ドメイン知識に基づくUNIX日本語インタフェースの構築

久保康司*・西島恵介**・藤田米春**・田中稔***

Construction of Japanese Interface for UNIX System using Domain Specific Knowledge

Kouji KUBO, Keisuke NISHIJIMA, Yoneharu FUJITA and Minoru TANAKA

Abstract

One of the current research for human-computer interface is natural language interface. It reduces users' amount of study of computer systems and helps users with little knowledge of computer to use computer systems.

This paper describes the construction of a Japanese user interface for UNIX file handling operations. The user interface accepts Japanese sntence and generate a sequence of UNIX commands. This system includes specific knowledge of several domains: knowledge of functions of UNIX commands, methods for command combinations, and functions of Japanese words on a specific domain. Domain specific knowledge is used in two part of the system, the syntax and semantics analizer and the code generator. The syntax and semantics analizer selects an appropriate description for each word from the dictionary based on word restrictions and generates internal representations for the input Japanese sentences. The code generator transforms an internal representation into a UNIX command sequence, and when needed, uses pipe and redirection to modify the commands. During code generation, the system checks the commands execution results based on command dictionary description.

1 はじめに

計算機システムの利用に必要な努力を軽減させ，計算機システムに対する知識の少ないユーザでも簡単に使用できるインタフェースの１つの形態として，自然言語を使用したインタフェースが研究されている．

自然言語を利用したユーザインタフェースは主にデータベース検索において研究されていた [1]．これ以外にもUNIXシステムを対象としたシステムでは，ユーザに対してコンサルタントを行うシステム [3]，日本語の要求文に対して単一のUNIXコマンドを生成し，実行するシステムなどが研究されている [2]．

本システムは，UNIXコマンドに間する知識，コマンドの組み合わせに関する知識，特定の操作対象を扱う場合の語彙に関する知識をドメイン知識として使用し日本語の要求文からUNIXコマンド列を生成し実行する．この時，要求文に応じてパイプ，リダイレクトを使用してコマンドの加工も行う．また，コマンド生成時にコマンド辞書の記述を基に，コマンドの実行がシステムに与える影響をチェックするため，複数のコマンドによって実現される要求文からもコマンド列を生

*大学院システム工学専攻
**大分大学工学部知能精報システム工学科
***知能情報システム工学科

成することが可能である.

以下, 2ではUNIX日本語インタフェースの構成とその動作の概要について, 3では要求文の構文・意味解析の手法について説明する. 4では本システムで使用する内部記述の表記について説明し, 5ではこの記述を用いて作られた要求文の内部表現からコマンド列を生成する方法を述べ, 6でまとめを行う.

2 UNIX日本語インタフェースの概要

本システムは, UNIXシステムを使用してファイル操作を行う場合に, ユーザのUNIXコマンドの持つ細かい構文や出力に関する学習を不要にし, 単一または, 複数のコマンドで実現される操作を日本語を用いた要求文によって可能にする事を目的としている. このため, 本インタフェースを使用するユーザは, UNIXシステムを使用して実行可能な基本的な事柄,「ファイルは削除可能である」,「ファイルは複写可能である」の様な単一のコマンドで実行可能な事柄をあらかじめ学習している事を仮定している. ユーザから与えられる要求文と生成されるコマンドの関係において, 要求文を構成する1つの動詞句は, 基本的に1つコマンドに対応するものと考えて, 要求文の解釈, コマンド生成を行う.

本システムは, UNIXシステムのフロントエンドとして使用され, ファイル操作に関する処理要求文, 例えば「ファイルABCのサイズを表示して下さい.」を受け取り要求を満たすコマンド, "ls-il./ABC | awk'{print $5}'"を生成し, その実行をUNIXシェルに依頼する.

受理可能な入力文は,

- 単文による要求文.
- 「～して～する.」,「～した後～する.」の形の複文による要求文.
- 「～した～を、する.」の形の重文による要求文.

であり, コマンド生成可能な要求文は,

- 複数のコマンドの連続実行.
- リダイレクトによってコマンドの出力場所の変更.
- パイプを使用してフィルタコマンドを接続したコマンド.

によって実現可能な要求文である.

システムの構成をFig1に示した. 各モジュールの機能を以下に説明する.

入出力モジュール：ユーザからの入力文の取得と, コマンドの実行結果などの応答を表示する.

構文・意味解析モジュール：取得した要求文の構文・意味解析を行い内部表現を生成する.

コマンド生成モジュール：個別のコマンドの機能を記述したコマンド辞書を参照して, 構文・意味解析処理で生成された内部表現からコマンド列を生成する.

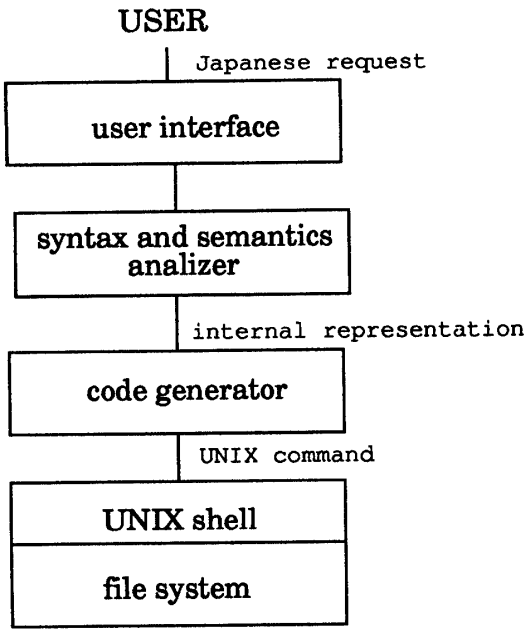

Fig 1：Software organization of Japanese interface for unix

3 構文・意味解析

本システムの構築に先だって収集した要求文の語数は長いもので10程度であったため, システムが対象とする要求文は語数が10程度の比較的短いものを想定している. 収集した要求文によると,「ファイルAを画面に表示する.」や「表示する, ファイルAを画面に」,「画面に表示する, ファイルAを」のような語順を入れ換えた文も多く使用され, これらの文は同じ操作を意図している.

構文・意味解析において, 要求文から簡単に必要な構造を生成するために, 以下の点を考慮する.

- 語順の違う文も同様に取り扱う.
- 構文解析と意味解析を同時に行う.
- プログラマが語彙毎に構文・意味解析の結果を制御できる.

上記の3つの文はFig2に示したように, 動詞「表示する」が格助詞〝を〟の直前の名詞句「ファイルA」を対象格に, 格助詞〝に〟の直前の名詞句「画面」場所格としてすることによって, 文を構成する句の位置が変わった場合でも同じ記述を得ることができる. また,「ファイルA」, においては,「A」が, 名詞「ファイル」の直後に位置することからファイルの固有名と解釈す

る.

また各々の語彙の持つ意味はそれが使用されるドメインによって変化するためドメイン毎に語彙の機能を定義する必要がある.

このような句構造生成を行うために，文を構成する各々の語彙は他の語彙を伴って句構造を形成すると考え，各々の語彙毎に辞書情報として，句構造生成のベースとなる語彙自身の格構造，句構造生成に必要な他の語彙に関する制約と句構造生成のための規則を記述する．制約には，語彙の並びに関するものと語彙の意味範疇によるものを使用する．また語彙には使用されるドメインによって制約や句構造生成規則が異なるため，特定のドメイン（ファイル操作やディレクトリ操作など）での記述を個別に作成しドメインラベルによって管理する．また，構文・意味解析を高速化するために，ドメイン毎に句構造生成時に，語彙の機能をどのような順序で使用するかを定義する．各々の語彙の記述はドメイン毎に独立しており，プログラマが自由に定義できる.

Fig3-4に「ファイルAを画面に表示する」の構文・意味解析に使用する辞書情報およびドメイン知識を示した．Fig3中でword項がファイル，domain項がfileOpとなっている行は，ファイル操作領域で使用されるファイルに関する記述である．この記述において，sem. markerは語彙の意味範疇を，activityはこの行の語彙の記述を使用する順序を定義するためのラベルである．structure項目にはファイルに関する句構造を生成するために必要な属性が書かれており，句構造の型枠となるものである．restrict sem.，restrict pos. が語彙が句を構成する場合に必要な他の語彙に関する制約である．restrict sem. にはファイルが句を生成する場合に，自身と未知語を使用する事をリストで表現している．restrict pos. とmethodは共にrestrict sem. のリスト中の位置を使用して記述され，restrict semは自身(ファイル）と未知語がファイル，未知語の順で隣接して並ぶという制約を示している．methodは，ファイルの属性〝名前〟の値として，未知語を使用する事を記述している.

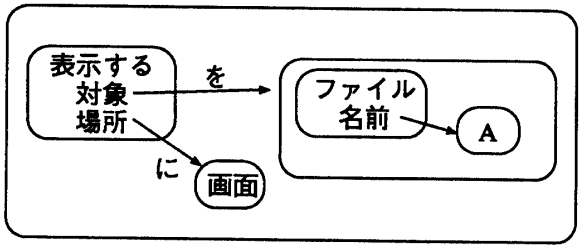

Fig 2：Case stucture of a Japanese sentence

Fig4にはドメイン知識として，入力文の解釈を行うドメインを決定するための条件，ドメイン名，構文・意味解析時にFig3の各々の語彙をどのような順序で使用するかを定義している.

構文・意味解析器はFig5に示すように，要求文の解釈を行うドメインの決定を行うドメイン選択器，格構造の生成を行う構文・意味解析器，格構造から内部表現に変換する変換テーブル，ドメイン知識，辞書情報によって構成される.

これらの辞書記述を使用した，格構造生成の手順を以下に示す.

1．入力文に対して最長一致法を用いて形態素解析を行い文を構成する語彙を抽出する.
2．Fig4のpredecessor to〝を〟項の語彙にしたがって，入力文を評価するドメインを決定する．(現在は，文中の動詞の対象格にあたる名詞によってドメインを定義しているため，格助詞〝を〟の直前の名詞によってドメインを決定している.)
3．決定されたドメインと同じdomain項を持ち，かつ入力文中語彙と同じword項をもつ語彙の辞書情報を使用して句構造を生成する．句構造の生

word	domain	sem. marker	activity	structure	restrict sem.	restrict pos.	method
ファイル	fileOp	記録物	1	[ファイル, 名前, 場所]	[this, 未知語]	seq(1,2)	setVal(1, 名前,2)
画面	fileOp, dirOp	場所	0	[画面]			
を	fileOp, dirOp	を	0	[を]			
に	fileOp, dirOp	に	0	[に]			
表示する	fileOp, dirOp	動詞	2	[表示する, 対象, 場所]	[記録物, を, 場所, に]	seq(1,2), seq(3,4)	setVal(this, 対象,1), setVal(this, 場所,3)

Fig 3：Example of word dictionary

predecessor to "を"	domain	application order of word function
ファイル	fileOp	[1,2]
ディレクトリ	dirOp	[1,2]

Fig 4 : Example of domain knowledge

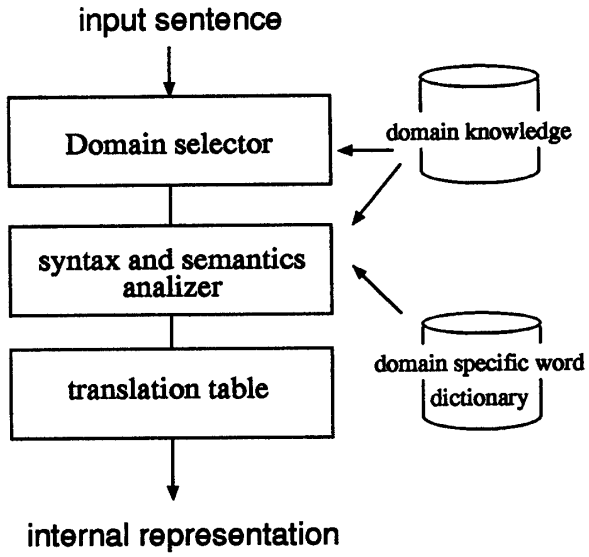

Fig 5 : Syntax and semantics analizer

成において要求文中の語彙は句構造を生成する働きを持つもの(能動語彙), 能動語彙が句構造を生成するために使用されるもの (受動語彙) の2つに分類される.

3の操作は, Fig4によって得られる語彙機能の適用順序に従いFig3の辞書情報を使用して以下の操作を行う.

1. 語彙機能適用順序の先頭の値と同じactivityを持つ辞書情報をもつ語彙を能動語彙として選択する.
2. 能動語彙は句構造を生成するために辞書情報のrestrict sem., restrict pos.を満足する受動語彙を選択する.
3. 能動語彙は辞書情報のmethodに基づい受動語彙を使用してて自身の句構造を書き換える.
4. 句構造生成に使用した受動語彙を入力文中から取り除く.

以後, 語彙機能適用順序にしたがって適用可能な全ての語彙の機能を適用した結果, 入力文中に残った語彙の辞書情報のstructure項に与えられた入力文の格構造が生成されている.

格構造の整形

このように生成された格構造は, 等価な要求文から生成されたものでも微妙に違ったものとなる. 同じ動詞を使用した等価な要求文について, 同じ処理を以後適用するために, 整形規則に従って格構造を整形し, 標準的な格構造に変換する.

```
matching template:
    [表示する,[対象,[ファイル,[名前,X],[場所,Y:.]]],
                [場所,画面]]
output description:
    out(content (file([name,X],[place,Y])),stdout)
```

Fig 6 : Structure translation table

内部表現の生成

構文・意味解析された要求文は構文的, 意味的に正しい事が保証されているため, この格構造を使用して内部表現を生成する. 生成には動詞の格構造とその内部記述の組からなる変換テーブルをFig6使用する.

要求文の格構造は, Fig6内のmatching templateに記述された格構造とユニフィケーションされる. この時, 必要な変数に対する値が要求文の格構造によって与えられない場合は, 変換テーブル内にコロンに続いて記述されたデフォールト値を使用して不足情報 (ファイルの存在する場所等) を補完される. ユニフィケーションによって, output descriptionに記述された内部表現の各々の変数に値が与えられ, これが要求文に対応する内部表現となる.

3.1 複数の要求を含む文の処理

収集した要求文は比較的短いものであったが,「ファイルABCにヘッダをつけて印刷する」, の様に2つ以上の操作を別々の単文で表現するのではなく, 重文を用いて1つの要求文で表現される例や,「ソートしたファイルXYZを表示する」, のように複文による要求文も多く含まれていた. これらの要求文のうち複数の要求を含む要求文についても考慮する.

重文の取扱い

「～して～する.」,「～してから～する.」,「～した後～する.」, の形の要求文, 例えば「ファイルABCにヘッダをつけて印刷する」は,「ファイルABCにヘッダをつける」,「ファイルABCを印字する」の2つの単文による要求文から構成されていると考える. この時, 動詞の連用形十〝て〞, 以下, +〝てから〞, +〝た後〞のような語彙の並びを文の切れ目とすることで, 重文→単文列変換を行う. 変換された, 単文列の間に順序関係を保存することで, 元の要求文の意味を保存する.

複文の取扱い

複文による要求文は,「ソートしたファイルABCを表示する」のように連体修飾句が特定の操作を要求するものと,「aを含むファイルの名前を表示する」のように非修飾句に条件を付加するものの2つに分類する事ができる．操作を要求する連体修飾句をもつ複文は単文列に分解し，各々の単文からコマンドを生成する．条件を付加する連体修飾句は修飾句は制約として使脚する．

重文／複文の単文列変換は，構文・意味解析の前処理として，形態素解析の後に行う．

4　操作対象，操作の記述

本システムの操作対象となるファイルをFig7に示した構造で定義する．Fig7中のapoラベルのついたリンクはa part of関係を表現しており，isaリンクはis a関係を示している．ノードの後にコロン以下は属性に対する制約を表現している．コマンド辞書記述において，ファイル，ディレクトリを指定する場合この構造を基に記述を行う．

特定のファイルやディレクトリは次の述語を使用して指定される．

file ([name, A], [place, P])
directory([name, A], [place, P])

ディレクトリhomeにあるファイルexampleを指定する場合は，

file ([name, example], [Place, home])

の様に記述する．また，Fig7に示したファイルの任意の属性を参照したい場合は，属性名を使用した関数で表現する．上で指定したファイルの内容を参照する場合は，

content (file ([name, example], [Place, home]))

と記述する．

システムに対する操作は内部表現として，プリミティブ動作を表現する手続き

out(Object, Place):ObjectをPlaceに出力する．

make (Object)　　:Objectを生成する．
delete (Object)　:Objectを削除する．
change(Object1,Object2) :
　　Object1をObject2に変更する．

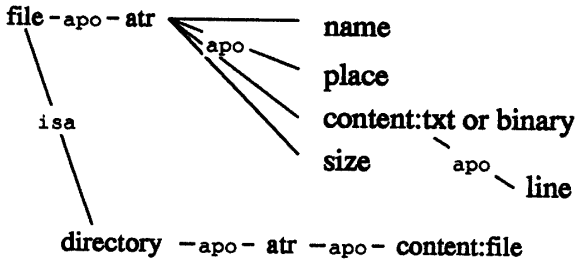

Fig 7：Tree structure of file

と，関数
sort(Object,Order,Key) :
　ObjectをKeyによってOrder順にソートする．

によって記述する．手続き，関数内のObjectにはファイル，ディレクトリの記述または，これらの属性を示す関数を記述する．これらの手続きと関数はファイル操作に関するUNIXコマンドの分析によって抽出された．

5　コマンドの生成

入力された要求文は構文・意味解析をへて内部表現変換された後コマンド列に変換される．コマンド生成機構はFig8に示すように，コマンド生成器，File System Simulator，およびコマンド辞書により構成される．

5.1　コマンド辞書

コマンド辞書にはFig9に示した各項目を記述する．UNIXコマンドは引き数によってコマンドの機能が変化するため，引き数の異なるコマンドは別のコマンドとして記述する．Preconditionにはこのコマンドを実行するために満たされなければならない前堤条件を対象の存在，非存在を示す述語Exist(File)，NotExist(File)を用いて記述する．InputStructureにはコマンドが入力を必要とする場合の構造を正規表現を用いて記述する．Actionは内部表現で使用したプリミティブ動作を表現する手続きおよび関数によって記述する．Output-Structureはコマンド実行によって出力されるテキストの構造で正規表現で記述する．Syntaxはこのコマンドを実行するためのUNIXコマンドである．Fig10はコマンド'cat P/A'の記述であり，コマンドを実行するためにはファイルAが存在している必要があること，コマンド実行によってファイルAの内容が標準出力に出力に出力される事，実行にはコマンドcat P/Aを使用する事を記述している．

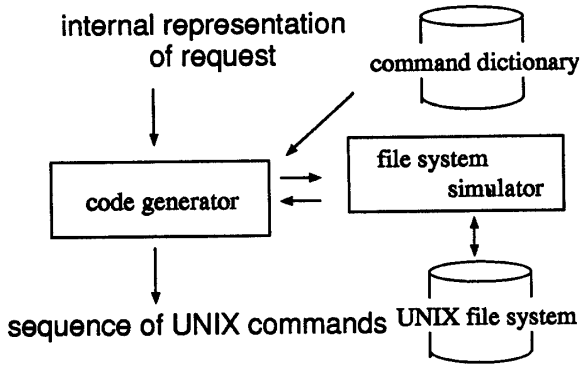

Fig 8 : Code generator

```
command(
  Name            : コマンドの一意名
    Precondition  : コマンド実行の前提条件
  InputStructure  : 入力を持つ場合のテキスト構造
  Action          : 実行による動作（手続き）
  OutputStructure : 出力を持つ場合のテキスト構造
  Syntax          : コマンド構文
)
```

Fig 9 : Description of command dictionary

```
command(
  Name            : cat 1
  Precondition    :
    Exist(file([name, A], [Place, P]))
  InputStructure  :
  Action          :
    out(content(file([name, A] [Place, P]))
      stdout)
  OutputStructure : text([*])
  Syntax          : cat P/A
)
```

Fig10 : Description of 'cat A' in command dictionary

5.2 コマンド生成器

コマンド生成器において，単一コマンドで実現可能な要求文は以下の手順に従ってUNIXコマンドに変換される．

1．要求文の内部表現と一致するaction記述をもつコマンドをコマンド辞書から候補コマンドとして選択する．
2．候補コマンドの前提条件が満たされているかどうかを後で述べるFSS:File System Simulatorに問い合わせて確認する．
3．前提条件が満たされたコマンドを生成コマンドとする．
4．以後のコマンド生成のために，生成したコマンドの動作をFSSに伝えコマンド実行時のファイルシステムの状態を模倣する．

複数のコマンドを連続して実行することで実現される要求文の内部表現からのコマンドの生成は，内部表現内の各々の動作手続きに対して順次上記の操作を行うことでコマンド列を生成する．この時，既に生成されたコマンドの実行によって変化するシステムの状態を現在のコマンド生成に反映させるため，FSS内の仮想ファイルシステムを使用してコマンドの実行条件の検査を行う．

上記の方法に加えて，コマンド生成の例外処理としてUNIXシステムの持つ2つの機能

●パイプ，リダイレクトを使用したコマンドの出力先の変更．

●awkフィルタを使用したコマンド出力の選択出力．

を使用した合成コマンドを生成し，コマンド生成能力を向上させる．これらの例外処理は候補コマンドの選択時に，要求文の内部表現と，コマンド辞書の記述の間の関係によって決定される．Table1-4に例外処理の条件となる，内部表現の記述と候補コマンドのAction記述，生成されるコマンドを示した．Table4は，コマンド出力の選択的表示に使用される．要求の内部表現が特定の属性（ファイルのサイズ等）の出力を要求している時に，要求された属性の上位にあたる属性を出力するコマンドが存在する場合，このコンドを用いて要求を実現する．この時属性間の上位／下位関係の検証は，前述のファイルの構造を参照して行われる．上位／下位関係が確認され，コマンドの出力項目が目的の属性を含んでいる場合，コマンドの出力項目から目的の属性の位置を計算し，この項目を出力するawkフィルタを接続する．

5.3 File System Simulator

File System Simulator（以下FSS）はFig11に示した構成をしており，コマンドの前提条件のチェックのためのファイルシステムの検査と，コマンド列生成時に以前に選択されたコマンドの影響を考慮した，正しい前提条件判定を行うために，コマンド実行に伴うファイルシステムの変化の予測を行う．このため内部にファ

Table 1 : Condition of add redirect

internal representation of request	out(Object, File2)
command description	out(Object, stdout)
generate commands	Command > File2

Table 2 : Condition of add pipe+lpr

internal representation of request	out(Object, printer)
command description	out(Object, stdout)
generate commands	Command | lpr

Table 3 : Condition of add pipe+sort

internal representation of request	out(Object, stdout)
command description	out(sort(Object,..), stdout)
generate commands	Command | sort

Table 4 : Condition of add pipe+filter command

internal representation of request	out(LowerObj, stdout)
command description	out(UpperObj, stdout)
generate commands	Command | filtercommand

イルシステムの変化を仮想的に実現する仮想ファイルシステムを持っている．仮想ファイルシステムは，現実のファイルシステム内ファイルの状況を部分的に記述した知識ベースで，過去に参照された各々のファイルについて，存在／非存在に関する記述を記録している．

FSSはこれらの機能をコマンド生成器に堤供するための２つのプロトコル

CheckFSS(*ExistFlag*, *FileType*, *FileType*, *FileName*, *Content*)

ChangeFSS(*ActionProcedure*)

を持っている．CheckFSSプロトコルは，FileType, FileName, Content, によって示されるファイルについてExistFlag (存在／非存在の値をとる) の示す状況をファイルシステムが満たしているかの問い合わせに使用される．

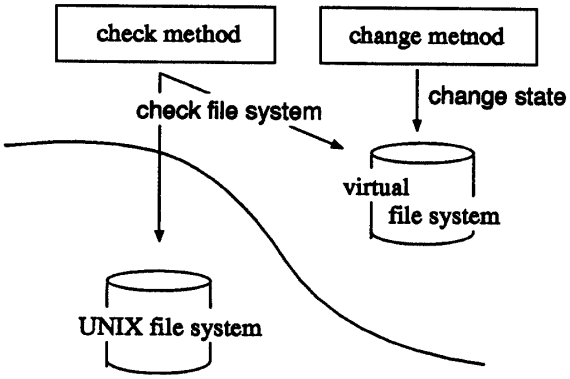

Fig 11 : FSS:File System Simulator

CheckFSSによる問い合わせが行われると，FSSは仮想ファイルシステムを使用して指定された状況が満たされいるかどうかを調査する．そして，仮想ファイルシステム内に指定されたファイルに関する記述に基づいて応答を行う．指定されたファイルに関する記述が仮想ファイルシステム内に存在しない場合は，現実のファイルシステムに対して指定されファイルに関する問い合わせ行ったうえで応答を行い，得られた状況を仮想ファイルシステムに登録する．

また，ChangeFSSプロトコルは選択されたコマンドの動作記述を引き数とし，コマンドの動作によって起こるファイルシステムの変化を仮想ファイルシステム内に実現する．Table5に引き数として渡される動作記述と，それによって，仮想ファイルシステムに加えられる知識，削除される知識を示した．

6 おわりに

本稿ではUNIX日本語インタフェースにおいて，入力要求文からUNIXコマンドを生成するメカニズムについて報告した．語彙の機能に関する知識を特定のドメイン毎に記述／管理することで，語彙の多義性の問題回避している．また構文意味・解析を行う順序に関する経験的知識をドメイン毎に用意する事で，計算コストを少なく抑えている．コマンド生成時に，コマンド実行によるファイルシステムの変化を考慮することで，複数のコマンドによって実現される要求文からもコマンド列を生成することができる．また，リダイレクトによるコマンドの出力先の変更，およびパイプを使用したフィルタコマンドの結合によって単一のコマンドで実現不可能な要求文からコマンド列を生成できる．

Table 5 : State change by change FSS

Action procedure	remove	add
delete(File)	Exist(File)	NotExist(File)
make(File)	NotExist(File)	Exist(File)
change(File1,File2)	Exist(File1) NotExist(File2)	NotExist(File1) Exist(File2)
out(Object)	—	—

謝 辞

研究に協力して下さった大分大学工学部知能情報システム工学科人工知能第一研究室の，神山文子技官，大島隆之君，末永祥子さん，矢野敏之君に深く感謝いたします．

参考文献

1) S. Jerrold Kaplan: Designing Portable Natural Language Databese Query System, ACM Tr. on Database Systems, Vol.9, No.1, pp.1-19 (1984)
2) R. Wilensky et al. : The Berkeley UNIX Consultant Project, Computatinal Linguistics, Vol.14, No.4, pp.35-84 (1988)
3) 青江潤一他：自然言語入力によるコマンドの理解 －UNIXシステムを対象にして－, NLC90-44 (1990)
4) 渡辺孝弘他：UNIX日本語インタフェースとコマンド結合，情報処理学会ソフトウエア工学研究会資料，No.77-21 (1991)
5) 渡辺孝弘他：UNIXシステム日本語インタフェースの構築，電気関係学会九州支部連合会大会，p.698 (1991)
6) 久保康司他：UNIX日本語インタフェースシステムの構築・コマンド辞書の記述とコマンドの選択、電気関係学会九州支部連合大会，No.1207, p.705 (1992)
7) 久保康司他：語彙の領域固有の機能に基づく日本語要求文の解釈，情報処理学会九州支部研究会資料，pp.1-6 (1993)

(平成5年10月15日受理)