ICOT Technical Memorandum: TM-0180

TM-0180

エキスパート・システム構築用ツール PHOENIXについて

麻生盛敏

June, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

エキスパート・システム構築用ツールPHOENIXについて

PHOENIX : A PROGRAMMING SYSTEM FOR BUILDING EXPERT SYSTEMS

麻 生 盛 敏
Moritoshi ASOU
財団法人 新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

[1] はじめに

最近、これまで研究室内でのみ研究されてきたエキスパート・システム、あるいはその構築用ツールであるエキスパート・シェル・システムが(以後これらを区別せずに、単にエキスパート・システムと呼ぶことにする)、研究室から飛び出し続々と実世界の業務に応用され始めており、未だ未熟ではあるものの、その広範な応用分野と可能性から注目を集めている。

このエキスパート・システムは、問題解決・推論、知識表現、知識獲得、知識ベース管理、知的マン・マシンインタフェース等、人工知能の研究における殆ど全ての要素技術を必要とする。既存のエキスパート・システムは別として、今後研究・開発されるシステムは、言わば人工知能の研究成果の集大成とでも言えるアプリケーションの一つと成るであろう。

我々は以上のような観点から、次世代のエキスパート・システムを研究・開発中であり、本論文ではこの一環として研究中のエキスパート・システム構築用ツールについて、特にその実験システムである PHOENIX-System V1 (Version 1) について述べる。

[2] PHOENIXについて

PHOENIX とは、Programming system for building HeterOrganized knowlEdge represeNtatIon of eXpert system の略称であり、読んで字の如くエキスパート・システムの知識表現、特に異種の知識表現を有機的に結合したシステムを設計することを目的としたプログラミング・システムである。また、PHOENIX-System V1 (Version 1) は、PROLOG で DECSYSTEM-20 上にインプリメント中のその実験システムである。

現時点では PHOENIX の全貌はまだ未定であるが、その最終的イメージは「広範な適用範囲を持つ知識表現言語を備えた、並列推論マシン上での分散/並列処理指向型のエキスパート・システム」である。

ところで、現実に稼働しているエキスパート・システムを概観してみると、その知識表現言語あるいはその知識表現の枠組みは、大雑把に言うとルール型かフレーム型(広義には意味ネットワーク)に分類されると言ってよい。そして最近になり、これらの異なる知識表現形式を一つに結合したハイブリッド型が現れ始めた。既存の枠組みには其々一長一短があり、これらを統合することによりその長所のみを取り入れたより良い知識表現形式を設計しようという試みと考えられる。この方式により設計されたシステムとして幾つかのものが提案されており、どの枠組みを中心に置くかにより其々特徴が出てくるわけであるが、その設計方針はどれもかわらないと言ってよいであろう。

しかしながら、これらのシステムに対して次のような批判がある[3]。「その発想は"FORTRAN, PASCAL, LISP, PROLOG 等のプログラミング言語には其々一長一短があるので、プログラミング環境として複数の言語を混在させて、ある言語から他の言語を呼び出せるようにする"というものによく似ている。つまり、これらのシステムでは単に知識の実体を表現する手段が複数用意されているだけに過ぎない。

ハイブリッド型システムは知識表現のレベルでの統合ではなく、その下のプログラミング言語レベルでの統合に過ぎない。この意味からすると"統合"というより"混合"と言うべきであろう」。

一方、フォン・ノイマン型のマシンの欠点が問題にされて久しい。即ち、「解くべき問題とアーキテクチャの間の隔たり、言い換えると高水準言語における概念とアーキテクチャにおける概念との間の隔たりであるセマンティック・ギャップがあるために、膨大なソフトウェアが必要とされる。そのために、コンパイラによって生成されるコードは問題を解くのに必要な部分より、その他の部分の方がはるかに多い」と言われている。同じようなこと(知識表現言語と並列推論マシンの機械語とのセマンティック・ギャップを埋めるということ)がエキスパート・システムを設計する際にも言えるであろう。分散/並列処理指向型のエキスパート・システムは、このシステムが稼働するマシン、即ち並列推論マシンのアーキテクチャと密接に関係しており、これを抜きにしては考えられない。言い換えるならば、エキスパート・システムの中核となる基本知識表現言語とそのアーキテクチャ(エキスパート・システム及び並列推論マシンのアーキテクチャ)は一体となって開発されなければならないということである。

また、仮に千~一万台程度のプロセッサからなる並列推論マシンが出来たとしても、ただ単に並列実行しただけではシステムの性能は大して向上するわけでもなく[2]、知識の有効利用、推論の制御戦略等が問題となる。

このように、問題は山積みしているわけであり、知識表現という問題一つ取ってみても一般的な知識表現の枠組みを設計しようとする場合には、その知識表現の範囲や表現のきめの細かさ、その評価尺度等、解決しなければならない問題が多々ある。そこで現在我々は図.1に示すような二段階の設計方針で、その基本知識表現言語を中心にPHOENIXシステムの研究開発を進めている。

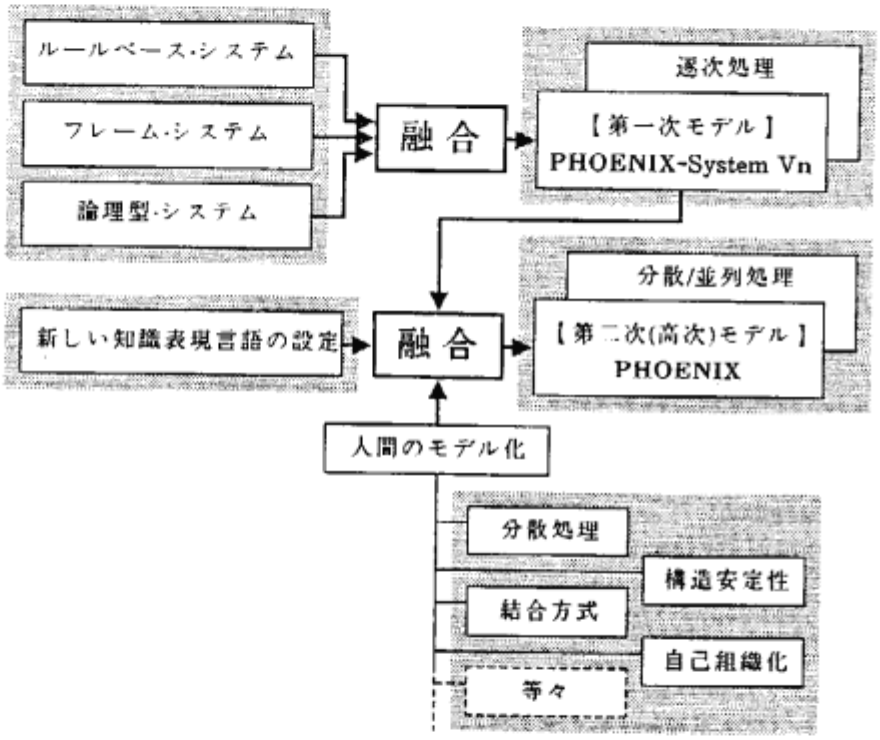

図.1 PHOENIXシステムにおける知識表現言語の設計方針

[3] PHOENIX-SystemV1について

前章で述べたように、PHOENIX-SystemV1は一次モデルとしてPROLOGでDECSYSTEM-20上に開発中の実験システムである。以下このシステムについて述べる。

[3.1] 知識表現言語の位置付け

一次モデルとしてのPHOENIX-SystemV1の開発にあたって、まずその基本となる知識表現言語の位置付けを図.2に示すように設定した。簡単に言うならば、論理型・システムの中へルールベース・システムとフレーム・システムの機能を取り込んだハイブリッド型システムと言うことができるであろう。この図.2については、ほぼ一目瞭然と思われるので、具体的な知識表現形式について次の節で述べる。

[3.2] 知識表現形式

PHOENIX-SystemV1のプログラムは、図.3に示すような各々独立したモジュール(**module**)の集合から構成される。各モジュールは、メタ記述部(**meta_description**)、リンク記述部(**link_description**)、

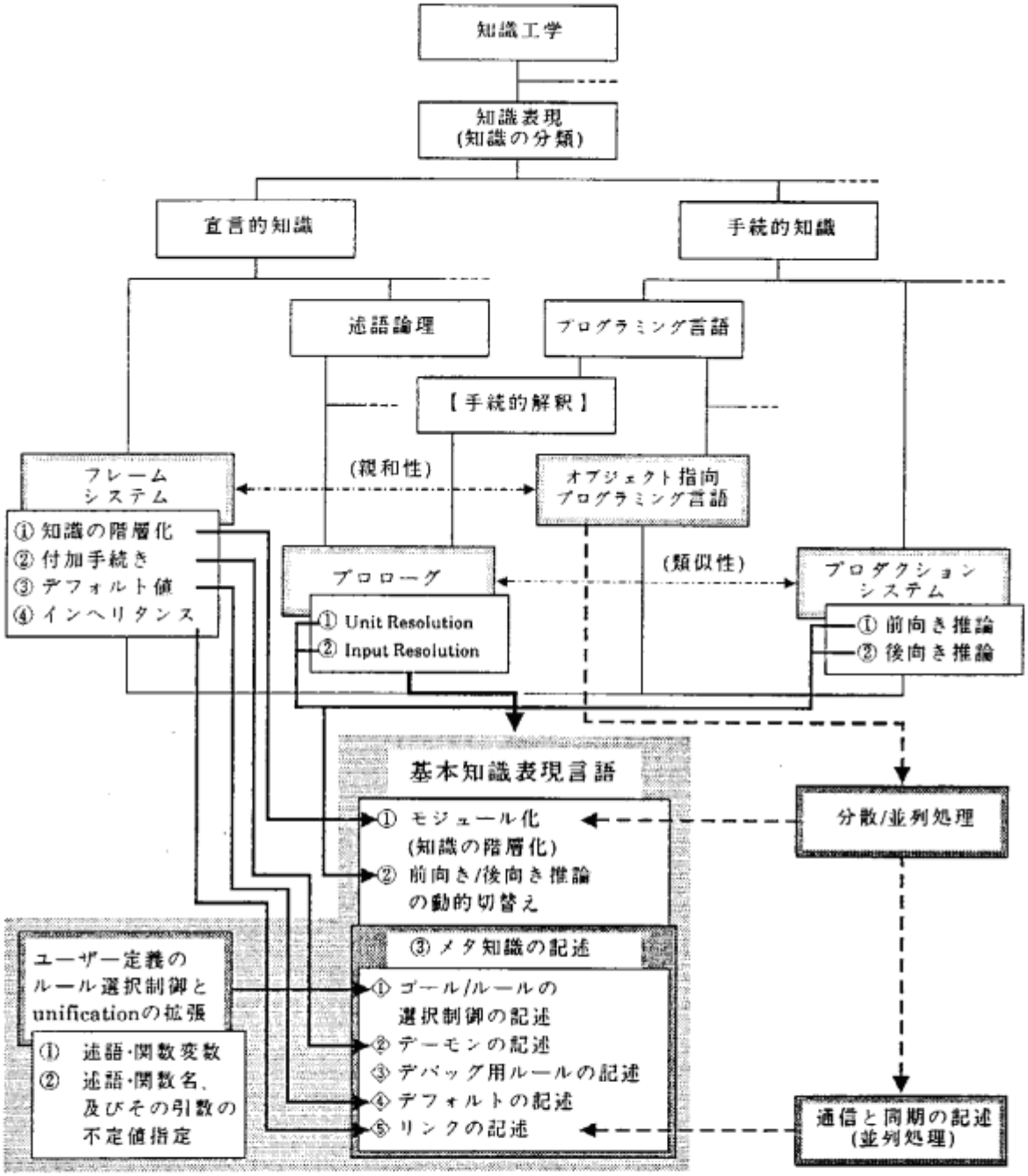

図.2 PHOENIX-SystemV1における知識表現言語の位置付け

ファクト記述部(**fact_description**)、ルール記述部(**rule_description**)、の四つの部分から構成され、更にメタ記述部は制御記述部(**control**)とデフォルト記述部(**default**)に分かれる。これら四つの記述順位は順不同であり、また必ずしも全ての記述部を書く必要はない。次に各記述部について説明する。

① ファクト/ルール記述部:**fact_description/rule_description.**

ファクト記述部には、PROLOGで言うところのいわゆるファクトを記述する。また、ルール記述部には、同じくファクト又はルールを記述する。このようにファクトは、ファクト記述部あるいはルール記述部どちらに記述してもかまわない。この違いは、後述のメタ及びリンク記述部の機能と合わせて、ファクト記述部に書かれたファクトと、ルール記述部に書かれたルールをそれぞれスロット及び付加手続きと見なし、

```
module  MODULE_NAME.
   meta_description.
      control.
         select_goal : GOAL :- CONDITION ; (true/fail/skip).
            .....
         select_rule : RULE :- CONDITION ; (true/fail/skip).
            .....
         demon       : (before/after) ; GOAL :- CONDITION ; ACTION.
            .....
         switch_No   : on.
            .....
         debug       : (before/after) ; GOAL :- switch_No  ; ACTION.
            .....
      end_control.
      default.
         DEFAULT     : CONDITION.
            .....
      end_default.
   end_meta.
   link_description.
      DATA (<--/<->/-->) LINK_NAME.
                  .....
   end_link.
   fact_description.
      FACT.
       .....
   end_fact.
   rule_description.
      RULE.
       .....
   end_rule.
end_module.
```

図.3 PHOENIX-SystemV1における知識表現形式

このモジュールをフレームとして扱うかどうかの違いによっている。またルールの形式には次の三つの形式があるが、これらの違いはそれぞれのルールの推論の向き(前向き/後向き/両方向)によるものであり、その意味はどれもPROLOGのルールと同じである。

① 前向き推論用ルール

[-] *HEAD* <== *BODY*.

ここで、"-"記号を付加した場合は*HEAD*部とマッチするファクトをWorking Memoryから削除することを意味する。

② 後向き推論用ルール

HEAD ==> *BODY*.

③ 両方向推論用ルール

HEAD <=> *BODY*.

② メタ記述部:meta_description.

メタ記述部は制御記述部とデフォルト記述部から成る。

Ⓐ 制御記述部:control.

① ゴールの選択制御

形式:select_goal : *GOAL* :- *CONDITION* ; (true/fail/skip).

意味:*GOAL*部とマッチするゴールの選択時に*CONDITION*部をチェックし、次の動作を行う。

ⓐ **true** : そのゴールを実行する。

ⓑ **fail** : そのゴールの実行を、この時点で無条件に失敗させる。

ⓒ **skip** : そのゴールの実行を後回しにする。

② ルールの選択制御

形式:select_rule : *RULE* :- *CONDITION* ; (true/fail/skip).

意味：*RULE*部とマッチするルールの選択時に*CONDITION*部をチェックし、次の動作を行う。

ⓐ **true**：そのルールを選択する。

ⓑ **fail**：他の候補も含めたルールの選択を、この時点で無条件に失敗させる。

ⓒ **skip**：そのルールの選択を放棄し、次のルールの選択を試みる。

④の場合も含めて、*CONDITION*部が失敗した場合には、そのゴール/ルールの選択は成功する。

③ デーモン文による制御

形式：**demon**：(**before**/**after**) ; *GOAL* :- *CONDITION* ; *ACTION*.

意味：*GOAL*部とマッチするゴールの実行前(**before**)あるいは実行後(**after**)に*CONDITION*部をチェックし、*ACTION*部で指定された動作を行う。
このデーモンの実行の失敗は、元のゴールの実行に影響を与えない。

④ デバッグ文による制御

形式：**debug**：(**before**/**after**) ; *GOAL* :- **switch**_*No* ; *ACTIONs*.

意味：*CONDITION*部が異なる以外は、③のデーモン文と同じである。**switch**_*No*：**on**時に*ACTION*部で指定された動作を行う。ここで、"*No*"部分には自然数が入る(0はシステムトレース用)。

⑤ ユーザの定義によるルールの選択制御

上記以外に特別のインタフェースが存在し、このインタフェースを通してユーザ独自のルール選択制御が出来るようになっている(PROLOGでプログラムする)。この中でのユニフィケーションは拡張され、述語/関数変数の使用と述語/関数名及びその引数の"不定値指定"が出来るようになっている。例えば、次のようなユニフィケーションが行える。

ⓐ **'$P'(X, '$F'(Y, Z)) :- q(X, Y), '$P'(Z, '$F'(X, Y))** と
p(A, f(b, C)) :- q(A, C), p(c, f(a, B)), 等 (**$**で始まる文字列が述語/関数変数に当たる)

ⓑ **'p1*3*'(*, a, *, X, *)** と
p123435(1, 2, a, 3, 4, 5, a, 6, 7, 8), 等 (*部分が"不定値指定"に当たる)

Ⓑ デフォルト記述部：**default.**

形式：*DEFAULT*：*CONDITION*.

意味：ファクト及びルールを用いて解が求まらない場合にこのデフォルト記述部を参照し、実行中のゴールが*DEFAULT*部とマッチし、かつ*CONDITION*部を満足すればこれを解とする。

尚、Ⓐ,Ⓑを通して*CONDITION*及び*ACTION*部の形式はルールの*BODY*部と同じであり、この中からファクト/ルール記述部を参照出来る。

③ リンク記述部：**link.**

形式 ①：*DATA* <-- *LINK_NAME*.
②：*DATA* --> *LINK_NAME*.
③：*DATA* <-> *LINK_NAME*.

意味：リンクはモジュール間を結ぶ論理的なパス(枝)であり、このパスを通してデータやメッセージの参照/被参照、通信、あるいは分散/並列実行時の同期等が行われる。更に、このリンクを通してフレーム・システムにおけるインヘリタンス機能を実現している。上記の*LINK_NAME*は、これも一つのモジュールの名前であり、このモジュール内のルール等によりこのリンクが特徴付けられる(リンクに対するアクセスが制限される)。また、上記矢印はモジュールに対するこのパスの方向を表し(<--：入力、-->：出力、<->：双方向)、このリンクに対するアクセスの前後では上記*DATA*部とのパターンマッチ(ユニフィケーション)が行われる。従って、*DATA*部を変数にし*LINK_NAME*に対するモジュールを定義しなければ、このリンクに対する全てのアクセスが許

される。図.4はこれらモジュールの関係を図示したものであり、三つのプログラム・モジュールがあり、MODULE#1とMODULE#2がPATH#1とPATH#2で、MODULE#2とMODULE#3がPATH#3で、それぞれリンクされている。また、モジュールPATH#3内にはこれを制限するルールは無く、PATH#4を通して無条件にMODULE#4を参照している(MODULE#4内のルールをインヘリットしていると考えられる)。

[3.3] 推論の方式

推論の基本は、Unit/Input Resolutionを用いた前向き/後向き推論である。ルールの適用に関しては、表.1のようにその時点のシステムの実行モードとルールの推論方向により決まる。ゴール/ルールの選択に際しては制御記述部による制御が可能であり、また後向き推論の場合には、ファクトから優先してリゾルベントを求める。ゴールとルールだけでは解けない場合にはデフォルト記述部を参照し、それでも解けない場合には参照可能なリンクを辿る。

表.1 ルールの適用の可否

ルールの推論方向 ＼ システムの実行モード	前向き	後向き	両方向(前向き,次に後向き)
前向き推論用ルール	可	不可	可
後向き推論用ルール	不可	可	可
両方向推論用ルール	可	可	可

[3.4] システムの構成

図.5に PHOENIX-SystemV1のシステム概念構成図を示す。基本的にはルールベース・システムのアーキテクチャと同じものであると考えられる[1]。プログラム(知識)は、ユーザ・インタフェース中の Translatorにより変換され、KBMS(Knowledge Base Management Subsystem) 下に KS(Knowledge Source) として格納される。このKBMS 下の知識KSは、PWMS(Production & Working Memory Management Subsystem) 及び推論システム中の Interpreter と Meta Controllerの制御により PWM(Production & Working Memory)上に活性化される。活性化されたKSは PWMSによりコンパイルされ、Interpreter と Meta Controller はこのコンパイルされた知識を基にして推論を実行する。この Interpreter と Meta Controller は複数の Unification Engine 上で稼働することを想定している。また、推論システム中の Scheduler, Agenda, 及び Task/Process Queue により候補となるルールの選択/優先度制御を行う。システム構成については今後さらに詳細を検討していく予定である。

[4] おわりに

現在の実験システムである PHOENIX-SystemV1 は、図.3に示した知識表現言語を中間言語(PROLOG形式)に変換するトランスレータと、これを直接解釈実行するインタプリータとからなる。
この PHOENIX-SystemV1 の基本となる知識表現言語を設計するに当たって考えた機能の内、今回インプリメントしなかった機能として曖昧な推論、仮説推論、時間に関する推論 等がある。更に、図.2に示したように、フレーム・システム、プロダクション・システム 等の特徴的な機能を取り込んで知識表現言語を設計したわけであるが、これらのシステムとの比較検討を含めた評価を行う必要がある。今後これらを含め、図.5のシステム構成に従って改良していく予定である。

最後に、日頃御討論して頂く第五研究室 岩下室長ならびに研究員諸氏に感謝する。尚、下記以外にも多くの文献を参考にさせて頂いたが、省略する。

[参考文献]

[1] Frederick Hayes-Roth, Rule-Based Systems. Communication of *ACM*, vol.28, no.9, pp.921-932, Sept. 1985
[2] 辻 三郎, 知識表現と推論制御. 情報処理, vol.26, no.12, pp.1475-1480, Dec. 1985
[3] 松原, 井上, 知識表現の枠組の統合に関する考察. 知識工学と人工知能研究会資料No.40, 情処研報, vol.85, no.1, pp.1-8, May. 1985

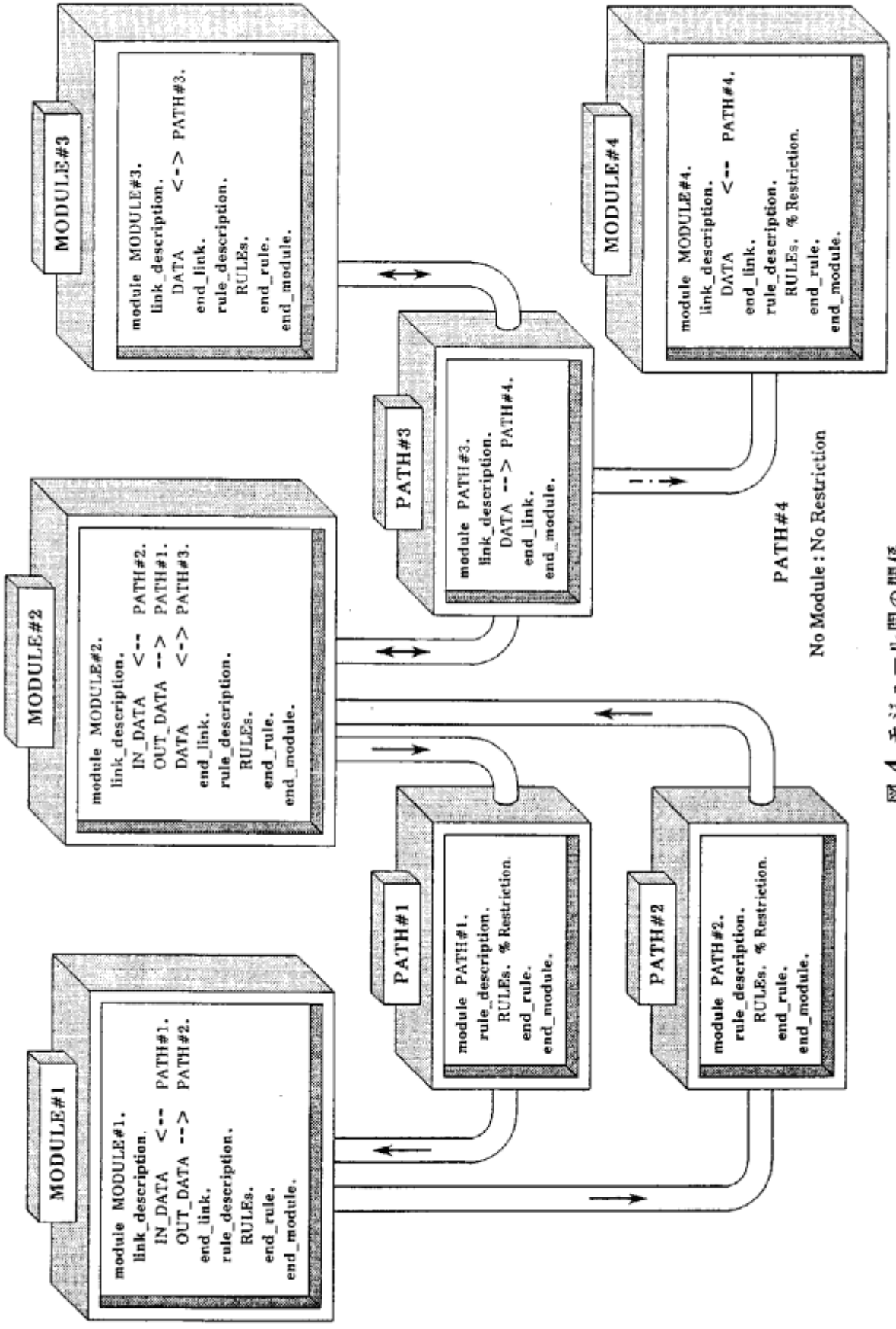

図.4 モジュール間の関係

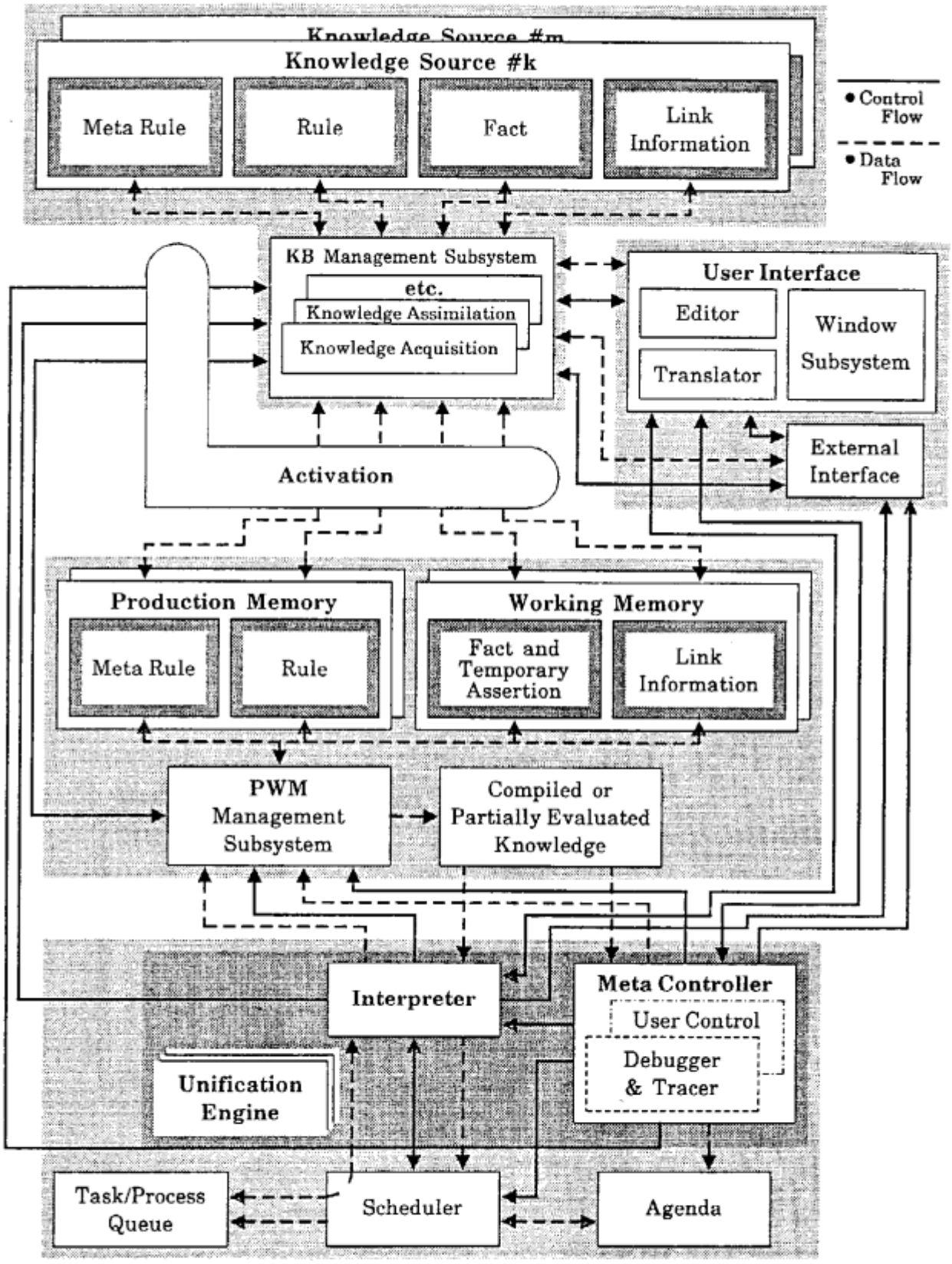

図.5 PHOENIX-SystemV1のシステム概念構成図