知識工学と人工知能 44－1
（1986. 1. 20）

音声理解システムにおける言語処理部の開発
－オブジェクト指向の概念に基づいたインプリメンテーションについて－

堀 雅洋　上原 邦昭　溝口 理一郎
角所 収　豊田 順一

（大阪大学産業科学研究所）

1．はじめに

計算機科学および工学における大きな目標の1つとして，人間と計算機あるいは一般に人間と機械との間のインターフェイスをより快適で自然なものにすることがあげられる．音声が人間にとって最も自然な意志伝達の手段であることから，音声による機械との対話の実現すなわち音声理解システムの研究は大変重要な課題となっている．音声理解システムについては1970年代の前半に米国で大規模な研究が行われたが，その後音声認識・自然言語処理・知識情報処理等の技術に大きな進歩が見られ最近新たな研究の可能性が開けてきている．

筆者らは，音声理解システムを音響・音韻・構文・意味・文脈等のさまざまな知識を用いて発話を認識するとともにその意味を理解する1つの知識情報処理システムとみなして研究を行ってきた[1]．ここでは特に膨大な知識の抽出が大きな問題となることから，まず知識の獲得に重点を置きある程度独立して扱うことのできる音響情報処理・高次情報処理という2つの部分課題に分割してシステムの設計・開発を行っている(図1)．これによって知識の獲得が効率的に行えるだけでなく，各サブシステムに固有の問題を十分検討した上でシステム全体のアーキテクチャを具体化することが可能になると考えている．

さらに一般に知識獲得は知識工学者が対象領域の専門家に繰り返しインタビューすることによって行われる．ところが音声理解について考えた場合，人間はそれ自身1つの理想的な音声理解システムとみなすことができる．したがってそこからの知識獲得とは，われわれが日常無意識に行っている音声理解の過程を内省しそこで用いられた知識を抽出することにほかならない．その意味で人間の音声認識あるいは言語理解の過程を検討し，そこで得られた知見を生かすという認知科学的立場からも研究を進めてきた[2]．

本稿では，Symbolics 社のリスプマシン上に Flavorsの機能を用いてインプリメントされた言語処理部のプロトタイプについて，特にオブジェクト指向の概念との関連において述べる．以下まず音声理解システム全体の概要，次に言語処理部の概要とそこで用いられる知識の分類・組織化，さらにインプリメンテーションについて述べ，最後に簡単な動作例を示す．

2．音声理解システムの概要

本システムでは，不特定話者による文節発声で3000語程度の語彙を想定している．タスクとしては比較的簡単な情景描写の文章を対象としているが，システムの設計においてはタスクに依存しないものをめざしている．文節発声された入力音声は音響処理[3]によって文節単位の音韻列に変換される．次に高次情報処理としてまず文節内での単語間の接続規則や用言の活用に関する規則を適用し語彙解析[4]を行なう．言語処理部はそれによって得られた候補単語列を入力とし言語情報に代表される高次情報を用いて正しい単語列を同定

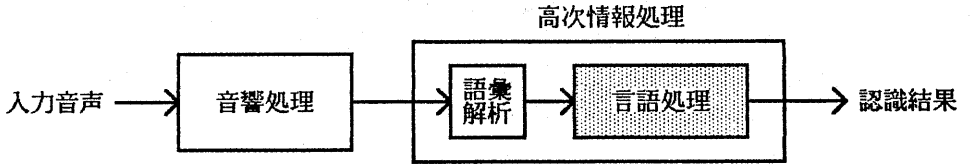

図1．音声理解システムの構成

し最終的な認識結果を出力する．表１にシステムへの入力およびそれに対応する音響処理の結果，さらに言語処理部の入力となる語彙解析の結果をそれぞれ示す．ただし各候補単語については文節内規則の適用によって，実際にはこの段階で品詞や活用形に関する情報も抽出されている．

表１からもわかるように各文節には複数の候補単語が対応し，多義性が実効的に増大したと考えることができる．したがって，ここでは通常の自然言語処理に比べて音韻認識における不確実さに起因するあいまいさの解消が特に重要で困難な問題となる．

３．言語処理部の概要

本システムでは，さまざまな役割を担ったデーモン[5](注) がその時点での処理経過が記述されているワーキングメモリを参照しそれを書き換えることによって処理が行われる．デーモンは各自が適用されるべき状況を知っている独立した知識であり，それぞれに固有の条件が満たされた段で対応する動作を実行する(図２)．したがって，これまで自然言語処理においてなされてきたように構文・意味・文脈解析を段階的に行うのではなく，さまざまな知識をその時点で適用可能なものから順に用いることができ，また通常の言語処理では文中の単語を左から右へ走査するのに対して，

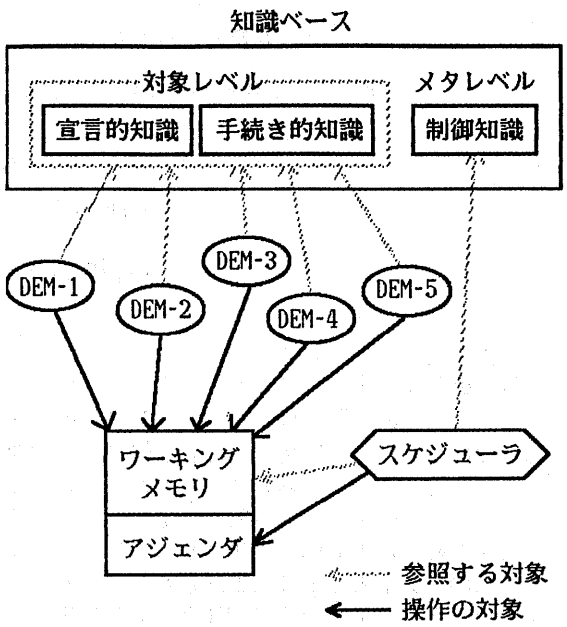

図２．デーモンによる処理の概念図

ある条件を満たす語や句あるいは節を核(島)として拡張可能なものから順次処理していく島駆動方式の処理を行うことができる．さらにトップダウンおよびボトムアップな処理をデーモンという枠組みのもとで統一的にとらえることによって両者を自然な形で融合することが可能となる．これらはいずれも音声理解に固有のあいまいさを効率的に解消するためのメカニズムを提供する．

また各文節には複数の候補単語が対応している

(注)ここでいうデーモンとは適用される知識の単位を意味し，フレーム理論における付加手続きとしてのデーモンとは異なる．またFlavors におけるメソッド結合のタイプとしてのデーモンとも直接の関係はない．

「のこぎりで	とうさんは	とびらに	ちょうど	いい	ながさに	まるたを	きります」
moporiribe	poohamwa	porirami	pjoubo	ii	marahami	mabupao	pibimahu
残り火	後半は	ローラに	ちょうど	見よ	丸太に	長く	煎ります
鋸ぎりで	倒産は	杭だのに	木ほど	言い	菜からさえ	まぶたを	切ります
	父さんは	扉に	道具を	いい	間からさえ	丸太を	要ります
	公算は	小枝に	気ほどを	居ろ	長さに	ながさを	入ります
	公判は	狩り場に	器ほどを		魔からさえ	歯形を	
	降参は	枯れ枝に	用ほどを		母さなに	名だけを	
	母さんは		器ほど				
	小枝には		気ほど				
	草には						

表１．システムへの入力および音響処理・語彙解析の結果の例

ことから，それらの組み合わせによってできるいくつかの解釈がそれぞれコンテクストとして管理される．新たなコンテクストは排他的な役割を担ったデーモンのうちの１つが実行された場合に生成される．そしてコンテクスト間のスケジューリングによって常に信頼性の高い解釈が注目される．したがって，基本的な処理として次の３つのレベルがある．

(1)複数の可能な解釈から１つを選ぶ
(2)適用可能なデーモンから１つを選ぶ
(3)デーモンの実行および消滅

このようなデーモン間・コンテクスト間のスケジューリングさらにそれらのうちのどちらに重点を置くかなどについてはシステムの知識ベース中に制御知識として記述される．

４．基本設計

各デーモンはそれらが生成される段階で必要な知識を知識ベース中から継承するように定義される．そこで，以下知識ベース中の知識の分類およびそれに基づく知識の組織化について，特にデーモンとオブジェクト指向の概念の融合という観点から述べる．

4.1. 知識の分類

知識ベースは対象レベルとメタレベルに分けられる．対象レベルについてはさらに宣言的知識と手続き的知識の２つに分類されるが，メタレベルの知識とは制御知識にほかならない(図２参照)．

・宣言的知識

文中の動詞・名詞あるいは文章の主題から連想される単語や関連する場面・主題等を１つの知識構造として宣言的に表わす．これには意味記憶(semantic memory)およびエピソード記憶(episodic memory)としての側面がある．意味記憶は同義・類義の関係に基づいて階層的に構成されるもので，ここでは分類語彙表[6] の記述に従う．そして分類語彙表中の分類項目および単語がクラスとして定義される．またエピソード記憶は，主題およびそれに伴われる一連の場面を単位とし，主題間の階層関係からネットワーク状の構造を持つ．ここでは主題・場面がクラスとなり，特に場面はactor, act, obj, instrなどのインスタンス変数をスロットとする格フレームの構造をもつものとして定義される．したがって場面を表わすフレームへのスロットフィリングは個々の場面クラスのインスタンスに対して行われる．

この２つの記憶構造中の対応する単語はそれぞれ両方向性のポインタによって結びつけられ，これによって主題からそれと関連の深い場面や単語を予測したり，逆に単語から関連する場面を連想することが可能となる．

・手続き的知識

手続き的知識は，基本的手続き・構文知識・一般の推論の３種類からなる．基本的手続きとしてはスロットフィリングを行うものが代表的である．構文知識とは，連体形の動詞が現れた場合に埋め込み構造の処理を行う一連のデーモンを起動したりするものである．また一般の推論とは，たとえば「太郎は東京へ行った」という場合，その時点で太郎はそれまでいた場所にはすでに存在しないと結論することであり意味記憶中の「移動」というクラスにメソッドとして定義されるが，この種の手続きは自発的にではなくデーモンの動作部などから呼び出される．

それに対して基本的手続き・構文知識はそれぞれに固有のクラスのメソッドとして定義される．クラスBASIC-DEMON-PROCには，すべてのデーモンに共通する手続きとして《条件部が満たされれば付随する動作を実行し消滅する》ことを記述したメソッドが定義される．それ以外の各クラスには条件部と実行部に対応する:test，:+actメソッドが定義され，それらがデーモンに継承される．

・制御知識

これはいわゆる focus attentionに用いられるもので，たとえばそれまでの処理によって入力に対して複数の解釈が可能な場合，そのうちのいずれを処理の対象とするかの評価を行うコンテクスト間のスケジューリング，およびあるコンテクスト内でデーモンの適用を制御するデーモン間のスケジューリングがある．

コンテクスト間のスケジューリングでは，デフォールトとして，排他的な役割を担ったデーモンが実行された段階で生成されるコンテクストが常に注目される．

またデーモン間のスケジューリングは，以下に示すデーモンの機能の分類に基づいて行われる．

(a)話題となっている場面の抽出
(b)抽出された場面に対応するフレームへの
　スロットフィリング
(c)共起関係の抽出
(d)隣接する文節間の係り受け関係の抽出

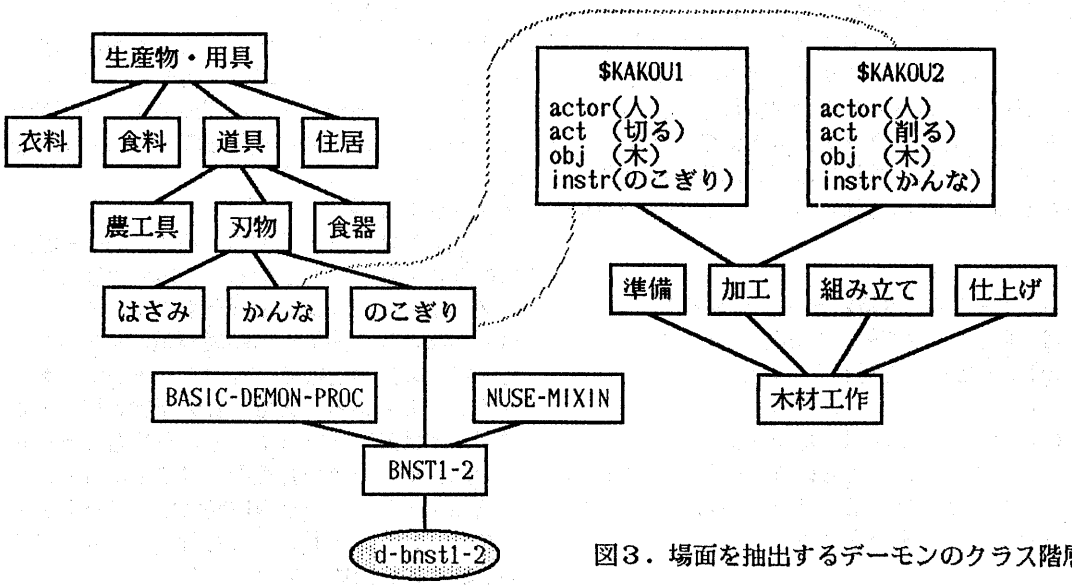

図３．場面を抽出するデーモンのクラス階層

このような機能の分類が実際にどのように用いられるかについては，'コンテクスト管理'の項で述べる．

4.2. 知識の組織化

知識の組織化とは各デーモンに必要とされるクラスを上位にもつようなクラスを生成し，新たなクラス階層を定義することをいう．たとえば話題となっている場面の抽出を行うデーモンについてはBASIC-DEMON-PROCの他に，場面抽出の手続きが定義されるクラスNUSE-MIXIN，および対応する文節の自立語のクラスを上位にもつクラスが定義される．図３に「鋸ぎりで」に対応するデーモンにおけるクラス階層を示す．インスタンスであるd-bnst1-2 は意味記憶中のクラス階層によって「のこぎり」の「刃物」あるいは「道具」としての性質を参照することができ，またクラス「のこぎり」からのポインタによってエピソード記憶中の関連する場面を参照することができる．

またスロットフィリングを行うデーモンについては，BASIC-DEMON-PROCに加えてスロットフィリングに関する基本的手続きのクラスであるFGAP-MIXINを継承するクラス FGAP1が定義される(図４)．FGAP-MIXIN中の :testメソッドは与えられた意味上の制約を満たす候補が入力系列中にあればＴなければ nilを返す． :+actメソッドは，スロットフィリングの対象となる場面フレームのインスタンス変数に :testによって得られた候補単語を代入する．ここで，注目すべきスロット，考慮すべき意味上の制約および場面のインスタンスは，各デーモンごとにインスタンス変数の値として与えられる．すなわちFGAP-MIXINにはスロットフィリングにおける純粋に手続き的な部分が定義されている．

このようにあらかじめ分類された知識は各デーモンの機能に合わせて組織化される．このときいくつかのデーモンに共通する知識については知識ベース中の同一の知識が継承される．これによって知識ベースの効率的な構成および利用が可能となる．

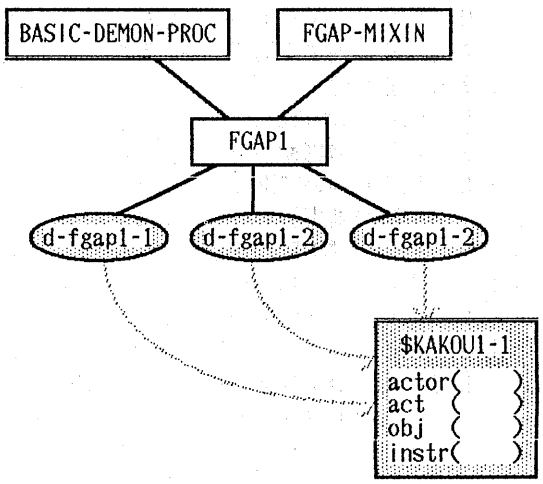

図４．FGAP1に関連するクラス階層

5．インプリメンテーション

言語処理部はリスプマシン上の Flavorsの機能を用いてオブジェクト指向プログラミングのパラダイムの下にインプリメントされている．これによってシステムはモジュール性・拡張性にすぐれるだけでなく，知識ベース中の対象レベルの知識や各コンテクスト間の関係をクラス階層を用いて自然にインプリメントすることができる．またデーモンは，Symbolics Lispの強力なマクロ機能を用いてクラス階層を動的に定義しそのインスタンスとして生成される．

図５に初期状態におけるオブジェクト間の関係を示す．図に示されたオブジェクトはそれぞれクラスに対応し，そのうち網がけされたものは特にインスタンスを直接生成していることを表す．したがってオブジェクト間のメッセージ交信の経路を表す破線は網がけされたオブジェクト間のみを結ぶ．また実線はクラス間の階層を表している．図からもわかるように，この言語処理部はINPUT-HANDLER，KNOWLEDGE-ORGANIZER，ROOT-CONTEXTの３つのオブジェクトに対応するモジュールからなる．

以下各クラスはINPUT-HANDLER・CONTEXT-MANAGERのように大文字で表し，インスタンスは *knowledge-organizer*・*root-context*のように前後に "*"をつけて表す．また "*"ではさまれたアトムは，プログラム中では大域変数であることを意味し，適当な宣言が行われる．さらにメソッドは :gen-init-dems，:runのようにコロンで始まるアトムによって表される．

5.1. INPUT-HANDLER

言語処理部は INPUT-HANDLERに:startメッセージを送ることによって起動される．INPUT-HANDLER は語彙解析の結果である候補単語系列を受け取り，それらに必要な処理の起動を他のモジュールに依頼する．

```
(send  *input-handler*  :start  s1)
```

引数s1にはたとえば図６に示すような候補単語列がバインドされる．

次に KNOWLEDGE-ORGANIZERに:gen-init-demsメッセージを送ることによって，引数candidatesによって与えられる候補単語列をデーモンとして初期化する手続きが起動される．

```
(send *knowledge-organizer*
      :gen-init-dems candidates)
```

以後の具体的な処理についてはKNOWLEDGE-ORGANIZERに一任され，INPUT-HANDLERはその内容に一切関与しない．

入力系列の初期化が終わると， INPUT-HANDLERはその時点で処理の対象として注目されているアクティブなコンテクストに対して:runメッセージを送る．

```
(send *active-context* :run)
```

これによってそのコンテクスト内の実行可能なデーモンが発火する．大域変数*active-context*には，アクティブなコンテクストに対応するインスタンス（正確にはインスタンスへのポインタ）が

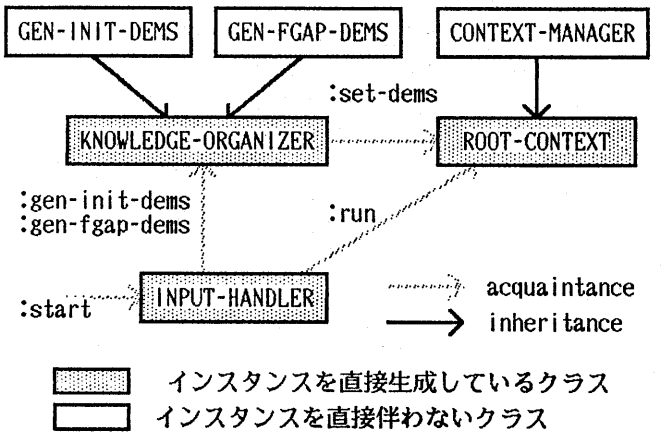

図５．オブジェクト間の関係

```
(( ((ms 残り火)(kz で))
   ((ms 鋸ぎり)(kz で)) )

 ( ((ms 後半)(kz は))
   ((ms 倒産)(kz は))
   ((ms 父さん)(kz は))
   ((ms 公算)(kz は))
   ((ms 降参)(kz は))
          •••••          )

 (  ((ms ローラ)(kz に))
    ((ms 杭)(hz だの)(kz に))
    ((ms 扉)(kz に))
    ((ms 小枝)(kz に))
          •••••          ))
```

図６．入力候補列の例

バインドされる．各コンテクストのうちいずれをアクティブにするかはコンテクスト間のスケジューリングによって決定される．したがってこの大域変数はアクティブ・コンテクストへのチャネルとしてとらえることができ，*root-context*がその初期値となる．

5.2. KNOWLEDGE-ORGANIZER

デーモンが生成される状況には，入力系列が文節ごとにデーモンとして初期化される場合と，デーモンに付随する動作によって生成される場合の２つがある．

入力系列を初期化する手続きは，INPUT-HANDLER によって起動され，特にデーモンの初期化に必要な手続きが記述されているGEN-INIT-DEMS クラスから継承したメソッドを順次起動することによって行われる．たとえば表１の第１文節の第２候補「鋸ぎりで」については，図７に示されたフォームに従ってBNST1-2 というクラスが処理中に動的に定義される．そしてそのインスタンスとしてこの文節に対応するデーモンd-bnst1-2 が生成され(図３参照)， :set-demsメソッドによってアジェンダに登録される．

つぎにデーモンに付随する動作によって他のデーモンが生成される場合について，特にスロットフィリングを行うデーモンの生成について説明する．*knowledge-organizer* に送られた:gen-fgap-demsメッセージは GEN-FGAP-DEMSクラスから継承した手続きを起動する．たとえば $KAKOU1-1という場面に対応するフレームへのスロットフィリングについては，図８に示したフォームによってクラス FGAP1が新たに定義される．さらに各空スロットごとにスロットフィリングを行うデーモンが FGAP1のインスタンスとして生成され(図４参照)，アジェンダに登録される．

```
(defflavor bnst1-2
           ((priority 4)
            (indep '(ms 鋸ぎり))
            (deps  '((kz で))))
           (basic-demon-proc
            nuse-mixin 鋸ぎり)
  :gettable-instance-variables
  (:settable-instance-variables priority))
```

図７．クラスBNST1-2の定義

```
(defflavor fgap1
           ((priority 3)
            (scene '$KAKOU1-1))
           (basic-demon-proc fgap-mixin)
  :gettable-instance-variables
  (:settable-instance-variables priority))
```

図８．クラスFGAP1の定義

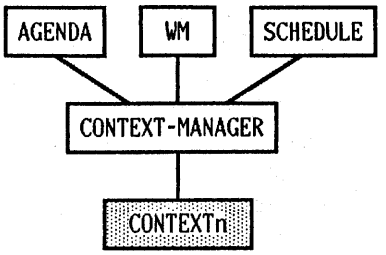

図９．CONTEXTnにおけるクラス階層

5.3. コンテクスト管理

コンテクストはアジェンダとワーキングメモリからなり，排他的な役割を担ったデーモンの実行に伴って適宜各コンテクストに対応するクラスが生成される．ROOT-CONTEXTは初期状態において存在する唯一のコンテクストであり，デーモンとして初期化された候補系列はROOT-CONTEXTのアジェンダに登録される．各コンテクストはAGENDA，WM，SCHEDULEという３つのクラスを上位にもつ CONTEXT-MANAGERを継承するインスタンスとして実現されている．これによって各コンテクストに共通する手続き，たとえばアジェンダへのデーモンの登録あるいは削除・ワーキングメモリへの処理経過の登録・デーモン間のスケジューリングの手続きなどを共有することができる．図９にｎ番目に生成されたコンテクストCONTEXTnにおけるクラス階層を示す．

またアジェンダやワーキングメモリの内容など各コンテクストに固有の情報は，基本的には各々のインスタンス変数の値として保持することが可能である．ところがコンテクスト間でのアジェン

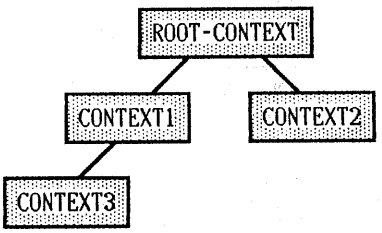

図10．コンテクスト間の階層構造

ダやワーキングメモリの内容の相違は，特に隣接するものについてみた場合，高々１つのデーモンの実行によって生じるものでしかない．そこで新たに生成されたコンテクストでは直前のコンテクストと異なる部分のみを記述し，それ以外の大部分の共通する内容については直前のコンテクストから継承するかたちで実現している(図10)．そして，クラスであるコンテクストの間で継承を行うためにクラス変数の機能を実現した(Flavorsにはクラス変数の機能がサポートされていない)．すなわち引き続くコンテクストに継承されるべき情報はそのコンテクスト・クラスのクラス変数の値として保持される．これによって，たとえば初期化されたデーモンについては，ROOT-CONTEXTのアジェンダに登録すれば他のコンテクストはその内容を継承することができ，新たにコンテクストが生成されるたびにその内容をコピーする必要がない．また処理中に生成されたデーモンについては，まずそれが生成されたコンテクストのクラス変数の値として保持し，以後そのデーモンが実行された段階でそのコンテクストにおいて同名のクラス変数を再定義しその値を nilとする．これによってそれ以後下位クラスとして生成されるコンテクストへの継承を打ち切ることができる．

実際にはアジェンダをagenda-4，agenda-3 …という４つのレベルに分けそれぞれを各コンテクスト中のクラス変数として定義する．そして各レベルは上に示した分類の (a),(b) … に順に対応する．:runメソッドによってあるコンテクストが起動されると，SCHEDULEクラス(図９参照)のメソッドによって処理のレベルが決定される．デフォールトのスケジューリングでは，(a)から(d)の機能をもったデーモンの適用が順にチェックされる．SCHEDULEクラスにメソッドを追加していくことによって，コンテクスト切り換えのタイミングも含めて，より柔軟な制御構造を実現することができる．

クラス変数は，各変数ごとにその値を保持するクロージャーを生成し，そのクロージャーにアクセスするメソッドを定義するマクロ関数 defclassvarによって定義される．クラス fooにクラス変数 barを初期値 bazとして定義する場合について図11に示す．これによって変数 barの値は fooおよびその下位クラスの各インスタンスに共通の値となる．値の変更は:set-barメソッドによって行う．図11(c) にその例を示す．

```
   (let-closed ((x 'baz)) #'(lambda () x))
=> #<dtp-closure...>
```

(a) クロージャーの生成

```
   (defclassvar foo (:bar 'baz))
==> (defmethod foo (:bar 'baz))
      (funcall #<dtp-closure...>))
    (defmethod (foo :set-bar) (val)
      (set-in-closure
          #<dtp-closure...> 'x val)
      val)
```

(b) マクロ展開の結果

```
(setq a (make-instance 'foo))
(send a :bar) =>  baz

(setq b (make-instance 'foo))
(send b :bar) =>  baz

(send a :set-bar 123) =>  123
(send b :bar) =>  123
```

(c) クラス変数の参照と変更

図11．クラス変数barの定義

６．動作例

表１に示した「鋸ぎりで父さんは扉にちょうどいい長さに丸太を切ります」に対応する入力系列がどのように処理されるか以下に示す．

まず INPUT-HANDLERに送られた候補系列をデーモンとして初期化するために KNOWLEDGE-ORGANIZERに:gen-init-demsメッセージが送られ，その結果がROOT-CONTEXTのレベル４のアジェンダに登録される．初期化が終わると INPUT-HANDLERはROOT-CONTEXTに:runメッセージを送りデーモンの発火を促す．するとSCHEDULEクラスに定義されたデーモン間のデフォールトのスケジューリングによって，まずレベル４のアジェンダ中のデーモンの適用がチェックされる．ここでは「鋸ぎりで」に対応する候補について，その自立語「鋸ぎり」からそれが道具としての役割を果たす場面 $KAKOU1が連想され，しかも付属部に道具格を表す格助詞「で」が伴われていることからデーモン D-BNSTO1-02が発火し，エピソード記憶中の『人が鋸ぎりで木を切る』場面が活性化される．ここでコンテクストの分岐が起こり，以後の処理はCONTEXT1において第１文節が「鋸ぎりで」に確定したものとして行われる．また場面の活性化によって $KAKOU1のインスタンスが生成されワーキングメモリに登録されると同時に，その空スロットにスロットフィリングを行うデーモンD-FGAP1-1，D-FGAP1-2 … が:gen-fgap-demsメソッドによって生成され

CONTEXT1のレベル３のアジェンダに登録される．次にレベル３のアジェンダがチェックされるが，この段階で各単語からの場面の抽出というボトムアップな処理から，抽出された場面へのスロットフィリングというトップダウンな処理に切り換わることになる．たとえば「切る」の対象を表すobjスロットにスロットフィリングを行うデーモンD-FGAP1-3ついては，「木」もしくは意味記憶中でその下位にある「用材・木材・丸太 … 」などの語が対象を表わす格助詞「を」とともに用いられていれば，それを objスロットに代入する．ここでこれらのスロットフィリングがうまくいかなければ，さらに別の場面あるいは主題の抽出を行うためにコンテクストが切り換わりレベル４のアジェンダがチェックされる．フレームへのスロットフィリングがうまく行われれば，さらにレベル２のアジェンダがチェックされ再びボトムアップな処理が行われる．そして「ちょうど」と共起

```
++++++ ROOT-CONTEXT ++++++++++++++++++++
>
> agenda-4 = (D-BNST01-01 D-BNST01-02 D-
2-03 D-BNST02-04 D-BNST03-01 D-BNST03-02
ST04-01 D-BNST04-02 D-BNST04-03 D-BNST04
> agenda-3 = NIL
> agenda-2 = (D-KYOU01-01)
> agenda-1 = (D-SYNT01-01 D-SYNT02-01)
> agenda-0 = NIL
>
> WM = NIL
>
 checking demon D-BNST01-01
 checking demon D-BNST01-02

===== executing D-BNST01-02 =====

------------------------------
     generating CONTEXT01
------------------------------
   activating $KAKOU1-01
   adding D-FGAP01-01 in agenda-3
   adding D-FGAP01-02 in agenda-3
   adding D-FGAP01-03 in agenda-3
   adding D-BNST01-02 in agenda-0

++++++ CONTEXT01 ++++++++++++++++++++++
>
> agenda-4 = (D-BNST02-01 D-BNST02-02 D-
3-01 D-BNST03-02 D-BNST03-03 D-BNST03-04
ST04-03 D-BNST04-04)
> agenda-3 = (D-FGAP01-01 D-FGAP01-02 D-
> agenda-2 = (D-KYOU01-01)
> agenda-1 = (D-SYNT01-01 D-SYNT02-01)
> agenda-0 = (D-BNST01-02)
>
> WM = ($KAKOU1-01)
>
 checking demon D-BNST02-01
 checking demon D-BNST02-02
 checking demon D-BNST02-03
**MORE**
```

図12．リスプマシン上での実行例

関係にある「いい」を検出するデーモン D-KYOU1-1が発火し「ちょうど」に隣接する第５文節中から「いい」が選択される．さらに，レベル１のアジェンダがチェックされ隣接する文節間の係り受け関係の検出が行われる．

図12にリスプマシン上での対応する実行例を示す．処理経過はリスプ・リスナの画面上に連続して表示されているが，画面をいくつかの区画(pane)に分割しそれぞれがシステム中のオブジェクトに対応するようなマルチウィンドウによる支援ツールを現在作成中である．それによってシステムの内部および動作がユーザにとって透明なものとなり，知識の獲得や修正がスムーズになると考えている．

７．むすび

音声理解システムの言語処理部について，特にオブジェクト指向の概念に基づいたインプリメンテーションとの関連において述べた．現在，この言語処理部は50余りのクラスからなるが，今後プロトタイプ上での実験を通して，知識ベースを拡充していく予定である．

（参考文献）

[1] 溝口 他:"音声理解システムー基本構造ー", 情報処理学会第29回大会, 4Q-8,1984.

[2] 溝口:"音声理解と認知科学",数理科学, Vol. 23,No.8,1985.

[3] 辻野 他:"音声理解システムの開発ー音響処理部の設計ー",秋季音響学会,1985.

[4] 河内 他:"音声理解システムにおける語彙解析部の設計",知識工学と人工知能研究会38-1, 1985.

[5] Lindsay,P.H. and Norman,D.A.:"Human Information Processing", Academic Press,1977.

[6] 国立国語研究所:"分類語彙表",秀英出版, 1982.