

ドメインオントロジーに基づく言語理解

A Framework of Domain Ontology-centric Language Understanding

船越 孝太郎 中野 幹生 長谷川 雄二 辻野 広司
Kotaro Funakoshi Mikio Nakano Yuji Hasegawa Hiroshi Tsujino

(株) ホンダ・リサーチ・インスティテュート・ジャパン
Honda Research Institute Japan Co., Ltd.

This paper describes a framework of natural language understanding for a broad range of applications. Without deep knowledge and experience of natural language processing, users of this framework can build a powerful natural language dialogue interface easily just by providing an ontology (a concept hierarchy, semantic frames and a lexicon), an onomasticon (a set of proper names) and API methods that define behaviour of a target application.

1. はじめに

本稿は、多様なアプリケーションに利用可能な自然言語理解のフレームワーク、KNOLU を紹介する。KNOLU を用いることで、自然言語処理に関する知識を持たないアプリケーション開発者でも容易に自然言語インターフェースを構築できる。KNOLU を用いて構築されたインターフェースはキーワード抽出や文パターンとのマッチングに基づく言語理解を行うインターフェースよりも高度な言語理解が可能でありながら、アプリケーション開発者に要求されるものはアプリケーションドメインに関するオントロジー知識の記述と基本的なプログラミング能力だけであり、複雑な文法の記述や管理を行う必要がない。

本稿の構成は以下の通りである。まず 2 節で、現在の自然言語処理パラダイムにおける自然言語理解について説明し、その中で我々のフレームワークの特徴を説明する。次に、3 節で提案フレームワークを用いて自然言語インターフェース（対話システム）を構築する方法を説明し、4 節で実際の適用例を二つ紹介する。最後に 5 節で結ぶ。

2. 自然言語理解

図 1 に自然言語理解の模式図を示す。現在の自然言語処理パラダイムにおける自然言語理解とは、自然言語表現から計算機が理解できるタスク表現への変換機能である。すなわち自然言語理解とは、大雑把に言えば、多種多様な曖昧性を含む自然言語表現を処理してユーザの要求を曖昧性のない意味表現で表し（図 1 (α)）、そのユーザの要求を達成する処理を実行するに足るタスク表現を意味表現から生成する（図 1 (β)）ことといえる。本論文では (α) の過程を意味表現生成とよび、(β) の過程をタスク表現生成とよぶ。

対話システムであれば、実際には言語表現からタスク表現への変換は一度に行われるのではなく、不足する情報を獲得するための補助的なタスク表現（例えば確認や明確化要求を行うための発話生成）が生成 / 遂行されながら徐々に意味表現が積み上げられ、十分な情報と確信が手に入ったところで主たるタスク表現が生成されることになる。意味表現とは言語・対話処理のための内部表現で、タスク表現とはタスク処理のための内部表現といえる。ロボット対話であればタスク表現はタスクプランナの入力となる目標状態（ゴール）で、データベース検索対

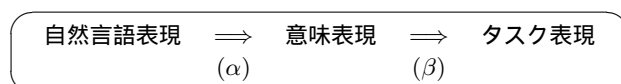


図 1: 自然言語理解の模式図

話であればタスク表現は SQL 文になるだろう。本稿では、図 1 に示した一連の流れを言語処理の専門家でなくとも容易に実現できるフレームワークを提案する。

2.1 意味表現生成

自然言語理解は意味表現生成方式の違いによって大きく二つの方式に分けられる。一つは、キーワードやパラメタライズされた表現パターンと入力発話との表層的なマッチングによって意味抽出を行う方式で、本稿ではこれをテンプレート方式とよぶ（例えば [Konashi 04]）。もう一つは、文法規則や統計データを用いて入力発話を構文解析し、得られた構文木から再帰的な手続きによって意味表現を得る方式で、これを構文解析方式とよぶ（例えば [Tokunaga 04]）。

テンプレート方式は構築・保守共に比較的容易で、構文解析方式に比べて頑健なシステムを実現しやすいため、音声対話システムでよく用いられる。そのかわり、キーワードの羅列で代替できる単純な表現しか扱えない。構文解析方式は、構文の再帰性、意味の合成性を正面から扱うため、複雑な要求を扱うことができる。反面、構築・保守共に容易ではなく、一般に頑健性も低い。

KNOLU ではテンプレート方式と構文解析方式を融合した意味表現生成 [船越 07] を行う。従って、構文解析結果に基づく合成的な意味理解が可能であると同時に、キーワードを羅列しただけの非文法的な表現も理解する。[船越 07] の手法は、構文解析を利用する従来の手法（例えば [山本 95, Milward 00, Dzikovska 03]）と異なり、意味表現生成のための変換規則を人手で記述する必要がない。

2.2 タスク表現生成

意味表現生成の方式に関わらず、アプリケーション開発者は意味表現からタスク表現を生成する機構を用意する必要がある。しかし、意味表現生成に関する研究は多いが、タスク表現生成についてはほとんど言及されていない。

[Glass 01] のように、明示的なタスク表現生成機構を持たないフレームワークでは、語句の意味解釈をタスク処理モジュールに委譲してしまう。例えば、「金曜日の天気を教えて」という表現の中の「金曜日」という語の意味は言語・対話処理モ

ジュールでは解釈されず、タスク処理モジュールに「金曜日」という事前に定められたキーワードとして送られるだけである。「金曜日」と表現された日がカレンダー上のどの特定の日の意味するのかがタスク処理モジュールが判断する。対話が行われている日から一週間先までの7日間だけしか扱わないシステムであればこのような方法でも良いかもしれないが、より広範な日時を対象としそれらに対し複雑な表現（例えば「来月の第三金曜」）を許すならば、タスク処理モジュールにそれらの表現の解釈を行わせるのは適当ではない。そのようなやり方をすれば、言語処理を複数のモジュールに分散させることになり、システム開発を複雑にしモジュールの再利用性を損なう。「金曜日」も「来月の第三金曜」も共に明示的なタスク表現生成機構によって一様に解釈され、カレンダー上のある特定の日付として（例えば整数値などの形で）タスク処理モジュールに渡されるべきである。自然言語理解フレームワークは、タスク表現生成機構をアプリケーション開発者が容易に構築できる仕組みを提供しなければならない。

我々のフレームワークでは、意味フレームと一対一で対応づけられた具体化関数という装置を用いることで、タスク処理と言語・対話処理を明確に分離しやすくしている。具体化関数については次節で説明する。

3. 自然言語インターフェースの構築

KNOLU はドメイン非依存の言語理解機構と対話管理機構を備えており、KNOLU を用いる開発者は以下の三種のドメイン知識を記述するだけで対話システムを構築できる。

KNOLU の実装には Java^{*1} と XML^{*2} を使用した。以下の説明は Java と XML の基礎的な知識を前提とするが、例を用いて説明するのでそれらに詳しくない読者でも直感的に理解可能だと考える。

3.1 オントロジーの定義

オントロジーは概念階層、概念を参照するための言語表現、個々の概念を規定する意味フレーム、意味フレームから具体化関数へのマッピングからなる。オントロジーは図2に示すように XML 形式で記述する。

図2に示したオントロジーは、crawler ロボットの移動を制御するタスクのために必要な四つの概念を含んでいる。crawler ロボットができることは指示された場所に、歩くか走って移動することだけである。OWL^{*3} とは異なり、KNOLU の概念階層は is-a 関係に基づく木構造でなければならない。top 要素は概念階層の頂 (top) を表す。top 要素には対象ドメインの名前と具体化関数を含む API (3.2 節参照) を実装した Java クラス名を指定する。図2の例の場合、ドメイン名は crawler で API クラス名は CrawlerAPI である。

各概念は、concept 要素によって定義され、概念を一意に特定する概念名 (name 属性) を与えられる。もしある概念が言語的に参照可能であれば、参照するために使用される言語表現 (概念表現) を term 属性として与える。

各概念は最低一つの意味フレーム (frame 要素) を持つ。各フレームの定義は具体化関数へのマッピング (method 要素) を一つ含む。フレームの名前は与えなくても良い。フレームの名前が未定義であれば、KNOLU が自動的に固有の名前を与える。下位概念は上位概念のフレーム定義を継承する。従っ

```

01 <top domain="crawler" apiClass="CrawlerAPI">
02   <concept name="move">
03     <slot name="goal" concept="place" clarif="どこに?"
04       spec="行き先 目的地" marker="に へ">
05       <frame name="moveFrame" class="Action">
06         <method name="moveFunc">
07           <param value="concept:this" class="String"/>
08           <param value="slot:goal" class="String"/>
09         </method>
10       </frame>
11     <concept name="walk" term="歩け 行け"/>
12     <concept name="run" term="走れ 急げ"/>
13   </concept>
14   <concept name="place">
15     <frame>
16       <method name="identity" class="String"/>
17       <param value="symbol:this" class="String"/>
18     </method>
19   </frame>
20 </concept>
21 </top>

```

図2: crawler オントロジー

て、図2の例の場合、概念 walk と概念 run は、同じフレーム moveFrame を概念 move と共有する。

概念 move の意味フレーム moveFrame は、具体化関数 moveFunc に二つの媒介変数 (param 要素) を伴ってマップされる。これらの媒介変数は、定義のままの順序で moveFunc 関数の引数となる。param 要素の value 属性は引数値の源泉を指定する。実際の引数値 (具体化された値) は Java インスタンスオブジェクトである。class 属性はこれらのインスタンスオブジェクトの Java クラスを指定する。

07 行目の値指定 concept:this は、moveFunc 関数の第一引数が、その関数を実際に呼び出した概念の名前であることを意味する。従って、moveFunc が概念 walk によって呼び出された場合、第一引数の値は “walk” という文字列オブジェクトとなる。概念 run に呼び出された場合は “run” である。08 行目の値指定は moveFunc の第二引数の値が goal スロットの具体化された値であることを意味する。17 行目の値指定は identity 関数の引数が関数を呼び出した概念のインスタンスのインスタンスシンボルとなることを意味する。例の場合、identity 関数は、3.3 節で定義されるインスタンスシンボルをその唯一の引数としてとる。identity は入力された文字列をそのまま出力するだけの関数である。値指定として term:this が指定された場合は、入力文中で実際に使用された言語表現が具体化関数に渡される。これは、概念の意味がその言語表現によって変化する場合に使用される。例えば概念 day は、その具体的な意味を表層表現に応じて変化させる (「今日」や「25日」など)。

slot 要素は意味フレームのスロットを定義する。スロット自身の定義は意味フレーム定義の外側でなされる。ある意味フレームがどのスロットを持つかは、具体関数へのマッピングの定義から後ろ向きに規定される。moveFrame の場合、moveFunc がスロットから値を受け取る引数をただ一つだけ持ち、そのスロットが goal であるので、moveFrame 自身が持つスロットは goal だけとなる。

スロットの定義は、その名前 (name 要素)、値の型 (concept 要素)、スロットを特定するための言語表現としてスロット指定 (spec 属性) と標示 (marker 属性)、明確化要求表現 (clarif

*1 <http://www.sun.com/java/>

*2 <http://www.w3.org/XML/>

*3 <http://www.w3.org/2004/OWL/>

```
public class CrawlerAPI extends API {
    ...
    public Action moveFunc(String action, String goal) {
        ...
    }
    ...
}
```

図 3: CrawlerAPI の実装

属性)を含む。スロット指定は、例えば「目的地は図書室」のような表現で話し手が情報の役割を指定することを可能にする。標示はスロット指定と同様の機能を持つが、スロット指定よりも頻繁かつ文法的に使用される。一般に標示は助詞などの機能語で、日本語の場合標示対象の直後に付加される。スロット指定、標示共にオントロジー定義において必須ではないが、これらを与えることでより正確な言語理解が可能になる。明確化要求表現はスロットの値が空のときに、KNOLU の対話管理機構によってユーザへの質問に使用される。明確化要求表現が定義されていない場合、KNOLU は既定値を使用する。既定値の与え方は三種類あるがそれらの説明は省く。

開発者は一つ概念に対して複数の意味フレームを定義することができる。複数の意味フレームを定義することで、KNOLU は同一の意味内容を異なる仕方理解することができる。例えば「3/4 から 3泊予約」と「3/4 から 3/7 まで予約」は異なる仕方同じ期間を指定している。

3.2 API の実装

API (Application Program Interface) ^{*4} はアプリケーションシステムの振る舞いを規定する。API は具体化関数、既定値解決関数、そして対話制御子からなり、すべて Java メソッドとして実装される。既定値解決関数は空のスロットに対して既定値を与え、対話制御子は対話のドメイン依存部分を制御する。

具体化関数は機械が理解可能な語句の意味を Java インスタンスオブジェクトとして与える。アプリケーション開発者は 3.1 節で説明したように意味フレームから具体化関数へのマッピングを定義し、全ての具体化関数を実装する。図 2 で定義した CrawlerAPI クラスは、図 3 中のようなコードとして実装される。開発者は crawler ロボットに移動させるためのコードを moveFunc の中に実装すれば良い。moveFunc は要求された行動 (walk あるいは run) を文字列オブジェクト action によって識別し、文字列オブジェクト goal によって目的地を知る。関数 identity は CrawlerAPI の親クラスである API に実装されている。

具体化関数は汎用プログラミング言語である Java によって実装されるので、開発者は高い自由度を享受できる。例えば、「95年の6月10日から9月20日までの東京の最高気温を教えてください」という要求に答えるシステムを、「最高気温」に対する具体化関数に SQL 文生成機能を実装することで容易に構築できる。

「明日」、「最後」などの相対的な言語表現はその具体的な意味を事前に静的な形で定義することが難しい。これらの言語表現の意味解釈を行う際の一般的な解決策は、これらの語句の意味を意味表現生成モジュールやタスク処理モジュールの中にハードコードしてしまうというものである。この問題に対して具体化関数はより柔軟な解決策を提供する。具体化関数のおか

*4 所謂 API (Application Programming Interface) ではない。

```
<world domain="crawler">
  <instance concept="place" symbol="room1"
    name="玄関 入り口"/>
  <instance concept="place" symbol="room2"
    name="書斎 図書室"/>
  <instance concept="place" symbol="room3"
    name="ガレージ 車庫"/>
</world>
```

図 4: インスタンスの定義

げで、アプリケーション開発者はシステムのドメイン・言語非依存な部分からこれらの言語表現の解釈を容易に分離できる。

開発したアプリケーションシステムを起動すると、KNOLU は与えられたオントロジーと API との間の整合性を検証する。もし何らかの不整合が見つければ、KNOLU はシステムを停止しその不整合を警告する。このような検証機構はシステムの開発とデバグの効率を大幅に向上させる。

KNOLU はオントロジーに定義された概念名を Java 1.5/5.0 の enum 定数として出力することができる。これらの定数を使用すれば、API 実装における概念名の綴り誤りが Java コンパイラによって自動検出されるため、シンボルプログラミングにつきまとう綴り誤りによるバグを減らし、開発者に与えるフラストレーションを軽減できる。Eclipse^{*5} などの IDE (統合開発環境) を用いればコード補完機能を利用して開発効率も向上できる。

3.3 インスタンスの定義

アプリケーション開発者は最後に、アプリケーションドメインに現れる、場所や人物/エージェントといったインスタンスを定義する。インスタンスには固有の文字列 (インスタンスシンボル) を与え、同時に固有表現を定義する。crawler ドメインのインスタンス定義の一例を図 4 に示す。

図 4 の例では place 概念の三つのインスタンスが定義されており、それぞれ room1, room2, room3 というインスタンスシンボルを持つ。例に示されているように、個々のインスタンスは複数の固有表現を持ちうる。例えば room1 は「玄関」とも「入り口」とも参照される。

3.4 実行例

以上で KNOLU を動作させる十分な知識を記述できた。例として、以下の文を crawler ドメインで起動した KNOLU に入力する。

玄関と車庫に行け (1)

KNOLU は意味表現生成機構 [船越 07] が出力する意味表現に従って具体化関数を呼び出し、タスク表現を生成することで言語理解を完了する。KNOLU は Java のリフレクション機構を用いて、以下の Java コードと同等の、一連の具体化関数の呼び出しを実現する。

```
CrawlerAPI.moveFunc("walk", API.identity("room1"));
CrawlerAPI.moveFunc("walk", API.identity("room3"));
```

4. 適用例

4.1 ホテル予約システム

図 5 に示すオントロジーを用いてホテル予約用対話システムを構築した。対話システム自体の構築に要したものはこのオ

*5 <http://www.eclipse.org/>

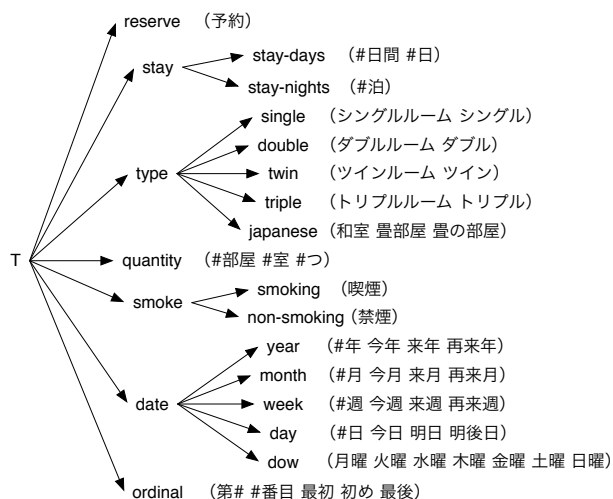


図 5: ホテル予約システムオントロジー (概念体系と概念表現)

ントロジー (95 行の XML 形式テキスト) と API クラス (150 行の Java プログラム) だけである。

被験者 9 名それぞれに予約課題 5 つに取り組みさせた。課題毎に宿泊期間などの条件を指定し、それらの条件を満たす部屋を予約するように被験者に指示した。対話は Web ブラウザ経由のキーボード対話である。被験者実験で得た対話の例を図 6 に示す。

発話 U1 はシステムが理解できない「7 人」という語句を含んでいるが、単純にそれを無視することでシステムは支障無く動作している。U3 は並列表現と文内省略 (ツインに対する禁煙の指定) を含んでいるが、それらは全て適切に処理されている。並列表現と文内省略の処理は KNOLU によって自動的に行われるので、上記の XML テキストと Java プログラムはそれらに関する処理や知識は含んでいない。S2 はドメイン非依存の対話管理の結果としてシステムが生成した。S4 と S6 はドメイン依存の対話制御子 (3.2 節) が生成した。

課題の達成率は 45 対話中 26 対話 (57.8%) であった。この課題達成率自体は高いとはいえないが、比較的単純な概念定義だけで図 6 の対話例が示すような複雑な要望も扱えることが確認できた。失敗のほとんどはドメイン外の要求と可算概念の高度な理解を必要とする表現が原因であった。

実験で得られた発話に対して意味表現生成手法の性能を調べたところ、フレームとそのスロットの値の関係にある二つの概念表現の組の 94.0% を正しく理解していた。従来のように構文解析結果に強く依存した意味解析を行った場合、例外処理などを導入しなければ、正しく理解できたのはおおよそ半分の 48.8% だけであった。

4.2 ロボット行動制御

自然言語理解システム K2 [Tokunaga 04] では、簡単な日本語を理解する 3 次元仮想空間内のキャラクター (アニメーションロボット) に対して音声で物体の移動などの物理行動を指示できる。このドメインでは物体を参照する表現に再帰性が現れるため、テンプレート方式では処理できない。

K2 システムの後継バージョンである K4 システムの言語理解部分を KNOLU を用いて置き換えた。照応解決などを除く基本的な言語理解を行うために必要な元々のコードの量と、KNOLU で同等の言語理解を行うために今回実装したコードの量をソースレベル (コメントは除外) で比較したところ、お

- U1: 8 月 15 日から 4 泊 7 人予約したい
- S2: どんな部屋をご希望ですか? シングル/ダブル/ツイン/トリプル, 及び和室 (4 人部屋) がございます
- U3: 喫煙のシングル 1 部屋, 禁煙の和室, ツイン一部屋
- S4: 以下の内容でよろしいでしょうか?
 項目 1: チェックイン:2006/8/15, チェックアウト:2006/8/19 (4 泊) 部屋種:シングルルーム 部屋数:1 喫煙
 項目 2: チェックイン:2006/8/15, チェックアウト:2006/8/19 (4 泊) 部屋種:和室 部屋数:1 禁煙
 項目 3: チェックイン:2006/8/15, チェックアウト:2006/8/19 (4 泊) 部屋種:ツインルーム 部屋数:1 禁煙
- U5: はい
- S6: 承知しました。料金は、108000 円です。

図 6: 実際の対話例

よそ 9 割削減できていた。

5. おわりに

本稿では、自然言語インターフェースを容易に開発できるフレームワーク、KNOLU を紹介した。KNOLU は、アプリケーションドメインの知識を XML と汎用プログラミング言語 (Java) で与えるだけで、並列表現や文内省略も理解する自然言語インターフェースを提供する。構文解析に基づく言語理解を行うが、ドメインに応じた文法規則や、構文解析結果から意味表現を生成するため変換規則を手で作成する必要はない。

KNOLU を用いてホテル予約システムを実装し、単純な記述で複雑な対話に対応できることを確認した。また KNOLU をロボット制御に適用し、基本的な言語理解の実現に必要なコードの量を大幅に削減できることを確認した。

今後の課題として、音声対話への対応、数に関する言語処理のドメイン非依存化などが挙げられる。また、開発環境を整備した上で、実際にアプリケーション開発者が自然言語インターフェースを容易に構築できるかどうかを評価する必要がある。

参考文献

- [Dzikovska 03] Dzikovska, M. O., Allen, J. F., and Swift, M. D.: Integrating linguistic and domain knowledge for spoken dialogue systems in multiple domains, in *Proceedings of IJCAI-03 Workshop on Knowledge and Reasoning in Practical Dialogue Systems* (2003)
- [船越 07] 船越 孝太郎, 中野 幹生, 長谷川 雄二, 辻野 広司: 構文解析を補助的に用いる意味解析, 言語処理学会第 13 回年次大会予稿集 (2007)
- [Glass 01] Glass, J. and Weinstein, E.: SpeechBuilder: Facilitating spoken dialogue system development, in *Proceedings of EUROSPEECH 2001*, pp. 1335–1338 (2001)
- [Konashi 04] Konashi, T., Suzuki, M., Ito, A., and Makino, S.: A spoken dialog system based on automatic grammar generation and template-based weighting for autonomous mobile robots, in *Proceedings of INTERSPEECH 2004*, pp. 189–192 (2004)
- [Milward 00] Milward, D.: Distributing Representation for Robust Interpretation of Dialogue Utterances, in *Proceedings of the 38th Annual Meeting of the ACL*, pp. 133–141 (2000)
- [Tokunaga 04] Tokunaga, T., Funakoshi, K., and Tanaka, H.: K2: Animated agents that understand speech commands and perform actions, in *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence*, pp. 635–643 (2004)
- [山本 95] 山本 幹雄, 伊藤 敏彦, 肥田野 勝, 中川 聖一: 人間の理解手法を用いたロバスタな音声対話システム, 情報処理学会論文誌, Vol. 36, No. 4, pp. 471–482 (1995)