知識工学と人工知能 46－3
（1986. 5. 9）

CLにおけるルール指向プログラミング

渡辺 正信* 岩本 雅彦* 山之内 徹** 出口 幸子*** 松田 裕幸****

*日本電気㈱C&Cシステム研究所 **日本電気㈱基本ソフトウェア開発本部
***日本電気ソフトウェア㈱ ****日本電気技術情報システム開発㈱

CL(Conceptual-network-based Language)におけるルール指向プログラミングに関して，その設計思想，概要，特徴等を報告する。特に，ユーザがルール言語を容易に拡張することができたり，競合解消をユーザが自由に定義できるためのドメイン言語プリミティブ（メタ言語）を提供することにより，ルール指向における柔軟性を高める。さらに，オブジェクト（フレーム）指向をベースに構造化されたワーキングメモリとルールの高速処理を実現するためのルールコンパイル方式を，逐次実行型ルールと，競合解消型ルールのそれぞれに対して提案する。その後，具体的なエキスパートシステム構築例を通して，CLがエキスパートシステム構築時の生産性を大幅に向上させること示す。最後に，今後の課題を述べる。

"Rule-Based Programming in CL (Conceptual-network-based Language)" (in Japanese)

by Masanobu WATANABE, Masahiko IWAMOTO (C&C Systems Research Laboratories,NEC Corp.,Kawasaki-shi, 213 Japan), Toru YAMANOUCHI (Basic Software Development Division,NEC Corp.), Sachiko DEGUCHI (NEC Software Ltd.) and Hiroyuki MATSUDA (NEC Scientific Information System Development Co.Ltd.)

This paper describes facilities in CL (Conceptual-network-based Language) for representing rules and for building expert systems with rule-based programming. In particular, the flexibility of rule-based programming is increased by providing Domain Language Primitives (a meta language) which allow users to both freely define conflict resolution strategies, and easily extend the rule language. For efficiency, two kinds of rule compilers are proposed. One translates sequentially executable rule sets into Lisp code to be compiled by a Lisp Compiler. The other utilizes a modified Rete algorithm to quickly derive instantiation sets. We conclude with a discussion of future work.

1．はじめに

エキスパートシステムに代表される知識ベースシステムの構築を支える重要な知識表現パラダイムのひとつに，ルール指向がある。ルール指向が，悪構造(ill-structured)，開いたドメイン(open-ended domain) の問題解決に対する柔軟な方法論を提供するからである[1]。

しかしながら，ルール指向が万能というわけではない。さらに，ルールの表現形式とか，他の表現パラダイムとの結合法が確立しているわけではなく，以下の問題が認識されている。

①ルール表現の拡張性の問題

ルール表現においては，ドメインの知識をルールとしてできるだけ自然に記述する方法と，ルール起動制御の指定方法が問題となる。開いたドメインに対して，この両方を固定することは非常に困難である。しかしながら，従来のシステムでは，ルール言語やルールの推論制御が固定されていたため，拡張やカスタマイズを柔軟に効率良く行ないにくいという問題があった。

②他の表現パラダイムとの統合化問題

ルール指向が万能でないことから，他の有効な表現パラダイム（オブジェクト／フレーム，データ，論理，手続き指向等）といかに統合するかが問題となる。特に，ワーキングメモリの構造化，構造化されたワーキングメモリを有効に活用するルール表現と制御構造等にルール指向から見た統合化問題がある。つまり，従来のルール指向におけるワーキングメモリでは構造化が不十分であったことや，ルールの表現力が弱いため不必要にルールの数が増大するといった問題があった。

③高速化問題

実用的エキスパートシステムを構築し，利用するためには，高速な処理系の実現が焦眉の急である。特に，ルール指向におけるルールコンパイラによる高速化がキーとなる。従来のルールコンパイラは，非構造のワーキングメモリを前提としていたので，構造化されたワーキングメモリを取り扱うのは困難という問題があった。

本稿では，以上の問題を解決すべく開発中の，ＣＬ[2] におけるルール指向知識表現パラダイムの概要と特徴について報告する。特に，その中で，ルール指向におけるドメイン言語プリミティブを提案する。

以下，２節でＣＬにおけるルール指向の設計思想を示し，３節で推論制御，ルール表現，ルールコンパイラ等のルール指向パラダイムについて述べ，４節でその適用例を示す。最後に今後の課題について論じる。

2．設計思想

ＣＬでのルール指向の設計思想は，以下の４点にある。

2.1 ルール指向におけるドメイン言語プリミティブの提供

ＣＬにおけるドメイン言語プリミティブの思想は，ユーザが開いた世界を記述するための言語（ドメイン言語）を構築するためのプリミティブ（メタ言語）を提供し，柔軟性を追求することにある。その基本は，ドメイン独立の汎用ツールを窮めるという立場にある。つまり，限定問題向けツールと対比される立場である。

さて，ルール指向でのドメイン言語プリミティブとは，①ルール表現，即ち，ルール言語の拡張用プリミティブと，②ルールの実行制御記述用プリミティブがある。前者には，ルール言語における基本条件関数や，基本動作関数，及び，パターンマッチング関数をユーザが拡張，定義可能とするためのもので，その具体例を 3.3項で示す。後者には，宣言的なアジェンダ／ルールセット記述や，競合解消方式をユーザに解放するプリミティブがあり， 3.2項で具体的に述べる。尚，この意味で，OPS5がシステムで固定にしていた競合解消方式を，OPS83[3]ではユーザに解放したことは注目に値する。

2.2 ワーキングメモリの洗練化

ルール表現言語を決めるポイントは，どのような構造のワーキングメモリを前提とするかにある。従って，表現力及び柔軟性が高いルール言語を提供するためには，高度に洗練化された構造を持つワーキングメモリが必須となる。ＣＬのルール指向においては，フレームベースの概念ネットワークをワーキングメモリとして利用して，洗練化を図る。図１は，ＣＬでのルール指向の概念構成を示したものである。そこでのワーキングメモリは，ＣＬでのオブジェクトをノードとした概念ネットワーク（ＣＬでの知識ベース）として構造化される。つまり，ワーキングメモリの表現，アクセス，更新は，ＣＬでの基本関数により実現され，動的に操作可能である。さらに，複数のワーキングメモリを独立に実現し，利用することが可能となる。

尚，ワーキングメモリを概念ネットワークで実現する時の弱点としてメモリの増大，スピード低下があげられる。これに対しては，後述するルールコンパイラを含む知識ベースコンパイラで対処する。

2.3 ルール表現の洗練化

ここでのポイントは，①洗練化されたワーキング

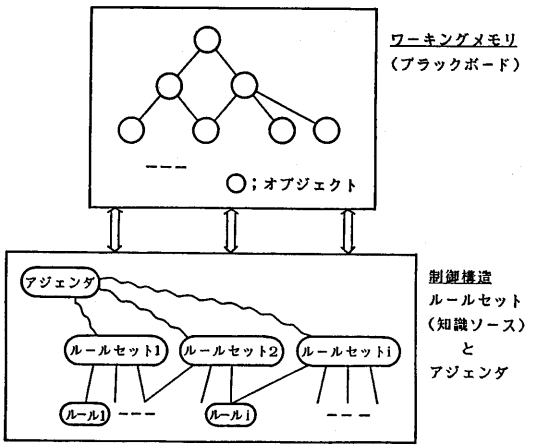

図1．CLでのルール指向概念図

メモリを十分に活用できるルール表現，と②ルール，推論制御表現自身の洗練化にある。

まず，前者①に関する基本的考え方は，ルールが適用されるべきオブジェクトの指示を，洗練化されたワーキングメモリの構造を十分に反映した形で自然に表現できるということにある。つまり，この指示の仕方そのものが，専門家のノウハウの一部とみなせる。そこでは，専門家が表現する自然な形でルールを表現できることが望まれる。さらに，このことは，ひとつのルールを複数のルールとして不自然に分割することを余儀なくさせることを防ぐ。図2は，ルールを適用すべきオブジェクトの代表的指示法を図式化したものである。つまり，ルールを適用すべきオブジェクトのクラスを直接指示する直接的指示法(Direct Indication)と，ルールを適用すべきオブジェクトのクラスを間接的に指示する間接的指示法(Indirect Indication)がある[4]。この具体例を 3.3項で示す。

次に，後者②は，ルール表現や，ルールセット，アジェンダ等の推論制御自身の表現を構造化することである。CLでは，図3，図4に示すように，ルール，ルールセット，アジェンダ自身も，ワーキングメモリと同じフレーム／概念ネットワークとして構造化される。

以上の考え方の下で，ルール指向とオブジェクト（フレーム）指向が統合される。

2.4 ルールコンパイラによる高速化

ワーキングメモリとルールを洗練化することによって柔軟性を高めることは前述した。ここでは，この柔軟性と高速性を両立させることを目指す。つまり，一般に，柔軟性を優先させると性能が犠牲になる。このことを防ぐために，ルール表現の柔軟さを保ったままで，ルール実行を高速化するコンパイラ

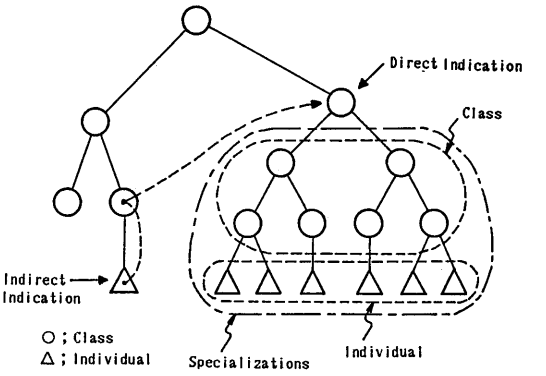

図2．ルールを適用すべきオブジェクトの指示法

を提供するわけである。ルールコンパイラには，逐次実行指定があるルールセットをコンパイルするものと，競合解消による実行指定があるルールセットをコンパイルするものがある。ルールコンパイラは 3.4項で述べる。

3．ルール指向パラダイム

CLでのルール指向パラダイムに関して，その全体構成，推論制御，ルール表現，高速化について述べる。その中で，ルール指向におけるドメイン言語プリミティブを提案する。

3.1 全体構成

図1は，CLでのルール指向の全体イメージを示したものである。ここでの特徴は，①制御構造，ルール，ワーキングメモリ全てがオブジェクト（フレーム）指向上に構築されていること，②ルールを含む制御構造とワーキングメモリが同じ形式の知識ベースとして実現されていることである。従って，ユーザは，任意の独立したワーキングメモリや，制御構造を生成，利用することができ，柔軟性が高い。

3.2 推論制御（ルール実行管理）

ここでは，ルールの実行管理を行なうルールセットと，ルールセットで代表されるタスクの実行管理を行なうアジェンダについて説明する。

3.2.1 アジェンダ

複数のルールセット（ルール集合）の実行制御を行なうオブジェクトがアジェンダである。図5は，ユーザが定義したアジェンダオブジェクトDKU-A の例である。このDKU-A は，CLが提供するAgendaオブジェクトの特殊オブジェクトとして定義されてい

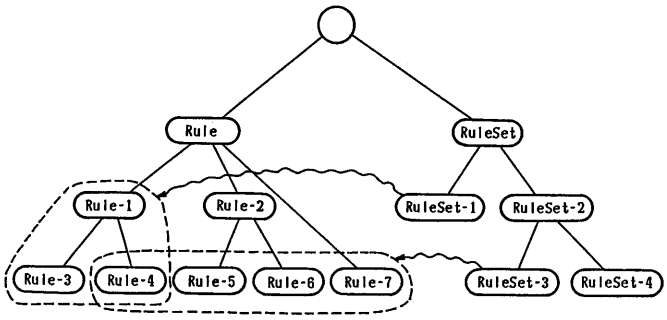

図３．ルールの構造化

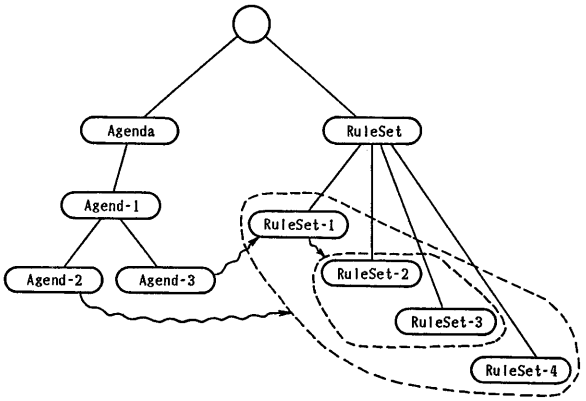

図４．アジェンダの構造化

る。ここで，NORMAL-TASKSスロットに示されたルールセット群は，このアジェンダが起動された時に，順次適用され，最後のルールセットの実行が終了した時点でアジェンダの実行を終了する。尚，アジェンダは，アジェンダオブジェクトのSCHEDULEスロットにメッセージを送ることによって起動される。

尚，ルールセット群の動的な実行制御をルールセット自身を使って実現することもできる。

3.2.2 ルールセット

ルールを機能単位にまとめたものがルールセットである[4][5]。ルールセットに基づく管理によって，ルールの維持／拡張が見通し良いものとなるだけでなく，推論時のルール探索を効率化することができる。つまり，ルールをいくつかのルールセットに分類して，ルール型知識をモジュール化し，ルールセットにひとつのまとまった処理／推論を行なわせるわけである。従って，ルール探索の無駄を極小化できる。さらに，ルールセットでは，個々のルールをどのように適用するかの実行制御を指定できる。
以下，ルールセット内でのルール制御に関してルールの適用基準と適用戦略について述べる。

```
Object Window
DKU-A
  Type : Individual
  Generalizations : Agenda
  NORMAL-TASKS : (Ruleset-Slave Ruleset-N7756
                                Ruleset-N7761
                                Ruleset-Master
                                Ruleset-Msp)
  DISPLAY-TRACE : *no-value*
  SCHEDULE FROM Agenda :
    (lambda (object slot facet value)
            (SCHEDULE-AGENDA (ObjectName object) nil))
  WHEN-INITIATED FROM Agenda :
    *no-value*
  WHEN-COMPLETED FROM Agenda :
    *no-value*
  TASK-COMPLETED FROM Agenda :
    *no-value*
  Documentation FROM Root :
    *no-value*
```

図５．アジェンダ例
(DKU-A)

```
Object Window
Ruleset-Slave
  Type : Class
  Generalizations : RuleSet
  NORMAL-RULES : (Slave-R-1 Slave-R-2 Slave-R-3)
  ONCE-RULES : *no-value*
  ALWAYS-RULES : *no-value*
  CONTROL-STRATEGY : RESTART-AFTER-FIRING
  KNOWLEDGE-BASE : *no-value*
  TRACE : *no-value*
  TERMINATION-CONDITION : *no-value*
  FILTER : *no-value*
  PAUSE : *no-value*
  DISPLAY-TRACE : *no-value*
  RULESET-VARIABLE-BINDINGS : *no-value*
  FIRE-ONCE-RULES FROM RuleSet :
    *no-value*
  TRACE-STORAGE FR
```

図６．ルールセット例
(Ruleset-Slave)

１）適用基準

次の３種類があり，それぞれに対応するスロットがある。

①NORMAL-RULES；記述された順に適用される。但し，一回のルールセットの実行中で同一の束縛環境で，ひとつのルールが２度以上実行されない。

②ONCE-RULES；NORMAL-RULESに先立って，記述された順に適用される。但し，一度実行されたルールは以後適用されない。

③ALWAYS-RULES；ONCE-RULES，NORMAL-RULESの次に順次適用される。但し，同一の束縛環境でも２度以上実行される。

２）適用戦略

適用基準に従って登録されたルールの適用順序の制御方法を決めるのが適用戦略であり，次の４つがある。

①RESTART-AFTER-FIRING
②CONTINUE-AFTER-FIRING
③LEAVE-AFTER-FIRING
④CONFLICT-RESOLUTION

ここで，①は，ONCE-RULES，NORMAL-RULES，ALWAYS-RULESの順で適用され，ひとつのルールが起動された後，再び最初からルールの適用が行なわれる。②は，起動されたルールの次のルールが適用される点と，③は，ひとつのルールが起動されるとALWAYS-RULESに制御が移り，ルールセットの実行が終了する点が，それぞれ①と異なる。④は，通常の競合解

消のことである[3]。ＣＬでの競合解消の指定法は次の項で説明する。尚，上記の適用戦略は，ルールセットのCONTROL-STRATEGYスロットに指定する。

図６は，ルールセットの具体例(Ruleset-Slave)を示す。ここでは，NORMAL-RULESスロットに３つのルール，CONTROL-STRATEGYにRESTART-AFTER-FIRINGがそれぞれ指定されている。

3.2.3 競合解消用ドメイン言語プリミティブ

複数のルールが適用可能な状態で，どのルールを実行するかを決定することを競合解消という。OPS5に代表される多くのルール指向支援ツールでは，この競合解消法はシステムで固定されており，ユーザには解放されていなかった。このため，不自然なルール制御を強いる結果となった。又，OPS5に続くOPS83では，この競合解消法をユーザに解放したものの，基本的に関連するデータ構造（インスタンシエーションセット関連）と低レベルプリミティブを提供したにすぎない。ＣＬでは図７で示す，より自然で，高レベルのプリミティブ（競合解消用ドメイン言語プリミティブ）を提供する。図７(a)が競合解消法の一例で，図７(b)がＣＬでの記述例である。これは，OPS83 の記述方式と比べて記述量が約1/10となり，可読性も高い。

3.3 ルール表現

図８は，ＣＬでのルール表現例である。つまり，Slave-R-1 は，ルール名でＣＬのRuleオブジェクトの特殊オブジェクトとして定義される。ＩＦ部（ＩＦスロット）には２つの条件節(Condition-1とCondition-2)が，THEN部（THENスロット）には４つの帰結節（Action-1…Action-4）がそれぞれ定義されている。

ここでは，限定条件節と，ルール言語拡張方式について述べる。

3.3.1 限定条件節

THERE-EXISTS等で始まる条件節を限定条件節と呼び，図８のCondition-1 とCondition-2 がその例である。ここでは，２番目の要素(-Anythingと-Parent)がオブジェクト変数，３番目が領域定義，４番目以降が述語である。従って，Condition-1 は，“Connected-Praents スロットの値が nilであるSlave-Diskのインディビデュアル(-Anything) が存在する”と読む。

ここで，I＊ は図２で示した直接的指示法に使われる関数で，Condition-2 の第３要素は間接的指示法の例である。

1．今まで一度も実行されていないルール
2．最初の条件節に出現する変数に対応するデータ要素が最新であるルール
3．使用している変数の多いルール（最も厳密なルール）
4．最近適用条件が満足されたルール

（ａ）競合解消戦略例

```
(RuleSelect
    ((equal 0 (ExecutionCount *i-other* 1)))
    ((> (ObjectAge *i-best* 1)(ObjectAge *i-other* 1))
     (> (RuleVariableCount *i-best*)(RuleVariableCount *i-other*))
     (> (InstantiationAge *i-best*)(InstantiationAge *i-other*)))
)
```

（ｂ）ドメイン言語プリミティブによる記述例

図７．競合解消用ドメイン言語プリミティブ例

```
Object: Instance
Slave-R-1
  Type : Individual
  Generalizations : Rule
  IF : (Condition-1 Condition-2)
  Condition-1 : (THERE-EXISTS -Anything
                              (I* (quote Slave-Disk))
                              (&= (THE Connected-Parents -Anything) nil))
  Condition-2 : (THERE-EXISTS -Parent
                              (MI* (THE Connectable-Parents -Anything))
                              (&< (THE Used-Port-Number -Parent)
                                  (THE Port-Number -Parent))
                              (&< (THE Used-Slave-Disk -Parent)
                                  (THE Slave-Disk -Parent)))
  THEN : (Action-1 Action-2 Action-3 Action-4)
  Action-1 : (&CONNECT -Anything to -Parent)
  Action-2 : (&INCREMENT (quote Used-Slave-Disk) of -Parent)
  Action-3 : (&INCREMENT (quote Used-Port-Number) of -Parent)
  Action-4 : (&WRITE (quote Slave-R-1)
                     (B 4)
                     (quote CONNECT)
                     B
                     -Anything
                     B
                     (quote to)
                     B
                     -Parent
                     N)
  RULESET FROM Rule :
    *no
```

図８．ルール例
（Slave-R-1）

3.3.2 ルール言語拡張用ドメイン言語プリミティブ

図８のAction-1に使われている&CONNECT関数は，ＣＬが標準に提供する動作関数ではなく，応用ドメイン固有の関数で，ＣＬユーザが次のルール言語拡張用ドメイン言語プリミティブを使用して生成したものである。

(rhs-language ACTION-FUNCTION
FUNCTION Arg-1…Arg-n)

ここで，ACTION-FUNCTION はＣＬでの動作関数名，FUNCTION以降は通常のLisp関数名とその引数である。これによって，ユーザがLispで定義した関数FUNCTIONをＣＬのルールインタプリタの中へ ACTION-FUNCTIONとして組込むことができる。

同様にＩＦ部でのルール言語拡張用プリミティブとしてlhs-language関数を提供する。

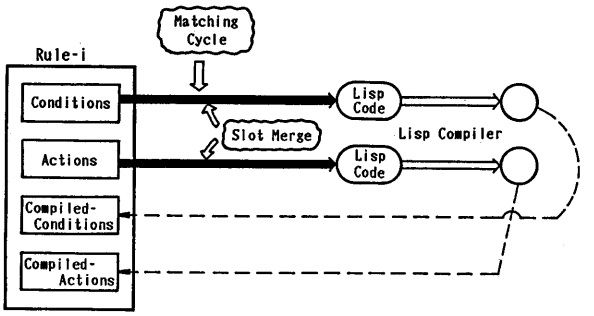

図９．逐次実行向ルールコンパイル方式

3.4 ルールコンパイラによる高速化

ＣＬでのルール指向では柔軟性を求めると共に高速化を促進するため，逐次実行向コンパイラと競合解消向コンパイラを提供する。

3.4.1 逐次実行向ルールコンパイラ

逐次実行型のルールインタプリタにおいては，競合解消の扱いがないため，処理効率を低下させる大きな要因は存在しない。むしろ，ワーキングメモリ上に存在するルールセットオブジェクトに対するアクセス方式の効率向上が望まれる。

そこで，ＣＬでの逐次実行向ルールコンパイラは，ルールインタプリタがルールオブジェクトの複数の条件節スロットと，帰結節スロットに対して毎回行なっている一連のサイクル（フェッチ→解釈→処理）を１回のＩＦ部処理と，THEN部処理に置き換える。図９は，ＣＬのコンパイル方式を図式化したものである。つまり，ルールオブジェクトの（複数の）条件節スロット，帰結節スロットをコンパイルしてそれぞれ１つのスロットにマージする。ここで，条件節のスロットマージを可能にするために，ルールインタプリタが行なっているマッチング機能をコンパイルされたスロットの中に埋め込んでいる。

さらに，１つのスロットにマージしたＩＦ部，THEN部はLISP関数としてLISPコンパイラによってコンパイルされ，ルールオブジェクトのCompiled-Conditions，Compiled-Actions スロットのバリューとして登録される。さらに，領域定義用関数によって求められる領域候補の値はどのルールも起動されなければ変化しないので，領域候補の値の再計算をしないようにコンパイルを行なっている。

以上のコンパイル方式を採用することで，ワーキングメモリ上のルールオブジェクトに対するルール実行過程の効率が大幅に向上する。

3.4.2 競合解消向ルールコンパイラ

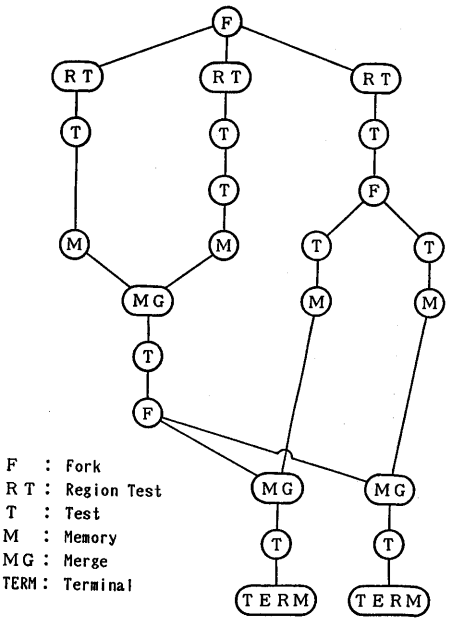

図10．コンパイルされたルールのネットワーク構造

ＣＬの競合解消向ルールコンパイラは，基本的にはReteアルゴリズム[6] を利用している。つまり，１ルールの帰結節の実行によって生じるワーキングメモリのオブジェクト単位での変化を，トークンとして記述し，コンパイルされたルールネットワーク上を，トークンが上から下へ流れ，最後のノードに到達したトークンによって競合集合の変化を求める。但し，ＣＬでは以下で述べる領域定義のテストノード（ＲＴ）の機能を追加拡張している。図10はＣＬにおけるコンパイルされたルールネットワークの構成図を示す。まず，Forkノード（Ｆ）に達したトークンは，全ての枝に流れる。次に，RegionTestノード（ＲＴ）では，変化のあったオブジェクトが領域定義を満たすかどうかのテストが行なわれる。領域定義には，①従来のReteアルゴリズムに対応するクラス名を指定し，そのインスタンスを領域とするものの他，②オブジェクトのスロットの値に指定されたオブジェクトの集合（リスト）を領域とするもの，③オブジェクトのスロットの値に指定されたクラスオブジェクトのインスタンスの集合を領域とするもの等が，それぞれ前述した直接的及び間接的指示法により指定できる。又，Ｔノードがルールの述語のテスト，Ｍノードがトークンの保存，ＭＧノードがＭノードに保存されていたトークンと新たに流れてきたトークンの合併，TERMノードが対応するルールの競合集合への登録／削除を行なう点は，従来のReteアルゴリズムと同様である。

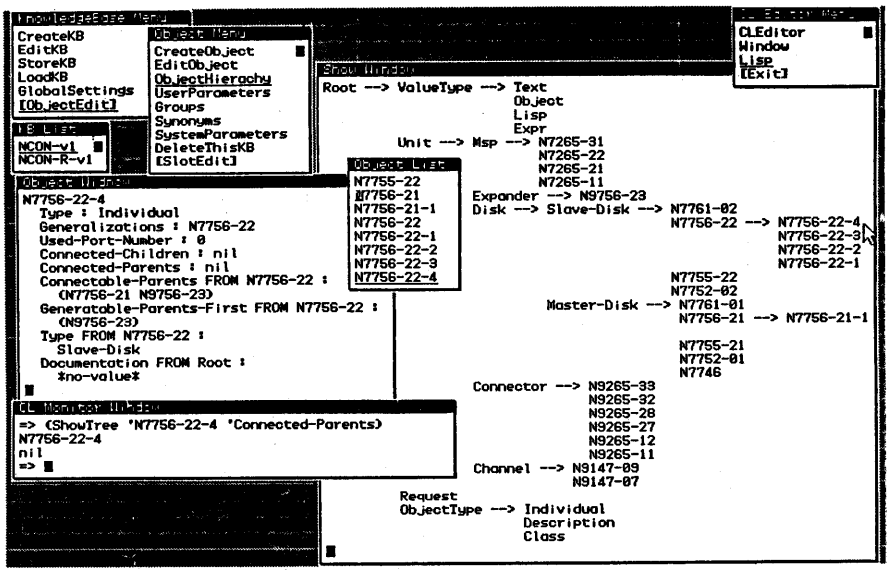

図12．初期ワーキングメモリ

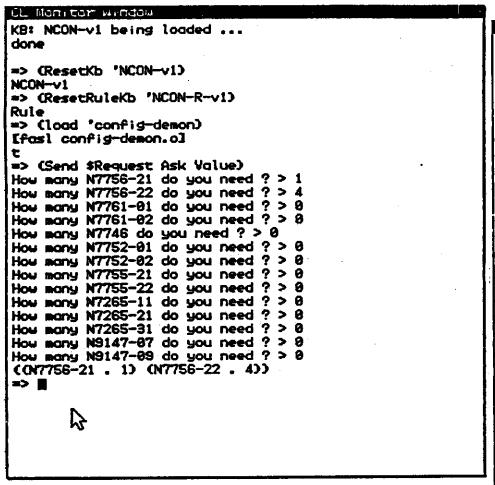

図11．要求入力例

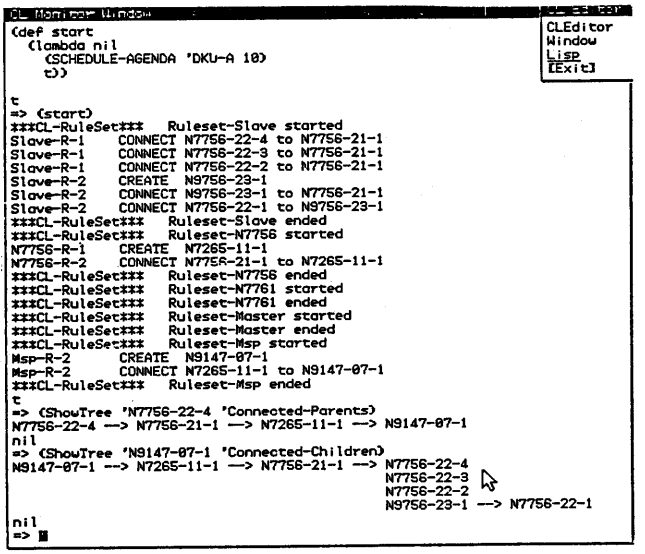

図13．実行トレース例

4．適用例

図11～13は，ＣＬを使って構築されたコンピュータシステム構成支援エキスパートシステムでの会話例をＥＴ[7] 画面で表示したものである。本エキスパートシステムは，ACOSに接続可能なデバイスを与えた場合，最適な接続関係を決定する。この時，不足なデバイスが追加される。まず，図11の例では，必要なデバイスのそれぞれの個数を入力して，初期ワーキングメモリ（図12）が生成される。図12は，初期ワーキングメモリのオブジェクト階層を Show Windowに，要求された磁気ディスクN7756-22-4の内容を Object Windowにそれぞれ示している。又，ＣＬ Monitor Windowでの(ShowTree 'N7756-22-4 'Connected-Parents) の実行値が nilにより，N7756-22-4が，この時点でまだ接続されていないことがわかる。図13は，本システムの実行トレースである。ここでは，まず， Ruleset-Slaveルールセットが起動され，その中の Slave-R-1ルールがN7756-22-4（スレーブ）をN7756-21（マスタ）に接続していること等がトレースされている。この実行後，図12で示し

たものと同じ関数（ShowTree…）を実行するとN7756-22-4→N7756-21-1→…→N9147-07-1という接続関係が表示され，最適な接続が行なわれたことがわかる。

尚，約20個のルールを持つ本システムは，LISP，ＣＬとも初めての人が，PROLOG上に既に作成されたルール[8] を解読して約３週間で構築された。

５．おわりに

ＣＬでのルール指向プログラミングに関して，その設計思想，概要，特徴等を報告した。特に，ルール言語拡張用や競合解消用のドメイン言語プリミティブを提供することにより，ルール指向における柔軟性を高めた。さらに，洗練化／構造化されたワーキングメモリとルールの高速処理を実現するためのルールコンパイル方式を逐次実行型と，競合解消型のそれぞれに対して提案した。最後に，ＣＬを使って構築されたエキスパートシステムの例を紹介した。このことにより，ＣＬを使った場合，エキスパートシステム構築時の生産性が大巾に向上することがわかった。

今後の課題として次の４点をあげる。

①ドメイン言語プリミティブの評価：
本稿で紹介したものは，ＣＬユーザに高い評価を受けている。一方，パターンマッチング記述用ドメイン言語プリミティブの洗練化を進めている。

②ルールコンパイラの実現と評価：
現在，ハンドコンパイルによって生成されたコードの評価を進めている。コンパイラ自身の実現とその評価を行なわなければならない。

③アジェンダ／ルールセット制御構造の洗練化：
現在，実現されているのは前向き推論だけである。今後，各種推論制御方式の拡充を行なっていく予定である。

④ルール，推論過程の説明機能の充実化：
ルール自身をよりわかりやすく説明する方法や，推論過程を要領良く説明する方式の検討が望まれる。

最後に，本研究の機会と貴重な助言を頂いたＣ＆Ｃシステム研究所山本昌弘部長，小池誠彦課長に深謝します。

参考文献

[1] 小林，“プロダクションシステム，”情処 Vol.26 No.12,1985,pp1487-1496.

[2] 渡辺，岩本，出口，“ＣＬ：ドメイン言語構築機能を強化したフレーム型知識表現システム，”情処　知識工学と人工知能研究会，1985.5.

[3] Forgy,C.L.,OPS83 User's Manual and Report, Production Systems Technologies,1985.3.

[4] Lafue,G.M.and Smith,R.G., “A Moduar Tool Kit For Knowledge Management,” Proc.of IJCAI'85,1985.8,PP46-52.

[5] Stefik,M.and et al, “Rule-Oriented Programming in LOOPS,”KB-VLSI-82-22(working paper),Xerox,1983.6.

[6] Forgy,C.L., “Rete：A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem,” Artificial Intelligence 19,1982, pp17-32.

[7] 松田，渡辺，“ＣＬ向け知識エディタ（ＥＴ），”情処第32回全大，1986.3,pp1159-1160.

[8] 出口，岩本，梅村，山本，“構成支援エキスパートシステムに於ける知識表現，”情処第30回全大，1985.3.,pp1475-1476.