TM-0063

KAISER/KIM Architecture
—Preliminary Version of Knowledge Interface Module—

北上　始，宮地泰造
古川康一，国藤　進

June, 1984

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

KAISER/KIM Architecture

[Preliminary Version of]
[Knowledge Interface Module]

1984.6/01

ICOT

第２研究室

北上　始，宮地泰造
古川康一，国藤　進

［概要］

ICOTの第５世代コンピュータプロジェクトでは、複数台のパーソナル推論マシン（PSI）および一台の関係データベースマシン（DELTA）を、LAN 経由で接続し、新しい知識情報処理の方向をさぐろうとしている。知識ベースのデータおよび知識は、関係データベースマシンばかりでなく、パーソナル推論マシン自身がもつ関係データベースにも蓄積されている。

本報告は、このような計算機環境下で、知識獲得システムの１モジュール KAISER/~~KSM~~ のアーキテクチャについて述べる。

目　次

１．はじめに

第五世代コンピュータ・プロジェクトでは、知識情報処理システムの構築を目標としている。現在、パーソナル型逐次推論(PSI) マシンと関係データベースマシン（Delta)の試作・検討を実施している。PSI マシンは、ESP と呼ばれる論理プログラミング言語で書かれたプログラムを実行するマシンであり、Delta は、関係代数レベルのコマンドから成るコマンド木(Queryなど）を実行するマシンである。

さらに、ＩＣＯＴでは、数台の推論マシンと１台の関係データベースマシンがLAN(local Area Network)により接続されている。上記、PSI マシンは、ネットワークサブシステムをもつが、LAN を経由したPSI マシンとDelta マシン間およびPSI マシン間の通信は、このサブシステムが提供するコマンドにより達成される。

図１に、本プロジェクトの第１ステージにおけるハードウェア構成を示す。この図の破線は、本プロジェクトの中期に達成予定の PSIマシンと、RDBM "Delta"間のDirect Coupling を、表わしている。このDirect Coupling の件は、今回、試作予定のKIM とは関係をもっていない。

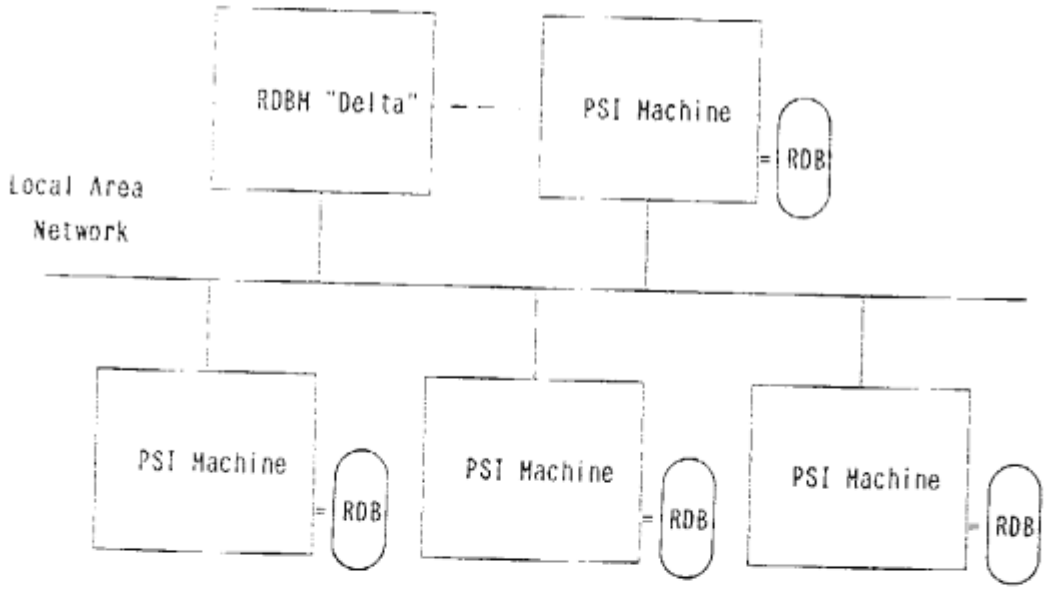

図１．第５世代コンピュータのシステム構成

関係データベースは、Delta ばかりでなく、各々の、PSI マシンにも存在している。

この様な、環境下において、PSI マシンが備えているPrologベースの論理プログラミング言語 (ESP)と、PSI マシン及びDelta がもつ関係データベース操作用言語（関係代数レベルの言語）の各々とインタフェースをとり、サイト間の情報交換を円滑に遂行するモジュールが必要である。このモジュールを KIM (Knowledge Interface Module) と呼ぶ。図２は、この KIMの前後に存在するソフトウェアモジュールの階層関係を図示したものである。

本報告書では、このようなソフトウェア環境下で位置づけられている KAISER/KIM のアーキテクチャについて、述べる。

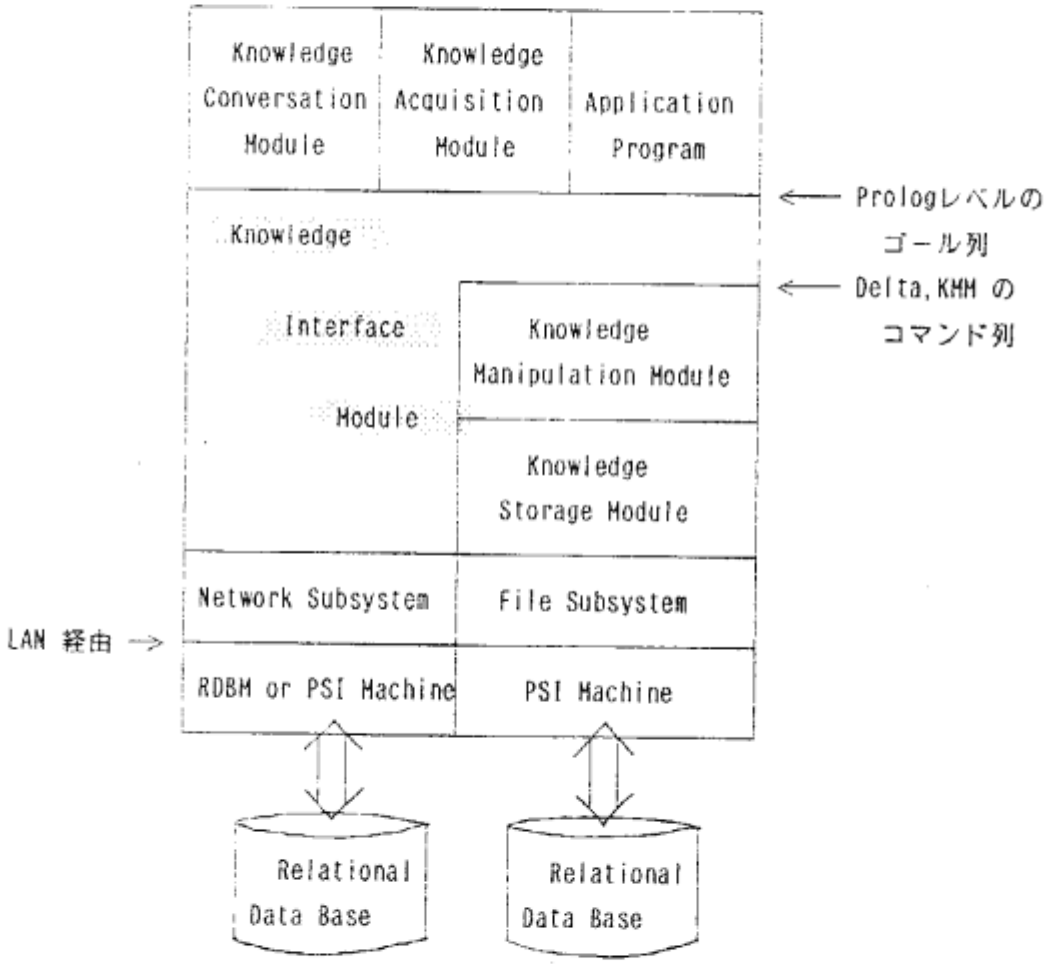

図２．ソフトウェアモジュールの階層関係

2．ＫＩＭに対する問題設定

ESP(Prologレベルの言語）と、RDB(関係代数レベルの言語）のインタフェースを実現することにより、KIM は、下記に示す分散知識(Rules,Facts)を使用し、ユーザが与えたPrologのゴール列を、実行解釈することができる。ここで、質問処理のためにユーザが直接コマンドを入力したサイトを主サイトと呼び、その処理に関連するサイトを、従サイトと呼ぶ。

(a) 主サイトで、Library に登録されているLibrary Program(Rules.Facts)。
(b) 主サイトがもつ Relational Database (Facts)。
(c) Delta に格納されている Relational Database (Facts)。
(d) 従サイトの PSIマシンにあるLiabrary Program(Rules, Facts) および Relational Database (Facts) 。

(a) ～(d) を直観的に説明するために、知識の分散配置関係を図３に示す。

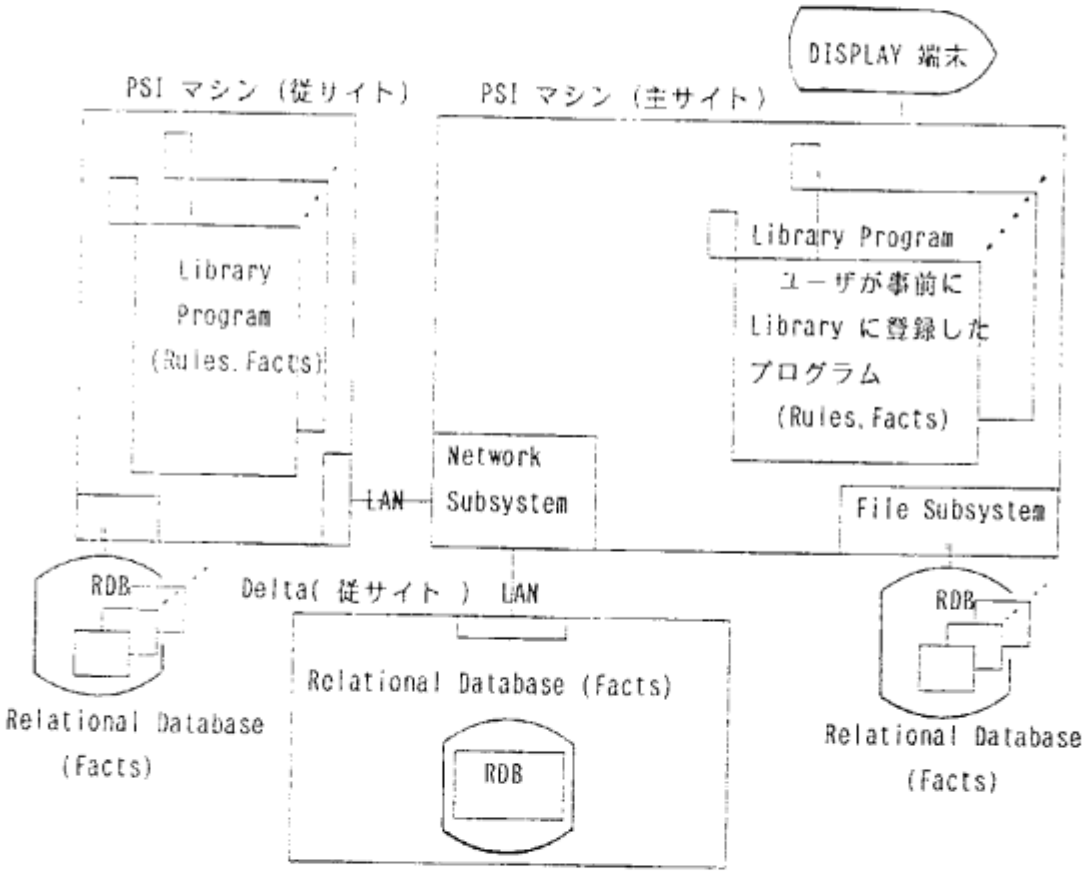

図３　知識の分散配置関係

KIM は、(a) ～(d) までのProgram およびRelational Database が主サイトから操作できる様にするメカニズムを必要としているが、(d) に関しては、中期に達成される協調問題解決メカニズムの一貫として、今後の作業に残しておく。
また、このRelational Database が、Facts のみならずRules をも蓄積できるようになると、さらに進んだ議論の展開が予想される。

KIM の設計に際し、Relational Database は、Frame の集合で構成されるとする。
この Frameは、ある特定のまとまりをもつRelationの集合であり、それらのRelationを管理する辞書は、その Frameの中に、Relationの形式で、蓄積されている。
この Frameは、KMM がもつ Knowledge Base の概念に等しい。Delta では、現在、ひとつの Frameしかもっていないとする。以後、 Frame, Relation(同じ関係名をもつFactの集合), Program 化されたRuleを総称して、オブジェクトとよぶ。

インプリメントの第１段階では、混乱を避けるため、各サイトの知識 (Rules,Facts)を更新するためのassert又はretract などの更新系の述語が、Library Program およびKIMへの入力ゴール列の中に、入っていないものとして、問題を整理した方が望ましい。

３．ＫＩＭの概略構成

図４に、KIM の概略構成図を示す。

KIM にPrologのゴール列が渡されると、KIM Manager は、次のサブモジュールの制御を実施し、Prologのゴール列を実行する。

Separator は、ゴール列をLibrary Program の上で部分評価し、他の場所で評価すべきゴール列を抽出する。抽出されるゴール列は、評価される場所単位に分類されている。

Translatorは、Separator の分類に従って抽出されたゴール列を、それぞれのインタフェース仕様に合わせて変換する。

Database Connectorは、データベース管理システムを使用するために前処理を行う機能をもつ。

Network Connector は、サイトの仕様に合わせて変換されたコマンド列をNetwork を経由して送信したり、その実行結果を受信したりするときに必要とされる前処理を行う機能をもつ。ここで、Network Connector の一機能である Cache Managerは、Network Subsystem とKSH , KSC 間でデータの受け渡しを行うサブシステムである。

以上のプロセスの実行により取得されたFacts は、ユーザが与えたゴール列の別解を与えるときに、うまく割り付けられていかなければならない。

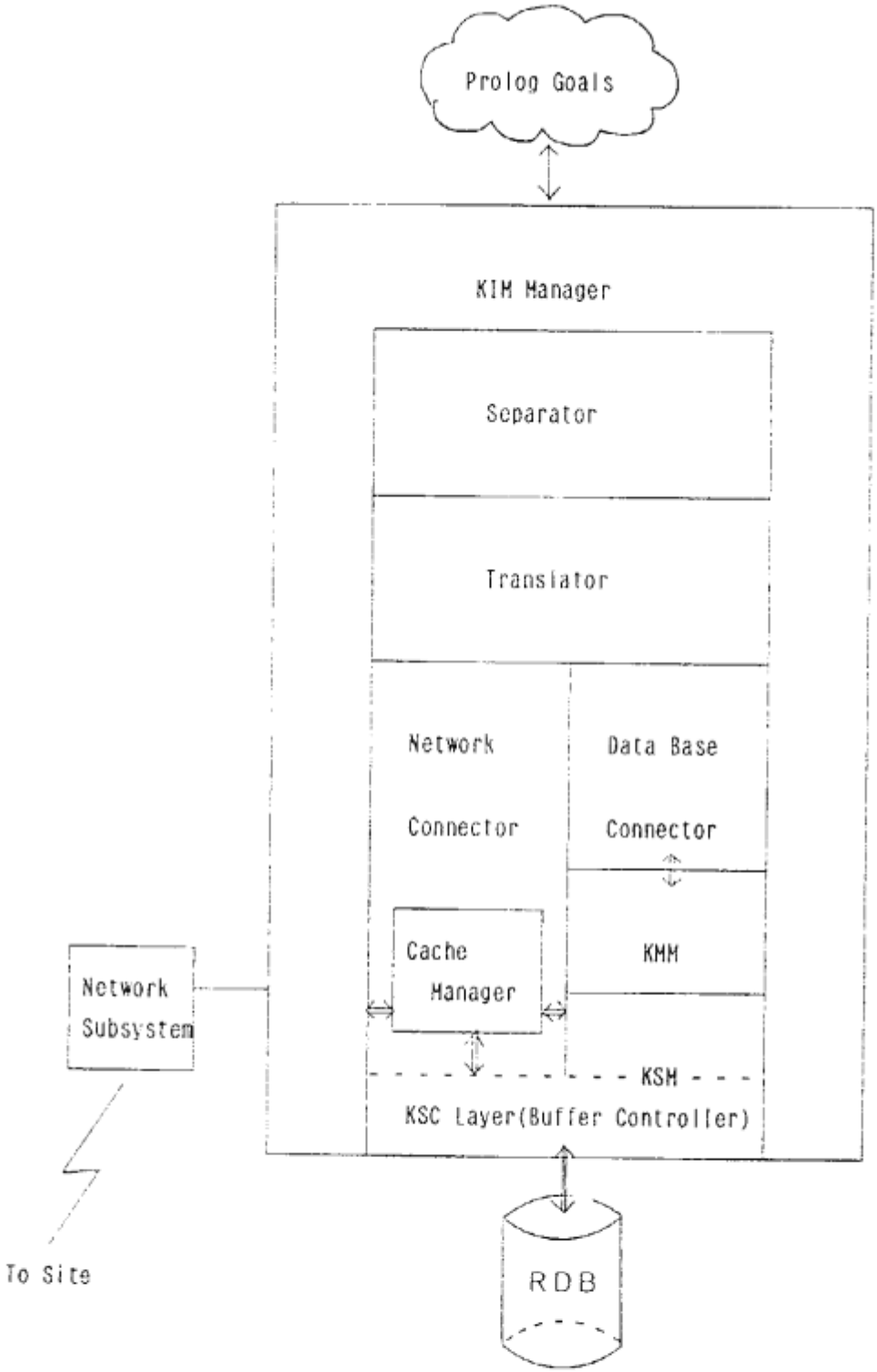
Prolog Goals
KIM Manager
Separator
Translator
Network
Connector
Data Base
Connector
Cache
Manager
KMM
Network
Subsystem
KSM
KSC Layer(Buffer Controller)
To Site
RDB

図４．ＫＩＭの概略構成

４．ＫＩＭの主要機能

KIM を設計するための第一目標として、各サイト間に分散しているオブジェクトに、サイト間のモジュラリティをもたせる（サイトの自治権を確立できる）。そのために、各サイトの各々のオブジェクトの名前に SWN (System Wide Name) をもたせることにする。SWN は、オブジェクトの創成時に定められる。

SWN ::= [知識創成者のUser-ID , 知識創成者がいるサイトの名前,
オブジェクトの名前, 知識が誕生したサイトの名前]

ここで、このサイトの名前は、そのサイトのTerminal Addressと一対一の対応がつけられた識別子であり、各サイトで、その対応関係が管理されている。また、SWN 内のUser-IDは、ユーザが任意に使用上の都合でつけた名前である。

オブジェクトの名前を使用するにあたって、直接、長い、SWN を使用することを回避するために、KIM に、次のコマンドを提供させた方がよい。

define_synonym (オブジェクトの同義語, SWN).

また、この様な定義がないときは、使用しているオブジェクトの名前は、次に示すSWNを、もつものとする。SWN=[利用者のuser-ID , 利用者側サイトの名前, 使用中のその知識名, 利用者側サイトの名前]。

以下に、各サブモジュールの主要機能について、概説する。

４－１．KIM Manager

KIM Manager は、知識インタフェースをとるための処理を統合・制御するサブモジュールである。この中で、System Catalog管理をはじめとして、KIM の実行環境に対するパラメータの設定・変更、Cache Manager の制御（従サイトの辞書をコピーとして蓄積・管理などを実施する）、KMM がもつ辞書とDelta がもつ辞書の規格化提供、Network Subsystem と KSM がもつサブモジュール間のデータ交換、セキュリティ管理、ジョブ管理、トランザクション管理、エラー処理などをおこなう。

複数サイトをもつネットワーク上で、知識情報処理を効率的に実施するために、サイト間でオブジェクトのコピーまたは移動を行う方法があるが、各々、固有の特徴をもっている。コピー方式は、同じ知識がネットワーク上に複数できるため、その知識に対するメンテナンスが、その知識を更新するたびに必要である。移動方式は、同じ知識をネットワーク上に作成しないので、そのメンテナンスが不要である。

その代り、あるサイトの処理効率を向上するために実施する知識移動操作が、他のサイトにおける処理効率を低下させる危険性を生じることがある。これは、サイトをどのような立場で使用するかという、トレードオフの問題を含んでおり、ユーザの適切な判断が必要である。 copy および migrateコマンドの形式は、

copy(コピー対象オブジェクトの名前，サイトの名前，オブジェクトの名前)，

migrate (移動対象オブジェクトの名前，サイトの名前，オブジェクトの名前)

である。

4－2．Separator

Separator は、ゴール列をLibrary Program の上で部分評価し、他の場所で評価すべきゴール列を抽出する機能をもつ。KIH へのコマンド入力は、 kim_solve(ゴール列) のような形式で受けつけられるとする。

Library Program で評価できないゴール列は、他の場所で評価されるが、そのゴール列を抽出するためには、次に示す２つの方式がある。

《方式１》

Library Program 又はゴール列の中に他の場所で評価するための指示を直接与える。そのためには、次に示す述語が必要である。

・connect(サイトの名前，オブジェクトの名前，Goal)．

"connect" 述語は、そのサイトの Relational Database又は、Library Program 内にあるオブジェクト上でGoalを評価する述語である。"connect" 述語中にあるサイトの名前は、サイトのTerminal_Address と一対一の関係をもつ論理識別子である。この関係は、System Catalogに登録されている。すなわち、次の形式のいずれかで、使用される。

- Head :- G1,G2,..,connect(...),... ----------> プログラム内。
- ?- kim_solve((G1,G2,..,connect(...),...)). --> 入力コマンド内。

《方式２》

Library Prgramにもゴール列にも他の場所で評価するための指示を与えない方式。Library Program 上で部分評価できずfailしたゴールが、主サイトにある Relational Databaseに存在しないときは、他のサイト（既に、この処理のために使用することが宣言されている）に、リレーションが存在するか否かを判定する。

もし、そのゴールが他のサイトで評価できるならば、それを抽出する。そのゴールが主サイトの Relational Databaseで評価可能なときも、それを抽出する。上記、判定のどれにも満足しないときは、そのゴール列は、宣言されているいかなる知識を対象にしても評価できないとする。KIM が、処理対象とするサイトおよびそのサイト内フレームは、次のコマンドにより定義されるとする。ただし、主サイトのLibrary Program は、この定義がなくとも実行対象になっている。

define_connection(サイトの名前, フレームの名前リスト).

また、ディフォルト値として、ゴール列の評価対象になっている知識は、主サイトの Relational Database又は、Delta がもつ Relational Databaseに蓄積されるとする。すなわち、define_connection(主サイトの名前, [])および
define_connection(Delta の識別名, [])のコマンドを、
実行させたのと同等である。[]は、そのサイトのすべての Frameを、処理対象とすることを意味する。他の知識領域との関係づけは、System Catalogに登録すべきである。

以上の説明において、方式１は高速処理が期待されるが、方式２は、簡易なユーザインタフェースが期待される。

いずれの特徴も、すてがたい面があるので、できれば、方式１及び２を双方、実現しておいた方が好ましい。

図５は、Separator を実現する上で中心的な役割を果す部分評価用のプログラム例である。図中の get_connector は、外部データベースの中に、述語 P が存在するときにかぎり、その P に対するサイト名およびフレーム名をconnect 述語で表現し、その述語を、Q に返す述語である。この外部データベースは、主サイト内で、その使用が宣言されている。

```
demo(Frame,true,d(X,X)) :- !.
demo(Frame,(P,Q),d(X,Z)) :-
    demo(Frame,P,d(X,Y)), demo(Frame,Q,d(Y,Z)).
demo(Frame,connect(Site,Frame1,P),d([connect(Site,Frame1,P) | X],X)) :-
    check _object(Site,Frame1,P) .
demo(Frame,P,d(X,Y)) :-
    system(P) -> P ;
    get _next_clause(Frame,P,Q),demo(Frame,Q,d(X,Y)).

get _next_clause(Frame,P,Q) :-
    X=..[Frame,Clause],X,
    (Clause=P -> Q=true ; Clause=(P :- Q) -> true ; get _connector(P,Q)) .
```

図５．部分評価プログラムの例

4－3．Translator

入力されるゴール列は、あるサイトで評価されるゴール列である。このゴール列が、Delta 又は主サイトがもつデータベースで評価できるものならば、各々の定められた形式の関係代数列に変換される。

Join, Cartesian などの演算を、ゴール列から抽出するためには、まずゴール間を結ぶ共有変数に着目し、ゴール列をクラスタリングしなければならない。同じ共有変数をもつゴールを寄せ集めて、クラスタリングすると、次の様なクラスタが抽出される。

①単一ゴール

②ゴール列とそれらを結ぶ共有変数

③ゴール列（共有変数なし）

①～③は、各々、Restriction , Join, Cartesian に変換される。

変換により形成された関係代数列の中に不要な引き数の受け渡しがある場合は、早めに Projection をかけておく。この他、関数代数レベルの最適化も可能な限り検討し、インプリメントするものとする。

以下の様に変換される関係代数は、評価するサイトによっても多少の違いがある事が多いので、変換ルールをルールベース化しておいた方が望ましい。

Planの作成方法に関する詳細なルールは、付録－１，２に示してあるが、各々、 KMM, Delta の使い方に関するルールであり、 DCG として、まとめられている。

4－4．Data Base Connector

KMM を使用するに当っての簡易インタフェース及びTranslatorで吸収できなかった処理をここで実現する。Frame の一元管理は、このモジュールで、実施すべきである。

4－5．Network Connector

Network 経由で取得されるData Dictionary を管理すると同時に、ネットワークサブシステムが用意するコマンドを使いながら他のサイトとの通信を行う機能をもたせなければならない。

以下にPSI マシンとRDBM間の接続において、考えられる作業項目を列挙する。

(1) ディクショナリ(Deltaのterminology では、スキーマに関するメタ情報を特殊なリレーションとして扱い、これをディクショナリと呼ぶ）の管理。

(2) ユーザのセキュリティの管理（アクセス権）
これは必要不可欠というわけではないが、Toolとして使うのであれば備えたい機能。これは、今年度の作業に含めない。

(3) フォーマット変換(Binary →Hexadecimal →Ascii 変換、Delimiter 挿入、アトリビュート名→アトリビュート番号変換（ディクショナリ・ルックアップによる））。

(4) 上位レベルの通信コントロール
コマンド木、（Delta からの）レスポンス（正常、異常）の送達確認必要ならばRetry 等。

(5) トランザクション管理
Delta では、start-transaction 、end-transaction でnestされる範囲をTransaction と定義し、この内部でのUpdateの単位性を保証している。Delta では、Transaction のLimit-time、SIM からのresponseのlimit-timeなどを設定して、時間監視をしているので、これに対応した管理が必要。非同期コマンドとしての sense-status （DEC20 のsystatのようなもの）とか、Delta コマンドのアボート（＾Cに対応）をする abort-processing の発行もこの管理といえるかも知れない。

(6) Bufferの管理
結果リレーションの受取、新タプルの挿入(GET、PUT コマンドに対応）を行う時のバッファリング．GET するリレーションをSIM 内のdouble-buffer で受ければSIM 内の処理と結果の転送がparallelに行なえる。

(7) Job 管理
Delta 自体ではTransaction を複数まとめたJob というような概念はサポートしていない。ユーザがJob というインタフェースを持ち（たとえばそのscope の中ではDBアクセスのauthorization （(2) ）は不要とか、論理チャネルは１度設定しておけばよいとか）これをサポートするならば必要

(8) エラー処理

エラー処理は、上述の機能各々に、いろいろなレベルで散在しているが、思いつくまま多少挙げてみると、

(a) Relation (Attribute) not found
(b) Data type mismatch
(c) illegal command-tree format
(d) illegal command sequence
(e) communication failed
(f) transaction timeout
(g) response timeout
(h) obsolete relation access tried
(e) その他

(9) PrologシンボルとBit ストリーム間の変換

通信バッファ内に入っているDELTA コマンドは、Bit パターンで格納されているのでPrologのシンボルとして使用するためには、変換が必要である。また、その逆の変換も必要である。

5．システム辞書

ユーザに、メタ的なデータ操作が容易にできるようにするために、次に示す５つのシステム辞書をテーブル形式で提供する。この辞書は、サイトの種類にかかわらず、同じ形式のシステム辞書としてみせる。Delta,KMM がもつ辞書の形式については、各々、“KSH 機能仕様書”，“関係データベース・マシン(Delta) の機能仕様書”を参照のこと。

(1) システムカタログ・テーブル

これは、オブジェクトを、任意のサイトからアクセスしても、高々、２回のサイトアクセスでみつけるられるように設計する。このテーブルは、次の形をしている。

kim_catalog(SWN, RSite, SSite).

RSite は、オブジェクトを、どのサイトから移動したかを、サイト名で記述する。誕生サイトでは、RSite=[]とする。

SSite は、オブジェクトを、どのサイトへ移動したかを、サイト名で記述する。主サイトに、オブジェクトがある場合は、SSite=[]とする。

なお、本テーブルは、オブジェクトのコピー・移動にともない、自動的にメンテナンスされるものとする。メンテナンスは、本リレーションに対する constraints (Trigger) として実施することができる。

(2) 同義語・テーブル

これは、 define _syninym コマンドで定義された SWNに対する同義語を管理するテーブルである。このテーブルは、次の形をしている。

kim_synonym(オブジェクトの名前, SWN)

オブジェクトには、現在、Frame,Relation(Facts),Ruleの３種類がある。

(3) フレーム・テーブル

これは、主サイトに存在する、すべてのフレームの名前を管理するテーブルであり、次の形をしている。

kim_frame(引き数の構成については、以下の項目を参照の事).

・第１引き数
　frame _name
・第２引き数
　creator _name
・第３引き数
　creator _site_name
・第４引き数
　birth _site_name
・第５引き数
　creator _date
・第６引き数
　access_permission
・第７引き数
　last_accessed_date
・第８引き数
　comments
・第９引き数
　semantics

(4) リレーション・テーブル

リレーションの枠組みが記述されており、次の形をしている。

kim_relation(引き数の構成については、以下の項目を参照の事).

・第１引き数
  relation_name
・第２引き数
  frame _name
・第３引き数
  number_of_fields
・第４引き数
  number_of_records
・第５引き数
  record_length
・第６引き数
  creator _name
・第７引き数
  creator _site_name
・第８引き数
  birth _site_name
・第９引き数
  creator _date
・第１０引き数
  access_permission
・第１１引き数
  reconstructed _date
・第１２引き数
  revised _date
・第１３引き数
  last_accessed_date
・第１４引き数
  comments
・第１５引き数
  semantics

(5) フィールド・テーブル

リレーションのフィールド構成を記述するテーブルであり、次の形をしている。

kim_field(引き数の構成については、以下の項目を参照の事).

・第１引き数
  relation_name
・第２引き数
  field _number
・第３引き数
  field _name
・第４引き数
  field _type
・第５引き数
  field _width
・第６引き数
  decimal _ point
・第７引き数
  comments
・第８引き数
  semantics

６．分散知識に対する更新

主サイトに存在する手続き（general clause）が、水平成分および、垂直成分とするリレーションで、定義されている時、その手続きをとおしてデータの更新を行う方法についてのべる。ここで、その手続きの定義に使われているリレーションは、従サイトのみならず主サイトにも存在するものとする。

主サイトのリレーションに対するデータの直接的な更新は、assert,retractの命令だけで達成できるが、上記、手続きを利用して、その定義に使われている分散知識を更新する方法としては、trigger の概念を導入することによって実現できる。これは、更新に関する不確定性を含まないように、ユーザが更新の詳細を記述できるので、妥当な方法である。このメタ知識（trigger ）の起動は、その手続きをとおして次の命令のいずれかが使用された時に実施される。

```
distributed_assert(Instanse-of-the-procedure).
distributed_retract(Instanse-of-the-procedure).
```

trigger* の syntax は、次のようにユーザが具体的な更新の方法を記述することになる。

```
trigger(Action,Procedure) :- member(Action,Action-List),!,
                             Revise-Condition, Revise-Specification.
```

しかしながら、もし、ユーザがこのような更新の方法に関して、十分な情報をもっていない場合は、システム自身の手によって、更新の仕方のいくつかの可能性を検出し、システムがユーザと対話をしながら、ユーザにその一つを選択させる方法も考えられる。その更新の仕方のいくつかの可能性は、上記の手続きを分析する事によっても抽出できる。

---

* この概念と類似性をもつメタ知識として、次のような形式のメタ知識がある。

```
is_inconsistent(Action,Procedure) :- member(Action,Action-List),!,
                                     Inconsistent_condition.
```

これは、外部から入力される知識に対する制約または、知識の削除，追加に対する制約である。

7．おわりに

本報告では、general-clause　の二次記憶への蓄積方法をはじめとして、排他制御，リカバリー，機密保護などの方式については、述べなかった。これらは、本格的な分散型知識ベースを構築する上で重要な問題であり、今後の課題である。

8．謝辞

本研究の機会を与えて下さったICOT研究所長・淵一博　氏および、有益な討論に参加していただいたICOT第一研究室・柴山茂樹，横田治夫，田口昭仁，宮崎収兄の各氏、ICOT第三研究室・服部隆　氏、三菱電気・溝口，三石の各氏（ICOT- KBIGのメンバー）、富士通研究所・手塚正義　氏に深謝いたします。

9．参考文献

[1] 知識獲得システム KAISER：電子計算機基礎技術開発成果報告書、
基礎ソフトウェアシステム編第１分冊、1983および1984.

[2] 田中譲　他：推論システムとデータベースシステムとの部分評価機構による結合、
第１回自動制御学会知識工学シンポジウム資料、1983.

[3] 横田治夫　他：Prologによる推論機構と関係データベースの結合、
情報処理学会研究会（知識工学と人口知能32-2），1983.

[4] SIMPOS CLASS仕様書（未公開）.

[5] 北上始　他: 関係データベースシステムRDB/v1の最適化技法.
情報処理学会論文誌　Vol.24 (1983).

[6] KMM 機能仕様書：電子計算機基礎技術開発成果報告書、
基礎ソフトウェアシステム編第１分冊、1984.

[7] KSM 機能仕様書：電子計算機基礎技術開発成果報告書、
基礎ソフトウェアシステム編第１分冊、1984.

[8] 関係データベース・マシン(Delta) 機能仕様書（未公開）.

[9] Bruce Lindsay: Object Naming and Catalog Management for a Distributed
Database Manager, Proc. 2nd International Conference on Distrubuted
Computing System, France (1981).

[10] R.williams et al.: R* An Overview of the Architecture,
Proc.International Conference on Database Systems,
Jerusalem, Israel (1982).

[11] 増永良文: 米国における最近の分散関係データベースシステム技術,
情報処理学会誌　Vol.25, No.5 May 1984.

付録－１

```
/*-------------------------------------------------+
|  DCG of SIM-RDB(KMM) Command Sequence.  1984.5.15  |
+-------------------------------------------------*/

sim_rdb_command(X,Y):-
      open_kmm_command(X,Z),
      kmm_command_group(Z,W),
      close_kmm_command(W,Y).

kmm_command_group(X,Y):-
      kmm_command(X,Y).
kmm_command_group(X,Y):-
      kmm_command(X,Z),
      kmm_command_group(Z,Y).

kmm_command(X,Y):-
      specify_kmm_command(X,Y).
kmm_command(X,Y):-
      data_def_command(X,Y).
kmm_command(X,Y):-
      list_command_tree(X,Y).
kmm_command(X,Y):-
      relation_command_tree(X,Y).
kmm_command(X,Y):-
      save_restore_command(X,Y).
kmm_command(X,Y):-
      print_display_command(X,Y).
kmm_command(X,Y):-
      error_message_command(X,Y).

list_command_tree(X,Y):-
      list_command(X,Y).
list_command_tree(X,Y):-
      list_command(X,Z),
      list_command_tree(Z,Y).

relation_command_tree(X,Y):-
      relation_command(X,Y).
relation_command_tree(X,Y):-
      relation_command(X,Z),
      relation_command_tree(Z,Y).

list_command(X,Y):-
      conversion_command(X,Y).
list_command(X,Y):-
      list_algebra_command(X,Y).

relation_command(X,Y):-
      relation_algebra_command(X,Y).
relation_command(X,Y):-
      picture_command(X,Y).
relation_command(X,Y):-
      aggregete_command(X,Y).
relation_command(X,Y):-
      boolean_command(X,Y).
relation_command(X,Y):-
      modification_command(X,Y).

/*---------------------------------+
|                                  |
|  (1) open/close_kmm command.     |
|  (2) specify_kmm command.        |
|  (3) save_restore command.       |
|  (4) print_display command.      |
```

```
|  (5) error_message command.      |
|  (6) data_def command.           |
|  (7) conversion command.         |
|  (8) list_algebra command.       |
|  (9) relation_algebra command.   |
| (10) picture command.            |
| (11) aggregate command.          |
| (12) boolean command.            |
| (13) modification command.       |
|                                  |
+----------------------------------*/

open_kmm_command([kmm_open|X],X).
close_kmm_command([kmm_close|X],X).

specify_kmm_command([kmm_specify|X],X).

save_restore_command([kmm_save|X],X).
save_restore_command([kmm_restore|X],X).

print_display_command([kmm_print_records|X],X).
print_display_command([kmm_display_records|X],X).

error_message_command([kmm_error_messages|X],X).
error_message_command([kmm_error_codes|X],X).

data_def_command([kmm_define_relation|X],X).
data_def_command([kmm_drop_relation|X],X).
data_def_command([kmm_define_index|X],X).
data_def_command([kmm_drop_index|X],X).
data_def_command([kmm_define_view|X],X).
data_def_command([kmm_drop_view|X],X).

conversion_command([kmm_relation_to_list|X],X).
conversion_command([kmm_list_to_relation|X],X).
conversion_command([kmm_index_to_list|X],X).
conversion_command([kmm_fetch|X],X).

list_algebra_command([kmm_reduction|X],X).
list_algebra_command([kmm_lfp|X],X).
list_algebra_command([kmm_cartesian|X],X).
list_algebra_command([kmm_natural_join|X],X).
list_algebra_command([kmm_outer_join|X],X).
list_algebra_command([kmm_difference|X],X).
list_algebra_command([kmm_append|X],X).
list_algebra_command([kmm_group|X],X).
list_algebra_command([kmm_sort|X],X).

relation_algebra_command([kmm_relation_reduction|X],X).
relation_algebra_command([kmm_relation_fetch|X],X).
relation_algebra_command([kmm_relation_join|X],X).
relation_algebra_command([kmm_relation_group|X],X).
relation_algebra_command([kmm_relation_sort|X],X).

picture_command([kmm_save_picture|X],X).
picture_command([kmm_restore_picture|X],X).
picture_command([kmm_get_picture|X],X).
picture_command([kmm_put_picture|X],X).
picture_command([kmm_free_picture|X],X).

aggregate_command([kmm_count|X],X).
aggregate_command([kmm_sum|X],X).
aggregate_command([kmm_average|X],X).
aggregate_command([kmm_min|X],X).
```

```
aggregate_command([kmm_max|X],X).

boolean_command([kmm_membership|X],X).
boolean_command([kmm_not_membership|X],X).
boolean_command([kmm_set_contain|X],X).
boolean_command([kmm_not_set_contain|X],X).
boolean_command([kmm_set_equal|X],X).
boolean_command([kmm_not_set_equal|X],X).

modification_command([kmm_clear_relation|X],X).
modification_command([kmm_delete|X],X).
modification_command([kmm_update|X],X).
modification_command([kmm_insert|X],X).
```

付録－２

```
/*-------------------------------------------------+
|  DCG of Delta Command Syntax.       1984.5.14   |
+-------------------------------------------------*/

delta_command(X,Y):-
     delta_header(X,Z),
     command_tree_group(Z,Y).

delta_header(X,Y):-
     zero(X,W),chain_id(W,Y).

command_tree_group(X,Y):-
     transaction(X,Y).
command_tree_group(X,Y):-
     transaction(X,Z),
     command_tree_group(Z,Y).

transaction(X,Y):-
     start_transaction(X,Z),
     transaction_body(Z,W),
     end_transaction(W,Y).
transaction(X,Y):-
     start_transaction(X,Z),
     transaction_body(Z,W),
     commit_transaction(W,U),
     end_transaction(U,Y).
transaction(X,Y):-
     start_transaction(X,Z),
     transaction_body(Z,W),
     commit_transaction(W,U),
     get_command_sequence(U,V),
     end_transaction(V,Y).
transaction(X,Y):-
     sense_status_command(X,Y).

transaction_body(X,X).
transaction_body(X,Y):-
     command_tree(X,Y).
transaction_body(X,Y):-
     command_tree(X,Z),transaction_body(Z,Y).

command_tree(X,Y):-
     algebra_command_tree(X,Y).
command_tree(X,Y):-
     data_definition_command(X,Y).
command_tree(X,Y):-
     get_put_command(X,Y).
command_tree(X,Y):-
     non_synchronized_command(X,Y).



start_transaction(X,Y):-
     ct_header(X,Z),stx(Z3,W),
     str_comm(W,U),
     etx(U,V),end_ct(V,Y).

commit_transaction(X,Y):-
     ct_header(X,Z),stx(Z,W),
     ctr_comm(W,U),
     etx(U,V),end_ct(V,Y).

end_transaction(X,Y):-
     ct_header(X,Z),stx(Z,W),
```

```
        atr_comm(W,U),
        etx(U,V),end_ct(V,Y).
end_transaction(X,Y):-
        ct_header(X,Z),stx(Z,W),
        etr_comm(W,U),
        etx(U,V),end_ct(V,Y).

algebra_command_tree(X,Y):-
        ct_header(X,Z),
        algebra_tree_body(Z,W),
        end_ct(W,Y).

algebra_tree_body(X,Y):-
        stx(X,Z),
        algebra_comm(Z,W),
        etx(W,Y).
algebra_tree_body(X,Y):-
        stx(X,Z),
        algebra_comm(Z,W),
        etx(W,U),
        algebra_tree_body(U,Y).

data_definition_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        data_def_comm(W,U),
        etx(U,V),end_ct(V,Y).

get_put_command(X,Y):-
        get_command(X,Y).
get_put_command(X,Y):-
        put_command(X,Y).

get_command_sequence(X,Y):-
        get_command(X,Y).
get_command_sequence(X,Y):-
        get_command(X,Z),
        get_command_sequence(Z,Y).

get_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        get_comm(W,U),
        etx(U,V),end_ct(V,Y).

put_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        put_comm(W,U),
        etx(U,V),end_ct(V,Y).

sense_status_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        sense_comm(W,U),
        etx(U,V),end_ct(V,Y).

non_synchronized_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        sense_comm(W,U),
        etx(U,V),end_ct(V,Y).
non_synchronized_command(X,Y):-
        ct_header(X,Z),stx(Z,W),
        abort_comm(W,U),
        etx(U,V),end_ct(V,Y).

ct_header(X,Y):-
        soh(X,Z),
```

```
      ctid(Z,W),cmdn(W,U),cl(U,Y).
end_ct(X,Y):-
      eot(X,Y).

/*-------------------------------------+
|                                      |
| (1) control information.             |
| (2) str/ctr/atr/etr command.         |
| (3) algebra command.                 |
| (4) data_def command.                |
| (5) get/put command.                 |
| (6) nonsyn command.                  |
|                                      |
+-------------------------------------*/

zero([0|X],X).                  % tag of packet
chain_id([id|X],X).             % chain ID ( ASCII 1,2,... ).

soh([soh|X],X).                 % Start of heading ( ASCII '01H' ).
ctid([ctid|X],X).               % command tree ID.
cmdn([cmdn|X],X).               % number of commands
cl([cl|X],X).                   % character length of command tree
stx([stx|X],X).                 % start of text ( ASCII '02H' ).
etx([etx|X],X).                 % End of text ( ASCII '03H' ).
eot([eot|X],X).                 % End of transmission ( ASCII '04H' ).

str_comm([str|X],X).            % start transaction
etr_comm([etr|X],X).            % end transaction
ctr_comm([ctr|X],X).            % commit transaction
atr_comm([atr|X],X).            % abort transaction

algebra_comm([pro|X],X).        % projection
algebra_comm([sel|X],X).        % selection
algebra_comm([joi|X],X).        % t-join
algebra_comm([njo|X],X).        % natural join
algebra_comm([sjo|X],X).        % semi join
algebra_comm([unn|X],X).        % union
algebra_comm([int|X],X).        % intersection
algebra_comm([dif|X],X).        % difference
algebra_comm([cpr|X],X).        % cartesian product
algebra_comm([cnt|X],X).        % count
algebra_comm([sum|X],X).        % summantion
algebra_comm([max|X],X).        % maximum
algebra_comm([min|X],X).        % minimum
algebra_comm([avg|X],X).        % average
algebra_comm([add|X],X).        % addition
algebra_comm([sub|X],X).        % subtraction
algebra_comm([mul|X],X).        % multiplication
algebra_comm([div|X],X).        % division
algebra_comm([equ|X],X).        % equal
algebra_comm([cta|X],X).        % contain
algebra_comm([del|X],X).        % delete
algebra_comm([ins|X],X).        % insert
algebra_comm([upd|X],X).        % update
algebra_comm([grp|X],X).        % group by
algebra_comm([slg|X],X).        % select group
algebra_comm([cop|X],X).        % copy
algebra_comm([srt|X],X).        % sort
algebra_comm([uqe|X],X).        % unique
algebra_comm([cls|X],X).        % classify

data_def_comm([cre|X],X).       % create relation
data_def_comm([pge|X],X).       % purge relation
data_def_comm([ren|X],X).       % rename relation
```

```
data_def_comm([app|X],X).      % append attribute
data_def_comm([drp|X],X).      % drop attribute

get_comm([get|X],X).           % get
get_comm([gtn|X],X).           % get next
get_comm([gte|X],X).           % get end
put_comm([put|X],X).           % put
put_comm([ptn|X],X).           % put next
put_comm([pte|X],X).           % put end

sense_comm([sns|X],X).         % sense status
abort_comm([abp|X],X).         % abort processing
```