styles and the design of network-based software architectures, PhD Thesis, Citeseer (2000).
11) Russell., A.: Comet: Low Latency Data For Browsers. (2006).
12) Nakano, T., Fujie, S. and Kobayashi, T.: Extensible speech recognition system using proxy-agent, *ASRU2007*, pp.601–606 (2007).
13) 中野鐵兵，藤江真也，小林哲則：Proxy-Agent を用いた音声認識対応ウェブアプリケーション開発フレームワークの提案と実装 (音声基盤技術・インタフェース)，情報処理学会研究報告. SLP, 音声言語情報処理， Vol.2008, No.12, pp.83–88 (20080208).