ソフトウェア工学 77−21
（１９９１. ２. ８）

ｕｎｉｘ日本語インタフェースとコマンド結合

渡辺孝弘、曽根崎健一、神山文子、藤田米春
大分大学工学部

ＵＮＩＸ日本語インタフェースにおいて必要な機能と構築上の問題点について述べる。自然言語インタフェースの構築においてまず考えなければならないことは、ユーザの立場からの仕事の概念の割り振りと計算機の側からの処理概念の割り振りすなわちコマンドシステムとの食違いである。第２の問題は、談話処理一般で問題になっている常識および文脈の処理である。第１の問題では必要な知識の表現と利用法について考察し、第２の問題では常識／文脈が要求処理にどのように係わるかを考察し、これらを考慮して構築中のＵＮＩＸ日本語インタフェースについても述べる。

A JAPANESE INTERFACE FOR UNIX AND CONSTRUCTION OF COMMAND COMPLEXES

Takahiro Watanabe Kenichi Sonezaki Fumiko Kamiyama Yoneharu Fujita

Faculty of Engineering, Oita University

Dan-noharu 700, Oita 870-11, Japan

This report describes some functions which are necessary to a Japanese interface for UNIX and some problems in implementation of them. The most important problem of natural language interface for an operating system is the difference between the conceptional partitioning of world of users and that of operating systems. The second problem is well known one of common sense and contexts in discource understandings. Some methods for expressing and using knowledges to resolve the first problem are described and also a plan for installing the common sense and context processings in a prototype interface system is described.

１．まえがき

１．１　要求文とシステムの応答

コマンドの概念を持たない人にとって抵抗のないシステムの応答は次のようなものであろう。

①ユーザ　：この前作ったファイルを見せて下さい。
②システム：（ユーザが指していると思われるファイルを検索、表示して）これでよろしいですか。
③ユーザ　：いいえ、これじゃありません。これより前に作ったファイルです。
④システム：（もう１度検索し直して）では、これですか？
⑤ユーザ　：そうです。それをコンパイルして下さい。
⑥システム：（このファイルをコンパイルして）コンパイルしましたが、１８行と４７行にエラーが出ました。
⑦ユーザ　：もう１度見せて下さい。
・・・・・・・・・・・・・・・・・・

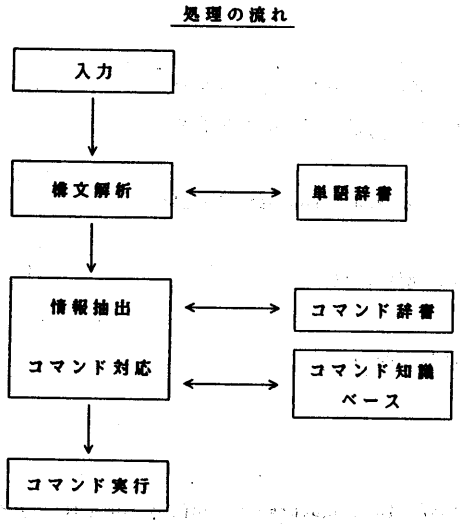

図１　日本語要求文処理システムの処理の流れ

一般に、ユーザとシステムの対話におけるユーザの要求は上例の①③⑤⑦の文のように、不完全なものである。そこで②④⑥のような応答や問い合せを行うことになる。また、要求がシステムのコマンド体系と１対１になるとは限らず、単文一つの要求文でもいくつかのコマンドの組み合わせになることもある。したがって、自然言語による処理要求においては、単文で表される要求の解釈およびこれに対応するコマンドの組み合わせを生成する問題と、代名詞や省略のような単一の文の範囲では処理できない常識や文脈に係わる問題とがある。[1] ここでは現在構築中の図１に示すＵＮＩＸ日本語インタフェースで問題になった点について述べる。

２．要求文とコマンド

２．１　要求文

１．１で示したように自然言語の処理要求は一般に曖昧でありかつコマンド体系を念頭に置かずになされるものである。例えば、処理についての次のような要求がされることがある。

a.「ディレクトリdirectの下のファイルリストをすべて詳しく見たい」
b.「ファイルfile1とfile2を一つにまとめて印刷してください」
c.「ディレクトリdirectの下のディレクトリのみを見たい」
d.「ユーザfujitaのホームディレクトリはどこにありますか」

a.の場合、文としては単文であり、コマンドとしても "ls -al direct" のように一つになりオプションパラメータ " -al" には修飾詞「すべて詳しく」が対応している。したがって、単文を一つのコマンドに対応させるシステムではうまく処理できるが、そのようなシステムは、c.、d.に示すような、単文が一つのコマンドにならない場合には処理できない。とくに、文が「ＡのＢ」の形の句を含む場合さまざまな解釈が可能で、解釈によって複数のコマンドに対応することも多い。たとえば上記の「directの下」の場合、「direct」がディレクトリであって「下」が位置に関する名詞であるので構造または関係を表すと考えられるが、「ファイルの一覧」といった場合には、「一覧」はコマンドに対応するので、文全体として複数のコマンドに対応することもある。一般には、「ＡのＢ」の形の表現はＢが構成要素または属性を表すとき、Ｂが何等

かの意味で関数になることが多い。ただし、「最大のファイル」のように、Aが属性を表す場合には、Aが関数になる。

b.の場合、文としては重文になっており、"cat file1 file2 | lpr" のように自然にｕｎｉｘのパイプ機能を使用することが予想される。しかし、c.、d.の場合には文としては単文であり、1段(ls -al direct | grep "^d")あるいは2段(ypcat passwd | grep "^fujita" | awk -F":" '{print $6}')のパイプ機能を使用することを予想できない。以下では、これらの場合の処理についても検討する。

2．2　文形とコマンド

2．1　で示したように、文形がコマンドの形を反映している場合とそうでない場合がある。

（1）単文が一つのコマンドに対応する場合。

修飾詞がオプションパラメータに対応する：

2．1のa.の例

必要な情報と知識

○ <ディレクトリ>の<上下>のX

<上下>=上 → <ディレクトリ>の上のXを再帰的に探す。

<上下>=下 → <ディレクトリ>の下のXを再帰的に探す。

○「ファイルリストを見る」。

コマンドはｌｓ。

○コマンドがｌｓの場合「すべて」はオプション－ａ。

○コマンドがｌｓの場合「詳しく」はオプション－ｌ。

（2）複文が複数のコマンドに対応する場合。

①各構成要素の単文が一つのコマンドに対応し、一本のパイプになる例：

「ディレクトリdirectの下のオーナがfujitaになっているファイルをリストしてください」

(ls -al direct | awk '$3=="fujita" {print}')

必要な情報と知識

○ｌｓコマンドは－ａｌオプションで実行すると「オーナ」、「ファイル名」を含む情報を出力する。「オーナ」は3列目に出力される。

○ａｗｋはコマンド出力のフィルタである。

ａｗｋコマンドの起動ｆｏｒｍａｔ。

○コマンドの出力の第3項目はａｗｋの変数$3にはいる。

○パイプについての知識。

②各構成要素の単文とコマンドは対応するが、一本のパイプにならない場合。

（3）重文が複数のコマンドに対応する場合。

①各構成要素の単文が一つのコマンドに対応し、一本のパイプになる例：

「ディレクトリdirectの下のファイルをまとめて印刷してください」

(cat direct/* | lpr)

必要な情報と知識

○「まとめる」の対象が「ディレクトリの下ファイル」なら「ディレクトリの下のすべてのファイル」を指す。

○ｃａｔコマンドはファイルをまとめ標準出力に出力する。

○「すべてのファイル」はワイルドカード＊で表される。

○ｌｐｒコマンドは標準入力からの文字列を「印刷」する。

○パイプ「｜」についての知識。

②各構成要素の単文とコマンドは対応し、単なる逐次処理になる例：

「ファイルFileを表示し、印刷しなさい」

(cat File; lpr File)

必要な情報と知識

○ｃａｔコマンドは引数ファイルを「標準出力装置に出力(表示)」する。

○ｌｐｒコマンドは引数ファイルを「印刷」する。

○逐次処理「；」についての知識。

③各構成要素の単文とコマンドは対応するが、一般的なシェルスクリプトになる場合。

「ディレクトリdirectの下のファイルがディレクトリならばその下のファイルリストを表示し、通常のファイルならばその内容を表示しなさい」

(ls -al | (if (^$1=="d") then "ls -al
$8" else "cat $8")))

必要な情報と知識

○「ファイルがディレクトリならば」からファイル情報が必要であること。

○ｌｓは引数ディレクトリの下のファイルリストを出力する。

○ｃａｔは引数ファイルの内容を出力する。

（４）文形とコマンドが対応しない場合。

これは、（１）で修飾詞の付き方でパイプ処理の要／不要が決まる場合などである。

これらの場合で、文がコマンドまたはパイプ構造に対応する(1)、(2)①、(3)① の場合には、まず、各単文に対応するコマンドを次に述べるコマンド知識ベースより探し出し、これに修飾詞に基づく適当なオプションパラメータを設定する。この結果作成されるパイプの実行結果の記述がユーザの目的に合致すればこれを実行する。合致しなければ、(4) の場合とみなし、コマンドの入出力記述をもとにコマンドを結合してパイプを構成する。そのような、パイプが構成できないとき、(2)②、(3)③ および (4)の一般の場合と見て関連するコマンド知識をユーザに提示する。

２．３　コマンド知識ベース

上記のように、コマンドやコマンドのパイプを生成するには、さまざまな知識が必要である。その主なものは、コマンドの起動と入出力データのシンタックス、セマンティクスなどである。この知識をコマンド知識ベースとして蓄える。コマンド知識は次の形とする

```
[コマンド名
 [C,[構造,ﾊﾟﾗﾒｰﾀｼﾝﾀｯｸｽ],
    [意味,内容のｶﾃｺﾞﾘ/意味記述] ]
 [I,[構造,ﾃﾞｰﾀｼﾝﾀｯｸｽ],
    [意味,内容のｶﾃｺﾞﾘ/意味記述] ]
 [O,[構造,ﾃﾞｰﾀｼﾝﾀｯｸｽ],
    [意味,内容のｶﾃｺﾞﾘ/意味記述] ] ]
```

ここに、Ｃ、Ｉ、Ｏはそれぞれコマンドの呼び出し、入力、出力に関する知識のスロット名である。

[例]

```
[ls
 [C,[構造,"ls -[a][l] [<ファイル名>]"]
 [I,[構造],[意味]],
 [O,[構造,
      [if(prameter=="-l")
       then Line("<モード> <リンク数> <オーナ>
                  <サイズ> <月> <日> <時刻｜年>
                  <ファイル名>") ] ],
    [意味,Mode(<モード>),Links(<リンク数>),
          Owner(<オーナ>),Size(<サイズ>),
          Month(<月>),Day(<日>),
          Time(<時刻｜年>)\/Year(<時刻｜年>)")
 ] ]
```

３．単一の要求文の処理

３．１　要求文の構造と生成規則

一般の世界での自然言語全体の扱う文法は、ＵＮＩＸのコマンド世界を扱うのには範囲が広すぎる。ここでは、sub-languageを抽出して専用の処理法を考える。まず、実際のユーザとして本研究室の学生を対象に、単一の要求文の収集を行った。その数は約１００である。ただしそれらの文の語尾は「～したい」の形に限定し、それらの文から考えられる言い換えや類義語を考慮した要求文を新たに加えて整理した。その結果、要求文は基本的に次の３つの構造をとることがわかった。

構造１：<対象１><を><動詞>。

構造２：<対象１><の><動名><を><したい>。

構造３：<対象１><を> <対象２><に><動詞>。

ここで、<対象１>、<対象２>には「ファイル」や「ディレクトリ」などの操作の対象となる語が入る。<動詞>には「見たい」や「コンパイルしたい」などの操作を表す動詞句が入り、<動名>には<動詞>と同じ働きをする「削除」や「コンパイル」などの操作を表す動詞の名詞形、あるいはサ変動詞の語幹が入る。<を>には助詞の「を」または「が」、<に>には「に」あるいは「で」、<の>には「の」がそれぞれ入る。

さらに、<対象１>、<対象２>は次のような正規表現

で表されるような繰り返しでもよい。

<対象１>[<と><対象１>]*

<対象２>[<と><対象２>]*

また、<対象１>、<対象２>の後には、「名前」や「内容」などの<対象>の内容を表す<名内容>と、「リスト」や「タイプ」など<対象>の属性の中で操作の目的語となる<目的語>が入ってもよい。したがって、<対象１>、<対象２>は次のようになることがある。

<対象物１><の><名内容>

<対象物１><の><目的語>

<対象物２><の><目的語><の><名内容>

・・・・・・・・・・・・・・・

さらに、それぞれのカテゴリの前には修飾語を表す<修飾子>が付くこともあり、<修飾子>は繰り返し現れてもよい。

以上のことから１～３の構造を解析するための文法を作成した。その一部を次に示す。

<要求文>→<対象部><操作部>

<対象部>→<対象１><を>

<対象部>→<対象１><を><対象２><に>

・・・・・・・・・・・・・・・

この生成規則で要求文を解析することによって、要求文から構文的かつある程度の意味情報を含んだ解析結果を得ることが出来る。

３．２　要求文解析結果とコマンドの対応

次に、生成規則による要求文の解析結果から、意味のまとまりや同義語などを考慮しながら、コマンド決定に必要な情報を取り出し、その情報から対応するコマンドを導く。そのとき、コマンド決定に必要な情報とコマンドを対応付けるコマンド辞書を用意する。その一例を次に示す。

```
[[[動詞,見る],[対象物１,Ｘ],[名内容,内容]],
 [命令,if(File(Ｘ))
       then if(Text(Ｘ))
            then "cat Ｘ"
            else if(Binary(Ｘ))
                 then "od -x Ｘ"] ]
[[[動詞,表示する],
  [対象物１,ファイル],[目的語,リスト] ],
  [命令,
   "ls ",if(詳しく∂modifier([動詞,表示する]))
          then {"-l";
                if(すべて
                   ∂modifier([動詞,表示する]))
                then "-a" } ] ]
[[[動詞,コピーする],[対象物１,Ｘ],[対象物２,Ｙ]],
 [命令,"cp Ｘ Ｙ"] ]
```

以上述べてきたように、本システムのおおまかな処理手順は要求文を生成規則を使って解析し、その解析結果とコマンド知識ベースとのマッチングによりコマンドを導き出すものである。

３．３　実用的なシステムへの考慮

１．および２．で述べたように、ユーザの要求文とコマンドが一意に対応しない場合も多い。現在コマンド辞書としては、実際に１５個程度の簡単なコマンドに対応するものが用意されている。しかし、これだけでは１つのコマンドを１つの文で言い換えたにすぎず、実用的なシステムとは言い難い。２．で示したようなコマンド結合処理の他に、自然言語の持つ曖昧性、コマンドの曖昧性についての処理が必要である。

次の４．で、このシステムにおける言語の曖昧性処理、コマンドの曖昧性処理について言及する。

４．常識、文脈処理

このシステムにおける情報補填と曖昧性の処理について述べる。また、１．１節で示したような対話形式における処理の問題（代名詞、省略など）も考察する。

４．１　常識処理

曖昧な文の例

①file1をコピーしたい。

②file1とfile2とfile3をfile4とfile5にコピーしたい。

③file1の名前をfile2にコピーしたい。

④file1をしたい。

⑤file1で削除したい。

上の①～③は構文論的には曖昧ではない。しかし、意味論的／語用論的には曖昧である。①ではコピー先が欠けているし、②では数が合っていない（file1をfile4に、file2をfile5にコピーすると、file3が残る）か、file1とfile2とfile3をまとめてfile4とfile5にコピーするのか一意に決まらない。③においてはそのままでも実行できるが、このようなことを要求するのは非常にまれである。この場合、この文の意味することは次のようなものであるかもしれない。

「file1の中の名前のリストをfile2にコピーしたい。」

④、⑤も曖昧である。④については、file1 の何をするのか分からないからである。しかし、人間が常識的に考えるとまず第一に一般的な可能性として、「file1を実行をする」、「file1をコンパイルをする」などが推論される。そしてさらに、前後の状況によってそのほかのことも推論されるかもしれない。⑤については、ユーザの間違いの方の可能性の方が高いであろう。しかし、この場合でもユーザが何か入力することには必ず意図があるはずであるから、その部分をなるべくくみ取る解釈をする必要がある。例えばこの場合なら、「で」は「を」の間違いで、「file1を削除したい」や、または、「file1の中で、ある部分を削除したい。」などである。

このように、本システムに曖昧な入力があった場合、コマンド決定に欠けているところは入力し直してもらったり、１つの文で何通りにもの解釈の仕方がある場合は、その入力から考えられる解釈を推論して可能性の高いものからユーザに聞き返して、ユーザに選択してもらう形式を取っている。その動作例を、次に示す。

ユーザ　：file1を変更したい。
システム：内容の変更ですか？（Ｙ／Ｎ）
ユーザ　：Ｎ。
システム：では、名前の変更ですか？（Ｙ／Ｎ）
ユーザ　：Ｙ。
システム：変更先の名前を入力して下さい。
ユーザ　：file2。
システム実行（mv file1 file2）

４．２　文脈処理

４．１のように要求の曖昧性は問い合せをすることにより解消可能であるが、あまり多くをユーザに聞き返すのはかえって実用的ではない。したがって文脈を考慮することによって、いく通りにも推論できる解釈を絞り込んだり、問い合せを効率的にする必要がある。

４．１の例で、第１のユーザ入力の直前に、file1をコンパイルしてエラーが出ていれば内容の変更すなわちファイルの編集の可能性が高いが、viにおいてサブコマンド「w file1」を用いて「File exists - use "w! file1" to overwrite」のメッセージが出ていればファイル名の変更の可能性が高い。したがって、これらの文脈により、「名前の変更ですか？」と「内容の変更ですか？」の問い合せの順序を変更して問い合せの数が少なくなるようにすることが望ましい。

曖昧な入力に対するいく通りもの解釈は、今のところ人間があらかじめ予想したいくつかの曖昧な入力についての解釈をデータとして与えている。しかし、人手によるデータ作成と入力には限界があり、なんらかの自動化が必要である。

５．むすび

自然言語インタフェースは柔軟性が重要であり、このために必要となるコマンド結合機能、常識処理機能、文脈処理機能について述べた。現在、テストプログラムをｐｒｏｌｏｇを用いて作成しており、単文からコマンドを生成実行する部分を作成し改良中である。文法を含むプログラムは約７００行であり、単語辞書はファイル処理関係のもの約２６０語、コマンド辞書は約１００行である。コマンド知識ベースは現在作成中である。

参考文献

[１]R. Wilensky, Y. Arens & D. Chin,"Talking to UNIX in Enhglish: An Overview of UC",CACM,Vol.27, No.6,pp.574-593(1984).

[２]M. Helander,"Handbook of Human-Computer Interaction", North-Holland(1988).