3E2-05

対話と言い換えを用いた言語表現によるプログラミングシステム

Programming System by using Linguistic Expression with Interaction and Paraphrasing

金子 望[*1] 鬼沢 武久[*2]
Nozomu KANEKO Takehisa ONISAWA

[*1]筑波大学 システム情報工学研究科
Systems and Information Engineering, University of Tsukuba

[*2]筑波大学 機能工学系
Institute of Engineering Mechanics and Systems, University of Tsukuba

This paper aims to propose an end-user programming system that applies interaction and paraphrasing with non-programming language to computer programming. Users can build computer programs inputting the procedures of problem solving expressed by non-programming language into the presented system. The interaction between users and the system bridges gaps between users' understanding and computer's one caused by ambiguity of non-programming language. Paraphrasing, which changes an undefined expression into defined one, is effective for a computer to understand users' intention. The usefulness of the presented system is confirmed by some experiments.

1. はじめに

近年, コンピュータの著しい普及に伴なってコンピュータのユーザが必ずしもコンピュータのエキスパートとは限らない状況が増えてきている. しかし, ソフトウェアを記述するプログラミング言語は, 論理的な正しさが優先されるため必ずしも人間にとってわかりやすいものではなく, そのためこれまでのソフトウェアはプログラミングに関する特別な知識を持ったエキスパートによって開発されてきた. このような状況では, プログラミングの知識に乏しいユーザは, コンピュータに対する多種多様な要求を満たすために, 既存のソフトウェアだけに頼らなければならないことが問題となる. これは, 従来のプログラミング言語の考え方が, 人間の側にその言語を習得することを要求し, コンピュータ側には人間の言葉を理解する必要はないというコンピュータ本位の考え方に基づいているためである.

このような問題を解決するために, これまでエンドユーザプログラミングに関する様々な研究が行われてきた. その一つに, 例示によってプログラムを自動生成する Programming by Demonstration (PBD) 技術がある [1]. しかし, PBD には終了条件や条件分岐を示せないことや, 推論エラーを修正できないなどの問題点が指摘されており, その能力には限界がある. 一方, エンドユーザにとって最も理解しやすい言語は日常使用している自然言語であるため, ユーザインタフェースに言語表現を用いた知的システムは人工知能の初期から盛んに研究されている. 例えば初期の代表的なシステムとして, 積木の世界において人間と自然な英語で対話を行う SHRDLU [2] がある. また, ソフトウェアの信頼性・生産性向上を目標とする自動プログラミングの分野では, 形式仕様からプログラムを生成する際に言語表現がしばしば用いられている [3]. 最近では, 従来の数値に基づく情報処理に代わってあらゆる情報処理を日常言語で行う「日常言語コンピューティング」という概念が提案されている [4]. 特にエンドユーザプログラミングに言語表現を用いたものとして, 英語による入力から Java 言語の構文木を生成するシステム [5] や, オントロジーを用いて概念レベルで問題解決を行うシステム [6] などがある. しかしこれらのシステムは, 世界を限定するか, あらかじめ莫大な知識を用意しなければならないという問題を持っている.

本研究では, 人間同士の日常的なコミュニケーションにおける対話の機能に注目する. 人間同士の日常的なコミュニケーションでは, 状況に応じて柔軟な対話が行われるため, 伝えたい内容のすべてを完全に表現する必要はなく, 曖昧な表現でもコミュニケーションが成立するという特徴がある. また未知の知識も対話を通じて学習がなされ, あらかじめすべての知識を持っていなくてもコミュニケーションを行うことができる. そこで本研究では, ユーザの意図をシステムが適切に推論するために対話を利用することを考え, 言語表現による対話の実現を目標とする. このようなシステムが実現されれば, エンドユーザー自身が理解可能な言語表現を用いてプログラミングを行えるようになり, より効率的にコンピュータを利用することができると考えられる. 本研究では, プログラミングの経験がないユーザでも理解可能な対象として特に, テキスト編集を行う状況を想定して, 言語表現による対話からユーザの要求に合ったプログラムを自動的に生成・実行するシステムを構築する.

連絡先: 〒 305-8573 茨城県つくば市天王台 1-1-1
E-mail: nozom@fhuman.esys.tsukuba.ac.jp

2. 言い換えによるプログラミング

言語表現から機械実行可能言語への変換は, 2 言語間の機械翻訳とみなすことができる. ここで, 文献 [7, 8] を参考に, 翻訳プロセスを「(狭義の) 翻訳」と「言い換え」に分けて考える. その概念図を図 1 に示す.

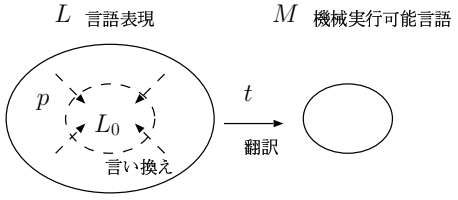

図 1: 翻訳と言い換え

(狭義の) 翻訳は, 特定の言語表現と機械実行可能言語の間の写像 t である. 言語表現の集合 L は無限に要素が存在するのに対して, 機械実行可能言語 M は有限集合であり, 要素の数もずっと少ない. そこで, M にほぼ 1 対 1 に対応する L_0 を定めて, L_0 と M の間でのみ写像 t を定義する. そして, 言い換え p によって任意の言語表現を翻訳可能な表現に置き換える. なお p は一般に L から L への写像となり, 複数の言い換えを行うことで最終的に L から L_0 への写像を得るものとする. 言い換えは同一言語内の変換であるため, プログラミングの知識のないユーザでもその妥当性を判断できるという利点がある. ただし翻訳事例に関しては, ユーザはその妥当性を判断できないためユーザによる追加は行わず, あらかじめ必要な事例を用意しておくことを前提とする.

3. システム構成

本システムは図 2 のような構成をしており, ユーザの抽象的な要求を含む言語表現を, 言い換えと翻訳を用いた対話を通じて段階的に具体化することで, 最終的にプログラムを生成する. 言い換えと翻訳には事例ベース推論を用いており, ユーザは言い換え事例を追加することができる.

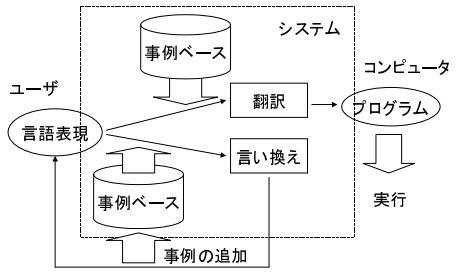

図 2: システム構成

言語表現からプログラムを生成するために, データベース中の言語表現とのパターンマッチングを行い, 類似した事例を検索する. 事例には翻訳事例と言い換え事例があり, それぞれ言語表現とプログラム, 言語表現と言語表現の対応関係を記述したものである. そして, パターンマッチングの結果を基に言い換え, または翻訳を出力する. なお, 本システムは, テキストエディタ GNU Emacs の Windows 移植版である Meadow Version 2.00pre1 上で, Emacs Lisp を用いて実装している.

3.1 構文解析

類似事例の検索では, 入力された言語表現に対して形態素解析と構文解析を行って構文木を生成し, 構文木に基づいて事例に含まれる言語表現と比較する. 形態素解析には茶筅 [9] を用いており, 分かち書きされていない日本語テキストから品詞, 活用型, 活用形*1, 基本形, 読みが付加された形態素列が得られる. 構文解析では, 文献 [10, 11] を参考に, 表 1 に示す文脈自由文法を定義し, これらの文法規則に基づいて複数の語句から構文木を構成する. 構文解析の結果として, 句には特徴が付加される. 特徴は, 句に含まれる語の属性と, 句を生成する際の文法規則から求められる. 語の属性から得られる特徴には, 表 2 に示すものがある. これらの特徴は, より上位の句へと継承され, 最終的に節全体の特徴として統合される. 一方, 名詞句と格助詞から補足語を生成する文法規則が実行された場合, 句には「を格」「に格」などの文法的な特徴が付加されるが, これは上位の句へは継承されない. これらの特徴は, 後にパターンマッチングと出力生成の際に使用される.

3.2 パターンマッチング

構文木が得られると, 入力された言語表現に対して事例に含まれる言語表現とのパターンマッチングを行い, 入力された言語表現に類似した事例を検索する. パターンマッチングは以下のアルゴリズムに従って行う.

- 節同士では, 述語と補足語に分けて比較する. ここで, 補足語とは文の組み立てにおいて述語を補う働きをする部分のこと [11] で, 具体的には, 「〜を」「〜に」のように「名詞+格助詞」の形式で表される部分のことである.
- 句同士では, 句に含まれる語を先頭から順に比較する. ただし, 名詞句同士は最後の名詞を, 動詞句同士は最初の動詞を比較し, 一致すれば全体として一致したものとする.

*1 活用型とは五段活用などの活用の種類を表すものであり, 活用形とは終止形や未然形のような活用における形を表すものである.

表 1: 文法規則

文法規則		
名詞	→	名詞句
動詞	→	動詞句
名詞句 (連体化)+名詞句	→	名詞句
名詞句+格助詞	→	補足語
名詞句+判定詞	→	述部
名詞句+助詞	→	名詞句
連体詞+名詞句	→	名詞句
数+助数詞	→	名詞句
サ変名詞+する	→	動詞句
サ変名詞+。(文末)	→	動詞句
動詞句+非自立動詞	→	動詞句
動詞句+助詞	→	動詞句
動詞句	→	述部
述部+助動詞	→	述部
補足語+述部	→	節
補足語+節	→	節
接続詞+節	→	節
副詞+節	→	節
名詞 (副詞可能)+節	→	節
感動詞	→	節
節+名詞	→	名詞句
節	→	文
節+文	→	文

表 2: 語の属性から得られる特徴

品詞	属性	値	得られる特徴
助動詞	活用形 活用型	仮定形 特殊・ナイ	仮定, 従属, 接続 否定
助詞	品詞 品詞	副助詞／並立助詞／終助詞 接続助詞	疑問 接続
副詞	基本形	すべて ｜ 全て ｜ 全部	すべて

- 語同士では, 品詞および基本形の読みを比較する. これは漢字変換の違いでマッチングに失敗するのを防ぐためである.
- 事例に含まれる可変部分を変数と呼び, 変数は型が一致する任意の単語または句と一致する. 変数の型は数値と文字列の 2 種類がある.

パターンマッチングにおいてパターンに含まれない語句が入力された場合, その語句は無視される. これは, 例えば『行頭が?x<文字列>だ』というパターンと『もし行頭が「-」ならば』という言語表現を, 『もし』という余分な語の有無に関わらずマッチングさせるためである. 逆にパターンに存在する語句が入力に存在しない場合は, その入力はパターンに対して不十分であるため, マッチング不成功とする. パターンマッチングの結果, 一致した事例の候補が複数得られた場合は, パターンマッチングで完全に一致しなかった部分の個数 (不一致数) が少ないものを優先する. 完全に一致すれば不一致数は 0 である. 不一致数が等しい候補が複数あった場合は変数の数が最も少ないものを選択する.

3.3 出力の生成

翻訳事例とのパターンマッチングによって類似事例が得られた場合, 事例の翻訳結果に含まれる変数に入力された言語表現の対応する部分を代入し, プログラムとして出力する. この処理を変数の解決と呼ぶ. 翻訳に用いられるデータベースは表 3 のような形式であらかじめ用意されている. 変数は, 言語表現では『?x<型>』のように表され, 翻訳結果として出力される対象言語では `lambda` 式 (無名関数) のパラメータとして表現される.

複数の節を含む複文の翻訳では, 節の特徴が参照される. 節の特徴に (仮定, 従属) という特徴が含まれる場合, 従属節は条件を表す式に変換される. この場合, 節の特徴に (すべて) が含まれていれば全体としては while 節に翻訳され, 含まれていなければ if 節に翻訳される. これらの特徴を持たない節は並列節

となり, それぞれの節の翻訳結果を順に連結したものが全体の翻訳結果となる.

表 3: 翻訳事例データベース (抜粋)

	言語表現	翻訳結果
1	左に移動する	(lambda nil (backward-char))
2	右に移動する	(lambda nil (forward-char))
3	?x<数字>文字左に移動する	(lambda (x) (backward-char x))
4	?x<数字>文字右に移動する	(lambda (x) (forward-char x))
5	前の行に移動する	(lambda nil (previous-line 1))
6	次の行に移動する	(lambda nil (next-line 1))
7	?x<数字>行上に移動する	(lambda (x) (previous-line x))
8	?x<数字>行下に移動する	(lambda (x) (next-line x))
9	前の単語に移動する	(lambda nil (backward-word 1))
10	次の単語に移動する	(lambda nil (forward-word 1))
11	行頭に移動する	(lambda nil (beginning-of-line))
12	行末に移動する	(lambda nil (end-of-line))
13	文頭に移動する	(lambda nil (goto-char (point-min)))
14	文末に移動する	(lambda nil (goto-char (point-max)))
15	?x<文字列>を検索する	(lambda (x) (re-search-forward x nil t))
16	?x<文字列>を?y<文字列>に置換する	(lambda (x) (replace-regexp x y))
17	?x<文字列>を挿入する	(lambda (x) (insert x))
18	?x<数値>文字削除する	(lambda (x) (delete-char x))
19	選択を開始する	(lambda nil (set-mark-command nil))
20	選択を解除する	(lambda nil (deactivate-mark))
21	選択範囲を削除する	(lambda nil (and mark-active (delete-active-region)))
22	行頭だ	(lambda nil (bolp))
23	行末だ	(lambda nil (eolp))
24	文頭だ	(lambda nil (bobp))
25	文末だ	(lambda nil (eobp))
26	コピーする	(lambda nil (kill-ring-save (point) (mark)))
27	貼り付ける	(lambda nil (yank))
28	切り取る	(lambda nil (kill-ring (point) (mark)))
29	行頭が?x<文字列>だ	(lambda (x) (save-excursion (beginning-of-line) (looking-at x)))
30	行末が?x<文字列>だ	(lambda (x) (save-excursion (beginning-of-line) (looking-at (concat ".*" x "$"))))

言い換えも翻訳と同様に行うことができるが, 出力がプログラムではなく言語表現であるという点が異なる. また, 複文のうち 1 つでも言い換え事例に一致する節があれば, 全体としては言い換えを行うことになる.

3.4 言い換えの追加

パターンマッチングによって類似事例が見つからなかった場合, ユーザは事例中に存在しない言語表現を用いたものと考えられる. この場合, その言語表現の意味はシステムには理解できないため, ユーザに別の表現に言い換えてもらうことで翻訳を続行しようとする. これは言い換え事例の追加にあたり, 追加された言語表現はその後の対話で使用することができる. 言い換え事例の追加を行う対話例を図 3 に示す. このように, 本システムは使用に先だって定義を必要とするこれまでのプログラミング言語とは対照的に, 使用することよってその言語表現の意味を定義するという考え方に基づいており, より自然言語に近い性質を持っているといえる.

```
> この行を引用する
候補なし: 『この行を引用する』を言い換えてください

> 行頭に「> 」を挿入する
実行しました: 『行頭に「> 」を挿入する。』
(progn
  (beginning-of-line)
  (insert "> "))
==>nil

>
```

図 3: 言い換え事例の追加

4. 実験

本システムの有効性を検証するために実験を行う. 実験は, 図 4 に示す編集前の文章を, 図 5 に示す編集後の文章に変換するための言語表現を入力することで行われる. 被験者には (1)JPEG ファイルは IMG タグに, AVI ファイルは A タグに変換したい (2) それ以外のファイル (~や.BAK で終わるファイル, Thumbs.db) は不要なため消したい, という要求を持って実験を行ってもらう.

```
photo(花見 2004)/hanami2004-1.JPG
photo(花見 2004)/hanami2004-1.JPG~
photo(花見 2004)/hanami2004-2.JPG
photo(花見 2004)/hanami2004-3.JPG
photo(花見 2004)/hanami2004-4.JPG
photo(花見 2004)/hanami2004-5.JPG
photo(花見 2004)/hanami2004-6.AVI
photo(花見 2004)/hanami2004-7.JPG
photo(花見 2004)/Thumbs.db
photo(卒業式 2004)/DSCF1.JPG
photo(卒業式 2004)/DSCF2.JPG
photo(卒業式 2004)/DSCF3.JPG
photo(卒業式 2004)/DSCF4.JPG
photo(卒業式 2004)/DSCF5.JPG
photo(卒業式 2004)/DSCF6.JPG
photo(卒業式 2004)/DSCF6.JPG.BAK
photo(卒業式 2004)/DSCF7.JPG
photo(卒業式 2004)/DSCF8.JPG
photo(卒業式 2004)/DSCF9.JPG
photo(卒業式 2004)/DSCF10.JPG
photo(卒業式 2004)/Thumbs.db
```

図 4: 編集前の状態

```
<IMG SRC="photo(花見 2004)/hanami2004-1.JPG"><BR>
<IMG SRC="photo(花見 2004)/hanami2004-2.JPG"><BR>
<IMG SRC="photo(花見 2004)/hanami2004-3.JPG"><BR>
<IMG SRC="photo(花見 2004)/hanami2004-4.JPG"><BR>
<IMG SRC="photo(花見 2004)/hanami2004-5.JPG"><BR>
<A HREF="photo(花見 2004)/hanami2004-6.AVI">hanami2004-6.AVI</A><BR>
<IMG SRC="photo(花見 2004)/hanami2004-7.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF1.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF2.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF3.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF4.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF5.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF6.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF7.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF8.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF9.JPG"><BR>
<IMG SRC="photo(卒業式 2004)/DSCF10.JPG"><BR>
```

図 5: 編集後の状態

被験者実験の結果を表 4 および表 5 に示す. この結果から, それぞれの被験者で異なった言い換え事例が獲得できていることがわかる. また, 被験者 1 で最終的に得られたプログラムは図 6 のようになり, これは編集前の文章の各行に適用することで当初の目標を達成できるものとなっているため, 本システムを用いてユーザの要求を満たすプログラムを作成できることが確認された.

5. おわりに

本稿では, エンドユーザを対象とした言語表現を用いたプログラミングシステムを提案した. 提案されたシステムでは, 言語表現による対話によってユーザの抽象的な要求を段階的に詳細化することで, 既存の言語知識のみでプログラムを作成することができる. 作成されたプログラムは言語表現で記述されるため, 既存のプログラミング言語を用いて作成したプログラムと比べて理解しやすい. 今後は, より自然な対話を実現するために, 対話インタフェースの改良を行っていく. 具体的には, 対話の状態を考慮した対話制御の方法について検討している. また, 現在は翻訳事例の数が十分とはいえず, 生成できるプログ

表 4: 得られた言い換え事例 (被験者 1)

	言い換え前	言い換え後
1	イメージタグ１を行う。	行頭に移動する。「<IMG SRC="」を挿入する。
2	イメージタグ２を行う。	行末に移動する。「"> 」を挿入する。
3	イメージタグをつける。	イメージタグ１を行う。イメージタグ２を行う。
4	イメージタグを付ける。	イメージタグをつける
5	リンクタグ１を行う。	行頭に移動する。「<A HREF="」を挿入する。「/」を検索する。
6	リンクタグ２を行う。	選択を開始する。行末に移動する。コピーする。「">」を挿入する。貼り。付ける。
7	リンクタグ３を行う。	行末に移動する。「</A>」を挿入する。
8	リンクタグを付ける。	リンクタグ１を行う。リンクタグ２を行う。リンクタグ３を行う。
9	１行削除する。	行頭に移動する。選択を開始する。行末に移動する。選択範囲を削除する。１文字削除する。
10	タグ判断２を行う。	行末が「AVI」なら、リンクタグを付ける。
11	タグ判断３を行う。	行末が「JPG」なら、イメージタグを付ける。
12	タグを挿入する。	タグ判断２を行う。タグ判断３を行う。
13	削除１を行う。	行末が「.BAK」なら、１行削除する。
14	削除２を行う。	行末が「~」なら、１行削除する。
15	削除３を行う。	行末が「Thumbs.db」なら、１行削除する。
16	余分な行を削除する。	削除１を行う。削除２を行う。削除３を行う。
17	処理を行う。	タグを挿入する。余分な行を削除する。

表 5: 得られた言い換え事例 (被験者 2)

	言い換え前	言い換え後
1	行頭に「<IMG SRC="」を挿入する。	行頭に移動する。「<IMG SRC="」を挿入する。
2	行末に「"> 」を挿入する。	行末に移動する。「"> 」を挿入する。
3	IMG タグを付ける。	行頭に「<IMG SRC="」を挿入する。行末に「"> 」を挿入する。
4	「JPG」だ。	行末が「JPG」だ。
5	イメージ処理をする。	「JPG」なら、IMG タグを付ける。
6	行末に「">」を挿入する。	行末に移動する。「">」を挿入する。
7	行頭に「<A HREF="」を挿入する。	行頭に移動する。「<A HREF="」を挿入する。
8	A タグの行末処理をする。	「/」を検索する。選択を開始する。行末に移動する。コピーする。行末に「">」を挿入する。貼り。付ける。
9	行末に「</A> 」を挿入する。	行末に移動する。「</A> 」を挿入する。
10	A タグを付ける。	行頭に「<A HREF="」を挿入する。A タグの行末処理をする。選択を解除する。行末に「</A> 」を挿入する。
11	「AVI」だ。	行末が「AVI」だ。
12	アンカー処理をする。	「AVI」なら、A タグを付ける。
13	タグ処理をする。	アンカー処理をする。イメージ処理をする。
14	この行を削除する。	行頭に移動する。選択を開始する。行末に移動する。選択範囲を削除する。選択を解除する。
15	「BAK」だ。	行末が「BAK」だ。
16	「BAK」の行を消す。	「BAK」なら、この行を削除する。
17	「db」だ。	行末が「db」だ。
18	「db」の行を消す。	「db」なら、この行を削除する。
19	「~」だ。	行末が「~」だ。
20	「~」の行を消す。	「~」なら、この行を削除する。
21	消す処理をする。	「~」の行を消す。「db」の行を消す。「BAK」の行を消す。
22	目標を達成する。	タグ処理をする。消す処理をする。

ラムの種類には限界があるため, 十分な量の翻訳事例を用意することは今後の課題である.

```
(progn
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "AVI" "$")))
      (progn
        (beginning-of-line)
        (insert "<A HREF=")
        (re-search-forward "/" nil t)
        (set-mark-command nil)
        (end-of-line)
        (kill-ring-save (point) (mark))
        (insert "">")
        (yank)
        (end-of-line)
        (insert "</A>")))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "JPG" "$")))
      (progn
        (beginning-of-line)
        (insert "<IMG SRC=")
        (end-of-line)
        (insert "¨><BR>")))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" ".BAK" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1)))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "~" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1)))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "Thumbs.db" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1))))
```

図 6: 生成されたプログラム (被験者１)

参考文献

[1] Henry Lieberman: 『Your Wish is My Command: Programming by Example』, Morgan Kaufmann Publishers (2001).

[2] テリー・ウィノグラード: 『言語理解の構造』, 産業図書 (1976).

[3] 原田実: “自動プログラミングの新パラダイム”, 人工知能学会誌, Vol.6, No.2, pp.19-23 (1991).

[4] 岩爪道昭, 小林一郎, 杉本徹, 岩下志乃, 高橋祐介, 伊藤紀子, 菅野道夫: “日常言語コンピューティング（第 2 報）– 日常言語に基づく計算機資源の管理・実行環境を目指して–, 人工知能学会論文誌, Vol.18, No.1, pp.45-56 (2003).

[5] David Price et al.: “NaturalJava: A Natural Language Interface for Programming in Java”, Proc. of 5th International Conference on Intelligent User Interfaces , pp.207-211 (2000).

[6] 瀬田和久, 池田満, 島輝行, 角所収, 溝口理一郎: “問題解決オントロジーに基づく概念レベルプログラミング環境 CLEPE”, 電子情報通信学会論文誌, D-II, Vol.J81, No.9, pp.2168-2180, (1998).

[7] 乾健太郎: “言語表現を言い換える技術”, 言語処理学会第 8 回年次大会チュートリアル, pp.1-21 (2002).

[8] 山本和英. “換言処理の現状と課題”, 言語処理学会 第 7 回年次大会併設ワークショップ 「言い換え／パラフレーズの自動化に向けて」論文集, (2001).

[9] 松本裕治 他: 形態素解析システム『茶筅』version 2.3.0 使用説明書, 奈良先端科学技術大学院大学 (2003).

[10] 吉村賢治: 『自然言語処理の基礎』, サイエンス社 (2000).

[11] 益岡隆志, 田窪行則: 『基礎日本語文法–改訂版–』, くろしお出版 (1992).