Vol.2013-HCI-151 No.4
Vol.2013-SLP-95 No.4
2013/2/1

短文を投稿するための音声対話システム

竹内 誉羽[1,a)]　藤堂 祐樹[2]　中野 幹生[1,b)]

概要：ツイッターなどのマイクロブログに，歩行中や運転中にテキストを投稿するためにはアイズフリー・ハンズフリーであることが必須である．このためにユーザが音声対話システムと対話することによってテキスト入力する方法を考案した．これにより，ユーザは定型的なテキストと，自由に編集できるテキストの両方を入力することができる．この方法では，複数の定型的なテキストの入力，ならびに自由なテキストの入力，さらに投稿のための対話制御といった複数のドメインの対話をシームレスに行うことができる．この方法を，スマートフォン向けの音声対話アプリケーションとして実装した．このアプリケーションは，我々が構築したマルチドメイン対話システム開発のためのツールキット HRIME を用いて実装した．

1. はじめに

ツイッター[*1], や Facebook[*2]など, 多くの Social Network Service が日常的に使われるようになってきている．また最近，災害時のこうしたサービスの有用性も指摘されるようになった．これらのサービスは，全てテキストベースでありユーザが何らかの方法でテキストを作成することによって利用できる．そのため原則的には歩行中や車を運転中には利用できないサービスである．もし運転中にテキストを作成しようとして，編集画面を凝視したりすれば，多くの国で法令違反であるばかりか危険である．また歩行中に携帯電話などの端末を操作することもやはり危険である．しかし歩行中や運転中のユーザからの情報は他の歩行中，運転中のユーザにとって有効な情報が多いはずである．したがって歩行中や運転中でもテキストベースの情報発信を可能にする手段を開発し提供することにはニーズがあると考える．その際，アイズフリー・ハンズフリーであることは必須条件である．

最近のスマートフォンでは, Apple 社による iOS の Siri[1] をはじめとして音声で SMS(Short Message Service) などのテキストメッセージを送る手段がある．しかし実際の入力では，テキストを音声で入力した後に，そのテキストを画面で確認して，必要なら手で操作して修正する必要がある．これではアイズフリー・ハンズフリーという条件を満たすことは困難である．

あるいは運転中などに特定の場所を通過すると，自動でその場所にいるという情報を SNS に投稿するシステムがある．また特定の場所を通過すると「投稿するか否か」を聞いてきてユーザが「はい」と言えば投稿し，「いいえ」と言えば投稿をしないという方法で決めらたテキストを投稿するアプリがある[*3]．いずれにしてもユーザの望むタイミングで投稿することは難しく，またあらかじめ決まったテキスト以外は投稿できない．

自動車向けのテキスト入力装置として，音声入力とダイヤル型の入力デバイスとを組み合わせたものがある[*4]．音声によるフィードバックとダイヤル操作で誤認識した部分を特定して，音声で訂正することができる．ダイヤル式の入力デバイスは必ずしも凝視しなくても操作できるため，ハンドルに取り付けるなどすれば各国・地域の法令に準拠できる可能性が高いが，やはりハンズフリーの方が望ましい．

以上の具体的なアプリケーション以外の関連研究としては，目の不自由な人向けの入力デバイスの研究などが多く（例えば [2]），本論文の目的とは異なっており，タッチディスプレイ上でのジェスチャー等を使う．そういった研究以外では，SMS の音声テキスト入力において，コンテキストを利用して音声認識率の向上などにつなげる研究がある [3]．これは SMS のように相手との関係でコンテキストが固定されるような場合に有効な手段になると思われるが，本報告では必ずしも双方向のメディアだけが対象ではないため，方向性が異なる．

1 (株) ホンダ・リサーチ・インスティチュート・ジャパン
埼玉県和光市本町 8–1
2 豊橋技術科学大学情報工学系
愛知県豊橋市天伯町雲雀ヶ丘 1–1
a) johane.takeuchi@jp.honda-ri.com
b) nakano@jp.honda-ri.com

*1 Twitter Home page: http://twitter.com/
*2 Facebook Home Page: http://www.facebook.com/
*3 iPhone app: Aqua Social Drive: http://aquadrive.jp/
*4 http://www.nuance.com/products/dragon-drive/index.htm

ⓒ 2013 Information Processing Society of Japan

User:「雨だ」
System:「天候，雨を記録しますか？」
User:「はい」
System:「天候，雨を記録します．」

図 1 天候に関する定型的なテキストを入力する際の対話例
Fig. 1 An example dialogue for posting a boilerplate text about weather.

User:「今から10分ごとに，場所を記録して」
System:「10分ごとに，記録しますか？」
User:「よろしく」

図 2 場所に関する定型的なテキストを入力する際の対話例
Fig. 2 An example dialogue for posting a boilerplate text about locations.

我々はマルチ・ドメインな音声対話システムによってアイズフリーかつハンズフリーなテキスト入力を実現することを目標として，対話戦略や試作アプリを構築した．本報告では，まず定型的なテキスト（決められたテキストではない）の入力と自由テキストの入力を可能にする音声対話戦略を提案する．これによりユーザの望むタイミングで定型的なテキストを投稿でき，時には自由なテキストを音声対話によって入力し投稿することができる．また以上の対話戦略に基づいて，スマートフォン向けの音声対話アプリケーションの試作を行った．このアプリケーションは，HRIME（HRi Intelligence platform based on Multilple Experts）と呼ばれるマルチ・ドメイン音声対話システムツールキットを基盤にして実装されており，音声対話の状況に合わせて複数種類の定型的な文章の入力と，音声コマンドによる自由な文章を入力するモードへの移行がシームレスにできるようになっている．

2. テキスト入力のための音声対話戦略

本節では，試作したアプリで実現されている音声対話によるテキスト作成について概説する．

2.1 定型テキストの入力

ツイッターなどのマイクロブログでは，投稿できるテキストの文字数に制限があって，定型的なテキストを投稿するだけで目的にかなうことが多い．例えば ForSquare などいくつかのツイッターを使用するサービスでは，位置情報を付けて「I'm at 和光市駅.」などという定型的なテキストを投稿できる．また運転中，道路工事をしているとか，渋滞しているとかということを信号待ちなどの停止中に対応するボタンを押すことで，定型的なテキストを投稿できるアプリがある[*5]．また先に述べたように特定の地点を通り過ぎるごとに，そのことを示す定型的なテキストを生成して投稿するアプリがある．このようにマイクロブログでは定型的なテキストを投稿することは既に当たり前に行われている．問題はタッチパネルのボタンを押したりせずにユーザの望むタイミングでユーザの望む定型的なテキストを投稿できる仕組みがないことである．

[*5] Waze, http://www.waze.com/

User:「文章の入力」
System:「文章の入力モードに移ります．このモードでは，入力したい文章を短かく区切りながら言ってみてください．」
User:「和光市駅のあたり」
System:「和光市駅のあたり」
User:「結構人がいる」
System:「結婚式」
User:「訂正」
System:「もう一度お願いします」
User:「人が多い」
System:「人が多い」
User:「保存」
System:「テキスト，『和光市駅あたり，人が多い』を保存します．文章入力モードを終わります．」

図 3 自由テキストを入力する際の対話例
Fig. 3 An example dialogue for creating a free formatted text.

そこでシステムと対話をすると，必要に応じてアプリが対話内容に応じた定型的なテキストを生成する音声対話戦略を考案した．例えば**図 1** ように，ユーザ側からスマートフォンに話しかけることで，その地点の天候について，投稿することができる．以上の対話の結果，「天候：雨 2012 年 5 月 6 日 14:30」などといったテキストが作成され，この対話がなされた場所の位置情報も属性としてつけることができる．

その他道路上の工事や，交通事情などについて投稿することができる．また，対話によって例えば 10 分ごとに自分のいる位置を投稿するようにセットすることもできる．例えば**図 2** のような対話になる．この結果，10 分ごと「今ここにいます．」といったようなテキストに位置情報の属性がついたデータが生成される．

以上によってユーザが意図したタイミングで画面を見たり操作したりすることなく，定型的なテキストを生成してツイッターをはじめとしたマイクロブログに投稿することができる．

2.2 自由テキストの入力

定型的なテキストだけでなく，自由なテキストを入力できることも時には必要である．そのために，音声コマンドを用いてテキストを入力できるようにした．これを行うには，まず「文章の入力」あるいは「テキストの入力をお願い」などの言葉で明確に指示をする必要がある．そうすると，音声でのテキスト入力モードに移行する．

ⓒ 2013 Information Processing Society of Japan

表 1　自由テキスト入力モードのための音声コマンド．
Table 1　Voice commands for creating free formatted texts.

音声コマンド	意味
「訂正」	ひとつ前の入力をやり直し
「取り消し」	ひとつ前の訂正を取り消し．
「保存」	入力したテキストを保存して自由テキスト入力モードを終わる
「破棄」	入力したテキストを破棄して自由テキスト入力モードを終わる

図 3 のように，テキスト入力モードでは，全ての文章を一気に入力するのではなく，短いセンテンスをユーザが話し，そしてアプリがその認識結果を返しながら入力していく．また音声認識誤りが起こった場合や，一度入力したテキストを入力し直したい場合，「訂正」と発話することで入力しなおすことができる．このモードでは，こうしたいくつかの音声コマンド（**表 1**）によってユーザの望むテキストを編集する．最後にユーザが「保存」と発話すれば，入力したテキストを保存して，あとで，email やマイクロブログなどに投稿できる．またユーザが「破棄」と発話すれば，アプリからの確認後，テキストを破棄して，このテキスト入力モードを終了する．この自由テキスト入力モードでも，画面を見たり操作したりすることなく，テキストの入力を行うことができる．

3. 音声対話によって短文を投稿するアプリケーション

前節で述べた対話戦略をもとに**図 4** に示すような，スマートフォン (Android[*6]) 向けのアプリケーションを試作した．我々は本試作アプリを「話してツイート」と呼んでいる．本節では，このアプリの音声対話機能について述べる．

3.1 対話システム構築プラットフォーム HRIME

本アプリは，HRIME というマルチドメイン音声対話システム・ツールキットをベースとして作成されている．HRIME は，オブジェクト指向言語 Java でコーディングされており，PC 上で動作することもできるし，中核部分のソースコードを変更することなく Android OS 上で動作させることもできる．さらに，Apache Tomcat[*7]などのサーブレットアプリとしても実装することができる．「話してツイート」は，Android クライアント・バージョンで動作させている．

HRIME を用いて構築できる音声対話システムの基本構成を**図 5** に示す．HRIME の基本的な構成は，一般的な音声対話システムに共通するものである．まずマイクからの音声信号が音声認識器によってテキストになり，それから言語理解器によって意味表現に直される．この意味表現を

図 4　試作アプリのスクリーンショット．今のバージョンでは日本語のみをサポートし，Google Maps を使用している．
Fig. 4　Screen shot of the created Android application. The current version only supports Japanese. This app uses Google Maps.

使って対話管理部が必要に応じて，システムの応答を返す．

HRIME では，対話管理部の主要モジュールが，対話ドメインごとに用意された複数の Expert モジュールからなっており，これにより複数ドメインの対話に対応できるようになっている (**図 6**)．この各 Expert モジュールに対話ドメインを担当させる方式は，特にユーザ発話がドメイン間で同じような場合に音声対話システムの開発者の負担を大きく下げ，従ってロバストな音声対話システムになる [4], [5]．例えば，システム側からユーザにある事柄の確認を行った場合，それに対するユーザ発話は多くの場合「はい」か「いいえ」である．そうするとこの発話が何に対するものであるかを判別するのは，その直前に確認のためのシステム発話を行った Expert モジュールに対するものであるとして

*6 http://www.android.com/
*7 http://tomcat.apache.org/

ⓒ 2013 Information Processing Society of Japan

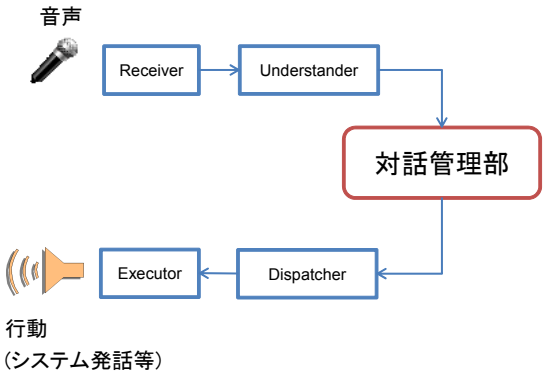

図 5 HRIME によってサポートされる音声対話システムの基本構成図．Receiver モジュールは音声認識の結果を非同期に受け取り，Unserstander モジュールは，その音声認識結果を意味理解表現に変換する．Dispatcher モジュールは対話管理部からのシステムアクションのコマンドを受け取り，Executor モジュールはそのシステムアクションコマンドを非同期に実行する．

Fig. 5 The basic structure of spoken dialogue systems supported by HRIME. The Receiver module receives the results of the automatic speech recognizer. The Understander module translates speech recognition results to semantic representations. The Dispatcher module receives system action commands from the Dialog Manager. The Executor executes the received system action commands.

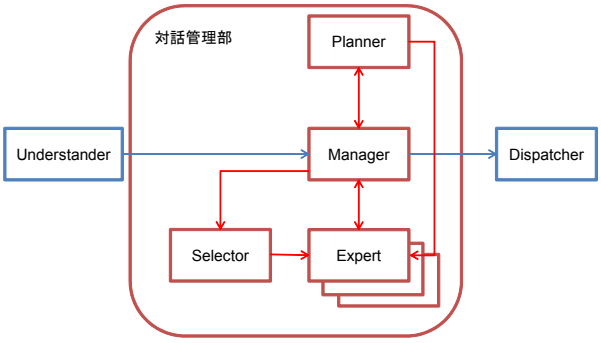

図 6 HRIME の対話管理部の構成．Manager モジュールは，HRIME の中心的なモジュールであり，複数ある Expert モジュールを管理する．Selector は，あるユーザ発話に対して対応する Expert モジュールを選択する．Planner は開発者が設計し，対話の計画に使うことができる．

Fig. 6 The structure of the dialogue management unit in HRIME. Manager is the actual central unit of HRIME and basically manages Expert modules. Selector is to select an Expert module responsible for handling each speech recognition result. Planner is for planning dialogue.

処理できる．このように対話ドメインごとに処理を分けることで，システム側の対話管理がスムーズになる．

どの Expert モジュールが処理を担当するかは原則として，毎回のユーザ発話ごとに判断される．また前の発話を担当した Expert モジュールに，優先権があり，これにより Expert の不用意な遷移を低減する．この時のユーザ発話の言語理解を行う部分は MLUTK(Multiple Language Understand Tool-Kit)[6] によって実装されている．MLUTK は，複数の音声認識器と複数の言語理解方式を扱うことができ [7], [8]，これらを実行時に設定することができる．音声認識が N 個あって各音声認識が最大 n 個の認識結果を返し，言語理解方式が M 個あって，その理解結果が各々最大 m 個あるとすれば，合計で最大 $N \times n \times M \times m$ 個の言語理解結果を得る．本アプリでは，認識器は一つ言語理解方式も一つであるが，それぞれが複数の結果を返すため $n \times m$ 個の理解結果からシステムの動作が決定されている．また必要に応じて，認識結果そのものを使うことも当然できる．本アプリの自由テキスト作成では，編集コマンド発話の理解には言語理解結果を使い，テキストの発話には認識結果を使用している．

この HRIME は，以前 RIME-TK と呼ばれていたツールキット [9], [10] をもとに，そのコンセプトは引き継ぎつつ，実装を大幅に書き直したものである．RIME-TK では，複数のスレッドが常にポーリング動作をしていた．これは一つには能動的な AI システムを目指したからで，これによって前節で示した定期的に投稿するような動作も容易に実装できた．しかしながら常にスレッドが動いているという状態は，つまり常に CPU を使い続けることであり消費電力の増加につながる．また，複数のスレッドが動作していることを前提にすると，実際の実装が複雑になりがちであり時に予測できない動作結果になることがあった．HRIME では，基本を受動的なメッセージ駆動型に書き換え，非同期動作が必要な部分だけを別スレッドで動作するようにした．この結果，開発者が動作を予測しやすくなり，処理するべきメッセージがないときにはスレッドが休止状態になるため，消費電力の抑制につながる．このことは特にモバイルデバイスにおいて重要である．そして定期的に投稿するような動作は，遅延メッセージ処理機構によって実現できるようにした．例えば，Expert モジュール中で 10 分後に自分自身にメッセージを送るように指定することで，定期的な動作も必要に応じて行える．また，非同期で動作する出力部分の拡張を開発者が行いやすいようにした．これにより本アプリではシステム発話と，email の送信，またはツイッターへの投稿やその検索を非同期に行っている．

3.2 言語理解のための記述形式

本アプリでは音声認識には Android Speech API を用いている．これはいわゆる大語彙音声認識であり，文法記述を必要としない．前述のとおり，認識結果の言語理解は MLUTK で行う．以前は，言語理解にも音声認識用の SRGS 等の文法記述を使用するようにしていたが，今回新

ⓒ 2013 Information Processing Society of Japan

(a)ontology.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ontology>
    <is-a-relation child="weather_rain" parent="weather" />
    <is-a-relation child="weather_clear" parent="weather" />
    <is-a-relation child="..." parent="..." />
</ontology>
```

(b)dictionary.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dictionary>
   <entry concept="weather_rain">
      <system word="雨" pron="アメ"/>
      <user word="雨" />
      <user word="大雨"/>
      <user word="霧雨"/>
   </entry>
   <entry concept="person_name">
      <raw-text/>
   </entry>
</dictionary>
```

(c)phrases-list.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phrase-list>
   <group id="here">
         <phrase>ここ[(まで|のところ)]</phrase>
         <phrase>このあたり</phrase>
         <phrase>現在地</phrase>
   </group>
   <group id="calendar_date">
         <phrase>…</phrase>
   </group>
</phrase-list>
```

(d)sentences.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sentence-list>
   <action name="report" domain="weather">
      <example>ここは雨</example>
      <example>晴れている</example>
      <pattern>*[{here}]*{weather|*weather}*</pattern>
   </action>
</sentence-list>
```

(e)variation-list.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<variation-list>
   <expression orth="今">
         <variation>いま</variation>
   </expression>
   <expression orth="雨">
      <variation>あめ</variation>
         <variation>アメ</variation>
   </expression>
</variation-list>
```

図 7 HGSF の例
Fig. 7 A simplified example of HGSF.

たな記述形式を導入した．この新しい記述形式では以下の点を考慮して作成した．

コンセプトの共通化 言語間・アプリ間で共通なコンセプトを積極的に纏めて共通化しやすいようにする．

マルチドメイン対話への対応 あるテキストが，どのドメインに該当し，どのようなユーザ発話 (User Action) として理解されるか明確に記述できる．

表記ゆれへの対処 音声認識結果の表記ゆれに対処するために，認識器ごとに別に管理できるようにする．

以上挙げた点は，音声認識用の文法形式でも実現できなくはないが，実際の研究・開発の経験から記述形式として明確にサポートすることにした．

現在 5 つの XML ファイルからなり，まとめて HGSF（Hrime Grammar Specification Format）と呼ぶ．**図 7** にこれら XML ファイルの簡単な例を示す．図 7（a）はコンセプトのオントロジーを記述する例を示す．今のところ，コンセプトの親子（is-a）関係を設定できる．図 7（b）でオントロジーで示されたコンセプトの言語表現を記述する．ここで，`<raw-text/>`は，認識結果そのものがこのコンセプトの言語表現になることを示すものである．続いて図 7（c）は，そのアプリまたは特定の言語で使われる言語表現に ID をつけるために使われる．ある文字列に名前を付けて管理する場合，上記 ontology+dictionary に記述するか，この phrase-list に記述するかは基本任意である．しかし例えば多言語に展開するアプリケーションを開発する場合などには，共通のコンセプトを ontology として記述し，言語特有の表現などは phrase-list に記述するようにした方が多人数で作業する場合の効率は上がると考えられる．図 7（d）に示す XML ファイルには，言語理解に使われる文パターンと例文を定義する．pattern エレメントに文パターンを記述する．“*” は任意の文字列・単語列にマッチする．また，音声認識文法でよく使われる“`[ ]`”（0 ないし 1 つ）や，“`( | )`”（|で仕切られた文字列のいずれか）を使うことができる．そして，`{here}`は，上記の ontology か phrase-list に示される言語表現にマッチすることを示す．特に`{weather:*weather}`は，スロットになる．この場合，スロット名が `weather` になり，その値は ontology の `weather` の子コンセプトのうち dictionary に記述された言語表現にマッチした場合，その `weather` の子コンセプトの名前が入る．もし，ontology に `weather` が記述されていない場合には，phrase-list から探索される．最後の図 7（e）は，音声認識での表記ゆれに対処するための記述形式である．

3.3 言語生成のための記述形式

またシステムの発話生成の記述には**図 8** に示すような XML によるフォーマットを用意した．ここで，`<template id="reply">`はプログラム中で，id 引数に`"reply"`を指定することで実行される，出力のテンプレートを意味する．実際に実行される内容は`<madl>`エレメント中に記述される．この中で，`<utterance>hello</utterance>`などと記述すれば，システムの音声合成が hello と発話する．またこの中の longitude などは，変数であり実行時にプログラムより `longitude="123.345"`と指定されれば，この値に置き換わる．

以上のような柔軟な入出力フォーマットを記述すること

ⓒ 2013 Information Processing Society of Japan

表 2　Expert モジュール．
Table 2　Expert modules.

名前	実装クラス	役割
text-edit	TextEditExpert	自由テキスト入力
weather	UserRecordExpert	定型文：天候
location	UserRecordExpert	定型文：位置情報
traffic	UserRecordExpert	定型文：道路状況 (工事，渋滞)
location-repeat	UserRecordWithSlotsExpert	定型文：位置情報の繰り返し投稿
publish	PublishExpert	投稿の制御
chatbot	ChatbotExpert	chat 的な会話，対話エラー処理

```
<?xml version="1.0" encoding="UTF-8"?>
<templates>

<generation type="confirm-record">
    <madl>
        <utterance>天候、{weather}を記録しますか？</utterance>
    </madl>
</generation>

<generation type="prompt-record">
    <madl>
        <utterance>この地点の天候，{weather}を記録します。</utterance>
        <publish title="天候"
            latitude="{latitude}" longitude="{longitude}"
            timestamp="{timestamp}" text="{weather}"/>
    </madl>
</generation>

</templates>
```

図 8　簡単化した template の例
Fig. 8　A simplified example of output template.

で，同じ実装クラスが，複数種類の Expert モジュールとして，動作することができる．また，入出力フォーマットを記述することで，容易にシステムを拡張することができる．このため，本論文で示した定型文以外にも，容易に定型文を作成する Expert モジュールを追加することができる．

3.4　「話してツイート」の構成

今回試作したアプリ，「話してツイート」では，音声認識に Android Speech API を使用し，言語理解には WFST[11] を用いている．WFST では，HGSF で説明した．“*” や`<raw-text/>`にマッチした場合ペナルティを課すように設定した．従って，すべてが`<raw-text/>`等にマッチしてしまうような理解結果はスコアが低くなる．音声合成には HOYA サービス株式会社による Voice Text[*8]を使用している．

試作アプリ「話したツイート」で使用している Expert モジュール名と，その役割を表に示す．この表中で，実装クラスは Java でのクラス名を表し，これを見ると分かる通り，いくつかの Expert モジュールが共通のコードで実装されており，その入出力について別途記述することで，それぞれ別の Expert モジュールとなっている．

特に自由なテキスト入力モードを担当する Expert モジュールが一度選ばれると，ユーザが明示的に「保存・破棄」などの音声指示を行わない限り，他の Expert モジュールに遷移しない．

[*8] http://voicetext.jp/index.html

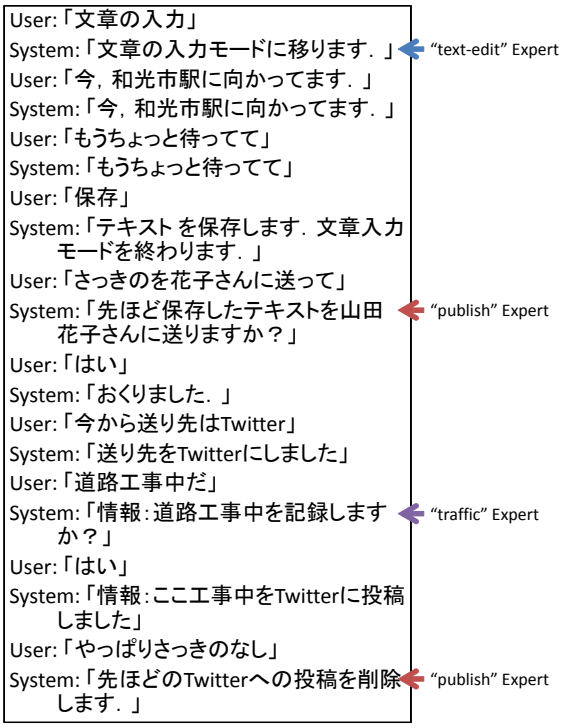

図 9　対話例と対応する対応する Expert モジュール
Fig. 9　Example Dialogue and each corresponding Expert module.

3.5　「話してツイート」による対話例

「話してツイート」によって実現されるユーザ・システム間の対話例を**図 9** に示す．ユーザ発話によって，Expert モジュールが適宜選択されることによって，定型的なテキストの入力と自由なテキストの入力，さらに投稿の制御を自然な対話の中で行うことができる．また試作アプリではあるが，実際に gmail を使用したメールの送信や Twitter への投稿などを，対話を通じて行うこともできるように実装されている．

ⓒ 2013 Information Processing Society of Japan

4. まとめと今後の予定

歩行中や運転中に，SNS に投稿するための音声対話アプリ「話してツイート」の概要と，その対話戦略とシステムの概要を述べた．こうしたアプリによって，歩行中や運転中の情報がツイッターなどに増えていけば，さらに発展したツイッター情報の活用も広がっていくと思われる．今後の予定としては，被験者実験による評価を行っていきたい．また，ここで紹介した HRIME は，このアプリの例に限らずマルチドメインの対話システムの構築基盤として様々な応用が考えられる．ドキュメントの充実と，HRIME を使用した対話アプリ作成のための指針作りを進めていく予定である．

参考文献

[1] Bellegarda, J. R.: Spoken Language Understanding for Natural Interaction: the Siri Experience, *the 4th International Workshop on Spoken Dialogue Systems (IWSDS 2012)*, pp. 3 – 14 (2012).

[2] Bonner, M., Brudvik, J., Abowd, G. and Edwards, W.: No-Look Notes: Accessible Eyes-Free Multi-touch Text Entry, *Pervasive Computing* (Floreen, P., Kruger, A. and Spasojevic, M., eds.), Lecture Notes in Computer Science, Vol. 6030, Springer Berlin Heidelberg, pp. 409–426 (2010).

[3] Winter, U., Ben-Aharon, R., Chernobrov, D. and Hecht, R. M.: Topics as contextual indicators for word choice in SMS conversations, *Proceedings of the SIGDIAL 2011 Conference*, SIGDIAL '11, pp. 185–193 (2011).

[4] Lin, B., Wang, H. and Lee, L.: Consistent Dialogue across Concurrent Topics based on An Expert System Model, *Proc. Eurospeech-99*, pp. 1427–1430 (1999).

[5] Komatani, K., Kanda, N., Nakano, M., Nakadai, K., Tsujino, H., Ogata, T. and Okuno, H. G.: Multi-Domain Spoken Dialogue System with Extensibility and Robustness against Speech Recognition Errors, *Proc. SIGDIAL 2006*, pp. 9–17 (2006).

[6] 竹内誉羽，中野幹生，森祥二郎，駒谷和範：音声対話システムにおいて複数の言語理解モデルの利用を容易にするツールキット：MLUTK，第 89 回音声言語情報処理研究会 (SIG-SLP) (2011).

[7] 勝丸真樹，中野幹生，駒谷和範，船越孝太郎，辻野広司，尾形哲也，奥乃博：複数の言語モデルと言語理解モデルによる音声理解の高精度化，電子情報通信学会論文誌，Vol. J93-D, No. 6, pp. 879–888 (2010).

[8] Komatani, K., Katsumaru, M., Nakano, M., Funakoshi, K., Ogata, T. and Okuno, H. G.: Automatic Allocation of Training Data for Rapid Prototyping of Speech Understanding based on Multiple Model Combination, *Proc. COLING 2010*, pp. 579–587 (2010).

[9] Nakano, M., Hasegawa, Y., Funakoshi, K., Takeuchi, J., Torii, T., Nakadai, K., Kanda, N., Komatani, K., Okuno, H. G. and Tsujino, H.: A multi-expert model for dialogue and behavior control of conversational robots and agents, *Know.-Based Syst.*, Vol. 24, pp. 248–256 (2011).

[10] 中野幹生，船越孝太郎，長谷川雄二，辻野広司：オブジェクト指向に基づくロボット・エージェントのマルチドメイン対話行動制御モジュール構築ツール RIME-TK，人工知能学会研究会資料 SIG-SLUD-54，No. 54, , pp. 21–26 (2008).

[11] Fukubayashi, Y., Komatani, K., Nakano, M., Funakoshi, K., Tsujino, H., Ogata, T. and Okuno, H. G.: Rapid Prototyping of Robust Language Understanding Modules for Spoken Dialogue Systems, *Proc. IJCNLP 2008*, pp. 210–216 (2008).

ⓒ 2013 Information Processing Society of Japan