TR-IT-0263

# 定型的タスク用音声対話システム設計試案

# Some Proposals on Designing Spoken Language Dialogue Systems for Fixed-Form Tasks

荒川直哉
ARAKAWA Naoya

1998年7月

**Abstract**

This report presents some proposals on designing spoken language dialogue systems, which perform fixed-form tasks such as hotel reservation and on-line help. The report proposes an architecture for spoken dialogue systems, a way to control tasks, and an interface module between task-independent natural language processing modules and natural language-independent task processing modules. The proposals are in accordance with standard methods in software engineering, and a proposed natural language processing module makes use of task knowledge obtained from object analysis during software design.

# 目次

# 第１章　はじめに

このレポートでは、定型的音声対話システム構築に関するいくつかの提案を行う。ここで定型的音声対話システムとは、音声言語による入出力インタフェイスを持ち、定型的な業務（タスク）を実行するソフトウエアシステムのことを指す。定型的業務の種類については第４章で詳しく述べるが、例えば予約業務やオンラインヘルプを包含する。本論での提案は次の３つからなる。

１）アーキテクチャの提案（第２章）
２）タスク制御方法に関する提案（第３章、第４章）
３）音声言語処理とバックエンドの間のインタフェイスモジュールの提案（第５章、第６章）

「１）アーキテクチャの提案」では、対話システムのモジュール分割を行い、２）と３）の提案で検討するソフトウエアモジュールの対話システム全体における位置付けを行う。提案されるアーキテクチャでは、システムは音声言語を処理する部分と、知識ベースとタスク制御の機能を持つ自然言語に依存しない部分に分割される。

「２）タスク制御方法に関する提案」は２つの章からなる。第３章では下準備として定型的音声対話システムが扱うであろうタスクの種類について検討し、第４章では第３章で検討したタスクを制御するモジュールの実現方法についての提案を行う。典型的なタスクを非手続き的な表現を解釈することにより遂行することが提案される。

「３）音声言語処理とバックエンドの間のインタフェイスモジュールの提案」では、言語処理を行う部分と言語に依存しない部分の間でインタフェイスをとるモジュールについて検討を行う。提案されるモジュールは、タスクに関する知識を用い、自然言語に依存する表現を自然言語に依存しない表現に多義性解消を行いつつ変換する。この際、ユーザに対する対話的多義性解消も行われる（第５章）。また、提案されるモジュールは、言語に依存しないモジュールからのメッセージを自然言語表現に近い形に変換する（第６章）。

このレポートでは、音声対話システムの要素技術（音声認識・合成、言語解析、応答生成、マルチモーダル統合技術など）として既存のものを仮定している。個別の技術の詳細については、それぞれの分野の参考書を参照していただきたい。音声対話システムの設計に関して包括的に扱った本としては [Bernsen '98] を是非参照されたい。

以下、本レポートを読む際のいくつかのポイント（キーワードとその説明）を挙げる。

## 1.1 オブジェクト指向

現在では、オブジェクト指向プログラミングは大規模なソフトウエア開発を行う際の規範的な手法となっている [シュレイア '95][ランボー '92]。音声対話システムはそれ自体大規模ソフトウエアであるから、その開発はオブジェクト指向により行われると考える。オブジェクト指向開発は、単にオブジェクト指向言語でプログラミングを行うということではなく、関連するデータやシステム内のモジュールや諸プロセスもオブジェクトとして分析するということを意味する。第５章では、対話システムのタスクで扱う諸データのオブジェクト指向による分析が自然言語の取り扱いにも有用であることを示す。

## 1.2 エージェント指向

ここで「エージェント指向」と呼ぶのは、いくつかの「ソフトウエアエージェント」（ある程度汎用的な言語によるメッセージの交換を行う独立したソフトウエアモジュール）により目的の機能を実現しようという手法である [西田 '95]。このレポートでは、エージェント指向を前面に打ち出しているわけではないが、音声対話システムを個別の自然言語に依存する部分と、バックエンドの自然言語に依存しない部分に分割して、その間を抽象化したメッセージのやりとりで記述するとい

う提案や、自然言語処理のための情報を知識ベースへの問い合わせによって獲得するといった提案（第５章）はエージェント指向の考え方の影響を受けている。実際、第４章と第５章では、モジュール間メッセージの記述例に既存の「エージェント通信言語」を用いて解説を行っている。

### 1.3 タスク分析

開発するソフトウエアにどのような機能が要求され、他のシステムとどのようなインタラクションを行うかということの分析は、すべてのソフトウエアの設計に際して必要なことである。（タスク指向の）音声対話システムの構築に当たっては、システムがユーザとの対話を通して遂行すべきタスクの分析が行われる。まずタスクのゴールを明確にすることが重要であり、次に人間によるタスクの実行例の分析により、タスク中のデータやオブジェクトの種類、データの流れ、データ要求・提示などのアクションの生起条件などを明確にする（☞ [DeMarco '86][ランボー '92]）。また、マルチモーダルシステムの設計では、ユーザとの情報のやりとりのうちどれをどんな場合にどのモダリティによって行うのがよいのか、といった考察も行わなければならない。

### 1.4 データ分析

タスク分析と共にタスクに関連するデータの分析が行われる。オブジェクト指向分析ではデータは何らかのクラスに属するものとみなされる。クラスにはクラス階層が定義される。また、通常オブジェクトは何らかの属性を持つ。オブジェクト指向分析の結果、タスクオントロジ、すなわちタスク中で言及される諸オブジェクト全体をクラスや属性、オブジェクト、属性間の関係によって記述したものが産み出される。これは従来のソフトウエア工学でデータディクショナリと呼ぶものをより詳細化したものといえる。タスクオントロジ中の知識は第５章で述べるように自然言語処理に用いることができる。

音声対話システムで用いるデータは、通常のソフトウエア設計で分析されるデータだけではなく、辞書や音声・言語統計ファイルなどの音声・言語関連データを含む。こうしたデータの多くはタスクあるいはそれが属するドメイン（分野）に依存するものであるから、音声対話システムの設計に際しては、音声・言語関連データの準備も視野に入れておかなければならない。データの一部（例えば自然言語語彙と計算機システム内での表現の語彙との対訳辞書）は人手で作らなければならないかもしれないが、統計データは人間によるタスクの実行例に伴う会話の録音およびその書き起こし（コーパス）などから得ることができる。

### 1.5 タスクとドメイン

タスクとはあるゴールとそれを達成するために行わなければならないことのの総体とみなすことができる。一方、ドメイン（分野）はタスクが生起する環境（相互に関連する空間、オブジェクトの総体）である。例えば旅行のドメインに対して、ホテルや飛行機の予約、旅行のプランニングなどのタスクが考えられる。医療のドメインでも病院の予約業務があるように、異なるドメインの間でも構造的に似かよったタスクがある。

音声対話システムは大規模なシステムであるから、タスク／ドメインに依存する部分としない部分を明確に分離し、別のタスク／ドメイン向けのシステムを作成する場合は変更部分を最小限に押さえ、依存しない部分を再利用することが望ましい。特にプログラムのコードの変更は最小限にとどめたい。本論ではこの点に留意し、言語処理に関する部分はタスク／ドメインによるコード変更が必要でないようなアーキテクチャを提案をしている。また、タスクの制御に直接かかわる部分でも汎用的なタスク制御機構を提案する（第４章）などして、プログラムのタスク／ドメイン依存性を減らすような考慮をしている。

第１章の文献

[Bernsen '98] N. O. Bernsen, H. D. Dybkjæer & L. Dybkjæer: *Designing Interactive Speech Systems.* Springer (1998).

[DeMarco '86] T. Demarco：*構造化分析とシステム仕様,* 日経マグローヒル (1986).

[西田 '95] 西田豊明："ソフトウエアエージェント"、人工知能学会誌 Vol.10 #5 (1995).

[シュレイア'95] S．シュレィアー & S.J．メラー：*オブジェクト指向システム分析,* 近代科学社 (1995).

[ランボー '92] J．ランボー, M．ブラハ, W．プレメラニ, F．エディ & W．ローレンセン：*オブジェクト指向方法論OMT－モデル化と設計,* プレンティスホール・トッパン (1992).

# 第2章　アーキテクチャ

この章では、後続する章での議論のために音声対話システムの構造化を提示し、モジュール群を定義する。構造化は、対話の主導権、およびモジュールの個別言語への依存性という2つの観点によって行う。

## 2.1 対話の主導権とアーキテクチャ

この節では対話システムのアーキテクチャの問題を論じるために対話のインタラクションの主導権（initiative）の問題に触れる。ここで、「対話の主導」とはインタラクションを駆動（initiate）する要求の発信（例えば質問の発話）を意味する。ソフトウエアとユーザの対話的インタラクションにおいては、場合によってソフトウエア側が主導権を握ってユーザに質問をすることもあれば、ユーザが主導権を握ってソフトウエアに指示や質問をしたりすることもある。

ユーザが主導権を握る対話状況はユーザがソフトウエアに要求を送り、ソフトウエアが要求への返答を行うという図式で定式化できる（図 2.1）。

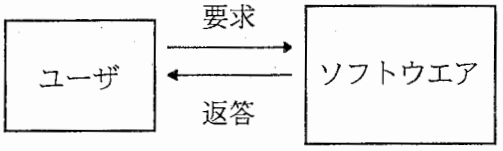


図 2.1

この図式は、ソフトウエア側を知識ベースとみなし、ユーザは知識ベースに問い合わせを送り、返答を受け取っていると解釈することもできる。（実際、データベースの検索・操作タスクをはじめ、コンピュータを用いた多くのタスクは知識ベース検索・操作タスクとして形式化できる。ほとんどのデータのブラウジングや問い合わせはデータベースの検索としてとらえられるし、予約業務も一種のデータベースの検索・操作タスクである。さまざまな装置の制御も装置をオブジェクトと考えれば、状態の観測をオブジェクトデータベースからのデータの取りだし、制御をデータベースの操作とみなすことができる [KQML manual]。）

一方、ソフトウエアが主導権を握る場合、ソフトウエア側で状況判断を行い、適切なタイミングでユーザに質問や指示を行う必要がある（図 2.2）。

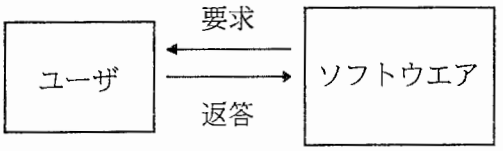


図 2.2

この図式ではユーザ側を知識ベースとみなすこともできるが、ソフトウエア側は主導権をもって働かなければならないので単なる知識ベースとみなすことはできない。

以上のことから、主導権が交代するような対話（Mixed Initiative Dialogue）を行う対話システムは、知識ベースに加えて自律的にユーザに対して質問や要求を発信するモジュールを持つものとしてモデル化できる。このようなモジュールを「対話タスクハンドラ」と呼ぶと、対話システムは次のような抽象的構造を持つことになる（図 2.3）。対話タスクハンドラは例えば特定の手続きに沿って対話を実行したりする。

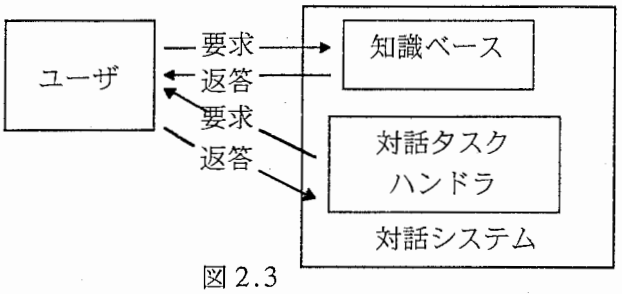


図 2.3

ここで、主導権の交代の前後で談話が関連性を保つためには、対話タスクハンドラはユーザが主導権を持つインタラクションをモニタしている必要がある。さらに、対話タスクハンドラも知識ベースが持つ情報を利用できることが望ましい。従って図 2.3 は次のように修正される（図 2.4）。

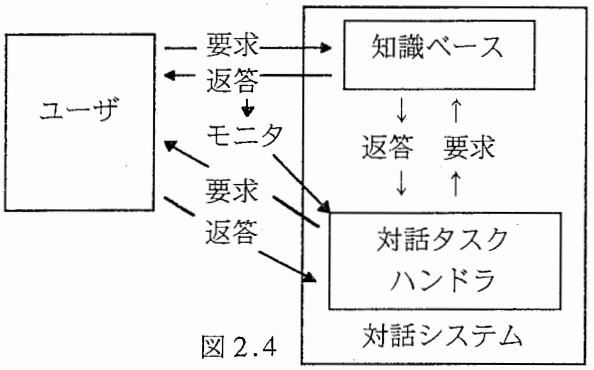


図 2.4

この図は主導権の交代という観点からの対話システムの構造のモデル化になっている。

## 2.2 自然言語によるインタラクションとアーキテクチャ

2.1 では対話システムの主導権の交代という観点からの対話システムの構造のモデル化を行ったが、ここではさらに自然言語によるインタラクションについて検討する。この小論は音声対話システムの設計を論じているのであるから、ユーザと対話システムの間のコミュニケーションには音声言語が用いられる（ＧＵＩについては後述する）。一方、2.1 で仮定された知識ベースはかならずしも音声言語を理解しない。従って、ユーザと知識ベースとの間には何らかのインタフェイス（翻訳装置）が必要になる。そこで、音声言語処理を考慮したモジュール分割は例えば次のようになる（図 2.5）。

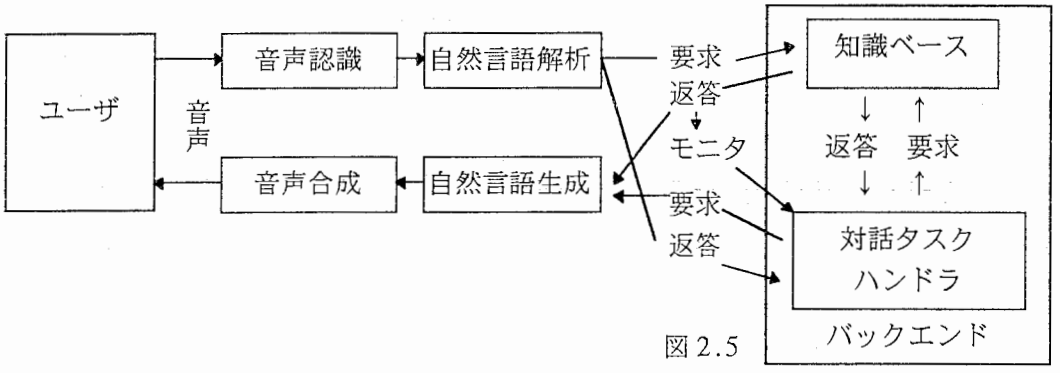


図 2.5

ここで、音声認識モジュールは、音声入力から形態素列や（部分）構文解析結果を出力する。自然言語解析モジュールは形態素列やパーザからの出力を元に意味・談話構造を作成する。自然言語生成モジュールはシステム内部で用いられる言語の表現を自然言語の表層構造（形態素列など）に変換する。音声合成は自然言語の表層構造（形態素列など）を音響系列に変換する。

自然言語解析に際して文脈情報が多義性の解決などの役に立つことがある。従って自然言語解析はシステム側の発話もモニタできることが望ましい。さらに、自然言語解析機能が多義性解消のた

めにユーザに対し自ら質問を行うことも考えられる。また、知識ベースなどへの問い合わせ言語の形式と、自然言語解析・生成モジュールが扱う自然言語の意味表現の間にはかなりのギャップがあることがある。こうしたことから、自然言語解析・生成モジュールと図 2.5 で示されるシステムの間にインタフェイスをとるモジュール（「自然言語インタフェイス」と呼ぶ）を置くことにする（図 2.6）。

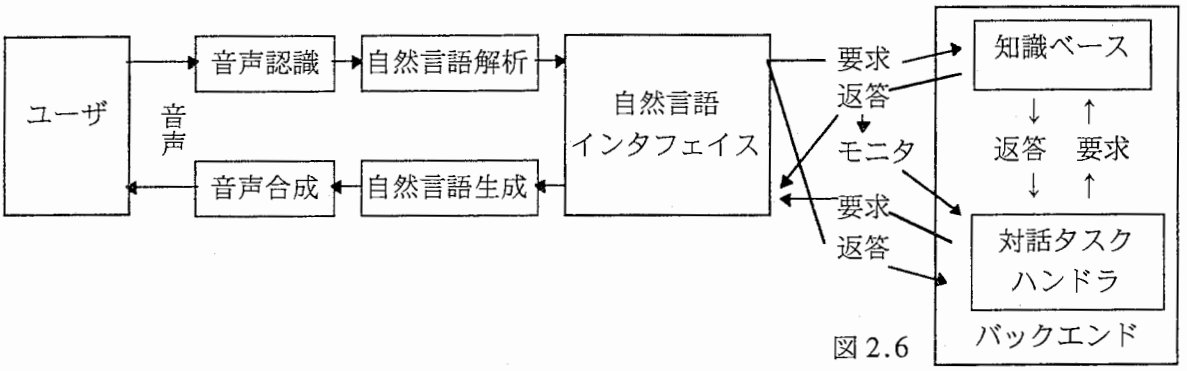


図 2.6

「自然言語インタフェイス」は、知識ベースや対話タスクハンドラから得たタスクに関する情報および文脈の情報を用いて 1）自然言語解析モジュールからの入力の多義性解消を行い、2）知識ベースや対話タスクハンドラへのメッセージを生成し、3）対話タスクハンドラからのメッセージを自然言語生成モジュールの入力形式に翻訳する一方、4）自ら多義性解消などのためのユーザへの質問を自然言語生成モジュールへ送る（第 5 章）。（機能（4）は対話の主導権を持つという対話タスクハンドラの機能を一部取り込むものであるように見える。自然言語インタフェイスの守備範囲となるような対話は本来のタスク遂行からははずれたものであり、このような対話の主導権を自然言語インタフェイスへ譲ることにより、対話タスクハンドラに言語的な知識を要求する必要がなくなる。これは入出力に近いモジュールが独自に出力を決定するという点で subsumption architecture 的 [Brooks '91] であるといえる。）

自然言語インタフェイスと図中「バックエンド」と書かれた部分で交換されるメッセージは、個別の自然言語やタスクに依存しないような形式としたい。また、ここで交換されるメッセージは、自然言語の表層的な表現に関する情報を捨象し、できるだけ純粋に意味レベルの情報を運ぶものとしたい[1]。これらの要求は次のような理由によっている。

1) 自然言語処理部をタスク独立に、バックエンド部を自然言語独立にすることにより、システムの設計上のモジュラリティを高めることができる。特に、バックエンド部の設計から個別言語に関する知識を極力排除し、自然言語処理部の設計はタスクに依存しないようなものにしたい。
2) バックエンド中の知識ベース（複数かもしれない）への直接的なインタフェイスをとることができる統一的な形式を用いたい。特に複数の知識ベースにアクセスする場合、知識ベースごとに異なるインタフェイス形式を考えるのは煩雑である[2]。
3) メッセージ交換を抽象化することにより理解しやすくすることができる。

バックエンド部が自然言語に依存しないとした場合の問題点として、バックエンド部がユーザへ

---

[1] バックエンドが自然言語に依存しないようにしたいといっても、まったく自然言語の表現がバックエンドに侵入しないということではない。例えば、特定の単語を含む書類の検索などの場合、バックエンドに文字列を渡すことによりタスクが遂行される。

[2] 知識ベースの統一アクセス形式として、KQML のようなエージェント通信言語を考えることができる。実装上もこうした統一形式を用いることが望ましいが、そうでない場合でも概念的なレベルで統一形式を記述目的に用いることにより議論を簡単にすることができる。

のメッセージを単純に自然言語の文字列などの形で発行できないということを挙げることができる。確かにバックエンド部からユーザへのメッセージを自然言語の文字列で発行すれば、自然言語生成過程の負担は小さくなる。しかしその場合、自然言語インタフェイスはそれらのメッセージの内容を（自然言語解析モジュールで解析してもらわない限り）文脈情報として利用できなくなるという問題が発生する。いずれにせよ、本レポートではバックエンド部が自然言語に依存しないと仮定した上での諸問題を議論することにする。

自然言語インタフェイスが、ユーザからのメッセージとユーザへのメッセージを扱う単一のモジュールとしているのは、次のような理由による。

1) ユーザからのメッセージの多義性解析（☞第５章）および、自然言語の生成における表現の選択（☞第６章）において文脈情報が役に立つため、バックエンド部からユーザへのメッセージも同時に管理することが望ましい。
2) 自然言語に依存する表現を自然言語に依存しない表現に変換するためのデータは、逆に自然言語に依存しない表現を自然言語に依存する表現に変換するためにも用いることができる。
3) ユーザからのメッセージの多義性解析において、システム内の情報で解決できないような多義性はユーザに対して質問を行うことにより解決を図ることができる。この際、自然言語の生成が必要になる。

むろん、これらの理由は絶対的なものではなく、自然言語インタフェイスを解析モジュールと生成モジュールに分割することは可能であろう（実際、第５章では自然言語インタフェイスの解析面を、第６章では生成面を扱う）。

## 2.3 ＧＵＩと「アプリケーション」

ＧＵＩを含めたモジュール構成は次のようになる（図 2.7）。ここで、ユーザがＧＵＩの表示するグラフィックオブジェクトの操作を音声言語を介して行いたい場合、ＧＵＩもまた知識ベースとして抽象化すればよい（例えば、ＧＵＩに対してグラフィックオブジェクトの属性を問い合わせたり、オブジェクトの操作を指示したりすることができる）。

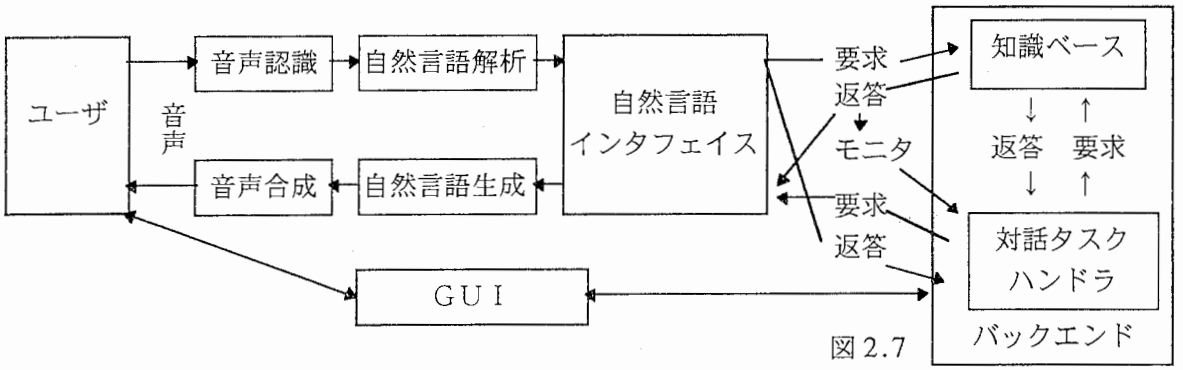


図 2.7

もっとも、ＧＵＩが表示するグラフィックオブジェクトは通常何らかのアプリケーションプログラムが使用するオブジェクトを表現しているので、グラフィックオブジェクトはむしろアプリケーションプログラムに属していると考えるのが自然かもしれない。こうしたグラフィックオブジェクトを（ＧＵＩを介して）表示するアプリケーションもまた知識ベースとして抽象化できるが、以下では逆に知識ベース、対話タスクハンドラを含めて「アプリケーション」と呼ぶことにする。グラフィックオブジェクトを表示しＧＵＩを介してユーザとインタラクションを行うようなソフトウエアは「知識ベース」というより「アプリケーション」と呼ぶほうが理解しやすいからである。こうして、知識ベース、ＧＵＩ、対話タスクハンドラを「アプリケーション」という形でまとめてしまうと、図 2.7 は次のように単純化できる（図 2.8）（なお、アプリケーションや知識ベースは今まで１

つしか図に表示していないが、実際は１つの対話システム中に複数あるかもしれない。また、それらのアプリケーションや知識ベースはネットワーク上に分散しているかもしれない）。

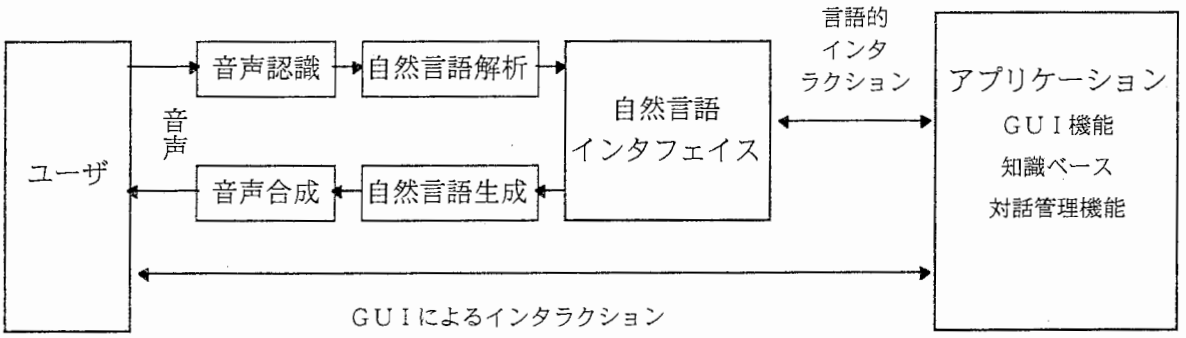


図 2.8

第２章の文献

[KQML Manual] The DARPA Knowledge Sharing Initiative External Interfaces Working Group: "Specification of the KQML," http://www.cs.umbc.edu/kqml/kqmlspec/spec.html (1993).

[Brooks '91] R.A. Brooks: Intelligence without representation," *Artificial Intelligence*, Vol.47, pp.139-159 (1991).

# 第3章　タスク

この章では、定型的音声対話システムがユーザとの対話によって行うタスクについてどのようなものがあるかを検討する。システム設計上必要なタスク分析では、システムが扱うタスクをサブタスクの組み合わせとして表現するので、音声対話システムの扱うサブタスクにはどんなものがあるかを知っておくことは重要である。以下、検討に当たって主導権の問題にも注意を払う。

## 3.1　メタコミュニケーションタスク

メタコミュニケーションタスクとはコミュニケーションを確実に行うためのタスクであり、例えば聞き取れなかった部分の聞き返しや、聞き返しに対する応答などを含む。また、対話的多義性解消や相手の発話が自分の持つ信念と矛盾する場合に行う確認の発話などもメタコミュニケーションタスクである。

メタコミュニケーションタスクは、ユーザと対話システムのどちらかの主導により遂行される。対話システムは、ユーザの発話が聞き取れなかったり、多義性を含んでいたり、自らの持つ信念と矛盾していたりした場合、主導権を持ってユーザに確認の質問を行う。ユーザも、対話システムの発話が聞き取れなかったり、多義性を含んでいたり、自らの信念と矛盾していたりした場合、主導権を持って確認の質問を行う。対話システムはユーザが主導権を持って発した質問に対し、適切に応答できなければならない（応答もメタコミュニケーションタスクの一部である）。

第2章のスキーマでは、システムによる確認やユーザからの確認への応答は、多くの場合「自然言語インタフェイス」（図2.6）によって遂行される。自然言語インタフェイスは言語解析結果をバックエンドのアプリケーションに対する問い合わせなどに変換する機能を持つが、変換がうまくいかない場合、言語生成機能を介してユーザに確認の質問を送り出す機能を持つ。一方、ユーザの聞き返しなどの質問に対しては、直前の対話システムの発話情報を保持することにより適切な応答を行うことができる（☞6.4節）。

## 3.2　タスク操作タスク

タスクの立ち上げや中止などのタスク操作タスクでは、通常ユーザが主導権を持つ。ただし、オンラインヘルプなどのアプリケーションから別のアプリケーションを立ち上げる場合、対話システム側が主導権を持ってユーザを誘導することもありうる。

ユーザのアプリケーションプログラムの立ち上げ、中断（suspend）あるいは終了（exit）に関する指示はＧＵＩやコマンドインタフェイス、音声言語インタフェイスを介してＯＳ（シェル）に伝えられる。各ソフトウエアモジュール内のサブタスクの立ち上げ、中断あるいは終了は、ユーザの指示によるか、あるいは各モジュールが必要に応じて行う。

## 3.3　知識ベース関連タスク

知識ベースがユーザの指示に基づいて行うタスクをいくつか挙げる。これらのタスクは専らユーザ主導で遂行される。

・オブジェクトの検索・同定

　知識ベース中のオブジェクトを属性、あるいは他のオブジェクトの関係によって指定する。

・情報提示

　指定されたオブジェクトの属性あるいは他のオブジェクトの関係をユーザの要求に応じて提示する。

・オブジェクトの操作

知識ベース中のオブジェクトの属性の変更、オブジェクトの削除などを行う。

・オブジェクトの生成

指定されたクラスと属性を持つ新規オブジェクトを作成する。生成されたオブジェクトは単に外部に出力されるかあるいは知識ベース中に追加される。

### 3.4 手続き的情報取得

手続き的情報取得は多くの場合対話システム主導で遂行されるタスクであり、しばしばより大きなタスクのサブタスクとして出現する。

・対話的スロット充填

例えば検索あるいは新規作成されるオブジェクトの種類（クラス）がわかっている場合、ソフトウエアはユーザに質問をし、応答を得ることによってオブジェクトの属性値の充填を行うことができる。

・対話的決定

ユーザから情報を得て行う決定（診断・推定）問題では、ソフトウエアはユーザに質問をし、応答を得ることによって情報を獲得することができる。

・不定個数情報取得

ある終了条件が満たされるまで不定個数件数の情報をユーザに要求するタスク。プランニングのための条件指定を不定個数集めるなどの場合があてはまる。

### 3.5 教示（instruction）

対話システムが、手続きに沿ってユーザに指示を与えるようなタスクである。教示タスクにおいては、対話システムが主導権を発揮する場合が多い。対話システムは、指示において主導権を持つばかりでなく、どんな指示をどのようなタイミングで出すかを決定するための情報を求めてユーザに質問をするという点でも主導権を発揮する。もちろん、ソフトウエアの教示に対しユーザが質問をすることもあるので、主導権は常に対話システム側にあるというわけではない。

### 3.6 応用タスク

次に３つの応用タスクを示す。これらは上で述べたサブタスクの組み合わせにより実現される。

・予約・登録／変更業務

予約・登録業務は多くの場合、対話的スロット充填とデータベースへの問い合わせ・操作指示から構成される。システムは、タスク遂行に必要な情報を対話的スロット充填タスクによりユーザに問い合わせ、ユーザの要求とデータベース中の情報が一定の制約を満たす場合に、データベースの更新を行う。さらに注文業務（オンラインショッピング）などの場合は、不定個数のアイテムの注文を受けるために不定個数情報取得タスクが必要になる。

・オンラインヘルプ

オンラインヘルプは、ユーザに何をしたいのかということを入力してもらい、関連する情報をヘルプデータベースから引き出すという一種の検索問題と考えることができる（この点で、オンライン百科のようなオンラインリファレンスも同様に抽象化できる）。対話的ヘルプでは、ユーザが行いたいことを、ソフトウエア側がある程度主導権を持つことにより対話的に同定していくことが求められる（対話的決定タスク）。また、オンラインヘルプは取り出された情報を単に提示する（オンラインマニュアル）だけでなく、ユーザに順を追って教示を行うかもしれない。

・プランニング

プランニングはいくつかの条件を与えて適切な解を生成させるような知識ベース検索・操作タスクとして形式化できる。知識ベース検索・操作タスクでは通常ユーザが主導権を持って遂行するが、プランニングタスクでは、条件を指定するために対話的なやりとりを必要とすることが多く、ユーザが主導権を常に持ち続けるということは少ない［Rich & Sidner '97］。プランニングのための条件は通常不定数個与えられるので、上記の不定個数情報取得タスクがサブタスクとして与えられる。さらに、プランニング中には条件を指定するためのスロット充填タスクやユーザの好みなどを決定するための対話的決定タスクがしばしば埋め込まれる。対話的なプランニングでは各条件は与えられた時点で成立するかどうかチェックする（その時点でそのプランは成立しなくなる）ことができることが望ましい。

第5章の文献

[Rich&Sidner '97] C. Rich & C. L. Sidner: "COLLAGEN: When Agents Collaborate with People" in *Readings in Agents*, M. Huhns & M. Singh eds. Morgan Kaufmann (1997).
Also see "COLLAGEN: A Toolkit for Building Collaborative Interface Agents" TR-97-21 (1997)　URL="http://www.merl.com/projects/collagen/index.html"

# 第４章　対話タスクの制御

この章では対話タスクの制御を行う「対話タスクハンドラ」（☞図 2.4）の機能を記述するための一方式を提案する。第３章で論じた定型的音声対話システムのサブタスクのうち、ここで扱うサブタスクは「手続き的情報取得」と「タスク操作タスク」である。特に「手続き的情報取得」に関して、仕様を記述することによりタスクごとに新たな手続きを記述する必要がないようなタスク制御方式を提案する。なお、第３章で述べたサブタスクのうち「メタコミュニケーションタスク」は第５章で述べる「自然言語インタフェイス」が扱い、「知識ベース関連タスク」は「自然言語インタフェイス」と知識ベースの直接的なメッセージ交換により実現できるので、ここでの検討対象とはしない。

## 4.1　機能要求

「対話タスクハンドラ」の機能の記述言語には以下のようなことが望まれる。

・逐次的な手続きの表現

　例えば教示において、対話タスクハンドラはいくつかのアクションを逐次的に行う必要がある。

・対話タスクハンドラの内部状態に応じたアクション（サブタスク）の実行の記述

　ユーザからの入力によって変化した内部状態に応じたアクションを記述することにより、インタラクションを記述することができる。

・さまざまなオブジェクトの表現

　対話タスクハンドラはさまざまなタスクで扱うオブジェクトの情報をやりとりする。

・典型的なサブタスク（スロット充填など）の記述には手続きを書かずにすむこと

　タスクごとに同じような手続きを書く無駄をはぶくために、定型的なサブタスクについては、仕様のみを与え、自動的に解釈、実行を行わせるようにしたい。

・タスクの構造が見てわかりやすいこと

以下で提案する手法では、対話タスクハンドラの機能を「タスク実行オブジェクト」というモジュール内オブジェクトおよびプロダクション規則により記述する。「タスク実行オブジェクト」は上記の要求項目のいくつかを満たすために導入される。実際、「タスク実行オブジェクト」は定型的なサブタスク（主に手続き的情報取得）の仕様を記述し、プロダクション規則は非定型的なタスクを担当することになる。

## 4.2　タスク実行オブジェクト

タスク実行オブジェクトは手続き的情報取得サブタスク（スロット充填、ケース分岐、不定個数情報収集［第３章参照］）とその遂行状況を表現する。対話タスクハンドラは「タスク実行オブジェクト」を解釈すると同時に、その状態をチェックしてアクションを決定したり、状態を書き換えたりする。タスク実行オブジェクトは入れ子のフレーム構造であり、その上位クラスオブジェクトは次のような属性を持つタスクオブジェクトである。

・活性値（述語式）
・タスク名
・上位クラス

また、タスクオブジェクトは次のようなオプションの属性を持つこともできる。

・生成された時刻
・最後に参照された時刻
・変更特権を持つユーザの集合

・参照特権を持つユーザの集合
・定義変更特権を持つユーザの集合

タスク実行オブジェクトはタスクオブジェクトの属性に加えて次のような属性を持つ。
・タスクリスト
・ソートキー（オプション）
・ソート関数（オプション）

タスクリストの要素はタスクオブジェクトである。タスクオブジェクトには次のような種類がある。
・タスク実行オブジェクト
・属性充填オブジェクト
・単純実行オブジェクト
・ケース分岐オブジェクト
・不定個数情報収集オブジェクト

対話タスクハンドラはタスクリストを調べ、活性値が真であるようなタスクオブジェクトについて該当するタスクの実行を試みる。この際、リストの上位にあるタスクが優先される（オプション属性のソートキーやソート関数は、この順序をダイナミックに変更するためのものである）。タスク実行オブジェクトをタスクリストに指定できるので、タスク実行オブジェクトは入れ子をなすことができる。タスクリストのすべてのタスクオブジェクトの活性値が偽になると、そのタスクリストを持つタスク実行オブジェクトの活性値も（デフォルトで）偽にセットされる。以下、属性充填オブジェクトから順に、個々のタスクオブジェクトについて説明する。

・属性充填オブジェクト

属性充填オブジェクトはスロット充填タスク用のタスクオブジェクトであり、次のような属性を持つクラス定義を持つ。
・クラス名＝「属性充填オブジェクト」
・上位クラス＝タスクオブジェクト
・属性名
・属性値（または属性関数）
・値が定められているかどうか（真理値）
・属性値のクラス（インスタンスのリストまたはクラス名）
・データ源（リスト）
・必須属性になるための条件（述語式）

属性充填オブジェクトの活性値は、値が決まっている場合は常に偽である。対話タスクハンドラは、活性値が真であるような属性充填オブジェクトを見つけると、そのオブジェクトに書かれているデータ源に対し、問い合わせを発行する。対話タスクハンドラは、外部から特定の属性充填オブジェクトの値を指定する有効な入力があった場合に、その値を変更する。これらの動作は自動的に行われ、ユーザが手続きを指定する必要はない。また、属性値の充填はかならずしもタスクリストの順に行われるわけではない。ユーザへの値充填の要求はタスクリストの順に行われるが、充填は外部から属性値を指定するメッセージがあった際に行われる。

属性充填オブジェクトは、次のようなオプションの属性を持つこともできる。
・デフォルト属性値（関数）
・他の属性値（他のオブジェクトの属性も含む）との間になりたつ関係（述語式）
・属性値変更の確認の要不要（真理値）

・デフォルト値使用の確認の要不要（真理値）
・同報リスト
・マスタＤＢ

属性値変更の確認が必要とされている場合、対話タスクハンドラは属性値を変更する要求に際してユーザに確認を行う（属性値間の関係が設定されている場合、属性値が別の属性の値により変更されることもあるが、このような場合にも確認を行う）。デフォルト値使用の確認が必要とされている場合は、デフォルト値を指定してよいかどうかの確認を行う。値の変更が他の属性値との間になりたつ関係に指定された制約に違反する場合、対話タスクハンドラはユーザに再指定を促す。同報リストが指定されている場合、属性充填オブジェクトの値の変更をリストで指定されているモジュールに通報する。マスタＤＢが指定されている場合、オブジェクトの値の変更は随時マスタＤＢに連絡し、マスタＤＢのオブジェクトの値の（別のモジュールによる）変更は、随時タスク実行オブジェクトに反映する（変更のタイミングを管理するために、変更時刻属性を用いることができる）[3]。

・単純実行オブジェクト

単純実行オブジェクトは属性として何らかのアクションの記述を持っている。対話タスクハンドラは、活性値が真であるような単純実行オブジェクトを見つけると、指定されたアクションを実行し、活性値を偽にする。単純実行オブジェクトは、例えば外部へのモジュールの送出を行うなどの、逐次的手続きを記述するために用いられる。単純実行オブジェクトは次のような属性を持つクラス定義を持つ。

・クラス名＝「単純実行オブジェクト」
・上位クラス＝タスクオブジェクト
・アクション

・ケース分岐オブジェクト

ケース分岐オブジェクトは次のような属性を持つクラス定義を持つ。

・クラス名＝「ケース分岐オブジェクト」
・上位クラス＝タスクオブジェクト
・分岐キー　属性充填オブジェクト
・分岐リスト　　{[値リスト <<値のリスト>>
　　タスクリスト <<タスクリスト>>]
　　....
　}

対話タスクハンドラは、分岐キーで指定される属性充填オブジェクトの属性値を評価し、分岐リスト中のその値を含む値リストに対応するタスクリストを評価する。このことにより、タスク実行オブジェクトは状況に応じての拡大が可能になる。属性充填オブジェクトの値が不定であれば、まずその値を外部への問い合わせにより解決する。第３章で述べた、対話的決定タスクはケース分岐オブジェクトの使用により実現可能である。

・不定個数情報収集オブジェクト

不定個数情報収集オブジェクトは次のような属性を持つクラス定義を持つ。

---

[3] ここで提案した属性充填オブジェクトは、ある種のデータベース管理システムにおけるデータ更新用画面（フォーム）の記述に似ている。

・クラス名＝「不定個数情報収集オブジェクト」
・上位クラス＝タスクオブジェクト
・終了条件　　述語式
・オブジェクト条件 述語式
・インスタンスリスト　　収集されたオブジェクトのリスト

対話タスクハンドラは、終了条件が満たされるまで、オブジェクト条件に合致するオブジェクトを外部データ源（ユーザ等）から取りだすことを試みる。（不定個数情報収集オブジェクトは、従ってデータベースから条件に合致するものを取り出すタスクにも用いることができる。）

### 4.3 規則記述

4.1 のタスクオブジェクトで記述できないような非定型的動作は、別に記述される規則群により実行される。これらの規則は、タスク実行オブジェクトの状態に依存して発火する。規則の条件部にはタスク実行オブジェクトの属性値に関する条件が与えられ、アクション部には（パス指定により）指定されたタスク実行オブジェクトの属性値の変更や、オブジェクトの追加や削除、さらに外部へのメッセージの出力が指定される。規則の競合は規則に与えられたスコアなどにより解決される。

プロダクション規則の記述および実行メカニズムは一般的な事柄なのでここではこれ以上追及しない（☞［石田 '96］[Arakawa '97]）。また、プロダクション規則（条件→アクション規則）は特殊なプロダクションシステムを使わずとも一般のプログラミング言語で記述可能である。

### 4.4 例題：タスク実行準備

音声対話システムを起動する際に必要になるような手続きのいくつかを例題として対話タスクハンドラに行わせてみよう。ここでの手続きは次のようなタスクからなるとする。

画面モード、音声モード使用の確認（画面と音声の両方のモードによる）
メインタスク種別判定

最初の段階で、タスク実行オブジェクトは図 4.1（次頁；素性構造で記述）のようになっている。対話タスクハンドラはタスクリスト中のタスクオブジェクトを順に実行する。属性充填オブジェクトについては、活性値がＴかつ値定不定がＦなので、指定されているデータ源「ユーザ」に向けて値の問い合わせを行う。ユーザからの返事、例えばエージェント通信言語 (KQML) の文

```
(tell    :language 属性リスト
         :content (属性問い合わせ :属性名 画面モード :属性値 T)
         :sender ユーザ
         :receiver 対話タスクハンドラ
         :reply_with 4436)
```

によって得られた情報により、属性値が決定され、活性値は偽になる。本タスクの種類を選ぶケース分岐では、分岐キーで指定された属性充填オブジェクトの属性値（ホテル検索・更新または旅行プランニング）をまず求め、その値によってタスクリスト中のタスクを名前に持つタスクオブジェクトを呼びだして実行する。ここではオブジェクトの定義中にマスタＤＢが指定されているので、オブジェクトの値の変更は随時マスタＤＢに連絡し、マスタＤＢのオブジェクトの値のＧＵＩなどによる変更は、随時タスク実行オブジェクトに反映する。

```
[タスククラス タスク実行準備
 上位クラス   タスクオブジェクト
 活性値       T
 タスクリスト {[ 活性値          T
                 タスククラス    属性充填オブジェクト
                 属性名          画面モード
                 属性値          NIL
                 デフォルト値    T
                 属性値クラス    真理値
                 必須            T
                 値定不定        F
                 データ源        ユーザ
                 マスタDB        MDB
                 デフォルト確認 T]
               [ 活性値          T
                 タスククラス    属性充填オブジェクト
                 属性名          音声モード
                 属性値          NIL
                 デフォルト値    T
                 属性値クラス    真理値
                 必須            T
                 値定不定        F
                 データ源        ユーザ
                 マスタDB        MDB
                 デフォルト確認 T]
               [ 活性値          T
                 タスククラス    ケース分岐オブジェクト
                 タスク名        本タスク種別取得
                 分岐キー        [ 活性値        T
                                   タスククラス  属性充填オブジェクト
                                   属性名        本タスク種別
                                   属性値        NIL
                                   属性値クラス  {ホテル検索・更新 旅行プランニング}
                                   必須          T
                                   値定不定      F
                                   データ源      ユーザ
                                   マスタDB      MDB]
                 分岐リスト      {[値リスト {ホテル検索・更新}
                                   タスクリスト {ホテル検索・更新}]
                                 [値リスト {旅行プランニング}
                                   タスクリスト {旅行プランニング}]
      ]}
]
```

図 4.1

## 4.5 特殊処理（中止（cancel）、終了）

対話タスクハンドラは、サブタスクを指定して中止することを求められれば、そのサブタスクを中止して、中止モードにはいり、ユーザの指示を待つことになる。サブタスクが指定されない場合、現在実行している最も低いレベルのタスクを中止して、ユーザの指示を待つ。

終了の場合、必要があれば関連するアプリケーションに終了のメッセージを送る。

## 4.6 確認とエラーリカバリ

属性充填オブジェクトの項で述べたように、対話タスクハンドラは、属性充填オブジェクトが「変更の確認が必要」としている場合、値の変更に対して確認を行う（上記の仕様では確認のタイミングの指定ができないが、できるように仕様を拡張してもよい）。また、属性充填オブジェクトが「デ

フォルト値使用の確認が必要」としている場合、やはり確認のメッセージを生成する。

対話タスクハンドラは少なくともタスク実行オブジェクトの解釈に関する限り、必須とされている情報が得られるまで質問を繰り返す。これはあまり知的なふるまいではない。これを避けるための方法の１つはプロダクション規則を用いることであるが、もう１つの方法として、属性充填オブジェクトに行った質問の回数に関する属性と、質問が一定の回数を超えた場合にとるべきアクションの指定に関する仕様を追加することが考えられる。

対話タスクハンドラは、自らが行ったデータベース検索結果が失敗に終わった場合、ユーザへの応答を生成しなければならない[4]。この処理は、タスク実行オブジェクトで記述することができる。例えば、ユーザが希望する条件に合致するホテルの部屋が見つからなかった場合の応答を考えてみよう。まず、ホテルの部屋の検索は、不定個数情報収集オブジェクトと外部データベースとの組み合わせで実現できる。検索結果がない（不定個数情報収集オブジェクトのサイズ属性が０）ということは、ケース分岐オブジェクトを用いて判定し、単純タスクオブジェクトによりメッセージ（例えば `(for_all (x) (⇒ (single x) (occupied x)))` ≒「シングルはただいま満室になっております」など）を生成するようにすればよい。

第４章の文献

[Arakawa '97] N. Arakawa & T. Morimoto: "Semantic and Discourse Processing with a Feature Structure-based Production System," in *Proceedings of NLPRS'97* (1997).

[石田 '96] 石田亨：*プロダクションシステムの発展*、朝倉ＡＩらいぶらり５，朝倉書店 (1996).

[4] ここでは、タスクに関するオントロジに違反するようなユーザからのメッセージは、対話タスクハンドラにやってこないものと仮定している。というのは、第５章で述べる「自然言語インタフェイス」がオントロジに違反するユーザからのメッセージへの対処を行うからである。

# 第５章　自然言語インタフェイスによる言語解析

## 5.1 はじめに

この章では、第２章で導入した自然言語インタフェイスの、自然言語に依存する形式のメッセージを自然言語に依存しない形式に多義性を解消しつつ変換する機能を実現する方法について検討を行う。より具体的には、上記の機能を実現するために、アプリケーション（☞第２章）が持つオブジェクトの知識を利用するような仕組みを提案する。このような仕組みを考える理由は次のようなものである。

1）自然言語解析における多義性解決や欠損情報の補完には言及されるオブジェクトの知識が役立つ。

2）アプリケーションが持つオブジェクトに関する知識を自然言語解析で用いることができれば、開発効率の観点から望ましい。

以下、簡単のために自然言語解析モジュールの出力と、自然言語生成モジュールの入力は同じ形式を持つものとし、その形式を「意味表現」と呼ぶことにする。「意味表現」は（自然言語に依存しない論理表現のようなものではなく）自然言語表現に対応する述語表現を持ち、多義性を表示する。

以下の節では、まずアプリケーションが使用するオブジェクトに関する知識について述べる（5.2）。さらに、自然言語インタフェイスとアプリケーションとの通信に用いられるメッセージについて (5.3)、および自然言語インタフェイスが自然言語解析・生成モジュールとの通信に用いられる意味表現について (5.4) 述べる。こうして準備ができたところで、自然言語インタフェイスの機能であるところの意味表現とアプリケーションとの間で用いられるメッセージの間の変換について説明を行う (5.5)。そこでは、アプリケーションが使用するオブジェクトに関する知識の他に、述語・属性対応表という変換知識を用いることが提案される。 (5.6) では自然言語インタフェイスが必要とするデータについて触れる。

## 5.2 アプリケーションオントロジ

アプリケーションオントロジとは、アプリケーションが扱うタスクが使用するオブジェクトのクラス定義やオブジェクトの間でなりたつべき関係の記述の全体を指すものとしよう。ただし、この定義においてアプリケーションオントロジはタスクごとに定義できるから、ここでいうアプリケーションオントロジは正確にはアプリケーションタスクオントロジと呼ぶべきである。

自然言語の解析において、アプリケーションオントロジは多義性解決や情報の補完のために役立つ。自然言語の解析結果は多義性を含むが、その解釈がアプリケーションオントロジの制約を満たさないような解析結果（候補）は棄却できるからである。また、オントロジ中の定義により属性値が与えられたときに属性名を求めるというようなことが可能になる。この章での提案では、アプリケーションは自らのタスクが使用するオントロジに関して自然言語インタフェイスからの問い合わせに返答を行うことになる。

現在の多くのソフトウエアはオブジェクト指向設計がなされており、設計時に行われるオブジェクト（クラス）定義は、そのアプリケーションのオントロジ（の一部）として用いることができる。同様にオブジェクト指向データベースで用いられるクラスやインスタンスの情報もアプリケーションオントロジにとりこむことができる。上で述べたようにアプリケーションオントロジは自然言語インタフェイスの処理のために有用であるが、オントロジ知識はアプリケーション設計に用いたものを利用することが、開発の無駄をはぶきかつ知識の整合性を保つうえで望ましい。

オントロジにおける最も重要な記述としてオブジェクトのクラス定義を挙げることができる。オ

ブジェクトのクラス定義は次の事項を含む。

上位クラスリスト

属性リスト

属性名

属性値のクラス、選択肢

属性値に関する制約（述語形式）

オントロジは述語と属性の間の関係定義、あるいはある述語が実は属性を表現するというような知識を含みうる。例えばオブジェクトクラス PAINT が属性 COLOR を持つということを（Ontolingua 風[5] に）

(has-slot-value 'PAINT 'COLOR ?x)

という形で記述しておけば、PAINT クラスのオブジェクトが属性 COLOR を持つことがわかる。

### 5.3 アプリケーションとの間のメッセージ

発話行為論によれば、メッセージは一般に陳述、質疑、命令などのメッセージタイプ（発話内行為タイプ）を表す要素と命題内容を表す要素に分解して考えることができる。ここでもメッセージを発話内行為タイプと命題内容および付随的情報に分けて考えることにする。

#### 5.3.1 発話内行為タイプ

自然言語インタフェイスとアプリケーション間のメッセージは、何を意図して発せられたものかを表す発話内行為タイプの情報を明示していることが望ましい。それによって、メッセージの受け手が取るべきアクションを決定しやすくなるからである。自然言語の発話行為論ではさまざまな発話内行為タイプのセットが提案されている [Allen&Core '97][友清 '94]。ソフトウエア間のメッセージ交換のために提案されているエージェント通信言語で用いる発話内行為タイプについてもいくつかの提案がある [KQML Manual][Shoham'93]。こうした提案は細部において異なる。自然言語インタフェイスが自然言語解析・生成モジュールから受け取ったり送ったりする意味表現もまた発話内行為タイプの情報を含むべきであるが、そこでの発話内行為タイプ表現と、アプリケーションとの間のメッセージで用いる発話内行為タイプ表現が異なる場合、自然言語インタフェイスはそれらの変換を行わなければならない。

#### 5.3.2 命題内容

命題内容は一般的に述語論理式で表記することができる。アプリケーションで扱うさまざまなオブジェクトは属性を持つが、これらは2項述語で表現できる。例えば、オブジェクト c が値 a の属性 b を持つということは、述語 (b c a) で表される。ただし前述したように、b が c の属性であるという情報はアプリケーションオントロジに記載しておかなければならない。述語表現の個体定数としては、例えばシステム中のオブジェクトをユニークに指定するパス名を用いることができる。述語論理式は、例えば高階の論理述語を用いて (exists x (and (ABC x) (DEF y))) などのように表記することができる（Ref. KIF (Knowledge Interface Format) [KIF Manual]）。不確定な命題は例えば次のような述語により表現される。

```
(uncertain P)       ; 命題 P は不確かである。
(probability P p) ; 命題 P の確率は p である。
```

---

[5] オントロジは一定のオントロジ記述言語を用いて記述されるはずである。この例では Ontolingua [Gruber'92] というオントロジ記述言語を用いている。

### 5.3.3 その他の情報

自然言語インタフェイスとアプリケーションで交わされるメッセージは、発話内行為タイプと命題内容の他にも、以下のような情報を運ぶ必要があると考えられる。

・メッセージ発信者

ユーザのメッセージの翻訳なのか、自然言語インタフェイスが問い合わせなどの目的で発信したメッセージなのかを判別するために必要である。

・メッセージ受信者

アプリケーションが自然言語インタフェイスにメッセージを翻訳してユーザへ送って欲しいのか、単に自然言語インタフェイスに情報を提供しているのかを判別するために必要である。

・メッセージＩＤ

問い合わせを行う際に、問い合わせと返答のマッチングを行うために必要である。

・オントロジの種類

後述するように、自然言語インタフェイスはタスクオントロジに依存するデータを使用する。アプリケーション内でオントロジを切り替える場合、あるいはアプリケーションを切り替える場合に、アプリケーションが使用するオントロジを指定する必要がある。

・メッセージの開始・終了時間

アプリケーションのＧＵＩなどの他のモダリティとの同期を取るために必要である。

参考）ＫＱＭＬ [Ref. KQML Manual]

DARPAの知識共有イニシアティブワーキンググループが提案したエージェント通信言語KQMLのメッセージ（performative）は次の一般的構造を持つ。

(performative-name key-pair*)

ここで、performative-name は tell, achieve など、メッセージの種類（発話内行為タイプ）を表す語であり、多くは知識ベースの問い合わせ／操作のためのもの（ask-if, delete, insert など）である。

一方、key-pair は key-word と何らかの表現（Lisp 式）の空白で区切られた組であり、メッセージの内容と付随情報を表す。Key-word は例えば :content, :sender など ":" で始まる語である。上で述べた項目とキーワードとの関連は次のとおり：

| | |
|---|---|
| :content | 命題内容（:language で指定される言語によって記述される） |
| :sender | メッセージ発信者 |
| :receiver | メッセージ受信者 |
| :reply-with | メッセージＩＤ（返答要求） |
| :reply-to | メッセージＩＤ（返答） |
| :ontology | オントロジ |
| :language | :content を記述する言語 |

## 5.4 意味表現

以下では、自然言語インタフェイスが自然言語解析・生成モジュールとのインタフェイスに用いる表現形式を「意味表現」と呼ぶ。

各発話の意味表現は次のような情報を持つ。

話者

発話開始・終了時間[6]

発話内行為タイプ

意味内容（命題内容に加え、法（mood）、態（voice）、時制（tense/aspect）、終助詞などの文法的情報を含む）

ここで、意味表現は、発話内行為タイプや意味内容に関して多義性を表示できる必要があり、できれば多義中の各選択肢に対し、尤度値情報を表示できることが望ましい。

以下の意味表現の例では、記述のために単一化文法で用いられている素性構造を用いる（記述方法の詳細については付録を参照されたい）。なお、素性構造の使用はあくまで例示のためであり、上記のような情報を表現できる限り、実装時にはどのような形式を用いてもかまわない。

## 5.5 意味表現に対するアプリケーションへのメッセージの生成

### 5.5.1 発話内行為タイプの変換

前述したように、意味表現における発話内行為タイプのセットとアプリケーションとの間で用いられるメッセージの発話内行為タイプのセットが一致しない場合、変換が必要になる。変換結果が多義性を持つ場合、自然言語インタフェイスはユーザに確認の問い合わせを行う。

例）ＫＱＭＬをメッセージ形式に用いた場合

ＫＱＭＬではパフォーマティブの種類（発話内行為タイプの種類）としていくつかが予約されている。これらの多くは知識データベースに対するメッセージの種類を表しており、人間どうしの発話の分類のために提案された発話内行為タイプとは一致しない。次にいくつかのＫＱＭＬ予約パフォーマティブ名とより一般的な発話内行為タイプとの対応を示す。命令や依頼は、埋め込まれている述語から知識ベースへの文の削除や追加命令であると判定できれば、deleteや insert などのパフォーマティブに変換されるが、そうでなければ achieve に変換される。質問は ask-if や ask-one などに変換され、通知は tell に変換される。

achieve: 命令（order）、依頼（request）
ask-if: 質問（TF-question）
ask-one: 質問（info-request）
delete: 命令（order）、依頼（request）
tell: 通知（inform）

ＫＱＭＬをアプリケーションとの間のメッセージ交換に用いるとすると、ＫＱＭＬの予約パフォーマティブ名に対応するものがないさまざまな発話内行為タイプを持つ発話は、必要であれば発話内行為タイプ情報を含んだ tell パフォーマティブに変換される（あるいはＫＱＭＬの方言を作って新しいパフォーマティブ名を定義してもよい。また、発話内行為を tell の :content を用いて伝達してもよい。例えば、ユーザの「おはようございます」という挨拶は次のような形でアプリケーションに伝達されるかもしれない。

(tell :sender user :receiver system :language KIF :content (greet $user morning))

また、アプリケーション側で必要としない発話内行為タイプを持つ発話（あいづち（acknowledge）など）についてはメッセージを生成しないことにしてもよい。

---

[6] 指示表現とＧＵＩ上のポインティングジェスチャのタイミングの一致を調べることにより指示対象を決定するためには、より細かい時間情報が必要になるかもしれない。

5.5.2 内容表現の変換

ここでは、意味表現の意味内容表現と自然言語インタフェイスとアプリケーションの通信のためのメッセージの内容表現の間の変換について検討する。内容表現の変換においてアプリケーションが持つオブジェクトに関する知識が利用される。自然言語インタフェイスは、変換の結果がアプリケーションオントロジの制約を満たしているかどうかを、アプリケーションの知識ベースに問い合わせる。アプリケーションオントロジの制約を満たさない変換結果を棄却することにより、不適確なメッセージの生成を防ぐことや多義性の解消が可能になる。オントロジ制約を満たす変換結果が1つも残らない場合、自然言語インタフェイスはユーザにその旨を伝え、再発話を促す（☞6.4節）。

i) 述語・属性対応表を用いた変換

意味表現中に現れてくる素性（属性）名や述語名がそのままアプリケーションとの間で用いられるメッセージの述語名（属性名）として用いることができる保証はなく、一般的には変換を行わなければならない。また、意味解析中の述語表現を属性に変換するためにも述語と属性の対応付けをする必要がある。ここでは、自然言語インタフェイスは、意味表現とアプリケーションとの間で用いられるメッセージの内容表現間の述語・属性対応表を利用して、それらの間の変換を行うという方法を検討する。述語・属性対応表による変換結果（多義性を含む）のうち、アプリケーションオントロジに違反するものは棄却される。述語・属性対応表には5種類を考えることができる。

a) 述語→述語対応表

これは意味表現の述語とアプリケーションとの間で用いられるメッセージの内容表現の述語を対応付けするための表であり、述語名と引き数の対応付けが行われる。例えば次の表は意味表現での述語名 ABC がアプリケーションとの間で用いられるメッセージの内容表現での述語名 ABC' になり、第1と第2引き数が入れ替わることを表している。

| 意味表現 | 内容表現 |
|---|---|
| ABC | ABC' |
| arg1 | arg2 |
| arg2 | arg1 |

場合によっては引き数の数が減少したり、特定のタイプ（例えば発話者）のオブジェクトを指示対象に持つの引き数が付加されたりする。変換表中に述語名 ABC に関するエントリがなければ、述語 ABC はそのままアプリケーションとの間で用いられるメッセージの内容表現中の述語となる。

b) 述語→属性対応表

2項の述語は属性に写像されることもある。2項述語 ABC の片方の引き数値が、述語が埋め込まれている pred 素性を持つオブジェクトで、次のような変換表（述語→属性対応表）が定義されている場合、述語（素性構造で記述）

[reln ABC
 arg1 ?x01
 arg2 ?x02]

は属性（ABC' ?x01 ?x02）に変換される（ABC' はオントロジで属性名として定義されているものとする）。

| 意味表現 | 内容表現 |
|---|---|
| ABC | ABC' |
| arg1 | attr_holder |
| arg2 | attr_val |

c) 述語→クラス対応表

1項の述語はクラス名に写像されることもある。次のような変換表

| 述語名 | クラス名 |
|---|---|
| ABC | ABC' |

（述語→クラス対応表）が定義されている場合、意味表現中のオブジェクト表現

```
?x01:[pred ([reln ABC
            arg1 ?x01])]
```

（ABC であるような ?x01 と読む）からは（instance-of ?x01 ABC'）が生成される。

d) 属性→属性対応表

属性名と属性値の変換は次のような表を用いて行われる。

| 意味表現 | 内容表現 |
|---|---|
| ABC | ABC' |
| VAL1 | VAL1' |
| VAL2 | VAL2' |
| VAL3 | VAL3' |

自然言語の意味表現で、発話中で非明示的な情報（話者や言及された 人物の性別など）が素性として表現されることがある（例：彼＝[gender M]）。このような素性も属性とみなされ、属性→属性対応表による変換の対象となる。

e) 属性→述語対応表

属性から述語への変換は次のような表によって行われる。

| 意味表現 | 内容表現 |
|---|---|
| ABC | CDE |
| attr_holder | arg1 |
| attr_val | arg2 |

f) コプラ表現の翻訳例

論理的に多義を含むコプラを使った表現が、どのように述語・属性対応表を用いて変換されるか検討してみる。典型的なコプラ表現「AはBです。」の意味表現はA、Bが普通名詞である場合、例えば次のようになる。

```
[pred ([reln copula
        arg1 ?x01:[
             pred ([reln A
                    arg1 ?x01])]
        arg2 ?x02:[
             pred ([reln B
                    arg1 ?x02])]])]
```

・単称解釈

「AはBです。」が「Aであるようなオブジェクトが存在し、それはBである。」とい

う意味で、A, B に対する述語 A', B' あるいはクラス B' がそれぞれアプリケーションオントロジで定義されている時、述語・属性対応表を用いて例えば次のようなエージェント通信言語の内容表現を得ることができる。

(exists (?x01) (and (A' ?x01) (B' ?x01)))

「AはBです。」が「A（という記述で特定されるオブジェクトは）はBという述語に包摂される。」という意味で、A, B に対する述語 A', B' あるいはクラス B' がそれぞれアプリケーションオントロジで定義されている時、述語・属性対応表を用いて例えば次のようなエージェント通信言語の内容表現を得ることができる。

(B' $x023)

ここで、$x023 は (A' ?x01) と単一化可能なオブジェクトインスタンスであり、後述する指示対象決定プロセスにより選択される。

・全称解釈

「AはBです。」が「すべてのAはBです。」を意味する場合、述語・属性対応表を用いて例えば次のような表現を得る。

(forall (?x01) (=> (A' ?x01) (B' ?x01)))

g) 属性表現の翻訳例

属性に関わる自然言語の表現が、どのように述語・属性対応表を用いて変換されるか検討してみる。

1)「AのBはCです。」

日本語で属性は典型的にはオブジェクトAの属性Bの値Cを指定する「AのBはCです。」というような文中で表現される。

・「AのBはCである。」という主張の意味表現はA、B、Cが普通名詞である場合、次のようになる。

```
[pred ([reln copula
        arg1 ?x01:[
              pred ([reln B
                     arg1 ?x01]
                    [reln の
                     arg1 ?x01
                     arg2 ?x03:[
                           pred ([reln A
                                  arg1 ?x03])]])]
        arg2 ?x02:[
              pred ([reln C
                     arg1 ?x02])]])]
```

属性→属性対応表中で、B が属性として定義されている場合、プログラムは B をアプリケーションとの間で用いられるメッセージの内容表現中の属性 B' に、C を属性値 C'、A を述語名 A' として翻訳することを試みる。例えば翻訳結果は次のようになる。

```
(exists (?x01)
 (and (A' ?x01)
      (B' ?x01 C')))
```

この翻訳結果は、アプリケーション側で定義されるオントロジ制約に違反するならば棄却される。

・「AのBはCである。」という主張の意味表現はBが普通名詞、A、Cが固有名詞である場合、次のようになる。

```
[pred ([reln copula
        arg1 ?x01:[
              pred ([reln B
                     arg1 ?x01]
                    [reln の
                     arg1 ?x01
                     arg2 ?x03:[
                           pred ([reln name
                                  arg1 A
                                  arg2 ?x03])]])]
        arg2 ?x02:[
              pred ([reln name
                     arg1 C
                     arg2 ?x02])]])]
```

ここで、属性→属性対応表中で、B が属性として定義されている場合、プログラムは B をエージェント通信言語の内容表現中の属性 B' に、C を属性値 C' に翻訳することを試みる。例えば翻訳結果は次のようになる。

```
(exists (?x01)
 (and (name ?x01 A)
      (B' ?x01 C')))
```

この際、翻訳結果がアプリケーション側で定義されるオントロジ制約に違反するならば、その翻訳をキャンセルする。

2)「AはCです。」

「AはCである。」の意味表現はA、Cが普通名詞である場合、次のようになる。

```
[pred ([reln copula
        arg1 ?x01:[
              pred ([reln A
                     arg1 ?x01])]
        arg2 ?x02:[
              pred ([reln C
                     arg1 ?x02])]])]
```

これが「あるA（のB）はCという属性値を持つ。」という意味である場合、次のような述語表現を得たい。

```
(exists (?x01) (and (A' ?x01) (B' ?x01 C')))
```

もし、述語・属性対応表中で属性値 C に対しユニークに属性名 B が決まるのであれば、A, B, C の翻訳を用いて上記のような変換を行う。Cが固有名詞の場合も、同様の変換が考えられる。

3)「BがCのAはDです。」

「BがCのAはDです。」の意味表現はA、B、C、Dが普通名詞である場合、次のよう

に表される。

```
[pred ([reln copula
        arg1 ?x01:[
              pred ([reln A
                     arg1 ?x01]
                    [reln copula-の
                     arg1 ?x01:[
                           pred ([reln B
                                  arg1 ?x01])]
                     arg2 ?x03:[
                           pred ([reln C
                                  arg1 ?x03])]])]
        arg2 ?x03:[
              pred ([reln D
                     arg1 ?x01])])]
```

C が属性 B の値であるような変換が変換表中に存在するならば次のような変換を試みる（D を述語に変換する場合）。これは「属性 B の値が C で あるようなある A は D である。」という意味である。結果がアプリケーションオントロジに違反しないならば採用する。

(exists (?x01) (and (A' ?x01) (D' ?x01) (B' ?x01 C'))

また、D が 属性 A の値であるような変換も存在すれば、次のような変換結果を得る。

(exists (?x01) (and (A' ?x01 D') (B' ?x01 C'))

h) メタ述語

「色は属性です。」という発話は色という属性について語っている。「AはBです。」という発話で、Aの翻訳が属性名として定義されており、Bの翻訳が述語（あるいはクラス）として定義されている場合、次のような翻訳を試みる。

(B' A')

ここで B が属性を引き数として取る述語であれば、この翻訳は有効である。

属性や述語を引き数に取るような高階の述語は、述語・属性対応表上でマークしておき、それ以外の述語では上のような高階の翻訳が起こらないようにしておくことにより多義性を防ぐことができる。

ii) 多義性解消

自然言語インタフェイスにおける多義性解消の方法を以下に述べる。指示対象決定は広義の多義性解消に含まれるが、固有の問題を多く含むので iv) 節で議論する。

多義性には、意味解析結果中に含まれてくる多義性と、述語・属性対応表による多義性がある。述語・属性対応表に因る多義性は、設計段階で対応表に１対多の対応をできるかぎり避けることで減らすことができる。入力の多義性は「音声認識」、「自然言語解析」などのモジュールで統計情報などを用いることにより減少させることができる。特に、「自然言語解析」では係り受けの共起関係の統計などを用いて多義候補間の尤度付けを行うことができる。

a) オントロジ制約の利用

多義性解消でのオントロジ知識の利用方法として次の２つを考えることができる。

1) 意味表現を述語・属性対応表を用いて翻訳した結果をアプリケーションに送り、アプリケーションオントロジの制約を満たすかどうかを問い合わせる。制約を満たす結果のみを利用する。
2) 発話でのオブジェクトの指示に多義性がある場合、実際にアプリケーションの知識ベース中に存在するオブジェクトを表現する翻訳を優先する。

b) 文脈の利用

意味表現が多義である場合でも、発話の文脈で言及されているオブジェクトと関連する意義があれば、その意義が尤もらしいと考えられる。特に、システムがユーザに問い合わせを行っている場合、問い合わせ中で言及されるオブジェクトの情報により多義性の解消を行うことができる。例えば、システムのクレジットカード番号の問い合わせに対し、ユーザが「番号はＸＸＸです。」という発話を行ったとする。タスク中で「番号」という言葉が電話番号、部屋番号、クレジットカード番号を意味し、「番号」は同一の意味表現を持つが、その意味によって別々のアプリケーションとの間で用いられるメッセージの内容表現に対応するとしよう。「番号」に対応する意味表現中がどのアプリケーションとの間で用いられるメッセージの内容表現に対応するかということを決定するのは、一種の多義性解決問題である。ここでは、直前にシステムがクレジットカード番号オブジェクトに言及しているので、「番号」がクレジットカード番号を指すと判断できる。

c) 対話的多義性解消

自然言語インタフェイスがアプリケーションなどに問い合わせて得られる情報により解決できない場合（あるいは信頼度が十分でない場合）、ユーザに問い合わせることになる。前述したように、メッセージは発話内行為タイプと内容部分に分けることができるから、そのどちらかが不明な場合、わからない部分だけを聞き返すことができる。

例：発話内行為タイプを問う場合

「～ことについてどうなさりたいのでしょうか。」

（「～」は聞き取れた命題内容）

内容を問う場合（アクションの要求であることがわかっている場合）

「何をすればよいのでしょうか。」

内容部分は命題を表す場合と、単に属性値や引き数値を与えるだけの場合がある。命題内容は述語と引き数の値により表現されるから、命題内容が聞き取れない場合、述語か引き数の値あるいは両方を問い返すことになる。すでに命題がシステムからの質問などの形で文脈に与えられていれば、ユーザからのメッセージとして単に引き数の値が与えられることが期待される（例：「どちらに行かれますか？」「京都です。」）。

命題引き数の値としてのオブジェクトの指定は、その属性あるいは述語による記述によって行われる。オブジェクトの質問の仕方はいくつかある。すでに質問がなされていて、答えが聞き取れなかった場合のデフォルトの聞き直し方法は「もう一度お願いします。」などである。属性の一部が聞き取れた場合、例えば「赤い何ですか。」などの質問をする（「何」の部分は質問するオブジェクトの種類により「誰」などになる。また、選択肢があるのならば「どれ」や「どちら」を用いる）。述語による記述が聞き取れた場合、例えば「昨日来たどなたですか。」などの聞き返しが可能になる。

引き数がわかったが述語が不明な場合、「～についてどうするのですか。」「～がどうな

のですか。」などと聞き返すことができる。

日本語の場合、述語が「代動詞」化して「～をお願いします。」などの形になる。こうした場合の述語の補完は特別な処理を行う必要があるかもしれない。（多くの場合「～をお願いします。」という発話では「～」の部分だけが問題になっている。タスクが「～」がスロット値になるような質問を用意しているのならば、そのスロットを埋めてアプリケーションに返送することにより、述語の補完を行うことができる。）

iii) 情報の補完

話し言葉の解析では、発話時の省略やゼロ代名詞に加え、音声認識の間違いによる情報の欠落の問題が生じる。こうした問題は、発話中の情報を予期することである程度対処できる。

a) 特定範囲の値が予期される場合

属性値を表す表現が予期されている場合、その値を表す表現が意味解析結果に現れてくれば、それを値の候補としユーザに確認する。この方法を用いることができるのは、原則として属性値を問う質問の後であろう。例えば色を訊ねる質問の後で「！@＃＄黒＾＆＊」という入力（音声認識結果）があった場合、属性値として「黒」を指定していることが推定される。

b) 属性名、述語名の推定

属性値あるいは述語の引き数が判明しているとき、それらの値から（述語・属性対応表を用いて）属性名、述語名がユニークに決まるのであれば、それらを補完する。例えば「黒」という属性値に対し「色」という属性名を補完することができる。

iv) 指示対象決定

ここでは、自然言語インタフェイスにおける指示対象決定の方法を以下に述べる。指示対象決定と前節で述べた多義性解消は、独立の過程ではなく、互いに他の情報を必要とすることがある。

a) 照応処理

照応は、ある言語表現の指示対象が先行する談話中で言及された言語表現の指示対象と一致するという現象である。自然言語理解においては、照応を見つけることにより、複数の言語表現に含まれる情報を統合することができる。自然言語インタフェイスでは、指示表現の照応は、先行する談話の指示表現の変換結果（論理定項などのオブジェクト）と現在問題になっている指示表現の意味表現をリンクすることにより行われる。リンク可能なオブジェクトは、リンクの結果生成された内容表現がアプリケーションオントロジの制約を満たすようなものに限られる。また、時間的に近いところで参照されたオブジェクト表現が優先される。

b) 背景オブジェクトへの指示

アプリケーションオントロジ中には存在するが、今までの談話では指示されていないオブジェクトが指示対象となることがある。例えばホテルの予約業務で、ホテルそのものについては今まで言及されていなくとも、いきなり「そちらに行くにはどうすればよいですか。」などの文中で指示されることがある。こうしたオブジェクトを見つけるための１つの方法は、あらかじめそうしたオブジェクトの表現を自然言語インタフェイス中に取り込んでおくことである。これは例えば、アプリケーションオントロジ中で、言及されたオブジェクト（上の例では例えばホテルの部屋）に直接関連するオブジェクト（ホテルの部屋は特定のホテル

と所属関係を持つ）を自然言語インタフェイスへロードしておくことにより実現できる。もちろん、関連するオブジェクトをあらかじめロードするのではなく、指示対象が見つからない場合にのみ知識ベースの探索を行うようにしてもよいかもしれない。

（このレポートでは自然言語インタフェイスが統計的な処理を行うことを考慮していないが、特定の述語の引き数が統計的に特定の背景オブジェクトにバインドされやすいという情報も指示対象決定のために有用であろう。）

c) 新規インスタンスの導入

アプリケーションの知識ベース中に存在しないが、アプリケーションオントロジでクラス定義はなされており、新しいインスタンスとして導入しなければならないものがある。典型的な例は新規ユーザの登録である。もっとも、定型的業務における新規インスタンスは、しばしばタスクオントロジで予期されるオブジェクトである（例えば名前などの属性スロットが未定の「新規ユーザ」として予期される）。こうした予期される不定インスタンスは背景オブジェクトと同様に、タスクが決定した時点で自然言語インタフェイスにロードしておくことができる。

d) ＧＵＩオブジェクトの指示

ＧＵＩオブジェクトは、ポインティングデバイスあるいは言語的表現、さらにその両方を用いて指示することができる。ＧＵＩオブジェクトあるいはＧＵＩオブジェクトが表現するものをポインティングと言語表現の情報を用いて同定するためには、タイミング情報の利用や、ＧＵＩ属性（色のＲＧＢ属性や座標）と自然言語表現（「赤」などの属性表現や位置関係の表現など）のすりあわせが必要になる。このようなすりあわせはアプリケーションが行うことになると考えられる。（関連するトピックについて参考文献 [Mizunashi '95, '97] [Maybury '93] を参照されたい。）

## 5.6 必要なデータについて

この章では、アプリケーションで用いるオントロジを利用するような汎用的な自然言語インタフェイスを提案した。以下、このようなシステムが利用するデータについて補足しておく。

### 5.6.1 タスク依存データ

ここで述べてきた仕組みにおいて、自然言語インタフェイスが直接使用するアプリケーション依存データは述語・属性対応表だけである。アプリケーションに関するその他の情報は、アプリケーションに問い合わせることにより得られる。

アプリケーションで取り扱うオブジェクトに関する知識（アプリケーションオントロジ）の一部は、アプリケーションがオブジェクト指向ソフトウエアであれば、そこで用いられているオブジェクト定義から抽出することができる。また、述語・属性対応表で用いられる、アプリケーションとの間で用いられるメッセージ側の述語・属性のリストは、アプリケーションオントロジ中のオブジェクト定義から抽出することができる。自然言語側の述語・属性のリストは、タスクで用いられる語彙と自然言語解析／生成モジュールの仕様に依存する。

### 5.6.2 尤度情報の利用

アプリケーションに自然言語インタフェイス中で発生した多義性の適性チェックを依頼する際、自然言語インタフェイスは尤度の高いものからチェックに送り、最初にチェックにパスした解を用いるという戦略を取ることができる。意味解析結果中の多義性に、それまでの処理で尤度が付

加されている場合は、それを利用する。自然言語インタフェイス中で発生した多義性に関しては、例えば自然言語インタフェイス中にロードされている意味表現やアプリケーションからのメッセージの集合に対し、意味的な親和性（オントロジをシソーラス的に利用することなどにより計算できる）が高いものに高い尤度を与えるといった方策が考えられる。

第5章の文献


[Allen&Core '97] J. Allen & M. Core: "Draft of DAMSL: Dialog Act Markup in Several Layers," http://www.cs.rochester.edu/research/trains/annotation/RevisedManual/RevisedManual.html (1997).

[Gruber '97] T.R. Gruber: "Ontolingua: A Mechanism to Support Portable Ontologies," http://www-ksl.stanford.edu/knowledge-sharing/papers/README.html (1992).

[KIF Manual] The Interlingua Working Group of DARPA Knoledge Sharing Initiative External Interfaces Working Group: "Specification of the KQML," http://www-ksl.stanford.edu/knowledge-sharing/papers/README.html (1992).

[KQML Manual] The DARPA Knowledge Sharing Effort Interlingua Working Group: "Knowledge Interchange Format Version 3.0 Reference Manual," http://www.cs.umbc.edu/kqml/kqmlspec/spec.html (1993).

[Maybury '93] M. Maybury (ed.): *Intelligent Multimedia Interfaces*, The MIT Press (1997).

[Mizunashi '97] 水梨豪、ローケンキム・キュンホ、森元逞：「音声とポインティング・ジェスチャの統合意味解析」ATR 音声翻訳通信研究所テクニカルレポート TR-IT-0213 (1997).

[Mizunashi '95] S.Mizunashi, M. Tomokiyo, K. Loken-Kim, L. Fais & T. Morimoto: "Integration of Utterances and Gestures for Naturally-Spoken Dialogues on a Multimodal System" *NLPRS'95 Proceedings* (1995).

[Shoham'93] Y. Shoham: "Agent-oriented programming," *Artificial Intelligence*, Vol.60, pp.51-92 (1993).

[友清'94] 友清睦子：*対話行為ラベルとその自動付与*、ATR 音声翻訳通信研究所テクニカルレポート TR-IT-0069 (1994).

# 第6章　自然言語インタフェイスによる言語生成[7]

この章では、「自然言語インタフェイス」（☞第2章、第5章）による自然言語生成モジュールへの「意味表現」（☞第5章）生成について議論する。ここで「意味表現」は主に「アプリケーション」（☞第2章）からユーザへのメッセージの翻訳として生成されるが、第5章で論じたように、自然言語インタフェイスは解析に必要な情報が不足して、ユーザに自律的に問い合わせを発行する場合にも生成が行われる。

以下、6.1と6.2では、アプリケーションからのメッセージの翻訳について述べ、6.3では文脈による表現の選択、6.4では、ユーザフレンドリなエラーリカバリについて述べる。

## 6.1 発話内行為タイプの変換

5.5節で述べたように、意味表現における発話内行為タイプのセットとアプリケーションとの間で用いられるメッセージの発話内行為タイプのセットは必ずしも一致しないため、変換が必要になる。例えば、アプリケーションとの間で用いられるメッセージにおける命令は、ユーザに対するメッセージでは依頼の形を取ることになるかもしれない。また、意味表現にあってアプリケーションとの間で用いられるメッセージの発話内行為タイプにない発話タイプの情報は通知（inform や tell）の中に埋め込まれることになるだろう。

## 6.2 内容表現の変換

アプリケーションとの間で用いられるメッセージの内容表現から意味表現を生成する場合には、意味表現からアプリケーションとの間で用いられるメッセージの内容表現を生成する場合に用いた述語・属性対応表を逆向きに使用することができる。表が1対多になっている場合は、設計段階で、使用するものを1つ選んでおくことができる。

アプリケーションとの間で用いられるメッセージの内容表現用の述語は必要に応じて述語・属性対応表を用いて意味表現用の述語に変換される。アプリケーションとの間で用いられるメッセージの内容表現用の属性は、対応表を用いて意味表現用の述語または意味素性などに変換される。できるだけ自然な自然言語表現に変換されるように対応表上の指定を行う。

述語・属性対応表に対応項目がない場合、アプリケーションとの間で用いられるメッセージの内容表現中の属性名や、属性値、述語名はそのまま意味表現中に埋め込まれる。もちろん、適切な自然言語語彙の生成を行うためには、意味表現として扱われるこうした属性名や、属性値、述語名と自然言語の単語のマッピングが適切になされていなければならない。

例：「BがCのA」

属性対（B' C'）を持つクラスA'のオブジェクトは、名詞句「BがCのA」として表現されることが望ましい。この名詞句の意味表現を素性構造で記述すると例えば次のようになる。

```
?x01:[pred ([reln A
             arg1 ?x01]
            [reln copula-の
             arg1 ?x02:[pred ([reln B
                               arg1 ?x02])]
                               arg2 ?x03:[pred ([reln C
                                                 arg1 ?x03])]])]
```

[7] 言語生成一般に関する参考書としては [Kempen '87] を挙げることができる。

### 6.3 文脈による表現の選択

アプリケーションからの同一の形を持つメッセージは常に同じ形の自然言語表現に翻訳されるほうがよいとは限らない。文脈に応じて表現を変えたほうがよい、あるいは変えなければならない場合がありうる。

・代名詞化および省略

一度用いた表現を代名詞化したり、省略することにより発話はより自然になることがある。省略、代名詞化は、本レポートの枠組みでは通常自然言語インタフェイス中で行われる。どんな要素をいつ省略、代名詞化すればよいのかは検討すべき課題である。（なお、１単文中あるいは１複文中の名詞句の省略の一部は、文法的な制約を用いて行うことも可能である。この処理は文脈に依存しないので自然言語生成モジュール中で行ってもよい。☞ [久野 '78]）

・新旧情報マーカの選択

日本語のトピックマーカ「は」や、ヨーロッパ諸語の定／不定表現の選択は、表現される情報の新旧に依存する。トピックマーカは、構文（例：デフォルトとしての「ハガ構文」）や、オブジェクトの新旧により選択される。例えば、ある記号で表現されるオブジェクトそのものが新しい情報として提示される場合、その記号にトピックマーカ「は」は与えられない。何が新しい情報なのかを決定するためには、先行する対話を調べる必要がある。例えば「何が」を問う質問の答えとして与えられるオブジェクトは通常新しい情報である。

定表現はユーザが同定できるものを指すが、メッセージの論理表現中での個体定項や、属性値として表現されるものは定表現、存在量化された個体変数などは不定表現に対応するというふうにおおまかに分けることができる。既に出現したオブジェクトの論理表現は、タスク中で扱われる限り一時的な個体定項として登録されているはずであり、それらには定表現が対応する。なお、抽象表現など一部表現の定／不定の決定法は言語ごとに異なる。また、新しい個体を固有名詞などで導入する場合、「～という人／もの」などの導入表現を付加することが必要になる。

日本語では終助詞「ね／よ」の使い分けも、話し手と聞き手の情報の保持と移行の状態に依存するといわれている。実際には、終助詞「ね」は確認のマーカーなので、アプリケーションから確認として来たメッセージや、自然言語インタフェイスが独自に生成する確認発話に対しては「ね」を付与するようにすることにより、一次近似を行うことができる。なお、ここで扱うような業務対話では、やや乱暴な終助詞である「よ」を生成する必要はないだろう。

・態の選択

受動態、能動態などの「態（voice）」の選択は何を主語とするかに依存する。ユーザの発話中の格フレームを用いる場合は、その態をコピーするというヒューリスティクスを用いることもできるし、トピック（文フォーカス）を主語にするというヒューリスティクスを用いることもできる。生成元となる内容表現中で動作主が特定されない場合には、受動態や一般主語（英語の "you" フランス語の "on" ドイツ語の "man" など）、主語省略などを用いることができる。

・確定記述の選択

オブジェクトを同定するための記述をどこまで詳しく行うかという問題である。例えば「番号」というか「クレジットカード番号」というか「お客様のクレジットカード番号」というか、というような選択を例としてあげることができる。選択の１つの方法としては、言語インタフェイスの履歴中の複数のオブジェクトの表現を「意味表現」に変換し、多義性の問題が起こらないようなレベルまで詳しく記述するという方法が考えられる。もう１つの方法としては、ユーザが多義性を解決できず、質問をしてきたときにのみ追加的な記述を行うという方法が考えられる。

・ユーザモデルによる表現の選択

多様なユーザに対応する対話システムは、ユーザの背景知識を判断して使用する語彙や表現を変えることが求められる。もっとも、ここで検討する定型的音声対話システムの自然言語インタフェイスにはそこまでの能力は必ずしも求めない（☞7.2節）。丁寧さを表わすマーカーもユーザによって変わりうるが、これも定型的対話システムではあたりさわりのないレベルを設定することで対処できる（日本語の場合）。なお、文法的な性がある言語などでは、聞き手により生成する表現の性別が変化するが、これは自然言語インタフェイスか自然言語生成モジュールで対応する必要がある。

## 6.4 ユーザフレンドリなエラーリカバリ

自然言語インタフェイスはユーザが対話システムの発話を聞き取れなかった場合などの聞き返しに対して応答することができる。自然言語インタフェイスは文脈情報を保持しているので、ユーザが聞き返しているとわかれば、自らが前回生成した発話を繰り返したり、前回生成した発話の一部分の情報だけを含んだ発話を生成することや、文脈中の情報を補って発話の生成を行うことが可能である。

自然言語インタフェイスは、ユーザの発話からオントロジ制約を満たすメッセージを生成できない場合、ユーザにその旨を伝え、再発話を促さなければならない。自然言語インタフェイスは、生成された（オントロジ制約を満たさない）解釈のうち、一定値以上の尤度を持ち、かつ最も尤もらしいものに対して応答を行う。生成された解釈がどれも一定以上の尤度を持たないならば、自然言語インタフェイスは単に再発話を促す。なお、ユーザが主張している命題がオントロジ制約を満たしているが対話システムの持つ知識と矛盾する場合の応答は、知識ベースなどのアプリケーションモジュールが担当することになる。

自然言語インタフェイスはアプリケーション中の知識ベースとユーザの自然言語表現を仲介する役割を持つが、知識ベース中にデータがない場合や、知識ベースの更新に失敗した場合、標準的なメッセージを生成しなければならない。常に「探索（更新）に失敗しました。」「わかりません。」というようなメッセージを生成するのでは、あまりユーザフレンドリとはいえないので、少なくとも謝りの表現を付加するなどの工夫が必要であろう。テレフォンショッピングや予約業務の場合などのように、ユーザが対話的タスクハンドラを介してデータベース（知識ベース）にアクセスする場合、気のきいた応答の生成は対話的タスクハンドラの役割になる（例えば長い処理時間が予期される場合、対話的タスクハンドラは (request :receiver USER (wait USER)) =「お待ちください。」といったメッセージを自然言語インタフェイスに送る）。

自然言語インタフェイスの解析においても、多義性の解消ができなかったり、オントロジ制約を満たす変換結果を得られないなどのエラーが発生する。上で述べたように、自然言語インタフェイスはこれらのエラーに関して、ユーザに問い合わせを発行するが、この際にも適切な謝り表現などを用いることが「ユーザフレンドリ」なインタフェイスとして必要になるであろう。

より知的な対話システムはユーザの高次の目的を判定してエラーリカバリを行う。すなわち、ユーザのリクエストが成功に終わらなかった場合、よりユーザフレンドリな対話システムは、そのリクエストの背後にあるユーザの目的を判定して、それを実現するための方策（代案）を提案する。こうした機能は、本レポートの枠組みでは対話タスクハンドラに任される。（第４章の枠組みでは、ユーザの目的は対話的決定タスクにより判定される。対話システムには、その「知的度合」に応じ、ユーザの目的を推定するためのさまざまな推論を行わせるようにすべきであるが、このレポートではつっこんだ議論は行っていない。より詳しい議論のためには、次の章でも触れるユーザモデルの問題にとりくむ必要がある。）

第 6 章の文献

[Kempen '87] G. Kempen (ed.): *Natural Language Generation.* Martinus Nijhoff Pub. (1987).
[久野 '78] 久野すすむ：*談話の文法*、大修館書店. (1978).

# 第7章　おわりに

## 7.1 データの作成

ここでは本レポートで述べた諸モジュールで用いられるデータの作成手順の概要を述べる。

1) 模擬タスク会話コーパス

模擬タスク会話はダミーの対話システムプロトタイプを作り、Wizard of Oz 実験[8]を行うことなどにより収集することができる。模擬会話タスク会話コーパスは、以下に述べるように 1) タスク分析、2) 語彙の抽出、3) 統計情報の抽出、に用いられる非常に重要な資料である。

2) オントロジ

タスクの分析をオブジェクト指向的に行い、タスクで使用されるオブジェクトの種類と属性を定義する。本レポートで述べてきたように、この作業はアプリケーションの設計に必要であるのみならず、自然言語の解析にも有用である。タスク分析の後で必要に応じオブジェクトデータベースを定義し、タスク固有のオブジェクトインスタンスを定義する（オブジェクトデータベースにデータを入れる）。さらにオブジェクト間のさまざまな制約を記述して知識ベースに格納し、オントロジとして整備する。

3) 語彙（辞書）

a. タスク用語彙セット

模擬会話と概念フレーム辞書中に現れる形態素（特に内容語）を収集し、タスク用語彙セットとする。

b. 述語・属性対応表

模擬会話を調べ、タスク中で用いられるオブジェクトとその属性（値）、オブジェクト間の関係などがどのような自然言語表現に対応するかを決定し、第5章で述べた述語・属性対応表を作成する。

4) その他のデータ

タスク分析の産物は、オブジェクト定義だけではなく、タスクがどのようなプロセスからなるかという分析結果も含まれる。この分析結果から、例えば第4章で述べたようなタスク制御のためのプログラムを作成することができる。

語彙セットからは（ここでは詳しく述べないが）音声認識や自然言語解析／生成、さらに音声合成のための辞書が作られる。

模擬会話コーパスからは、音声認識用の統計（音響的統計、言語的統計）や、自然言語解析用の統計が抽出される。自然言語生成や音声合成も模擬会話コーパスからの統計情報を利用するかもしれない。

## 7.2 結語

このレポートの内容に関して、今後取り組むべき2つの課題がある。1つは、このレポートで述べた事柄を実装し、検証するということである。もう1つは、ユーザモデルの問題への十分な考察である。

実装に関しては、本論であまり議論をしなかった統計的あるいは意味的な選好情報を多義性解消にいかにうまく用いるかという点が、実装上の鍵になるかもしれない。

---

[8] 被験者には機械が応答していると教示しつつも実は機械の中に人が入って操作している様な実験である。

ユーザモデルは、ユーザの持つ知識、ユーザの選好、ユーザの現在のゴールなどのモデル化である。ユーザモデルの問題は対話システムの構築において非常に重要な事項である [Kobsa '88]。というのは、対話が対話相手のゴール[9]を推し量り、それに言語的に対処していくことであると考えられるからである。知的な対話システムは、ユーザモデルの情報（直接的には対話相手のゴールの推定であるが、推定のためにはユーザの知識や選好の情報が役にたつ）を用いて、適切なアクションを決定する。また、第6章で述べたように、適切な言語表現を生成するためには、ユーザの適切なモデルが必要になる。ユーザモデルの問題は（実装とは異なり）多くの研究課題をもたらしてくれるはずである。ユーザモデルの問題は非常に広範かつ深いので、詳細な議論は場所を改めて行うことにする。

第7章の文献

[Kobsa '88] A. Kobsa & W. Wahlster (eds.): *User Models in Dialogue Systems*. Springer-Verlag. (1988).

謝辞

本レポートの作成にあたり御意見をいただいた第4研究室の方々に感謝します。

[9] これは単なるおしゃべりにおいても存在する。ゴールは「親睦」だったりするが、そのサブゴールは「お互いを知り合う」ことかもしれない。もっと単純に情報交換が目的のおしゃべりもある。もちろん、その他の有目的的会話と同様、ゴールの表象が明確に意識されているとは限らない。

## 付録 「意味表現」の素性構造記述

第５章では、自然言語インタフェイスが自然言語解析・生成モジュールとのインタフェイスに用いる表現形式を意味表現と呼び、その意味表現の記述のために素性構造を用いた。ここでは、第５章の例で用いられた意味表現の素性構造記述の詳細を記す。本文でも述べたように、意味表現の記述に素性構造を用いることは、あくまで記述の利便のためであり、同様な情報が表現できれば他の記述法を取ってもよい。

まず、各発話の意味表現は次のような情報を持つ。

話者
発話開始・終了時間
発話内行為タイプ
意味内容

これらを素性構造の形で書くと例えば次のような形になる。

```
[speaker 話者オブジェクト
 time [start 発話開始時間
       end   発話終了時間]
 ift 発話内行為タイプ（inform, information-request など）
 sem 意味内容表現]
```

意味内容表現は例えば次のような形式のオブジェクト表現の形を取るものとする。

```
オブジェクト表現 ::= ?x01:[Feature_List]
```

ここで、?x01 はオブジェクト自体を表す変数であり、Feature_List は素性対（属性対）のリストである。素性対（feature pair）は素性名と素性値の対であるが、オブジェクトに対する用語としては属性名と属性値からなる属性対（attribute pair）と呼んだほうがわかりやすいだろう。「意味内容表現」の表現するものは通常命題内容であり、状況を表すオブジェクト表現である。

述語表現は素性 pred の値のリストに現れるものとする。以下、述語に関する代表的なのオペレータについて記述例を示す。

a) 述語の連言は素性 pred の値のリストとして書かれる。

通常の述語表現で ABC(?x01, ?x02) & CDE(?x01) などと書かれる表現は次のように表記する。

```
([reln ABC
  arg1 ?x01
  arg2 ?x02]
 [reln CDE
  arg1 ?x01])
```

簡潔な別表記法として、次のものを許してもよい。

```
((ABC ?x01 ?x02)
 (CDE ?x01))
```

b) 述語の選言は選言述語 or を用いて記述する。

```
pred ([reln OR
       arg1 [reln ABC
                  arg1 ?x01
                  arg2 ?x02]
       arg2 [reln CDE
             arg1 ?x01]])
```

または ((OR (ABC ?x01 ?x02)
(CDE ?x01)))

c) 述語の否定は否定述語 not を用いるか極性表記を用いる。

```
pred ([reln NOT
       arg1 [reln ABC
             arg1 ?x01
             arg2 ?x02]])
```

または (NOT (ABC ?x01 ?x02))

極性表記を用いると次のようになる。

```
  ([reln ABC
    arg1 ?x01
    arg2 ?x02
    pol  F])
```

(極性 pol の値のデフォルトは T であり、述語の値が不確定であることは pol が Q であることで表す。)

d) 述語の含意は含意述語 if を用いて表現する。

```
pred ([reln if
       arg1 ([reln A
              arg1 ?x01])
       arg2 ([reln B
              arg1 ?x01])])]
```

e) 全称命題は全称述語 forall を用いて表現する。

```
pred ([reln forall
       arg1 全称化される変数のリスト
       arg2 述語リスト])
```

例えば、次の表現はすべての A は B であることを意味する。

```
([reln forall
       arg1 (?x01)
       arg2 ([reln if
              arg1 ([reln A
                     arg1 ?x01])
              arg2 ([reln B
                     arg1 ?x01])])]
])
```

f) 単称命題は、存在述語 exists を用いて表現する。

```
pred ([reln exists
       arg1 ?x01
       arg2 ([reln ABC
                ?x01])])
```

あるいは (exist ?x01 ((ABC ?x01))) は、(ABC ?x01) であるような ?x01 で表示されるオブジェクトが存在することを意味する。

例：「これは猫ですか。」

```
[speaker $x001              ; 話者 ($001 はオブジェクト表現へのポインタ).
 time [start xxx1           ; 発話開始時間
       end   xxx2]          ; 発話終了時間
 ift TF-question            ; 発話内行為タイプ
 sem ?x01:[                 ; 意味オブジェクト
     situation +            ; 状況を表す。
     pred ([reln copula     ; pred (...) は述語の連言；reln は述語名
           arg1 ?x01:[      ; オブジェクト変数
                pred ([reln this
                       arg1 ?x01])]
           arg2 ?x02:[      ; オブジェクト変数
                pred ([reln cat
                       arg1 ?x02])]
           pol T])]]
```

「これ」の指示対象は、後述する指示対象推定処理によりアプリケーションが扱う特定のオブジェクトインスタンスにバインドされる。

なお、意味表現中の多義性はリストで表す。例えば ift に多義がある場合、
[ift (inform TF-question)] などと表す。多義性は一般的に尤度を伴うが、ここでは割愛する。