音声認識結果の意味表現への変換手法の開発

小柏 博行　荒木 雅弘　西本 卓也　新美 康永

京都工芸繊維大学 工芸学部 電子情報工学科

1 はじめに

近年、これまで人間のオペレータが行ってきた電話での応答サービスに自動電話応答システムが導入されるようになり、大幅に進歩してきている音声認識／合成の技術などを利用して、Web 上の情報へアクセスすることを可能とするボイスブラウザの要求が高まっている。この要求に応えるべく「VoiceXML フォーラム」(http://www.voicexml.org) はボイス・アプリケーションを記述するための標準マークアップ言語として VoiceXML 1.0[1] を発表した。

この言語においてユーザ発話の取得はユーザにプロンプトを提示し、応答を待つという形式で行われる。そのため、発話を受理する文法を記述するための形式として、VoiceXML 1.0 では Java Speech Grammar Format（JSGF）形式が用いられた。そして Version 2.0 Working Draft での文法形式は SRGS（Speech Recognition Grammar Specification）と呼ばれ、XML 形式と拡張 BNF 形式（Augmented Backus-Naur Form）を採用している。

しかし、VoiceXML の定義上での混合主導型対話(一発話で複数の変数値を与える発話を含む対話) を実現させるためには、現状の仕様では受理されたユーザ発話と出現した値を保存する変数とのマッピング処理のメカニズムが 不明確である。そこで本研究では SRGS 形式の音声認識文法を用いてユーザの発話を受理し、発話と変数がマップされた意味表現を出力する意味解析系の実装を通じて、現状の仕様でのマッピング処理のメカニズムに関する問題点を明らかにしていく。図 1 に本研究で作成した解析系の概要図を示す。

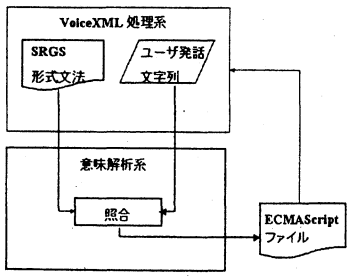

図 1: 意味解析系の概要図

2 音声対話パターン記述言語 VoiceXML

VoiceXML はボイスマークアップ言語と呼ばれるもので、XML をベースに開発された、音声で Web コンテンツにアクセスできるコンテンツ開発言語である。

2.1 VoiceXML での混合主導型対話の実現

VoiceXML の仕様 [1] では、混合主導型対話は以下のように定義されている。

- field 変数が任意の順番で埋めることができる
- 1 発話により、複数の field 変数を埋めることができる
- 1 回目の発話により埋まらなかった field 変数はシステム主導により埋める

例えば、図 2 の "reservation" という id 属性を持つ form は ⟨*initial*⟩ が存在するので、form の子要素である field の to, res, num の各編数値を任意の順序で、かつ複数同時に埋めることができる。図 2 の form で行うことのできる混合主導型対話の例を以下に示す。

```
Computer: 新幹線切符販売機です。
Computer: ご希望の切符をどうぞ。
User: 博多まで 1 枚。
C: 指定席ですか？自由席ですか？
U: 自由席。
C: 博多まで自由席を 1 枚ですね。
C: (ok.vxml に移動)
```

initial 要素が存在すると、その内部の prompt 要素を参照しユーザにプロンプトを出力する。それに対しユーザが「博多まで 1 枚」と発話したことにより目的地を示す field 変数 "to" に '博多'、枚数を示す "num" に '1 枚' という値が代入される。この時点で埋まっていない "res" field は通常通りのシステム主導型対話により埋められるので、「指定席ですか？自由席ですか？」という prompt が出力され、残った "res" field も埋まるので "reservation" form 内の全ての field 変数が埋まったことになり、アプリケーションサーバに全ての変数名と値が渡される。

連絡先：〒 606-8585 京都市左京区松ヶ崎御所海道町

```
<?xml version="1.0"?>
<vxml version="1.0">
<form id="greeting">
  <block>新幹線切符販売機です。</block>
</form>
<form id="reservation">
  <initial>
    <prompt>ご希望の切符をどうぞ。</prompt>
  </initial>
  <field name="to">
    <prompt>目的地を選んでください。</prompt>
    <grammar version="1.0" mode="voice">
      <rule id="destination">
        <one-of>
          <item>東京</item>
          <item>博多</item>
          <item>新大阪</item>
          <item>名古屋</item>
        </one-of>
      </rule>
    </grammar>
  </field>
  <field name="res">
    <prompt>指定席ですか？自由席ですか？</prompt>
    <grammar version="1.0" mode="voice">
      <rule id="seat_type">
        <one-of>
          <item>指定席</item>
          <item>自由席</item>
        </one-of>
      </rule>
    </grammar>
  </field>
  <field name="num">
    <prompt>何枚ですか？ 一度に購入できる枚数は 3 枚までです。</prompt>
    <grammar version="1.0" mode="voice">
      <rule id="num">
        <one-of>
          <item>1 枚</item>
          <item>2 枚</item>
          <item>3 枚</item>
        </one-of>
      </rule>
    </grammar>
  </field>
  <block>
    <value expr="to"/>まで<value expr="res"/>を
    <value expr="num"/>ですね。
    <goto next="./ok.cgi"/>
  </block>
</form>
</vxml>
```

図 2: 混合主導型 form の例

.. システム主導型対話と違い、混合主導型対話ではユーザ発話のどの部分に field 変数を埋める情報が含まれているか分からない。よって、混合主導型対話を成立させるためには発話中にシステムにとって意味のある情報が含まれているか、もし含まれていればそれはどのような情報であるかを解析する機能が必要となる。過去に行われた研究 [3] において、VoiceXML での混合主導型対話を実現させようとする試みがあった。その研究では非常に限定された形式でこの機能を実現させたが、実用面では問題があった。この問題と、本研究での解決法は次節で詳述する。

2.2 ユーザ主導発話の処理

VoiceXML の定義上の混合主導型対話は本来の定義よりもシステム主導型に偏ったものである。その理由はユーザ主導の発話への対処が不十分なことにある。例えば、ユーザからの聞き返しについて考えると、先の新幹線切符販売機のタスクであれば、ユーザは席の種類を決める際、双方の料金の差を尋ねてから判断するかもしれない。しかし、現在の VoiceXML の仕様ではその際の処理手法が確立していない。<initial>を使った対話でこの問題を解決しようとすると新たに文脈保持のためのスロットを設け、ユーザからの質問を受けた場合のための rule を用意し、ユーザ発話の種類によって sentence-type に値を振り分け、その値を参照することで処理を分岐させなければならない。だがその場合、どのスロットにすでに値が埋まっているかなどの情報が失われてしまい、ユーザからの質問に答えた後の処理に問題が生じる。本研究では <initial> を用いない対話においてのユーザ主導発話の処理にのみ対応した。

3 意味解析系の設計

3.1 ECMAScript オブジェクトによる意味表現

ECMAScript[2] とはヨーロッパの情報処理に関する標準化機関である ECMA が JavaScript などを元に標準化した規格であり、オブジェクトベースのスクリプト言語である。オブジェクトは基本的に「スロット名 : スロット値」の任意個の組から構成され、スロット値としてオブジェクトを指定することも可能である。

VoiceXML では、内部的に通常変数や、field 変数等の管理が ECMAScript のオブジェクトを使って行われており、本解析系からの意味表現の出力もこの ECMAScript のオブジェクト形式に則るものとする。

3.2 文法アノテーションを用いた意味解析

[3] では簡易的に意味表現を生成する処理を実現するために音声認識エンジンの文法のアノテーション（注釈）機能を用いた手法を試みた。アノテーション機能とはシステムにとって意味のあるデータに句を関連付ける機能である。スロット名を句として設定すれ

ばどのスロットに入るべきデータであるかが明らかとなり（VoiceXML 定義上の）混合主導型対話が可能となる。しかしこの手法には問題があり、同一の単語を grammar の要素として持つ field が複数ある場合に正しい field に値が埋まらない可能性があった。

3.3 SRGS-XML による文法表記

本研究では SRGS の 2 形式のうち、可読性の高さ、拡張の容易さ、パーサの実装のしやすさの点から、文法表記として XML 形式を選択した。

図 3 の例では *<rule>* の id 属性によって 規則のラベル付けをする。*<one-of>* は選択肢の集合を表し、個々の選択肢は *<item>* の中に記述する。*<item>* の中に *<tag>* が存在すれば、その *<item>* にマッチしたときには *<tag>* の内容が提出されることになる。この例であれば、「1 枚」という発話が認識されると、変数 num に 1 が代入される。

3.4 拡張 SRGS-XML を用いた意味解析

本研究では [3] の手法における問題を踏まえ、システムが必要とする部分だけを抽出し、スロットを埋めていくという手法を採る。文法の記述をする際、どのスロットに値を埋めるかを示す slot 属性を item 要素に付与することでこの問題を解決しようと試みた。

前節の例を 拡張 SRGS-XML で処理するための grammar を図 3 に示す.

この grammar に「東京 から 大阪 まで」という発話を入力すると、まず id が “from” である rule との照合が行われ、grammar にマッチする「東京 から」の部分が発話から抽出される。*slot* 属性を付与された *<item>* 要素があるのでそれを参照し、‘東京’ で “from” field を埋める。次に同様に id が *to* の rule との照合が行われ、「大阪 まで」が抽出される。そして同様に “to” field が ‘大阪’ で埋まることになる。

前節の手法は *<rule>* 要素にマッチした発話がそのまま field を埋めていたので、発話の構造的な概念を用いることができず、発話の中の抽出したい部分とその付加情報を一まとめにして解析することができなかった。ところが *slot* 属性を導入して field を埋める単位としての役割を担わせることにより、*<rule>* 要素は照合処理の最小単位としての役割に専念でき、解析処理には必要だが処理系に返す必要はない付加情報を有効に使えるようになる。

このことにより、例えば “から” や “まで” などの助詞も *<rule>* の中に含め照合処理の最小単位として扱う。一方、*<item>* の中の欲しい情報には *slot* 属性を付与することにより、特定のスロットをその値で埋める。このようにしてスロット名と値との関連付けを行い、[3] の問題の解消を試みた。

```
<grammar mode="voice">
  <rule id="train_order">
    <ruleref uri="#from"/> <ruleref uri="#to"/>
  </rule>
  <rule id="from">
    <one-of>
      <item slot="starting_point">東京</item>
      <item slot="starting_point">博多</item>
      <item slot="starting_point">新大阪</item>
    </one-of>
    <one-of>
      <item>から</item>
      <item>発</item>
      <item>行き</item>
    </one-of>
  </rule>

  <rule id="to">
    <one-of>
      <item slot="destination">東京</item>
      <item slot="destination">博多</item>
      <item slot="destination">新大阪</item>
    </one-of>
    <one-of>
      <item>まで</item>
      <item>へ</item>
      <item>に</item>
    </one-of>
  </rule>
  <rule id="res">
    <one-of>
      <item slot="seat_type">指定席</item>
      <item slot="seat_type">自由席</item>
    </one-of>
    <one-of>
      <item repeat="0-1">で</item>
      <item repeat="0-1">を</item>
    </one-of>
  </rule>
  <rule id="num">
    <one-of>
      <item><tag>1</tag>1 枚</item>
      <item><tag>2</tag>2 枚</item>
      <item><tag>3</tag>3 枚</item>
    </one-of>
  </rule>
</grammar>
```

図 3: 拡張 SRGS で記述した grammar

4 実行例・評価

4.1 VoiceXML における混合主導型対話の実行例

図 3 に示した文法が定義されているときに「東京 から 大阪 まで 指定席 を 1 枚」という発話を入力したときの出力結果は以下のようになる。

```
{
  from:{starting_point:"東京"}
  to:{destination:"大阪"}
  seat_type:"指定席"
  num:"1"
}
```

4.2 ユーザ主導発話の実行例

ユーザ主導発話の処理をするための VoiceXML を図 4 に、grammar を図 5 に示す。「席の種類を選んでください」の後にユーザ発話として「指定席 の 料金が知りたい」という発話が認識されたときの出力を図 6 に示す。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<vxml version="1.0">
  <form id="reserve">
    <field name="seat_type">
      <prompt>席の種類を選んでください。</prompt>
      <help>自由席、指定席、グリーン車から選ぶことができます。</help>
    </field>
    <filled>
      <if cond="sentence-type == 'answer'">
        <!-- 返答が返ってきた場合 -->
        <submit next="./ok.cgi" namelist="reserve">
      <elseif cond="sentence-type == 'question'"/>
        <!-- 質問が返ってきた場合 -->
        <subdialog namelist="question_type seat_type"
                   src="./question.cgi">
      </if>
    </filled>
  </form>
</vxml>
```

図 4: ユーザ主導発話を処理する VoiceXML

5 終わりに

本研究では、VoiceXML 処理系に組み込むことを前提とした、ユーザ発話の意味解析系の実装を行い、意味解析処理上の問題を grammar の形式の拡張によって解決した。混合主導型対話の実現には、ユーザ主導発話の処理には問題を残すものの VoiceXML における定義上のそれの実現には成功した。

参考文献

[1] VoiceXML Forum : "Voice eXtensible Markup Language Version 1.0", (2000)
http://www.voicexml.org/

[2] ECMA: "ECMA-262 ECMAScript language Specification 3rd edition", (1999)

[3] 秋田祥史 : "VoiceXML における混合主導型対話の実現", 2001 年人工知能学会全国大会論文集, 3F1-02, (2001)


```
<?xml version="1.0" encoding="Shift_JIS"?>
<grammar version="1.0" mode="voice">
  <rule id ="sentence-type">
    <one-of>
      <item> <ruleref="#answer"/> </item>
      <item> <ruleref="#question"/> </item>
  </rule>
  <!-- システムの質問に対する回答 -->
  <rule id="answer">
    <tag> sentence-type='answer' </tag>
    <one-of>
      <item slot="seat_type">自由席</item>
      <item slot="seat_type">指定席</item>
      <item slot="seat_type">グリーン車</item>
    </one-of>
  </rule>

  <!-- ユーザからの質問 -->
  <rule id="question">
    <tag> sentence-type='question' </tag>
    <ruleref uri="#seat_type">の
    <ruleref uri="#question_type">が知りたい
  </rule>
  <rule id="seat_type">
    <one-of>
      <item>自由席</item>
      <item>指定席</item>
      <item>グリーン車</item>
    </one-of>
  </rule>
  <rule id="question_type">
    <one-of>
      <item>料金</item>
      <item>車両数</item>
      <item>予約状況</item>
    </one-of>
  </rule>
</grammar>
```

図 5: ユーザ主導発話を処理する grammar

```
{
  sentence-type:"question"
  question_type:"料金"
  seat_type:"指定席"
}
```

図 6: ユーザ主導発話が入力されたときに出力されるオブジェクト