2003－SLP－46 (9)
2003／5／27

VoiceXMLをベースにした頑強な音声対話管理アーキテクチャ

大淵康成[1]，畑岡信夫[2]，赤堀一郎[3]、立石雅彦[3]、
Scott Judy[4], Teruko Mitamura[4], and Eric Nyberg[4]

1 日立製作所基礎研究所
2 日立製作所中央研究所
3 デンソー基礎研究所
4 Language Technologies Institute, Carnegie Mellon University

obuchi@rd.hitachi.co.jp

本論文では、カーテレマティクス向けの音声対話管理アーキテクチャについて述べる。カーテレマティクスシステムでは、ユーザーの自由な発話を処理し、車載クライアントとサービスセンターの間の通信回線の多様な状況に対応することが必要である。本システムは、VoiceXMLとHTTPによる通信に基づいており、サーバーシステムの構築には、VoiceXMLの拡張であるDialogXMLとScenarioXMLが用いられている。これらの拡張により、状態遷移をそのまま記述する方式での開発が可能になると同時に、様々な動的データベースへのアクセスが実現する。クライアントの音声認識システムは、複数の文法および語彙のセットによって記述され、個々の場面で必要となる文法・語彙のみが起動されることによって対話のコントロールが行なわれる。また、サーバー側のアプリケーションと同じインターフェースを持った簡易版アプリケーションをクライアントにも置くことによって、通信回線の状態に応じて効率的にサーバーサイドアプリケーションとクライアントサイドアプリケーションの切替えを行なう。これにより通信遮断時にも継続してシステムを動作させることが可能になる。

Robust Spoken Dialog Management Architecture Using VoiceXML

Yasunari Obuchi[1], Nobuo Hataoka[2], Ichiro Akahori[3], Masahiko Tateishi[3], Scott Judy[4], Teruko Mitamura[4], and Eric Nyberg[4]

1 Advanced Research Laboratory, Hitachi, Ltd.
2 Central Research Laboratory, Hitachi, Ltd.
3 Research Laboratories, DENSO CORPORATION
4 Language Technologies Institute, Carnegie Mellon University

This paper describes a spoken dialog management architecture for car telematics systems. It is required for the system to handle the user's spontaneous utterances and the variable communication conditions between the in-car client and the server. The communication is based on VoiceXML/HTTP, and the development of the server-side application is based on layered extensions of VoiceXML, called DialogXML and ScenarioXML. These extensions provide support for state-and-transition type programming and enable access to dynamic external databases. The client system includes a set of small grammars and lexicons for various tasks; only relevant grammars and lexicons are activated to control the dialog. The client system may include compact versions of the server-side applications. The system can switch between applications on both sides intelligently. This helps to reduce bandwidth utilization, and allows the system to continue even if the communication channel is lost.

1. はじめに

カーテレマティクスシステムの普及に伴い、車内での音声対話管理の重要性が高まっている。しかし、自然な音声対話管理を実現するためには、効率的な対話シナリオの生成、ユーザー発話の正確な解析、車載クライアントとサービスセンターの協調動作など、多くの解決すべき問題がある。一般に、車載クライアントではコンピューターリソースが限られており、サーバーとの通信回線も決して十分な帯域を持っていないことから、クライアントとサーバーの間で効率的にタスクを分担することが求められる。このような目的のために、我々はVoiceXML[1]に基づく対話アーキテクチャを提唱する。VoiceXMLにおいては、サーバーが対話の流れを導くために必要最低限の情報をXMLファイルの形でクライアントに送る。それに対してクライアントは、ユーザーの反応を記述するために必要最低限の情報をHTTPリクエストの形で返信する。

VoiceXMLを用いてサーバー側で対話管理を行なうアーキテクチャとしては、Carpenterら[2]によって提案されたものがある。VoiceXMLの枠組みにおいては対話の状態遷移を陽に記述することができないため、Carpenterらの方式では、サーバー側に存在する対話管理マネージャー[3]がすべての対話フローを管理し、その都度必要となる情報を細切れのVoiceXMLとして送るという方式をとっている。しかし、カーテレマティクスシステムにおいては、通信回線の品質が確保されていないことから、このように頻繁にVoiceXMLのやりとりをしなければならない方式は不利となる。この問題を解決するため、我々はDialogXMLおよびScenarioXMLと呼ばれるVoiceXMLの拡張言語を提案した[4]。DialogXMLは状態遷移を用いた高レベルの対話フロー管理を可能にし、ScenarioXMLは外部データベースへのシステマティックなアクセスを可能にする。これらの技術を用いることにより、これまでより大きな単位でのVoiceXMLドキュメントのやりとりが可能になり、通信回線の障害に対して頑強なシステムが実現される。実際には、システム開発者はScenarioXMLを用いて対話シナリオを記述するだけでよく、これをコンパイラによってDialogXML、さらにVoiceXMLへと変換する。この際、時々刻々と変化するデータベースの内容をVoiceXMLに反映させるため、Java Server Pages (JSP)[5]の技術が用いられている。

車載システムにおけるもう一つの問題は、ユーザー発話を解析して意図を抽出することである。コールセンターのような大規模CTIシステムにおいては、大語彙連続音声認識システムと自然言語解析システムの組み合わせにより、あらゆる発声からユーザーの意図を抽出しようという試みも可能かもしれない。しかし、車載システムのような限られたリソースのもとでは、より単純なシステムで対話を進行させることが必要である。我々は、正規文法に基づく小規模音声認識システムを用い、タスクや場面に応じた複数の小規模文法と語彙の使い分けをすることにより、同等の機能を実現させる。具体的には、ひとつの文法と語彙からなるペアがあるタスクを規定し、対話管理マネージャーが対話の流れに応じて必要とされる文法・語彙をアクティベートすることによってユーザー発話の解析を容易にする。この際、対話のフローが複数の分岐を持つ場合などには、複数の文法・語彙ペアを指定することによって対応する。このような文法や語彙は、もちろん人間の手によって書いても構わないが、より効率的に開発を行なうためには、コーパスを用いて自動生成することも可能である。この場合、あるタスクに該当する文ばかりを集めたコーパスを用いることにより、そのタスクに対応する文法が生成される。このような自動化プロセスを導入することにより、ユーザーの発話スタイルが変わっても意図を正確に抽出することが可能な対話システムが実現可能になる。

最後に、カーテレマティクスに固有の問題として、タスクの割り込みと通信の遮断についても考える必要がある。車内にはテレマティクス端末の他にも様々なシステムが存在し、車の運行状況に応じてテレマティクス端末に何らかの動作を求める場合がある。また、サーバーとの通信をともなう対話の途中で通信が遮断することも考えられる。このような場合に、サーバーとの対話を一時的に中断し、車載端末の側で必要な処理を行ない、状況が回復した段階で元の対話を復旧させることが必要である。

以下、本論文では、第２章でシステムの構成を簡単に述べたのち、第３章から第５章で本アーキテクチャの中心となる３つの概念について詳しく説明する。上で述べたように、一つめがVoiceXMLの拡張であるDialogXMLとScenarioXML、二つめが文法と語彙の設定、三つめが割り込みと通信遮断の制御である。その後、第６章で現在までに我々が開発したプロトタイプについて述べ、第７章でまとめと今後の課題を示す。

2. システム構成

図１に、本論文で述べるカーテレマティクス向け対話管理方式のアーキテクチャを実現するためのシステム構成を示す。クライアント側の中心となるのはVoiceXMLインタプリタで、音声認識(ASR)および音声合成(TTS)モジュールを通じてユーザーとのやりとりを行なう。ここで用いるVoiceXMLインタプリタは日立中央研究所で独自に開発したもので[6]、VoiceXML 2.0[1]で定義された機能の大半をサポートする他に、クライアント内の他のアプリケーション（例えばGPSモジュール）との非同期通信を行なうための入出力モジュールを持っている。そのため、通常のHTTPおよびVoiceXMLによる対話管理マネージャーとの通信の他に、クライアント内のアプリケーションのみで動作するスタンドアローンモードにも対応している。この機能を用いることにより、例えばレストランについての問い合わせの対話が進行している途中で、車が重要な交差点に差しかかった場合でも、GPSモジュールからの割り込み信号を検知することが可能である。

文法および語彙は、クライアントとサーバーの双方に保持されている。頻繁に用いられるものはクライアント内に保持しておくことによって通信量を削減する一方で、サーバー内には幅広いタスクに対応できる文法・語彙を用意しておき、必要に応じてVoiceXMLと一緒にクライアントに送付する。

サーバー側で中心となる役割を果たすのが、対話管理マネージャー(Dialog Manager)である。対話管理マネージャーは、あらかじめ保持された対話シナリオを元に、クライアントからのHTTPリクエストに応じて必要なVoiceXMLファイルを送付することによって対話の管理を行なう。また、動的なデータ（天気予報、交通情報、新規開店したレストランなど）にも対応できるよう、インターネットを通じて外部のデータベースとも通信を行なう。

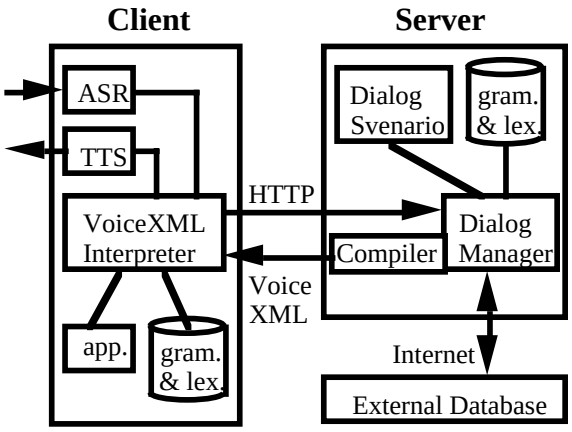

図１　システム構成

対話管理マネージャーは、ScenarioXMLおよびDialogXMLコンパイラをモジュールとして持ち、ScenarioXMLは自動的にDialogXMLを経てVoiceXMLにコンパイルされる。このVoiceXMLがクライアントに送られ、VoiceXMLインタプリタによって解釈されるわけである。また、対話管理マネージャーは、特定のURIを指し示すことによって特定の文法および語彙を指定し、対話フローの管理を行なう。このとき、文法および語彙はクライアント内にあるものでも、サーバーに保持されているものでもどちらでも構わない。

3. VoiceXMLの拡張

我々は以前に、VoiceXMLの拡張によって動的コンテンツを含む音声対話サービスの構築が容易になることを示した[4]。まず、DialogXMLの導入により、状態遷移のアルゴリズムをそのまま記述することによって対話システムを開発できるようになる。同等のシステムを直接VoiceXMLで記述することは不可能ではないが、非常に煩雑な手続きが必要になり、開発工数の増大をもたらすばかりでなく、できあがったVoiceXMLファイルも長くて複雑なものになり、メンテナンスも難しくなる。DialogXMLで書かれたドキュメントはDialogXMLコンパイラによってコンパイルされ、通常のVoiceXMLになるが、開発者はこの内容を確認する必要はなり。

次に、DialogXMLより更に上のレイヤーとして、ScenarioXMLを定義した。ScenarioXMLでは、様々な外部データベースから動的な情報を獲得することが容易にできるようになる。実サービスで必要となる情報というのは大半がこのような動的な情報であり、それゆえ実行時にこれらの情報をDialogXMLに埋め込んでやる必要がある。我々のシステムでは、JSP[5]の技術を用い、動的情報の埋め込みを実現している。これらの機能はテンプレート化され、Java関数で表記された情報獲得の結果がDialogXMLに埋め込まれる。さらに、それらのテンプレートを効率的に用いるためのいくつかの制御用のコマンドや、共通アークと呼ばれるグローバルコマンドの埋め込み方式も含

めて、ScenarioXMLが定義された。開発者が記述したScenarioXMLは、ScenarioXMLコンパイラによってコンパイルされ、更に実行時に得られた動的情報が埋め込まれた形でDialogXMLに変換される。

図２にScenarioXMLの例を示す。類似した状態のループが上位のレイヤーでまとめられていると同時に、Route.get(i)というJava関数が、引数を更新しながら繰り返し呼び出され、外部データベースの情報取り出しに用いられていることがわかる。図３に示したのは共通アークを表わす

```
<javaloopstates namabase="s" array="Route"
 final="sx" index="i">
 <action><prompt>
  <javaval expr="(String)Route.get(i)"/>
 </prompt></action>
 <arc>
  <grammar src="next.gram" type="application/
   x-hgf" fieldlist="next"/>
  <gotoloopnext/>
 </arc>
</javaloopstate>
```

図２　ScenarioXMLの例：
ループと外部データベースへのアクセス

```
<jumplist>
 <arc name="help">
  <grammar src="help.gram" type="application/
   x-hgf" fieldlist="help"/>
  <destination dialog="help.xml"/>
 </arc>
</jumplist>
```

図３　ScenarioXMLの例：共通アーク

```
<state name="s1">
 <action><prompt>
  Go straight on Fifth Avenue.
 </prompt></action>
 <arc>
  <grammar src="next.gram" type="application/
   x-hgf" fieldlist="go"/>
  <dest state="s2"/>
 </arc>
 <arc>
  <grammar src="help.gram" type="application/
   x-hgf" fieldlist="help"/>
  <push dialog="help.xml"/>
 </arc>
</state>
```

図４　DialogXMLの例

ScenarioXMLの例である。"Help"のように、対話中の任意の場面で呼び出される可能性のある対話については、個々の場面に記述するよりも共通アークとしてまとめて記述しておいた方が便利である。このようなしくみを共通アークと呼ぶ。

図４は、図２と図３に示したScenarioXMLから生成されたDialogXMLの一部を表わしている。一つの状態(state)は、アクション(action)といくつかのアーク(arc)で構成されている。この例では、図２のScenarioXMLから状態と一つめのアークが生成され、二つめのアークは図３のScenarioXMLから共通アークとして付加されている。たとえば経路案内のタスクなどは、似たような複数のステップから成っており、このような形式で記述することによって非常に簡明になる。

図５にもう少し複雑な例を示す。メインの対話には大きく二つの流れがあり、その分岐はアークに付与された文法の名前で管理されている。このような文法選択による対話フロー管理については、次章で詳しく述べる。

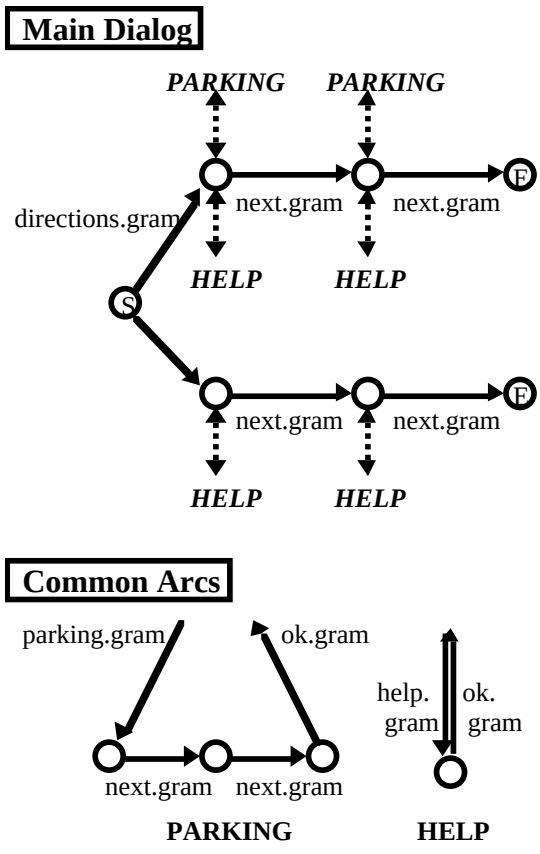

図５　状態遷移と文法の関係

4. 文法と語彙の設定

VoiceXMLにおいては、ユーザーが発声する可能性のある文全体の集合が文法によって規定される。ここでいう文法とは、厳密に言うと、品詞の連結規則を表わす「文法」と、用いられる単語の集合を表わす「語彙」とを合わせたものであるが、以下では簡略化のため、この「文法」と「語彙」の両者を合わせたものを広義の「文法」と呼ぶことにする。

ユーザー発話をカバーする最も広範な集合は、いわゆる大語彙連続音声認識(LVCSR)システムと、自然言語処理システムの組み合わせによって得ることができる。VoiceXMLにおいて上記の組み合わせを用いる場合、自然言語処理システムは、ユーザー発話から特定の属性(attribute)と値(value)の組み合わせを抽出するために用いられる。しかしながら、計算機リソースの限られた車載クライアントにこれらのシステムを搭載することは現実的ではない。我々はその代わりに、正規文法に基づく小語彙音声認識システムを車載クライアントの入力装置として採用した。

上記に加えて、文法には、対話の流れをコントロールするという重要な役割がある。図５に示したように、個々のアークにはそれぞれ".gram"で示された文法が割り当てられている。VoiceXMLでは同時に複数の文法を起動することができるので、このようにして容易に対話の分岐を記述することができる。開発者は、文法の一覧と、それぞれの文法で記述されている属性と値のペアについてのデータを持っていれば、あとはScenarioXML内で個々の場面に適当な文法を参照すれば良い。

書き言葉の自然言語処理とは異なり、音声言語処理システムにおいては、文法をできる限り小さくすることも重要である。なぜなら、必要以上に多くの文をカバーする文法を採用してしまうと、それらの不要な文が誤認識の原因となってしまうからである。すなわち、ユーザー発話の様々な可能性を包含できるよう文法を広げると同時に、誤認識を減らすように文法を狭めることが求められる。この要請に応えるための自動化方式について次に述べる。

図６は、コーパス[7]を用いた文法の自動生成（コンパイル）の流れを示している。まず始めに開発者が単一化文法[8]により文法を記述する。これは認識に用いる正規文法に比べて、様々な制約を効率的に記述することができ、人間の手による開発に向いた文法クラスである。次に、これらの制約をすべて品詞とみなして展開することにより、文脈自由文法に変換する。さらにこの文脈自由文法は、再帰の回数に制限を加えることによって、正規文法に変換される[9]。図に示すように、この正規文法は有限状態機械(FSM)として記述される。こうして得られた正規文法を音声認識用文法として用いることも可能だが、一般にこうして得られた文法は冗長である。そこで、対象となるタスクに特化した文ばかりを集めたコーパスを用意し、それぞれの文をこの文法で解析する。すべての文を解析した後、一度も使われなかった状態およびアークを取り除くことにより、文法を簡略化することができる。

文法のコンパイルと異なり、語彙のコンパイルは難しい問題である。コーパスで用いられた単語だけでは汎化能力に乏しい一方、語彙を単位とした拡張ではパープレキシティが大きくなりすぎ、音声認識性能が劣化する。シソーラスを使った単語のクラスタリングなどの研究も行なわれているが、現在のところ我々のシステムでは人間の手による単語のカテゴリー作成に留まっている。

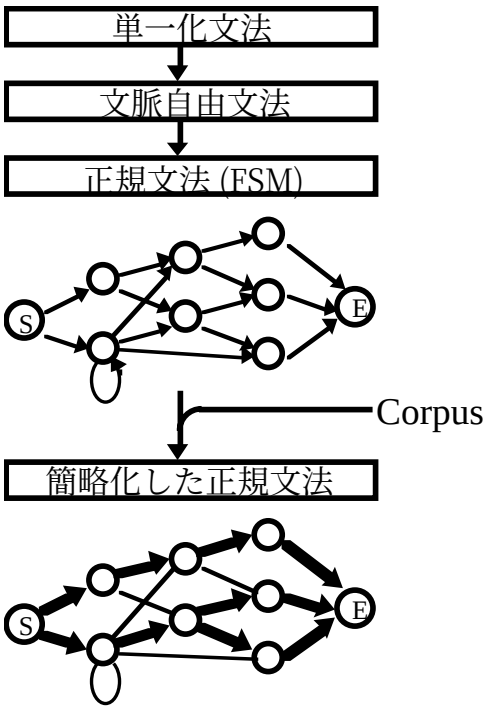

図６　文法のコンパイル

5. 割り込みと通信遮断

カーテレマティクスシステムにおいては、クライアントとサーバーの間の通信は一般に不安定であり、突然の通信遮断などへの対応が重要である。VoiceXMLの仕様では、"error.badfetch"というイベントが定義されており、所望のVoiceXML文書が送られてこないというエラーを検知することができる。これに対するイベントハンドラーの中で、クライアント内の簡易版アプリケーションへの切替えを行なう。例えば経路案内のタスクでは、現在地から目的地までの道順だけをクライアント内に持っておき、渋滞情報などの動的な情報はサーバーのみがアクセスできるとする。この場合、通信が途絶した状態でユーザーが渋滞情報を問い合わせると、「現在その情報へはアクセスできません」と応答した後、待機状態に入り、通信回線の再開もしくはユーザーの次のコマンドを待つ。その他、エアコンの操作のような静的で単純な対話については、すべてクライアントで処理し、通信コストを削減する。

第２章で述べたように、本システムで採用したVoiceXMLインタプリタには、クライアント内のアプリケーションと非同期に通信するためのモジュールが備えられている。これにより、通信回線の再開を検知するのみならず、GPSなどの状態に応じて割り込みを受け付け、対話を中断することも可能である。これにより、緊急性の低い対話が続いている途中で重要な交差点に差しかかったような場合でも、必要な情報を即座に提供することができる。

6. プロトタイプシステム

本論文で述べたアーキテクチャの評価を行なうため、プロトタイプの開発を行なっている。現在のところ、ScenarioXMLとDialogXMLのコンパイラ、文法選択による対話のコントロールなどが既に組み込まれている。ただし、第４章で述べた文法のコンパイルについては、別途独立したシステムとして開発中である。また、第５章で述べた機能の一部として、VoiceXMLインタプリタの非同期通信機能を用いてGPSモジュールからの割り込みを受け付ける方式も、既に実装されている。

7. まとめ

本論文では、カーテレマティクス向けの音声対話アーキテクチャについて述べた。システムはサーバーとクライアントから成り、VoiceXMLにより通信コストを低減するよう設計されている。開発者に対しては、ScenarioXMLとDialogXMLという枠組みが用意されており、状態遷移の記述と、外部データベースアクセスのためのテンプレートを用いて容易にシナリオを作成することができる。ユーザー発話はあらかじめ用意された文法の選択によって解析されると同時に、文法の選択により対話の流れがコントロールされる。文法は人間が直接記述する他に、コーパスを使って一部自動的に生成することも可能である。また、通信回線が不安定な場合にも、クライアント内のアプリケーションへの切替えを行なうことにより、対話を継続させることができる。これらの仕組みにより、様々な使用状況やユーザーのレスポンスに対しても頑強な対話システムが実現される。

参考文献

[1] W3C, “Voice Extensible Markup Language (VoiceXML) Version 2.0 Working Draft”, http://www.w3c.org/TR/voicexml20/

[2] B. Carpenter, S. Caskey, K. Dayanidhi, C. Drouin, and R. Pieraccini, “A Portable, Server-Side Dialog Framework for VoiceXML”, *Proc. of ICASSP 2002*

[3] R. Pieraccini, S. Caskey, K. Dayanidhi, B. Carpenter, and M. Phillips, “ETUDE: A Recursive Dialog Manager with Embedded User Interface Patterns”, *Proc. of ASRU 2001*

[4] E. Nyberg, T. Mitamura, P. Placeway, and M. Duggan, “DialogXML: Extending VoiceXML for Dynamic Dialog Management”, *Proc. of HLT 2002*

[5] Sun Microsystems, “JavaServer Pages”, http://java.sum.com/products/jsp/

[6] 鯨井俊宏、高橋久、天野明雄、畑岡信夫: “VoiceXMLインタプリタと連続単語認識エンジンの開発 音声ポータル向け音声認識技術の開発”, 情報処理学会研究報告SLP-33-12 (2000)

[7] M. Tateishi, I. Akahori, S. Judy, Y. Obuchi, T. Mitamura, and E. Nyberg, “A Spoken Dialog Corpus for Car Telematics Services”, *Proc. of Workshop on DSP in Vehicular and Mobile Systems* (2003)

[8] S. M. Shieber, H. Uszkoreit, J. robinson, and M. Tyson, “The Formalism and Implementation of PATR-II”, SRI International, Menlo Park, California (1983)

[9] A. Black, “Finite State Machines from Feature Grammars”, *Proc. of Int. Workshop on Parsing Technologies* (1989)