社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＨＩ－122 (5)
2007－SLP－ 65 (5)
2007／2／9

プロキシエージェントアーキテクチャによる音声認識アプリケーション用ユーザモニタリング機能の効率化

中野 鐵兵　梅本 暁　藤江 真也　小川 哲司　小林 哲則
早稲田大学 理工学部

あらまし：
音声認識アプリケーションにおける利用者の実際の振る舞いのモニタリング機能を効率的かつ効果的に実現する技術としてプロキシエージェントアーキテクチャを提案する．プロキシエージェントとは，アプリケーションと音声認識エンジンの間に入って，その連携を担当するソフトウェアであり，アプリケーションから音声認識エンジンに対する制御信号と音声認識エンジンの入出力に関する情報の収集を行う．プロキシエージェントは，その構造上の特性を活かし，システムに対する様々な付加サービスの提供を可能にする．本稿では，その中でもシステムのモニタリング機能に主眼を置いたプロトタイプシステムの開発について述べる．プロトタイプでは，モニタリングを可能にするためのアーキテクチャ，連携技術，及びデータに対するアノーテーション技術について実装を行い，実際にデータ入力用アプリケーションの開発に利用することで，その有効性を確認した．

Efficient Monitoring Capabilities of User Behavior for Speech Recognition Applications Based on Proxy-Agent Architecture

Teppei NAKANO, Akira UMEMOTO, Shinya FUJIE, Tetsuji OGAWA, Tetsunori KOBAYASHI
School of Science and Engineering, Waseda University

Abstract:
As a fundamental technology for speech recognition applications to realize a positive feedback loop between developers and users, we propose a new speech recognition application construction architecture; Proxy-Agent Architecture. Proxy-Agent is a software component located in between applications and speech recognition engines that takes charge of their collaborations. Especially in this paper, we focus on its monitoring capabilities of user behavior on speech recognition applications, and describe its prototype system development and experiment.

1 はじめに

音声認識システムの抱える問題点として，エンジンの性能・特性とアプリケーション設計とのミスマッチ，あるいは開発サイドの期待した使い方と利用者の望む使い方のミスマッチが報告されている [1]．このような問題を解決するためには，音声認識エンジンの性能向上に直接寄与する技術の研究だけではなく，アプリケーションに組み込まれた音声認識機能が，実際の利用環境で開発者が意図した通りに機能しているのかを分析する技術の研究，また，機能していないのならばそれを機能させるための技術開発が必要となる．そこで本研究では，音声認識システム用プロキシエージェントアーキテクチャなる音声認識システム構築に関する新しい枠組みを提案し，アプリケーション利用者の実際の振る舞いに関するモニタリング技術とフィードバック技術の開発を行う．

従来の一般的な音声認識アプリケーションでは，音声認識エンジンは単体のソフトウェアコンポーネントとみなされ，アプリケーションから直接参照される．例えば音声認識機能を搭載したオフィスアプリケーションやカーナビゲーションシステムは，音声認識用 API(Windows SAPI[2] や Java SAPI[3]) やプロプライエタリな API を用いて直接エンジンを操作する．しかしながら，これらのように音声認識エンジンを直接参照する方式では，モニタリング機能やフィードバック情報に基づいたモデルの自動更新機能を埋め込むために，アプリケーションに多くの機能追加が求められる．もしくはそのような機能を保持した特別なエンジンの利用が必須となる．

また，Open Agent Architecture[4] や，DARPA

Communicator Architecture[5] のような，従来から音声対話システム構築に用いられるアーキテクチャでは，音声認識エンジンはそのアーキテクチャの構成要素 (エージェント [4] やサーバ [5]) により包含される．この場合，上記の機能をその構成要素に埋め込むことは技術的には可能だが，その埋め込み方法は定義されておらず，構成要素の役割も不明確となる．また，これらは複数の対話機能モジュールを分散環境で統合し対話システムを構築するというシステム統合用アーキテクチャであり，デスクトップPCやPDA端末，車載情報機器等のアプリケーションでの適用可能性には疑問が残る．

そこで本研究では，音声認識アプリケーションにおける利用者の実際の振る舞いのモニタリング機能を効率的かつ効果的に実現する技術としてプロキシエージェントアーキテクチャを提案し，開発する．

本稿では，まず本研究で実現を目指す開発パラダイムについて述べ，その実現に有効なアーキテクチャとして提案アーキテクチャの概要を説明する．次に提案アーキテクチャの実装として開発したプロキシエージェントシステムについて述べ，そこで開発したモニタリング機能と，収集されたデータの分析を容易にするアノーテーション技術について述べる．最後に，実験用のアプリケーション開発とその実験について述べ，今後の課題についてまとめる．

2 双方向性型の開発パラダイム

本研究で解決を目指す開発サイドと利用者のミスマッチは以下のようにまとめられる [1].

1. 開発サイドが想定した限定された語彙とユーザの実際の指示語彙に差異がある
2. ユーザはマニュアルは読まず，装置に通じる語彙がわからない
3. 語彙を増やすと類似語も増加し，認識間違いも増加する

これは，開発初期に実験やテストプロセスを経て語彙を作成し，それを利用し続けるという現在の手法では，実際の利用者の使用語彙をカバーすることが困難であり，カバー率を向上させるために単純に語彙を増やすことも，認識率の低下の問題から有効な解決策とならないことを意味している．つまり，実際に利用される語彙が開発サイドで予測可能であるという仮説は成立しておらず，この仮説に基づいた現在の手法を抜本的に見直さなくてはならない．そこで本研究では，以下の2つのアプローチを採ることで問題の解決を図る．

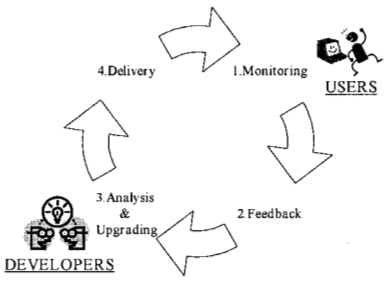

図 1: a positive feedback loop

フィードバックの実現 実際にユーザによって発話された語彙をアプリケーション毎に収集・分析し，UIや利用可能語彙を改善する

継続的な改善 フィードバック情報を分析し，実際に発話された語彙や認識誤りの多い語彙を抽出し，語彙やモデル・UIを自動更新する

このように，開発者が利用者に提供している語彙には，実際に利用される語彙の一部しか含められていないことを前提とし，それを運用中に逐次更新していく枠組みを導入することで問題の解決を図る．つまり，1. 利用者の実際の振る舞いをモニタリングし，2. その情報を開発者側にフィードバックし，3. それを分析して語彙やモデルを更新し，4. 更新情報を利用者側へ配布するという枠組みを導入する (図1)．このような，エンジン開発者／アプリケーション開発者／利用者の間の有機的な連携を可能にする枠組みを双方向性型の開発パラダイムと呼び，音声認識システムのようなパターン認識システムにおける精度の向上に大きく寄与するものと考える．本研究ではこの実現を最終的な目標とする．

2.1 アーキテクチャに対する要求

双方向性型の開発パラダイムを実現するために，以下に示す機能の提供を可能にする枠組みを検討する．

- 利用者の入力音声や振る舞いを含む，音声アプリケーションのモニタリング機能
- モニタリングした情報をネットワーク経由でサーバに送信するフィードバック機能
- フィードバックデータの分析に基づく，音声入力や振る舞いに関する統計データの提供機能
- 音声認識エンジンのモデルデータや機能モジュール，利用者に対する入力音声改善指示等の配信機能
- 利用者の識別情報を利用した，音声認識システムのパーソナライズ機能

上述の機能を，特定のエンジン，特定のアプリケーションに組み込むことは比較的容易である．しかしながら，このような個別対応では，フィードバック情報の集積がその機能を組み込んだ特定のエンジン，アプリケーションに限られてしまい，そのメリットを共有できるユーザが限られ，効果の広がりに欠ける．ここでは情報の収集と知見の共有とをより大規模に行うことを考え，多くのエンジン，アプリケーションで組み込み可能な形でモニタリング技術を提供するインフラストラクチャとして目的の機能を実現するアーキテクチャを設計する．また，上記機能の実現と，実際にその用途の広がりを可能にするために必要なプラットフォームとしての前提条件を以下にまとめる．

容易な導入 既存のアプリケーションに対する付加機能としての音声認識機能の導入に利用可能であり，必ずしも本格的な音声対話システムを対象としない

エンジンの自由選択 アプリケーション開発者が望む音声認識エンジンを利用

軽量・マルチプラットフォーム 車載情報機器やPDA端末を含む，小型デバイス上への導入

オフライン動作 ネットワークへの常時接続を必ずしも必要としない

自動更新機能 配信された情報を利用した自動更新機能をサポート

3 プロキシエージェントアーキテクチャ概要

本稿で提案するプロキシエージェントアーキテクチャについて説明し，前節の議論であげられた機能と前提条件を如何に満たすかを述べる．

まずプロキシエージェントアーキテクチャは，以下の4つの要素から構成される (図2).

- アプリケーション
- プロキシエージェント
- エンジンアダプタ
- デバイスアダプタ

プロキシエージェントとは，アプリケーションと音声認識エンジンの間に入ってその連携を担当するソフトウェアであり，アプリケーションから音声認識エンジンに対する制御信号と音声認識エンジンの入出力に関する情報の収集を行う．エンジンアダプタとは，1つ以上のプラグイン群から構成される仮想音声認識エンジンオブジェクトを表し，音声認識機能の実装が含まれる．プラグインは，音響モデルや発話区間検出機能モジュール等の音声認識エンジンの構成要素から，音声認識エンジンそのもの，またはアプリケーション開発者が用意したアプリケーション特有の音声認識エンジンの操作プラグインまで自由な粒度で構成される．認識対象となる入力データはデバイスアダプタから取得する．デバイスアダプタとは実際の入力デバイスからのデータ取得ロジックを包含したデータ提供オブジェクトであり，プロキシエージェントはデバイスからエンジンへのデータの流れを中継することで，実際に対象となるデータを収集する．デバイスアダプタもプラグインとして構成される．アプリケーションはプロキシエージェントとメッセージの送受信を行い，エンジンアダプタの機能を呼び出す.

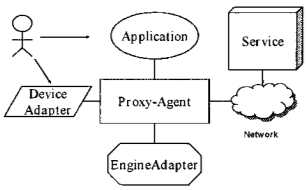

図 2: Proxy-Agent Overview

プロキシエージェントは，その名前の通り“プロキシ”として振舞う．つまり，プロキシエージェントがエンジンアダプタやデバイスアダプタのインタフェースを包み隠すのではなく，クライアントはそれらのインタフェースをそのまま呼び出せるようにする．また，エージェント自身のインタフェースを追加して公開することで，アプリケーションやエンジンアダプタから付加機能の呼び出しも可能にする．さらにプロキシエージェントはネットワーク経由で外部のサービスとの連携を可能にし，アプリケーション・音声認識エンジンのいずれにも属しにくい機能をプラグインとして実装可能にする．

プロキシエージェントアーキテクチャは，その構造上の特徴を活かし，前節で議論した機能の実現を容易にする．

まず，アプリケーション，エンジン，デバイスのデータの流れを中継するために，モニタリング機能は容易に実現可能となる．また，プロキシエージェントが追加APIを公開することで，アプリケーションの状態や認識対象の特徴，前後のユーザの操作等の，付加情報が付与されたモニタリングデータの収集が可能となり，収集データの分類・分析が容易になる．付加情報を利用したデータの分類方法に関し

ては5節で説明する．

さらに，外部のサービスと連携することによりフィードバック機能及びサーバからの配信情報の取得機能が実現される．フィードバック情報が一箇所に蓄積されることにより，多数のユーザを対象とした振る舞いの分析と統計データの算出が可能となる．最終的には，実使用環境に適した語彙や類義語の作成，モデルの構築，パーソナライズが可能となり，双方向性型の開発パラダイムの実現が見込まれる．

4 アーキテクチャの実装

アーキテクチャを実装するためのプラットフォームとしてEclipse RCP(Rich Client Platform)[6]を採用し，設計・開発を行った．

Eclipseは，巨大なオープンソースコミュニティによって支えられたJavaベースのオープンソース統合開発環境である．非常に拡張性の高いプラグインアーキテクチャ(Eclipse platform)によって構築されおり，開発環境本体を含め，数多くの機能がプラグインとして開発されている．ここで，Eclipse platformを稼動させるために必要な最小限のプラグインを集めた実行環境がRCPである．RCPで提供されたプラグインに独自のプラグインを含めることで，統合開発環境とは切り離された独自のアプリケーション(Rich Client Application)としての稼動が可能になる．

また，現在のバージョンのEclipseはOSGi(Open Service Gateway Initiative)[7]のプラットフォーム上に構築されており，ネットワーク経由でのプラグインの更新をサポートしている．OSGiは，情報家電や車載情報端末等のアプライアンス上で動作するソフトウェア部品をネットワーク経由で配布・管理するための標準的なプラットフォームの1つであり，Eclipseのプラグインはこの OSGiの配布単位であるバンドルとして実装される．

本研究ではプロキシエージェントを Eclipse プラグインとして開発し，RCPとしての実行，及びEclipse/OSGiの枠組みを用いた更新の枠組みの利用を可能にした．つまり，前節で前提条件としてあげた自動更新機能を，OSGiが提唱する標準的な枠組みに従った形式でサポートする．また，Javaベースの Eclipseを採用することによりマルチプラットフォーム性もサポートされる．軽量プラットフォーム上の動作に関しても，コミュニティでは組み込みデバイス用RCPの開発も進んでおり，今後の対応が見込まれる．

```
1 public class DemoFactory implements EngineAdapterFactory {
2
3     public EngineAdapter loadAdapter(
4         EngineContext context) throws Exception {
5         Sphinx4Engine engine =
6             new Sphinx4Engine(
7                 context.getClass().getResource("config.xml"));
8         EngineAdapter adapter =
9             new EngineAdapter(engine);
10        adapter.registerExtension(
11            new JSGFExtension(engine, "jsgfGrammar"));
12        return adapter;
13    }
14
15 }
```

図 3: An implementation of EngineAdapterFactory: (l.5–7: loading new engine instance, l.8–9: loading new adapter, l.10–11: registering the extension), both teh engine instance and the extension are included in other plugin.

4.1 エンジンアダプタの実装

アプリケーション開発者は開発するアプリケーション用に1つのエンジンアダプタプラグインを用意する．エンジンアダプタプラグインはEclipseプラグインとして実装され，プロキシエージェントによって定義されたエンジンアダプタ用の拡張ポイントへ登録される．拡張ポイントでは，エンジンアダプタを生成するインタフェース (EngineAdapterFactory)が公開されており，プラグインはその実装クラスを提供する (図3)．このプラグインは，音声認識エンジンを含む複数のプラグインから構成され，アプリケーションが利用する語彙や文法等が含まれる．

また，利用したい音声認識エンジンのプラグインが存在しない場合は，アプリケーション開発者はそのプラグインを開発する．プラグインは，Engineインタフェースの実装クラスとして開発でき，最低以下の処理が記述できれば自由にアーキテクチャに組み込むことが可能である．

- インスタンスの生成とメモリ確保
- エンジンのネイティブメソッドの Javaからの呼び出し
- 入力音声をバイト列を受け，認識結果を返す
- 終了後のメモリ開放

4.2 Proxy-Agent API

アプリケーションが利用するプロキシエージェントの APIは Proxy-Agent Application Programmer Interface(PA-API)として定義されている．アプリケーションは，このインタフェースを用いてプロキシエージェントと通信を行う．

アプリケーションとプロキシエージェントとの連携は，ローカルホスト上の TCP/IP接続，もしくは同一の Eclipse RCP上で Java APIの呼び出しによる，独自開発のバイナリ形式XMLの非同期メッ

セージ交換により実現する．アプリケーションからProxy-Agentに対する要求は基本的に音声認識エンジンに対する処理要求であり，その要求に対する処理は比較的時間がかかることが多いという特徴がある．そのために，アプリケーションとプロキシエージェントの連携を非同期で行えるように定義した．また，アプリケーションとプロキシエージェントの連携用に拡張性の高いXML形式のメッセージ交換プロトコルを定義することで，エージェントやエンジンに出す具体的な処理要求の内容を自由に拡張出来るようになっている．

5 モニタリング機能の実装

5.1 データ収集プロセス

プロキシエージェントに，認識エンジンに対する入出力を自動的に収集する枠組みを実装した．収集されるデータとしては，実際にエンジンが認識処理に利用した入力音声データの全てが保存される．具体的には，デバイスアダプタから読み取られたデータを自動的に保存用ディレクトリ (リポジトリ) に保存する．読み取られるデータの終端は，アプリケーションからの認識停止メッセージを受信するか，認識エンジンによって発話区間の終端が検出され，エンジンの認識処理が停止するまでとなる．また，エンジンからの出力である認識結果もプロキシエージェントが受け取り，それをアプリケーションに対して送信すると同時に，そのデータもリポジトリに保存し，それまでの入力データとの関連付けを行う．データ収集機能では，入力音声 (PCMとWAV形式)，認識結果 (バイナリ形式)，ログ情報 (XML形式) が保存可能である．

ログ情報とは，アプリケーションからの制御メッセージや，データに対するアノーテーションに関する情報であり，プロキシエージェントによってXML形式で保存される．アノーテーションは，データに対するメタ情報としてアプリケーションやエンジンアダプタ，プロキシエージェントから付与される．アノーテーションはその情報が有効となるスコープを持ち，そのスコープ内で保存された全ての入力音声に対して情報が付与される．スコープの種類とその範囲，アノーテーションが実際にデータに付与される様子を図4に示す．

5.2 アノーテーションを利用したデータ分類機能

プロキシエージェントにより収集されたアノーテーション付きデータを，そのアノーテーション及び認識結果を用いて自由な階層構造で出力する機能をプロキシエージェントに対するプラグインとして実装した．プラグインは，プロキシエージェントのコンテキストメニュー (ウィンドウズのタスクバーに表示されているプロキシエージェントのアイコンを右クリックすると表示される) に出力用のメニューを追加し，そこが選択されたら設定画面を表示する．設定画面では，出力するデータが収集された期間と，分類に利用するタグ名のリスト，及び出力先のディレクトリを指定する．タグ名のリストはデータの分類先のディレクトリ構造を表し，データはその末端のディレクトリに格納される．プロキシエージェントは出力要求を受けると，ログファイルをパースしながらタグの値にしたがってディレクトリを作成し，末端のディレクトリにデータを出力する．例えば，タグ名のリストが{A→y→x}であり (図5 (左))，あるデータのタグの値がそれぞれ{A=△, y=●, x=○}である場合，そのデータは{./△/●/○}以下のディレクトリに出力される (図5 (右)),.

この枠組みを用いることで，アプリケーション設計時にはデータに対してフラットなタグ付けを行いながら，分析時には必用なデータを分析しやすい階層構造で出力し，効率的なデータの利用が可能となる．

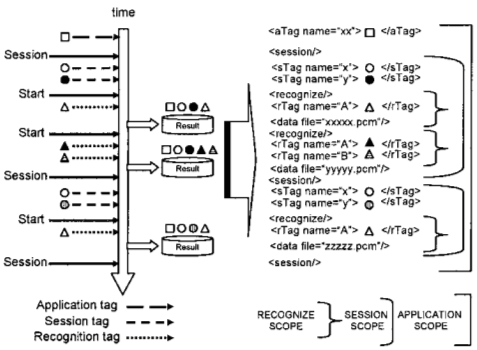

図 4: Annotation

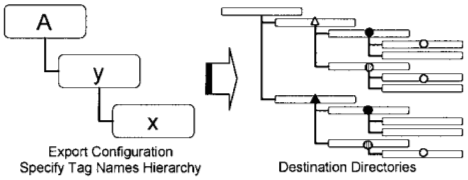

図 5: Export

5.3 データ収集実験システムの開発

音声による旅行情報のデータ入力アプリケーションを実験用アプリケーションとして開発した．プロ

キシエージェントとは別の独立したプロセスで動作し，TCP/IPの接続を用いて連携する．このアプリケーションでは，ユーザインタフェースの相違による音声入力の特徴の変化を調べるための，音声認識機能を用いたデータ入力を行う．比較対象のインタフェースは4種類でそれぞれ10の入力項目を持つ．入力項目毎に1つのネットワーク文法が割り当てられ，その項目の入力には割り当てられた文法を用いた音声認識処理が実行される．実験時には被験者毎の個人情報として，年齢と性別の入力を求める．

音声認識エンジンはSphinx-4[8]を用いた．アプリケーション用のプラグインとしては，Sphinx-4のプラグインとネットワーク文法をロードする拡張機能を用いた．また，文法の定義は全てプラグインにまとめ，アプリケーションからはその名前のみを指定した．

アノーテーションの情報としては以下の情報を付与するようにした．

- serialNumber 被験者の番号
- age 被験者の年齢
- gender 被験者の性別
- interface インタフェースの識別子
- category 入力項目
- grammar 音声認識に利用した文法

5.4 データ収集実験と結果の分析作業

ランチタイムの大学内のカフェテリアにて，昼食後の学生や職員のボランティアを被験者として実験を行った．実験では合計78名の被験者から，合計908回の音声入力データが得られた．収集された入力音声は，視聴によるフィラーの発生頻度と発話様式の変化の分析に用いられた．

分析用データは，インタフェースとカテゴリに付与されたタグで分類を行い，実際に視聴することでその比較を行った．分類のために新たなプログラムの作成等は必要なく，手軽にデータが分類された．また，アノーテーションさえ付与されていれば，設計時に予定しなかった組み合わせでもデータの比較を行えることも確認出来た．

実験を通して，アプリケーション利用者の入力音声や振る舞いの分析に必用な情報が，アプリケーション開発者の適切なタグ情報により，自動的かつ効率的に収集可能であることが示された．また，データの分析時には，タグ情報に従ったデータの出力機能を用いることができ，作業の効率化に有効であることを確認した．

6 むすび

本稿では音声認識アプリケーションにおける利用者の実際の振る舞いをモニタリングする技術としてプロキシエージェントアーキテクチャを提案し，Eclipse RCP上にそのシステムを構築した．プロトタイプでは，モニタリングを可能にするためのアーキテクチャ，連携技術，及びデータに対するアノーテーション技術について実装を行った．また，データ入力用アプリケーションを開発し，プロトタイプシステムのモニタリング機能が，データ収集と分析の観点から有効に動作することを確認した．

今後の予定は，収集したデータをサーバに対してフィードバックする機能を実装し，収集データの一元管理と，サーバ上でのデータ分析機能を実現する．また，サーバ上にウェブインタフェースによる分析・エクスポートサービスを実装し，アプリケーション開発者が，必要なデータをいつでも取得できる環境を提供する予定である．

参考文献

[1] 古井 貞熙, 他, “音声認識技術実用化に向けた先導研究,” NEDO 平成17年度成果報告書, 100007350, 2006.

[2] http://research.microsoft.com/research/srg/sapi.aspx

[3] http://java.sun.com/products/java-media/speech/index.jsp

[4] Adam Cheyer, David Martin, “The Open Agent Architecture,” Journal of Autonomous Agents and Multi-Agent Systems, vol. 4 , no. 1, pp. 143-148, March 2001.

[5] Alan Goldschen, Dan Loehr, “The Role of the DARPA Communicator Architecture as a Human Computer Interface for Distributed Simulations,” 1999 Simulation Interoperability Standards Organization (SISO) Spring Simulation Interoperability Workshop (SIW), Orlando, Florida, March, 1999.

[6] Rich Client Platform (RCP), Eclipse, http://www.eclipse.org/rcp/

[7] The OSGi Alliance, http://www.osgi.org/

[8] “Design of the CMU Sphinx-4 decoder,” Paul Lamere, et al., EUROSPEECH 2003, September 1-4, 2003.