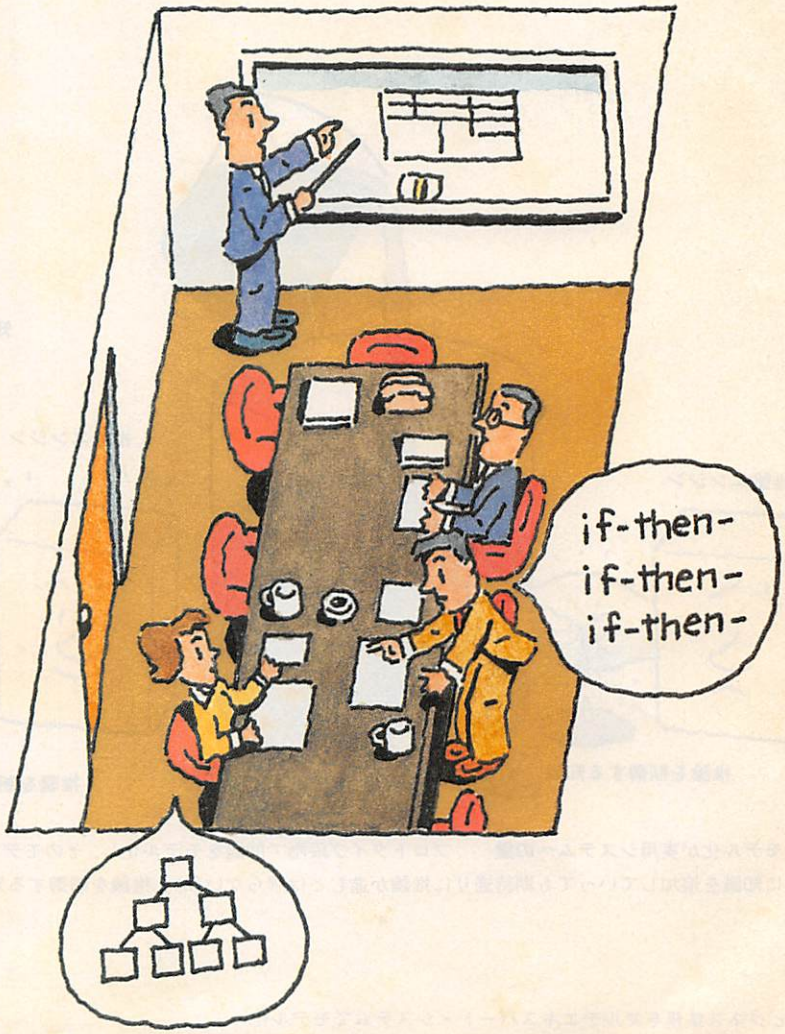
ビジネス業務をマルチエキスパート・システムでモデル化

マクロ/ミクロの2階層知識表現機能と黒板機能を採用

中所 武司
日立製作所　システム開発研究所

エキスパート・システム実用化の鍵を握るのは問題解決プロセスのモデル化である。エキスパート・システム構築ツールには，このモデルを素直に表現する機能が必要である。この要求にこたえるため，ビジネス分野向けのツールES/X90を開発した。さまざまな専門家の協調作業を，マルチエキスパート・システムとして実現する。業務全体の流れをオブジェクト指向の考え方と黒板機能を用いてマクロにモデル化していく。個々の業務は，ミクロなモデルとしてフレーム，ルール，述語論理で記述する。ツールの推論部はProlog，その他はC言語で記述した。2050上で動作する。（本誌による要約）



最近，ビジネス分野にもエキスパート・システムを適用しようとする動きがでてきた。故障診断などと違い，ビジネス分野には複数の専門家が協力して作業を進めるような複雑な業務が多い。それをエキスパート・システムで自動化する場合，解きたい問題をうまくモデル化できるかが成功への鍵となる。

エキスパート・システムの開発では，ふつう初めにプロトタイプ・システムを作る。そこに知識を追加していき，実用システムへもっていく。この方法は，初期の段階で大まかなシステムの動作を確認したり，本当に実現できそうかどうかを判断するのに，有効である。しかし，複雑な業務をエキスパート・システムにする場合，プロトタイプに知識を追加していけば実用システムになる，という考え方は必ずしも正しくない。

知識ベースの規模が小さければ，推論の流れは理解しやすい。新しい知識の追加も易しい。ところが，知識の量が増すに連れ，システム全体の動きが把握できなくなり，知識を追加しにくくなる。この段階で挫折してしまい，実用システムまで至らないことはよくある（図 1）。

実用システムへうまく拡張するには，その業務に関する事実とは別に，推論を制御する知識，つまり“知識を使うための知識”も入れておく必要がある。この知識を使うための知識とは，専門家がいかに問題を解決しているか，つまり問題解決プロセスのモデルである。

業務をマクロ/ミクロの２階層でモデル化

ビジネス分野のように，複数の専門家がかかわる大規模な業務をねらい，エキスパート・システム構築ツール ES/X 90 (Expert system building tool for 90's) を開発した[1]。その際には，実際の業務を分析し，その流れをモデル化したうえで，必要な機能を取り入れるという手順を採った。

ES/X90 の特徴は，マクロ・モデル/ミクロ・モデルによ

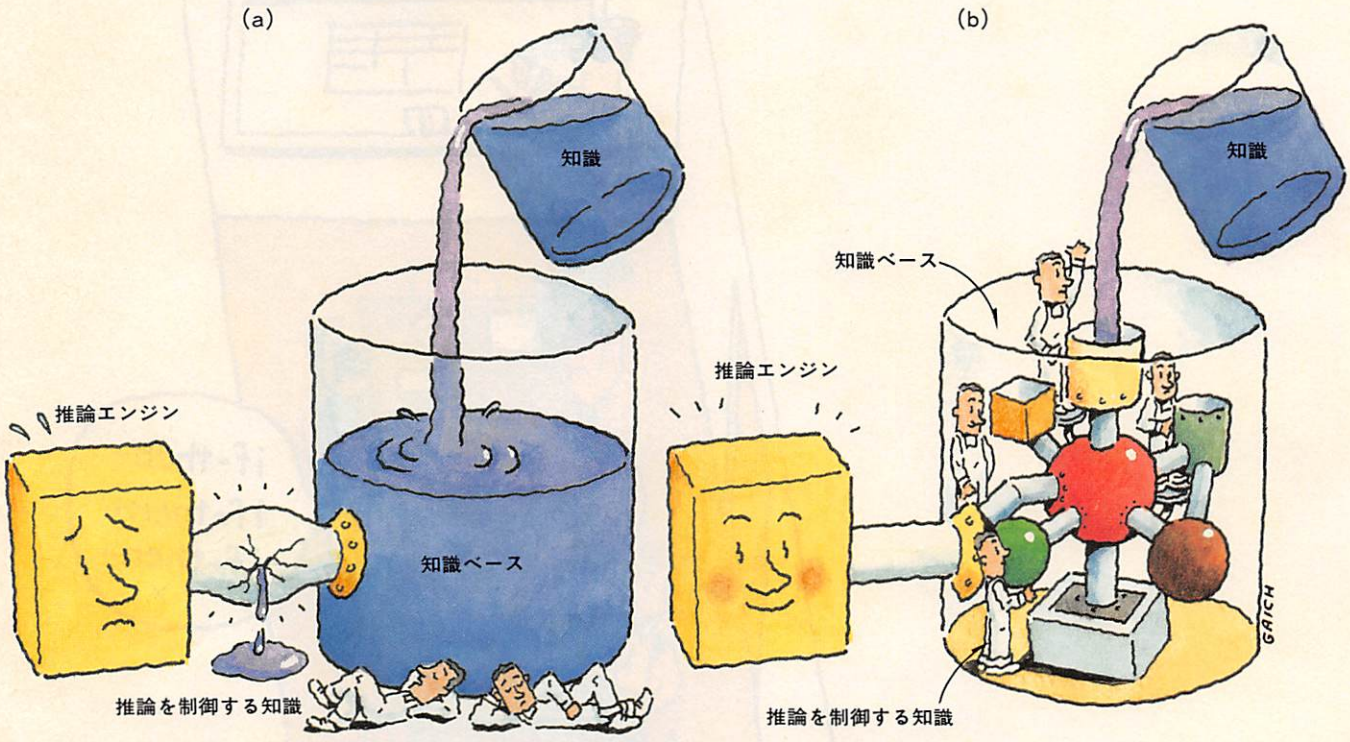


図 1　モデル化が実用システムへの鍵　プロトタイプ段階で問題をモデル化し，そのモデルを基に知識ベースを作ることが重要である。モデルを考えずに知識を追加していっても期待通りに推論が進むとは限らない(a)。推論を制御する知識（知識を使う知識）を加える必要がある(b)。

る知識表現[2]と黒板機能である。この二つの機能は，ビジネス業務にかかわるさまざまな専門家の作業を，複数のエキスパート・システムの協調動作として表すために使う。つまり，マルチエキスパート・システムを構成するわけである。詳細は後述するとして，各機能のねらいを簡単に示そう。

マクロ/ミクロのモデル化とは，実際の業務を二つの階層に分けて表すことである。マクロ・モデルでおおまかな仕事の流れを表し，ミクロ・モデルで個々の内容を表現する。実際の業務を素直に表すのがねらいである。

マクロ/ミクロ・モデルの表現にはオブジェクト指向の考え方を使った。全体の推論の流れは，マクロな知識である。これは，オブジェクト間のメッセージ送信機能で記述する。各業務を実行する専門家を一つのオブジェクトと考える。その人の知識は，ミクロな知識としてフレーム，ルール，述語論理などの知識表現機能で記述する（後述）。

黒板機能を取り入れたのは，複数の人間が参加する会議のような業務を表すため。黒板とは，共有の情報を掲示する場所である。複数のオブジェクト，つまり会議に参加している複数の専門家がそれを見ているようなイメージである。あるオブジェクト（専門家）が黒板に「キーワード」を書き込むと，それに関連したオブジェクトが推論を実行する。

そのほかに，ES/X90には，専門家自身が知識を入力しやすくするためのツールを用意した。データベースの情報を知識ベースに取り込む機能も持たせた（pp. 150-151の「ES/X90の概要：知識エディタ作成機能をもたせ，専門家でも知識ベースの保守を可能に」）。

実際の業務では複数の専門家が協調作業

ところで，ビジネス分野の業務（問題解決プロセス）とはどんな性質のものか。その例として，「不動産評価」をみよう（図2）。金融機関などが不動産の価値を評価する仕事

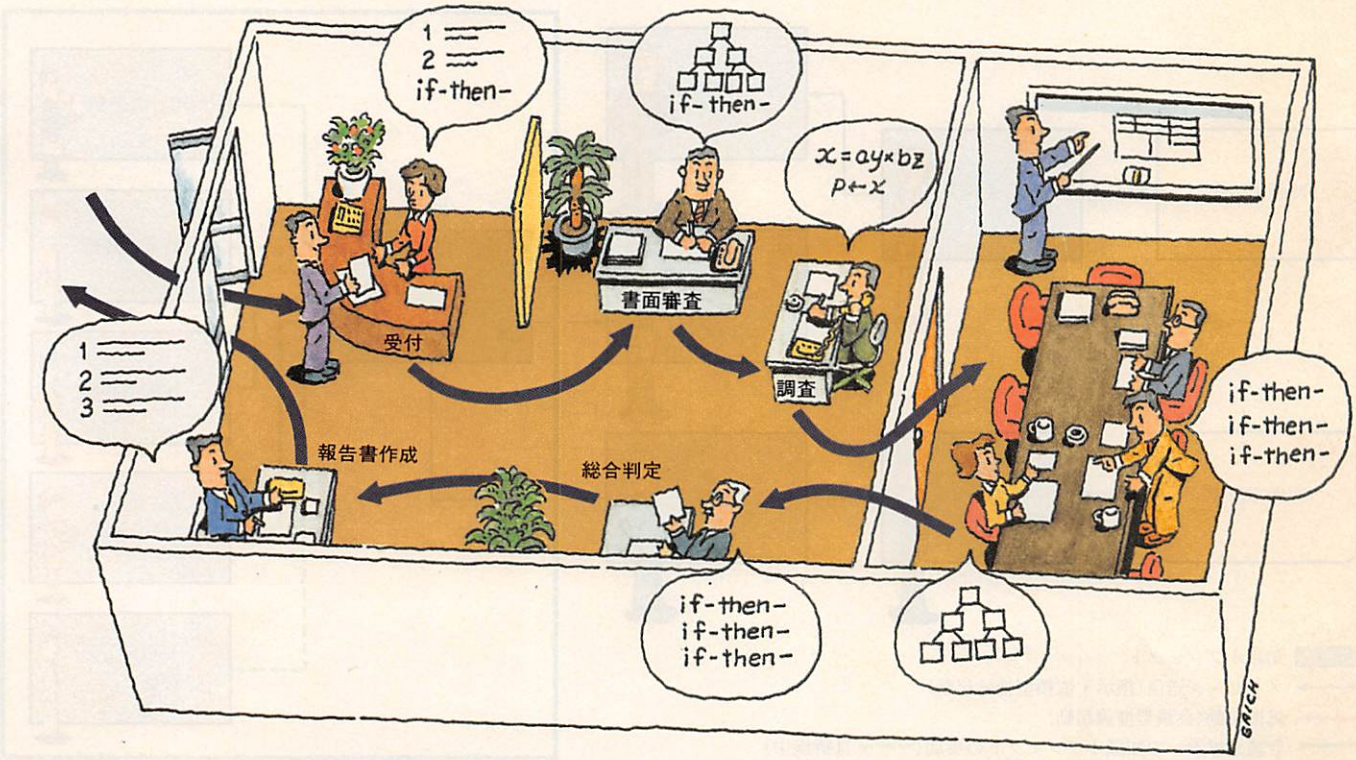


図2　ビジネス分野の業務の流れ　不動産評価業務を例に採った。複数の専門家が協調して業務を遂行する点が特徴である。協調のタイプは指示・依頼型と会議型の2種類がある。

である。さらに，それがどのようにモデル化できるかも示す。

不動産評価には，受付係，書類係，各種評価係，報告書作成係など多数の人間がかかわる。どの人も専門家といえる。この業務は，大きく見て，部単位あるいは担当者単位のサブ業務に分割できる。個々のサブ業務が互いに協力し合い，不動産評価という一つの大きな業務を遂行する形である。

こうした大規模な業務をエキスパート・システムとして作る場合，システム全体を一つのモデルで表すのは難しい。全体を，複数のサブ業務の協調作業と考えるとわかりやすくなる。この業務の分割がマクロ・モデルである。一方，個々の業務はミクロ・モデルである。

各サブ業務が協力し合うやり方には，二つのタイプがある。一つは，受付係→書類審査係→調査係のように業務の流れが一方向に進むタイプ。これを“指示・依頼型”と呼ぶ。このタイプでは，ふつう電話や書類の受け渡しによって業務を引き継ぐ。

もう一つは，複数の専門家が集まってそれぞれの意見を述べるタイプ。これを会議型と呼ぶ。不動産の評価では，都市計画予測係，住宅評価係などの専門家が物件に対して評価を加えるための会議を開く。

このように，実際の各業務はほぼ独立に進む。関連する業務間の引き継ぎ方法さえ決めておけば，各業務の担当者は他の業務の内容まで知る必要はない。一方，各々の業務はバリエーションに富む。分類作業が多い，計算作業が中心，というように仕事の性質も異なる。マクロ・モデルとして全体の業務をサブ業務に分割できたら，次にサブ業務単位でのモデリング（ミクロ・モデル）が必要となる。

マクロ・モデルはオブジェクト指向で表現

マクロ・モデルを記述するために，オブジェクト指向の考え方を利用した。不動産評価業務のマクロ・モデルをオブジェクト指向的に記述した例を**図３**に示す。サブ業務，

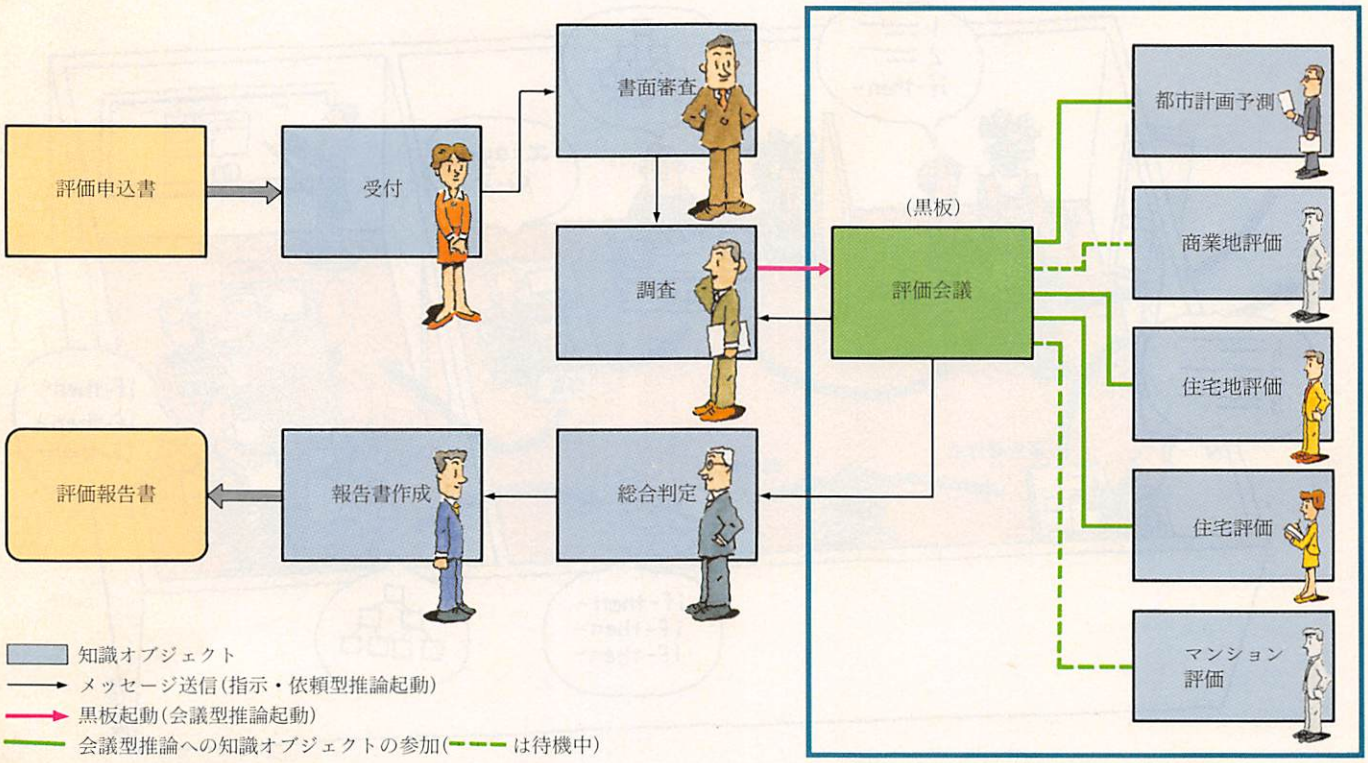


図３　不動産評価業務のマクロ・モデル　個々の業務をオブジェクトとして表す。指示・依頼型の業務はメッセージ送信で記述する。会議の記述には黒板機能を使う。

つまりそれを担当する専門家を一つのオブジェクトとみなすモデル化である。

指示・依頼型の業務は，オブジェクト間のメッセージ送信機能を使って記述できる。客から不動産評価の依頼を受けた「受付」オブジェクト（受付係）は受付業務が完了次第，「書面審査」オブジェクト（書面審査係）にメッセージを送信する，というような形で業務を引き継ぐ。各業務が独立していることもオブジェクト指向に向く理由である。「受付」オブジェクトを記述するとき，「書面審査」オブジェクトの記述形式を意識する必要はない。

ただし，このオブジェクト指向の考え方を使っても，会議型の業務は表せない。これを実現するために黒板機能を取り入れた。会議では，複数の専門家が共通の議題を議論する。黒板とは共通議題の提示場所である。会議を召集するオブジェクト（図3の例では調査係）が黒板にキーワードを書き込むと，そのキーワードと関係のあるオブジェクトが自動的に呼び出され，個別に推論を実行する。

都市計画予測係や住宅地評価係などのオブジェクトには起動条件となるキーワードのリストをもたせておく。「調査」オブジェクトが，黒板に“住宅”というキーワードを書き込む。すると，“住宅”を起動条件とする「住宅評価」オブジェクト，「住宅地評価」オブジェクト，「都市計画予測」オブジェクトの三つを呼び出し不動産評価を開始する。

オブジェクト

```
class 会員登録 ;                          フレーム
                                          (継承関係定義＋スロット部)
  inherit 業務 (without(実施日));   ②

  slots  会員数 (initial(0), on_read(会員数チェック)),
         定員 (default(100));

                                          プロダクション・ルール
                                          によるメソッドの記述
  rule_methods ;
    rules 会員数チェック forward(once);
      if #会員数 = 0 then write('会員未登録です');
      if #会員数 >= #定員 then write('満員です');

                                          述語論理による
                                          メソッドの記述
  predicate_methods ;
    入会(Name) :- send(会員, create(Name)),
                  X is #会員数 + 1,
                  write_value(#会員数, X);
    退会(Name) :- read_value(会員名簿#リスト,X),
                  X == Name,
                  send(Name,kill),
                  X is #会員数 - 1,
                  write_value(#会員数, X) ;
    退会(Name) :- write('この方は会員ではありません');
end.
```

①

図4　オブジェクトの記述　「会員登録」オブジェクトの記述例である。手続き的知識は，述語論理，プロダクション・ルール（①）で表している。上位オブジェクトとの継承関係は“inherit”で表す（②）。

ミクロ・モデルはフレーム，ルール，述語論理で記述

ミクロ・モデルは，一つのオブジェクト，つまりサブ業務に相当する。この部分は，いわばそれぞれが問題領域を限定した一つのエキスパート・システムといえる。ES/X90では，知識を表す形式として，フレーム，プロダクション・ルール，述語論理の3種類を用意した。適切な知識表現を選べるようにするためである。

ただし，そのこと自体は特に珍しくない。最近のツールでは，複数の知識表現形式をもたせるのは当たり前になっている。ミクロ・モデルのレベルになると，むしろ，機能をどう使いこなすかが問題である。専門家の頭の中にあるいろいろな知識を，宣言的知識はオブジェクトで，手続き的知識はプロダクション・ルールや述語論理で，というところが難しい。以下ではES/X90の用意した機能の使い方を説明しよう。

まずオブジェクトの記述形式を会員登録業務を例に採って示す(**図4**)。他の構築ツールでフレームと呼んでいる表現形式とほぼ同じである。図4では，手続き的な知識をメソッドとして，述語論理とプロダクション・ルールで表している（①の部分）。

述語論理機能を加えたのは，試行錯誤を必要とする探索は，ユニフィケーション機能とバックトラック機能を使った方が書きやすいため。他の述語を呼び出す範囲は自分のメソッド内に限定した。図4の“退会”メソッドは，会員名簿の会員リストから名前を読み出し，指定した名前と一

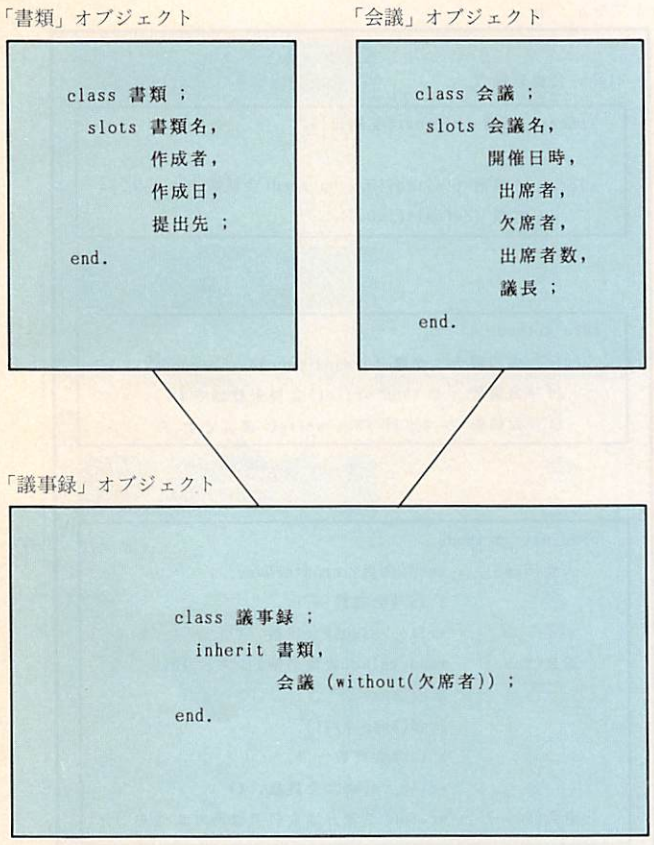


図５　多重継承の例　「議事録」オブジェクトは，「書類」と「会議」オブジェクトの性質，すなわち「書類」オブジェクトの全スロット，「会議」オブジェクトの“欠席者”以外のスロットを継承する。

```
class 会議 ;
  inherit 業務 ;
  slots 会議名,
        開催日時,
        出席者 (initial(池田、佐藤、田中、福田),
                on_delete(出席者数変更)),
        欠席者 on_add(出席者リスト削除),
        出席者数 on_read(出席者リストカウント),
        議長 default(業務#責任者) ;
end.
```

図６　デーモン機能の使用例　　部分がデーモン。on-delete, on-add, on-read は手続きの実行条件で，それぞれスロット値の‘削除’，‘追加’，‘読み出し’時に起動されることを示す。カッコ内は実行するメソッドの名前。

致すれば退会手続きを実行する。この手続きを一致する名前が現れるまで続ける。

オブジェクトの階層構造は，“inherit”という定義で記述する。“inherit 業務”で，この「会議登録」オブジェクトが「業務」というオブジェクトの性質を受け継ぐことを表している(②)。ただし，“without(実施日)”のように，“実施日”というスロットを継承しないことも指定できる(②)。

多重継承やデーモンで実際の知識を素直に表現

ES/X90 にはオブジェクトの多重継承機能をもたせた。この機能は他のツールにもある。一つのオブジェクトは多数の上位オブジェクトから性質を受け継ぐことができる。たとえば，「議事録」というオブジェクトを考えてみよう。このオブジェクトは，「会議」と「書類」という二つの性質を備えたもの，と考えるのが自然である。多重継承機能があれば，その関係を素直に表せる。**図５**に示したように，「書類」オブジェクトから“書類名”“提出先”などの書類に関するスロットを，「会議」オブジェクトから“欠席者”以外のスロット，すなわち“会議名”“出席者”などの会議に関するスロットを継承するように定義すればいい。こうやってフレームの多重継承機能を使うと個々のフレームに記述する項目が少なくてすむ。

```
class 住宅評価 ;
  rule_methods ;
    rules 中古住宅評価 on(住宅) ;
      if 報告書#建築年数 >= 15        then +(中古の度合, 0.8) ;
      if 15 > 報告書#建築年数 >= 5    then +(中古の度合, 0.5) ;
      if 報告書#工法 = 木造           then +(中古の度合, 0.2) ;
      if 報告書#集中暖房設備 = 無     then +(中古の度合, 0.1) ;
      if 報告書#屋根付き駐車場 = 無   then +(中古の度合, 0.1) ;
end.
```

図７　大局的ルールの記述例　「会議招集」オブジェクトが黒板に「住宅」というキーワードを書き込むと推論を開始する。条件を満たすすべてのルールを実行する。

表 1　局所的ルールの実行戦略

once	条件部が満たされた一つのルールのみを実行
once-all	条件部が満たされたすべてのルールを 1 回実行
multi	ルールを記述順に評価し，ルール実行後，先頭ルールに戻りすべて実行できなくなるまで評価を続行
multi-all	ルールを記述順に評価し，ルール実行後，そのルールの直後から評価を続行，先頭のルールに戻る

表 2　大局的ルールの競合解消戦略

種類	内容
最新性優先	ルール条件部で参照されているデータの値の設定された時点が新しいルールを優先的に実行
詳細優先	ルール条件部に記述されている条件が多いものを優先的に実行
確実性優先	ルールの確信度の大きいものを優先的に実行
記述順優先	ルールの記述順の早いものを優先的に実行
複合戦略	上記戦略の組み合わせ

さらに，スロットをアクセスするときに自動的にメソッドを起動させるデーモン機能を 4 種類（on-read, on-write, on-add, on-delete）用意した。デーモンとはソフトウエア割り込みの一種で，スロット値の読み出し，書き込み，追加，削除を実行しようとしたとき，その直前に必要な手続きを起動する機能である。

会議というオブジェクトを考えてみる（**図 6**）。出席予定者の一人が欠席届けを提出した場合，「会議」オブジェクトの“欠席者”スロットに欠席者の名前を書き込む。このとき，“欠席者”スロットの on-add デーモンが起動され，“出席者リスト削除”の手続きを実行し，“出席者”スロットのリストから欠席の届けがあった人の名前を削除する。その結果，さらに“出席者”スロットの on-delete デーモンが起動され，“出席者数変更”の手続きを呼び出すことになる。

汎用的な知識，業務固有の知識を別々のルール形式で

プロダクション・ルールは，手続き的な知識を表すのに使う。ルールの形式として 2 種類を用意した。ルールの参照範囲がそのオブジェクト内に限られる局所的ルールと，他のオブジェクトのスロット（ただし他のオブジェクトから参照してもいいと指定されている場合）も参照できる大局的ルールである。

局所的ルールは業務固有の処理手順を記述するのに使う。この型のルールでは，実行順序を陽に指定できたほうがいい場合も多い。簡易プログラミング言語的な使い方をするためである。実行戦略は前向き推論の場合 4 種類を用意した（**表 1**）。

手続き型言語の case 文のように条件を満たすルールを一つだけ実行したいときは once，if 文のような条件判断文が並び，条件が満たされたすべてのルールを実行するときは once-all を使う。ループのように繰り返し実行するときは multi を使う。multi と multi-all は，ルールの実行後に，先頭のルールから評価をし直すか，その直後のルールから評価を続けるか，の違いである。このほか，後ろ向き推論も可能である。

大局的ルールは，会議型モデルを実現するために用意した（**図 7**）。大局的ルールの起動条件はキーワードの形で記述する。この例では“住宅”である。会議を召集する役目を務めるオブジェクトが黒板にこのキーワードを書き込むと，図 7 に示したメソッドをもつオブジェクトが呼び出される。“住宅”で起動されるメソッドをもつオブジェクトはすべて呼び出される。

ルールの実行順序を決める競合解消戦略は 5 種類用意した（**表 2**）。この戦略は，会議を召集するオブジェクトが黒板に書き込む（なにも指定しないときは，最新性優先戦略を採用）。ただし，知識の量が増加すると，推論制御のために適切な戦略を選択することが難しくなる。マクロ・モデルを作る段階で性質の異なる業務を別のオブジェクトとして記述しておく必要がある。

参考文献

1) 中所ほか，「エキスパートシステム構築ツール ES/X90(1)～(9)」，『情報処理学会第 35 回（昭和 62 年度）全国大会講演論文集』，pp. 1733-1750，1987 年 9 月.

2) Chusho, T. and Haga, H., “A Multilingual Modular Programming System for Describing Knowledge Information Processing Systems,” *the 10th World Computer Congress IFIP'86*, pp. 903-908, 1986.

次ページに関連記事あり。

ES/X90の概要: 知識エディタ作成機能をもたせ，専門家でも知識ベースの保守を可能に

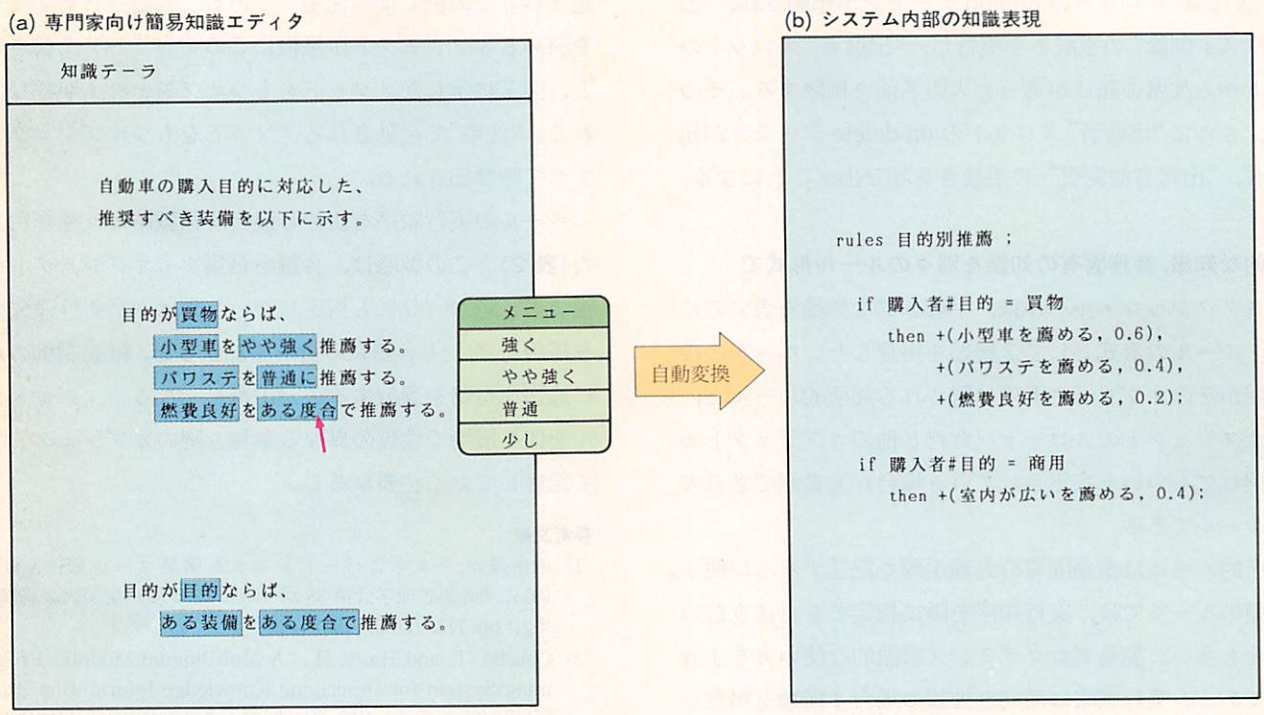
ES/X90は，ビジネス分野の業務をモデル化するのに必要な知識表現形式や推論機構のほかに，エキスパート・システムの開発過程を支援する開発環境を強化した点も大きな特徴である。専門家自身が知識を入力できる環境を作るための簡易知識エディタ作成ツールや，既存のデータベースのデータを知識ベースに取り込む機能，知識ベースの部分推論ができる検証機能などを用意した。なお，ES/X90は推論部分をProlog，それ以外をC言語で記述した。2050ワークステーション上で動作する。

専門家向けの簡易知識エディタを作成する

プロトタイプを作成するまでは，ナレッジ・エンジニア（KE）が知識を記述することが多い。実用システムに仕上げるために詳細な知識を入れたり，知識を保守し続けるためには，専門家自身が知識を管理できるほうが望ましい。専門家向けの簡易知識エディタ作成用ツールとして知識テーラを開発した。

簡易知識エディタからES/X90の知識ベースへの変換手順はシステム開発者（KE）が定義する。一度簡易知識エディタを作成すれば，その後の保守は専門家に任せることができる。

図Aは知識テーラを用いて作成した自動車購入相談エキスパート・システム用の簡易知識エディタである。自動車販売担当者は，枠内を埋めるだけで販売ノウハウの知識を入力すれば，自動的に知識ベースのプ



図A　知識エディタの入力を内部の知識表現形式に変換

ロダクション・ルールに変換できる。

また，KE向けには構文エディタを用意した。フレーム，ルール，述語論理などの知識表現形式を指定すれば，文法に則って，知識の記述を誘導する。文法誤りがなくなるうえ，キータッチの数が少なくなる。

データベースのデータを知識ベースに取り込む

ビジネス分野のエキスパート・システムは，既存のオフィス用ソフトウエアと結合して使用することが多くなる。C, Fortranなどの従来型言語で記述したプログラムをサブルーチンとして呼び出せる。また，リレーショナル・データベースのデータを，事前にあるいは推論中に知識ベースに取り込む機能を付けた。知識ベースを効率よく作成するのに有効である。

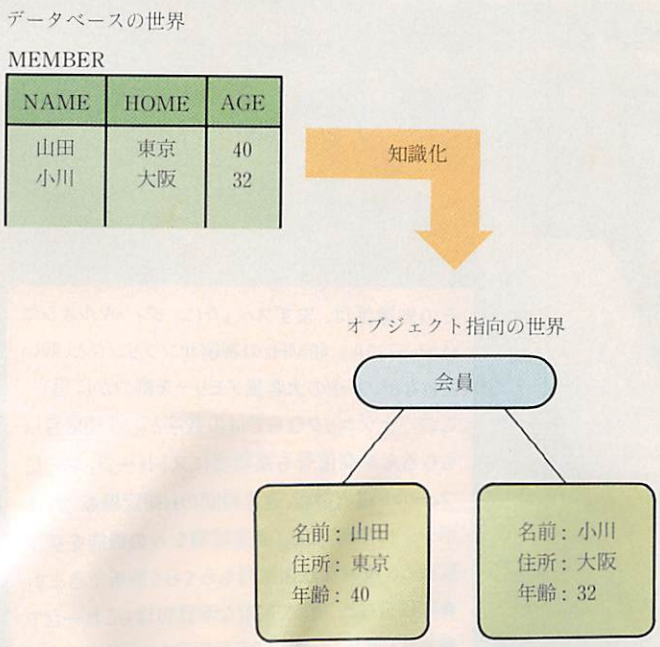
図Bに，リレーショナル・データベースの会員情報を「会員」オブジェクトのインスタンス・オブジェクトとして知識ベースに取り込む例を示した。

部分的な推論で知識ベースを検証する

知識ベースを拡張する段階で，入れた知識の正当性を検証する作業は欠かせない。この試行錯誤的作業を効率化するために，知識ベースを部分的に推論する機能を加えた。知識ベースの不良を発見するたびに最初から推論をやり直すのは効率が悪い。間違った知識があると思われる部分だけ推論を実行させれば，検証にかかる時間を短縮できる。

そのほか，推論の流れに従ってメッセージ送信やルールの発火，スロット値の変更などを表示するトレース機能，推論途中の各オブジェクトの状態を知るために推論を一時停止させる中断点設定機能，などがある。

(a) 概念図



(b) 記述例

```
class  会員;
  inherit   システム;
  is_TBL(rdb,'EMPLOYEE');
  class_part{
    predicate_methods;
      オブジェクト化:-send(self,load_DB([],rw));
      データベースへ格納:-send(self,save_DB);};
  instance_part{
    slots 名前 is_FLD('NAME'),
          住所 is_FLD('HOME'),
          年齢 is_FLD('AGE') ; };
end.
```

図B　データベースのデータを知識ベースに取り込む　(a)リレーショナル・データベースのテーブルと各タプルを知識ベースのインスタンス・オブジェクトに変換。(b)取り込むための手続き。