第10章

マルチパラダイム

★ *対象世界が複数の異なる種類の知識を用いて問題解決しているとき，対応するプログラムはマルチパラダイム型言語で記述されていると，わかりやすい。*

10.1 マルチパラダイムのメタファ

10.1.1 対象世界のモデル化

プログラミングパラダイムの研究は，従来，単一のパラダイムとその実現言語に関するものが多かった。その方が，意味仕様を形式的に厳密に定義することが容易であり，記述されたものの美しさや処理系の実現容易さなどの利点がある。しかしながら，対象世界は多様であり，適用分野によっては実用性に疑問が残る場合がある。

たとえば，実世界における次のような問題を考えてみるとよい。

- ある地点Aから目的地Bまでの経路と手段を決定するとき，いくつかの代替案を考え，各々の所要時間，費用，快適性などを考慮する。この時，手続き的知識，ノウハウ（ルール）的知識，算術計算式など，いくつかの異なる種類の知識を組み合わせて用いている。
- 忘年会の幹事が日時，場所，会費を決定するとき，その決定要因は多種

多様であり，配慮すべきことは多岐にわたる。この時，「名幹事」といわれるためには，その決定にいたるプロセスであらゆる種類の知識を駆使しなければならない。

このような対象世界の問題を素直にモデル化するためには，複数のパラダイムを組み合わせたマルチパラダイムの計算モデルが必要である。

10.1.2 キーコンセプトの進展

1つの試みとして，プログラミングの分野におけるマルチパラダイムに関連した「マルチ」の概念の変遷を大ざっぱに以下のように捉えることができる。

マルチリンガル→マルチパラダイム→マルチエージェント

(1970年代) (1980年代) (1990年代)

機能モデル 知識モデル 分散協調モデル

(1) マルチリンガル

1970年代は，ソフトウェアの大規模化に対応して，構造化設計，構造化プログラミング技法が開発された。そこでの核技術の1つは，「分割統治の原則」にしたがったモジュール化技法であり，「1つの機能を1つのモジュールに対応させる」ための種々のモジュール分割技法が提案された。

このような方式における主要な技術課題の1つは，各モジュールをそのモジュールの機能に適した言語で記述できるようにすることである。すなわち，システムの機能に注目し，その機能分割を中心とした開発技法では，機能モデルを実現するマルチリンガル（多言語）プログラミング環境が必要である。これは，見方をかえれば，任意のプログラミング言語で記述されたプログラムを統合して1つのアプリケーションソフトウェアを作り上げるような今日的意味でのリエンジニアリング技法ともいえる。

(2) マルチパラダイム

社会組織の高度化に伴い，製造，金融，流通，教育，医療，交通などの様々な分野で，種々の専門家が重要な役割を担っているが，このような専門家の育成は容易ではなく，優れた専門家の数は限られている。そこで，1980年代に

は，コンピュータをこれまでのような定形業務ばかりでなく，専門知識と問題解決ノウハウを必要とする非定形業務へ応用するエキスパートシステムの開発が活発に試みられ，今日では，プログラミング技法の1つとして定着するに至っている。

しかしながら，知識の抽出が十分には行えない複雑な問題や複数の専門家が協力して解決しなければならない問題については，エキスパートシステムの構築に関し，「プロトタイプ版の知識ベース内の知識を充実させていけば実用版になる」と言われているほど簡単ではない。このようなプロトタイプから実用への壁を乗り越えるためには，次の2点が重要である。

- ●専門家の推論過程が素直にモデル化できる。
- ●専門家が推論過程で用いる知識がそのモデルの中で素直に表現できる。

そこで，専門家の知識を素直に記述して知識ベースへ蓄積していくためには，その多様性に応じて複数の推論方式や複数の知識表現形式を可能とするマルチパラダイムの知識モデルが必要になる。

(3)　マルチエージェント

1990年代には，ワークステーションやパソコンをネットワーク接続した分散コンピューティング環境が一般化してきた。それに伴って，より高度なソフトウェアが要求され始めた。たとえば，1つのアプリケーションソフトウェアが，分散環境下の任意のリソースを自分のものとして用いることや，複数のアプリケーションソフトウェアが分散環境下で協調して問題を解くことなどである。すでに，CSCWやグループウェアと呼ばれるシステムが開発され始めている。

このようなソフトウェアの開発を，従来のような全体を集中制御する方式で構築するのは無理がある。このような分散環境下での，より知的な機能要求に対応するためには，分散協調モデルとしてのマルチエージェントモデルが必須となろう。

本章の以下では，マルチリンガル方式について簡単に述べた後，マルチパラダイムについて詳しく述べる。マルチエージェントについてはまだ研究段階のため第3編で言及するにとどめる。マルチパラダイム型言語の設計に興味のない読者は本節以下を飛ばして10.7節の記述例へ進むことをお勧めする。

10.2 マルチリンガルの実現方式

「1機能1モジュール」の機能分割技法を支援するモジュラープログラミング環境を実現するためには，次のような記述言語が必要である。

①モジュール間関係（インタフェース）の正確な表現
②各モジュールのそれぞれの機能に適した表現

まず，第1のモジュール間インタフェースの記述方式として，次の2つのアプローチがある。

(a) プログラミング言語の1機能として実現する。
(b) プログラミング言語とは独立した専用のモジュールインタフェース記述言語を導入する。

前者のアプローチは，モジュール間でのリソースの使用関係とモジュール自身の記述機能を1つの言語に含むため，言語仕様が複雑になる一方で，大規模ソフトウェアの開発には，両機能とも不十分になる場合が多い。後者のモジュールインタフェース記述言語の例として，過去にMIL (Module Interconnection Language)言語などが提案されてきたが，記述内容がリソース使用の規制に限定されており，モジュール自身の記述言語との連携も不十分だった。

次に，上記②の各モジュールのそれぞれの機能に適した表現を可能にするアプローチとして，以下の2種類がある。

(a) プログラミング言語の仕様を多機能化していく。
(b) 複数のプログラミング言語のモジュールをリンケージエディタで結合する。

前者のアプローチに基づく言語として，1960年代のPL/I，1970年代のFORTRAN77，1980年代のAdaがあるが，このように必要な機能を積極的に取り入れて万能言語にしていくアプローチは，当然ながら言語の複雑化による言語修得の困難さ，コンパイラの複雑化と巨大化，効率的なオブジェクトコード生成の困難さ，などの欠点を生じる。このような言語仕様の巨大化アプローチは「多機能，即，複雑」という弊害が避けられない。一方，後者のアプロー

チは，異なる言語間でのデータ型やパラメータメカニズムの不一致に関する対応が必要になるほか，リンケージオーバーヘッドによる実行効率低下の問題がある。

筆者は，これら双方のアプローチの欠点を克服すべく，次のような設計方針に基づく第3のアプローチを提案し，多言語モジュラープログラミング(MMP：Multilingual modular programming) と名付けた。

①モジュール間関係（インタフェース）の正確な表現のためにその専用言語を導入するが，コンパイラおよびリンケージエディタを含めた処理系の統合化により，プログラミング言語との連携処理を可能にする。
②各モジュールのそれぞれの機能に適した表現を実現するために機能別に複数の簡易言語を用意し，リンカーで統合した後にオブジェクトコードを生成する方式により，リンケージオーバーヘッドを回避する。

この多言語モジュラープログラミング用の処理系は，従来の**図10.1**(a)のような構成ではなく，同図(b)のようなコンパイラのコード生成系とリンケージエディタを統合したものにした。この処理系を LIGER* (Link-and-generator) と名付けた。主な機能は，モジュール単位の最適化，インライン展開などのモジュール間の最適化，異種データ型や異種パラメータメカニズムの処理コード付加，などである。

本方式を大規模システムに適用した場合の主な利点は，以下のようなものである。

①モジュール間インタフェースの検証による信頼性向上。
②各モジュールの機能に適した言語での記述による理解容易性向上。
③異種言語間リンケージの容易化。
④モジュール間の最適化による実行効率向上。

しかしながら，この方式の実用化のためには，次の2項目の技術課題の克服が必須である。

＊　liger は，lion（♂）と tiger（♀）の交配種

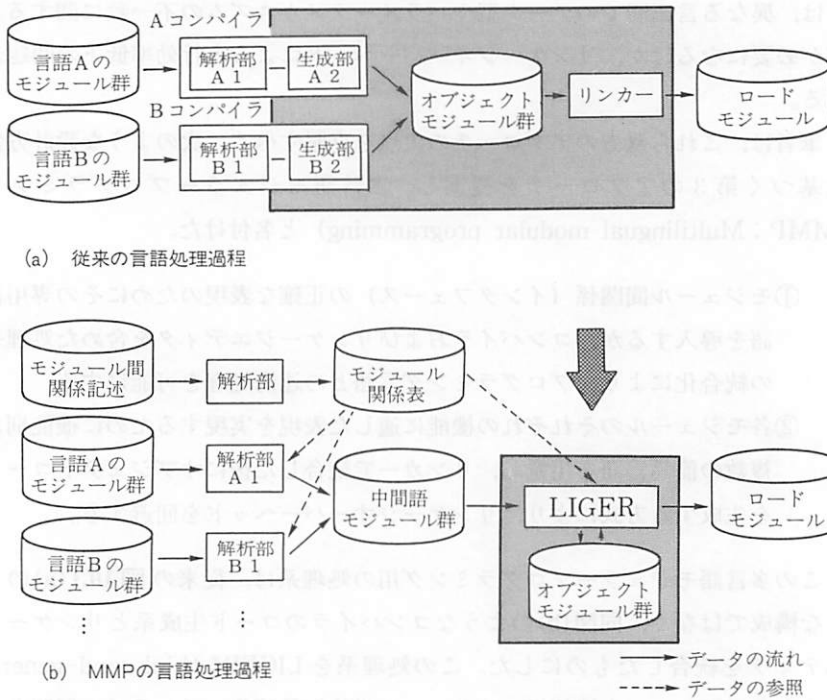


図10.1　多言語モジュラープログラミングMMPの処理系LIGER

●異種言語間共通のモジュールインタフェース記述言語の設定
●異種言語間共通の中間語の設定

モジュール間インタフェース記述言語に関しては，その後，仕様記述言語およびプログラム自動生成の分野で研究が続けられている。中間語の標準化についても，1950年代のUNCOL (UNiversal Computer Oriented Language) 以来，コンパイラの研究者の夢として，今日まで研究が続けられている。当初は，コンパイラのコード生成系を異種言語間で共通にするのが主目的だったが，その後，高度の技術と開発費用が要求されるコード最適化処理の汎用化，異種ハードウェア間でのソフトウェアの移行性，可搬性の確保，等へと必要性が拡大している。特に，最近のようにRISC型マイクロプロセッサの高性能化が急速に進み，そのグレードアップに対応してコストパフォーマンスの向上したワークステーションが次々と出回るようになると，アプリケーションソフトウェアの

異機種間での移行性はユーザにとって最重要課題になっており，中間語の統一がその鍵となる。

たとえば，UNIXの標準化団体のOSF（Open Software Foundation）では，ANDF（architecture neutral distribution format）という標準中間語を定めている。この目的は，どのようなアーキテクチャのハードウェアに対しても効率的にインストールできるようなソフトウェアの頒布形式を実現することである。従来はソースコードまたはオブジェクトコードによる頒布が一般的であったが，ソースコードによる方法は知的所有権の保護に関する問題がある。一方，オブジェクトコードによる方法では，ハードウェアアーキテクチャの種類に応じたものを作成する必要があり，ポーティング費用がかさむという問題がある。ANDFを用いた方法では，まず，ソースコードをANDF形式に変換し，そのレベルで最適化を行い，リンカーで結合したものを頒布する。ユーザは，そのアプリケーションパッケージを稼動したいハードウェア用のインストーラを用いてインストールする。

10.3　マルチパラダイムの実現方式

10.3.1　従来の知識モデルの問題点

エキスパートシステムをはじめとする知識処理システムの目的は，専門家の有する知識を用いて推論を行うことにより，専門家の問題解決能力を模擬することである。第9章で述べたように，従来のエキスパートシステム構築ツールは，特定の知識表現形式と特定の推論機構を備えることにより，その表現形式に合わせて記述した知識を入力するだけでエキスパートシステムが開発できることを特徴としていた。そのため，

- ●知識を専門家自身が入力，保守できる。
- ●知識を少しずつ充実させながら段階的に開発できる。

という利点があり，専門家自身がシステム開発に関与できるようになった。

その代表的なものは，if～then～形式で表現されたルール型知識を用いて前向き推論を行うプロダクションシステムである。この方式は，ルール数が100以下の比較的小規模なシステムを簡単に作る場合には適しているが，多くの知識

を必要とする複雑なシステムを開発する場合には，次のような問題がある。

●専門家からの知識の抽出（知識獲得）が妨げられる。
●専門家による検証が困難になる。

知識獲得は，エキスパートシステム構築における最大の課題と言われているが，その原因は，現在のエキスパートシステム構築ツールの未熟さのために，専門家に次のような努力が強いられるためである。

●自分の推論過程をツール固有の推論モデルに変換しなければいけない。
●自分の専門知識をツール固有の知識表現に変換しなければいけない。

専門家からの知識の抽出を容易にするためには，何よりもまず専門家の実際の推論過程（問題解決プロセス）に合った推論モデルの作成が重要である。

この推論過程のモデル化の重要性は，もう1つの本質的課題である検証についても言える。たとえば，実用化に際して，エキスパートシステムの機能が十分か否かを検証することを考えてみよう。いくつかのテストケースに対する推論結果が専門家並みであったとしても，そのシステムの機能が専門家並みである保証はない。また，あるテストケースで不十分な推論結果が生じた時，その原因が知識自身の不良のためか，あるいは知識の利用の仕方（推論制御方法）が悪いのかを見分けることは容易ではない。これらの検証技術が未熟な現段階では，エキスパートシステムの検証のための安全で確実な方法は，その開発者が推論過程を把握していることである。

このような課題を克服するためには，対象とする問題領域の知識のみならず，その知識の利用の仕方を規定する推論制御に関する知識（メタ知識）も蓄積できるように専門家の推論過程をマルチパラダイムでモデル化することが重要である。

10.3.2 2階層モデルに基づく融合方式

このようなマルチパラダイムのモデルに基づく言語の設計では，「多機能，即，複雑」とならないように構文仕様と意味仕様を決めることが重要な課題である。そのためには，全体的な知識構造や処理制御の記述にオブジェクト指向モデルを用い，個々の知識や処理アルゴリズムはそれに適したパラダイムで記

述するという2階層モデルが適している。2階層モデルとは，以下のようなオブジェクト指向概念をベースにした計算モデルである。

①全体の知識処理と知識構造はオブジェクト指向型言語で表現することにより，マクロなレベルでの柔軟で自然なモデリングを可能とする。

②局所的な知識はフレーム型，ルール型，述語論理型などの複数の形式のいずれでも表現できるようにすることにより，ミクロなレベルでの多様な知識の自然な表現を可能とする。

この概念図を**図10.2**に示す。

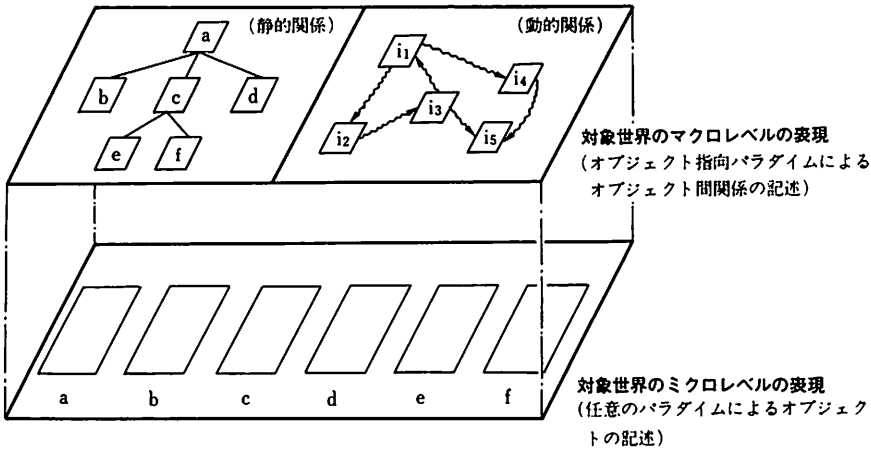


図10.2　マルチパラダイム型計算モデル(2階層モデル)の概念図

本章の以下では，筆者等が2階層モデルに基づいて開発したエキスパートシステム構築ツールES/X90 (Expert system building tool for 90's) に採用したマルチパラダイム型言語について紹介する。

なお，2階層モデルは，本システムの前身となった，オブジェクト指向言語と論理型プログラミング言語Prologを融合した知識処理言語S-LONLIの開発時の基本思想である。その基本図式を**図10.3**に示す。オブジェクト指向プログラミング (OOP) と論理型プログラミング (LP) の融合方式には，次の3種類がある。

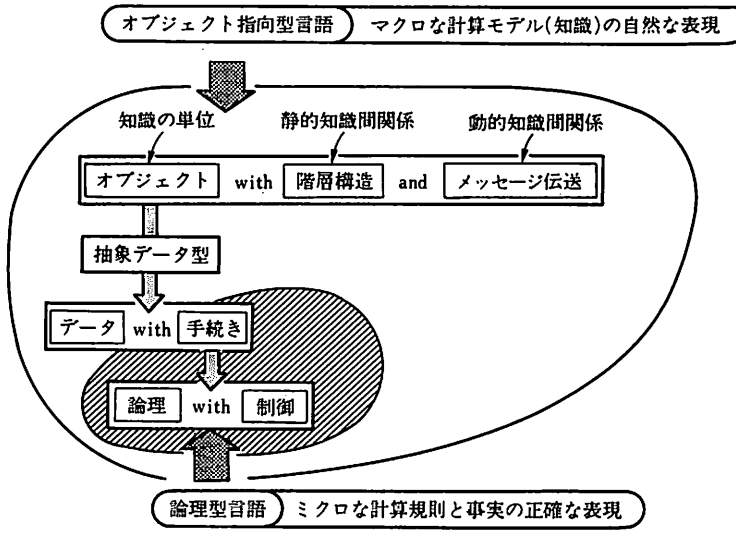


図10.3 マルチパラダイム型知識処理言語S-LONLIの設計思想

① LP and OOP
② LP with OOP
③ OOP with LP

従来は，両者を対等に融合する①の方式や論理型言語をベースにオブジェクト指向言語を取り込む②の方式が主であったが，S-LONLIでは2階層モデルの観点からオブジェクト指向言語をベースに論理型言語を取り込む③の方式をとった。ES/X90においてもこの設計思想を受け継いでいる。

(1) 推論モデル

推論制御方式は，メッセージ駆動とデータ駆動・イベント駆動の両方を導入する。メッセージ駆動は，外からの明示的な指示（メッセージ）によってオブジェクトを起動するものであり，受動的である。従来のプログラミング方式における手続き呼び出し，サブルーチン呼び出し，タスク起動，プロセス起動などはこの方式と考えられる。一方，データ駆動やイベント駆動は，外部状態の変化に伴ってある特定条件が満たされた時に自発的にオブジェクトが起動され

るものであり，能動的である。

これら2種類の起動方法は，メッセージをデータまたはイベントの一種とみなしたり，逆にデータの変化やイベント発生をメッセージとみなすことにより，どちらか1つの起動方法で擬似することは可能であるが，専門家の推論過程を素直にモデル化するという観点では好ましくない。一方，これらの機能を対等に扱って知識処理言語に組込んだ場合は，その意味仕様が複雑になる恐れがある。そこで我々は，メッセージ駆動を基本としたオブジェクト指向モデルをマクロなモデリングの基本的枠組みとして採用すると共に，その中で後者の実現をはかることにした。

(2)　知識表現モデル

いたずらに知識表現の種類を増やすことによって，1つの知識をいくつかの表現形式で記述可能とするような冗長性は複雑化による負の効果が大きいため，避けなければならない。我々は，各パラダイムはお互いに相補的な役割を果たすべきであるという考えから，以下のものをミクロレベルの必要最小限の表現形式として採用した。

①フレーム：知識の構造化，階層化により，大規模な知識の管理を容易にする。
②ルール：ノウハウ的な知識を if～then～形式で簡単に記述する。
③述語論理：事実や論理的規則などの宣言的知識および手順やアルゴリズムなどの手続き知識を同一形式で簡潔に記述する。

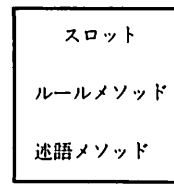
マクロレベルの知識処理の記述はオブジェクト指向モデルを基本にしたのに対し，ミクロレベルの知識の表現は，これら3種類を組み合わせて行う。この組み合わせの狙いは，事実的知識をフレームで表現し，規則的知識をルールで表現するという単純な知識表現方法を基本とする一方，この枠組みでは素直に表現できない手続き的知識などを述語論理（Prolog）で表現することである。知識処理の中で，部分的に従来の事務処理，計算処理などの情報処理が必要な場合は，既存の手続き型言語で記述された外部プログラムを呼び出す方式とした。

10.3.3 知識表現の基本構造

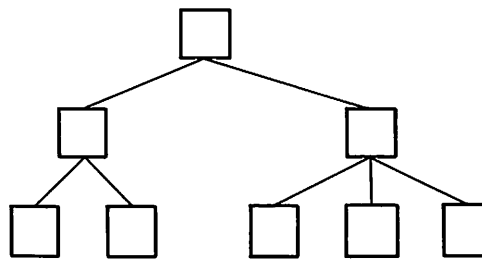
(1) 知識の静的構造

知識表現の基本単位をオブジェクトとして表現する。オブジェクト内は，図10.4(a)に示すように，以下の3種類の形式を記述可能とする。

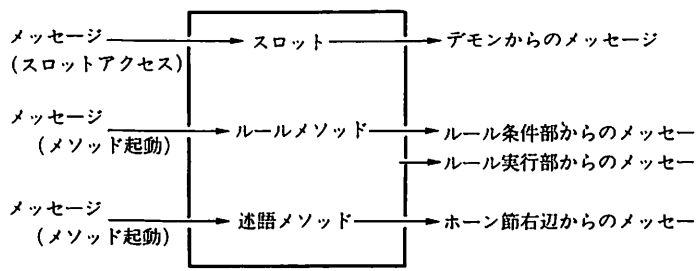
- ●スロット
- ●ルールメソッド
- ●述語メソッド



(a) オブジェクト



(b) オブジェクトの階層構造（静的構造）



(c) オブジェクト間の動的構造

図10.4 ES/X 90の知識表現の基本形式

スロットはフレームのスロットとオブジェクト指向における変数を兼ねたものである。ルールメソッドはプロダクションシステムにおけるルール群に対応する。述語メソッドはPrologプログラムに対応する。このオブジェクトは，オブジェクト指向やフレームと同様に図10.4(b)に示すような階層構造を有し，上下間に継承機能を持たせる。

(2)　知識の動的構造

実際の知識処理は，マクロレベルでは，図10.4(c)に示すような以下の2種類のオブジェクト間メッセージの送受信によって行われる。

●スロットへのアクセス
●メソッドの起動

ミクロなレベルの知識処理は，ルールメソッドおよび述語メソッドの実行によって行われる。

10.3.4　マルチパラダイム型言語設計の技術課題

このようなマルチパラダイム型言語の設計にあたって，「多機能，即，複雑」とならないように以下の方針を設定した。

①パラダイム相互の境界を明確にし，個々のパラダイムをできるだけ純粋な形で導入する。
②複数のパラダイムを必要としないシステムの記述に用いる時は，単一パラダイム型言語として使用可能とする。

第1の方針は，2階層モデルにおいて，マクロレベルでの知識オブジェクト間の関係の記述とミクロレベルでの各知識オブジェクト内部の記述を相互独立にすることによって達成する。第2の方針は，オブジェクトの記述に用いる3種類のパラダイムであるフレーム，ルール，述語論理の各々の仕様を各パラダイムだけで閉じたシステムが記述できる程度に十分なものにすることによって達成する。

しかしながら，もう一方において，全体の言語仕様が単一言語として統一化されていることも重要である。マルチパラダイム型言語の設計においては，こ

のような言語の統一性と先に述べたような設計方針が必ずしも一致しない点に難しさがある。

異なるパラダイムが有する概念間の関係は大きく次の3種類に分類できる。

●相補概念
●類似概念
●対立概念

言語仕様の統一性という観点からは，相補概念は比較的容易に両立が可能である。類似概念は，“似て非なるもの”の場合に混乱の可能性があるが，和集合的な拡張仕様によって基本的には両立可能である。最も大きな課題は対立概念の融合方式である。今回は特に2階層モデルに基づく言語仕様の設計という観点から，オブジェクト指向と他のパラダイムの間の融合方式が重要である。以下では，次のような，マクロな記述に用いるオブジェクト指向パラダイムとミクロな記述に用いるフレーム，プロダクションルール，述語論理の各パラダイムの間の融合方式について考察する。

●オブジェクト指向　vs.　フレーム
●オブジェクト指向　vs.　ルール
●オブジェクト指向　vs.　述語論理

10.4 オブジェクト指向とフレームの融合方式

10.4.1 概念の比較

オブジェクト指向型言語が計算モデルあるいはプログラミング技法として発展してきたのに対し，フレームは最初から知識表現形式として発展してきた。両者の目的は異なっていたが，結果的には類似概念が多い。オブジェクト指向型言語およびフレームの概念的特徴については，それぞれすでに8.1節および9.3節で述べたが，主要な概念の比較を表10.1にまとめた。

(1) 類似概念

表に示した類似概念について説明する。基本単位については，フレームをオブジェクトの一種とみなし，オブジェクトに統一する。階層構造と継承機能に

表 10.1　オブジェクト指向とフレームの概念比較

概念 ＼ パラダイム		オブジェクト指向	フレーム
類似概念	知識表現の基本単位	オブジェクト	フレーム
	階層構造と継承機能	有	有
	データ	変数	属性スロット
	手続き	メソッド	デモン
相補概念	手続き起動	メッセージ	データアクセス
	クラス・インスタンス	有（動的）	無または有（静的）
対立概念	名前の有効範囲がグローバルなもの	メソッド（変数は不可）	属性スロット
	継承の対象	メソッド	属性スロット

ついては，双方に同一の木構造を基本とし，継承機能を導入する。多重継承も可能とする。データについては，変数を属性スロットの一種とみなし，スロットに統一する。手続きについては，メソッドもデモンも同じと考え，メソッドに統一する。

(2)　相補概念

表に示した相補概念について説明する。手続き起動については，メソッドが外部からの明示的なメッセージの受信によって起動されるのに対し，デモンはスロットへのアクセスが発生した時に自動的に起動される。両者は相互に矛盾しないため，両方の機能を可能とする。

クラス・インスタンス関係については，1つのオブジェクトをひな型として同じ機能を持つオブジェクトを複数個生成する機能は，オブジェクト指向型言語では一般的である。フレーム型言語ではこのような概念の有無が明確でないが，矛盾するものではないので本機能を導入する。この結果，オブジェクト定義の基本構造は図 10.5 のようにする。

(3)　対立概念

表に示した対立概念に関しては，次節で詳細に検討するが，以下に内容の説明をしておく。

```
class<クラス名称>;
  inherit<上位クラス指定>
    class_part {
      <スロット定義> ;
      <メソッド定義> ; } ;
    instance_part {
      <スロット定義> ;
      <メソッド定義> ; } ;
end .
```

図10.5 ES/X90のオブジェクト定義の構文

① 名前の有効範囲：

オブジェクト指向では，データ抽象化あるいは情報隠蔽 (information hiding)の考えに基づき，外部からはメソッド名は見えるが，スロット名(変数名)は見えないのが一般的である。一方，フレームでは，スロットが中心なので，スロット名が外部から見える必要がある。

② 継承の対象：

オブジェクト指向では，オブジェクトを代表するものはメソッドであるため，継承の対象はメソッドであり，スロット（変数）は関連するメソッドの継承に伴って付随的に継承されるものである。一方，フレームでは，フレームを代表するスロットが継承される。

10.4.2 対立概念の融合

(1) 言語仕様の課題

言語仕様の定義という観点では，対立概念に関して次の項目を決める必要がある。

①スロット名とメソッド名の有効範囲

②スロットとメソッドの継承規則

③スロットとメソッドのアクセス規則

(2) オブジェクト指向に基づく仕様案

オブジェクト指向に基づく仕様案は次のようになる。

①オブジェクトの外部からはメソッド名だけが見える。
②継承の対象はメソッドだけとする。(付随的なスロットの継承を含む)
③オブジェクトの内部では，スロットへのアクセスもメソッドへのメッセージ送信も自由であるが，外部からは後者のみ可とする。

この場合，フレームシステムで中心的役割を果たすスロットへの制約が強すぎて，実質上，フレームシステムは作成できない。

(3) フレームに基づく仕様案

フレームに基づく仕様案としては，メソッドをスロットの一種とみなし，スロットとメソッドを全く対等に扱うことが考えられる。この場合，オブジェクト指向の観点からは次のような問題がある。

①スロットが外部から見えるため，データ抽象化の利点が損われる。
②スロットとメソッドの継承が独立に行われるため，スロットやメソッドの定義・参照関係が複雑になる。
③上記①，②に伴い，アクセス規制が複雑になる。

特に②に関しては，プログラムのわかりやすさに与える悪影響は大きい。その例を以下に示す。今，3個のオブジェクトA，B，Cが次のように定義されていたとする。

```
class A;              slot X; method p (...write(X)...); end.
class B; inherit A;   slot X;                            end.
class C; inherit B;   slot X;                            end.
```

メソッドpの中で参照しているスロットXは，AのXのように見えるが，このメソッドpは継承規則によりBとCでも使用可能であるため，BおよびCへメソッドpを起動するメッセージが送られた時は，各々BおよびCのXを参照することになる。

(4) 融合案

オブジェクト指向とフレームの間でのスロットとメソッドの概念の対立は，

論理的な矛盾に基づくものではなく，用途の違いによるものである。したがって，基本的には，双方の機能を和集合的に導入すると共に，その用途をユーザが明示的に指定するような融合方式が妥当であると考えられる。仕様案は次のようになる。

①スロットとメソッドは対等とする。
②スロットおよびメソッドへの外部からのアクセスの可否を明示する。
③スロットおよびメソッドの継承の可否を明示する。

この案では，オブジェクト指向プログラミングを行う場合は，スロットは外部アクセス不可，継承不可とし，メソッドは外部アクセス可，継承可とすればよい。一方，フレームシステム作成時はその逆の指定を行えばよい。しかしながら，スロットやメソッドを定義する毎に2種類の選択を行うような仕様は複雑であり，プログラムの理解容易性が低下する。そこで，2種類の可否の組み合わせは，可と可，不可と不可の場合が多いことに着目し，継承の可否は，外部アクセスの可否で兼ねることにする。すなわち，③は次のように改める。

③スロットおよびメソッドは外部アクセス可のものだけを継承可とする。

結局，言語仕様上は，スロットおよびメソッドの定義時に，外部アクセスの可否に応じて「公開」または「非公開」の区別をすることとした。いずれをデフォルトにするかは用途によるが，プログラムの信頼性を重視し，「非公開」をデフォルトとした。

(5) アクセス規則

このような融合方式に基づき，スロットおよびメソッドへのアクセス規則を次のようにした。

①公開スロットおよび公開メソッドを外部からアクセスする時は，オブジェクト名を指定する。
②自分のオブジェクトが上位オブジェクトから継承した公開スロットおよび公開メソッドをアクセスする時は，①のオブジェクト名の代わりにself を用いる。

```
class 会員 ;
 class_part  {
  slots 会員名簿(visible); } ;
 instance_part  {
  slots 名前(visible),
        住所(visible) ; } ;
end.
```

(a)　会員オブジェクトの例

```
class 業務 ;
 instance_part  {
  slots 責任者, 担当者, 実施日; } ;
end.

class 会員登録 ;
 inherit 業務 ;
 instance_part  {
  slots 会員数 (initial(0), on_read(会員数チェック),
        定員 (default(100));
  predicate_methods (入会、退会);
   入会(Name,Address) :- #会員数 := #会員数 + 1,
                          会員名簿登録(Name,Address);
   退会(Name) :- 会員名簿削除(Name),
                 #会員数 := #会員数 - 1;
   退会(Name) :- write('この方は会員ではありません');
   会員名簿登録(N,A) :- 会員 : create(N),
                        add_value(会員#会員名簿,N),
                        N #名前 := N,
                        N #住所 := A;
   会員名簿削除(N) :- 会員#会員名簿 == N,
                      N : kill,
                      delete_value(会員#会員名簿,N);
  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'); } ;
end.
```

(b)　会員登録オブジェクトの例

図10.6　マルチパラダイム型知識表現の例

③自分のオブジェクト内で定義されたスロットおよびメソッドをアクセスする時は，オブジェクト名および self による修飾は不要とする。

具体的な構文について，図 **10.6** の記述例を用いて以下に説明する。

(a) 公開スロットの宣言

図 10.6(a)の会員オブジェクトの 3 個のスロットに示すように，visible というキーワードを用いて公開を宣言する。

(b) スロットのアクセス

図 10.6(b)の述語メソッドの定義部分に示すように，スロット指定は上記①，②，③に対応して次のような構文とする。

① オブジェクト名#スロット名

② self #スロット名

③ #スロット名

スロットの値の読み出し，書き込み，追加，削除を行う時は，各々に対応した次の組込み述語を用いる。

read-value (スロット指定，項)

write-value (スロット指定，項)

add-value (スロット指定，項)

delete-value(スロット指定，項)

なお，頻繁に使用される read-value と write-value については，簡略構文 (syntax sugar)を導入する。すなわち，読み出し時はスロット指定を直接記述可とし，書き込みの時は代入を意味する「：＝」オペレータを用いてよい。

(c) 公開メソッドの宣言

図 10.6(b)に示すように，公開メソッドの場合は，メソッド定義の先頭のキーワード predicate_methods および rule_methods の直後にメソッド名を宣言する。この例では，入会と退会は公開メソッド，その他の会員名簿登録，会員名簿削除，会員数チェックは非公開メソッドである。

(d) メソッドの起動

メソッドの起動は，上記①，②，③に対応して次のような構文形式のメッセージ送信によって行う。

① send（オブジェクト名，メソッド呼出し）
② send（self，メソッド呼出し）
③ メソッド呼出し

ここで send は組込み述語である。メソッド呼出しは，メソッド名(実引数並び)の形式である。なお，頻繁に使用される①および②の形式については，簡略構文を導入し，以下の記法を許す。

① オブジェクト名：メソッド呼出し
② self：メソッド呼び出し

10.5 オブジェクト指向とプロダクションシステムの融合方式

10.5.1 概念の比較

オブジェクト指向型言語が情報処理のための計算モデルあるいはプログラミング技法として発展してきたのに対し，プロダクションシステムは，9.2 節で述べたように最初から知識処理のための計算モデル（推論モデル）として発展してきた。その結果，計算モデルとしては，前者がメッセージ駆動型，後者がデータ駆動型あるいは事象駆動型という相違がある。両者を言語として見た時の主要な概念の比較を 表 10.2 に示す。

(1) 類似概念

表に示した類似概念について説明する。基本単位については，ルール群をオ

表 10.2　オブジェクト指向とプロダクションシステムの概念比較

概念 ＼ パラダイム		オブジェクト指向	プロダクションシステム
類似概念	知識表現の基本単位	オブジェクト	ルール群
	データ	変数	ワーキングメモリ
	手続き	メソッド	ルール
相補概念	階層構造と継承機能	有	無
	クラス・インスタンス	有	無
	手続き起動	メッセージ駆動	データ駆動
対立概念	名前の有効範囲がグローバルなもの	メソッド (変数は不可)	ワーキングメモリ

ブジェクトの一種とみなし，オブジェクトに統一する。

データについては，ワーキングメモリの使用法には，固定的なデータ構造を前提とする場合とそうでない場合がある。前者については，オブジェクト指向における変数に対応づけることができるので，双方共にフレームの属性スロットの一種とみなして，スロットに統一する。後者については，後で導入する黒板モデルで代用する。

手続きについては，オブジェクト指向ではメソッドに対応し，プロダクションシステムではルールに対応する。このルールは原則的には個々に独立しているが，実際には関連するルールの集り（ルール群）によって1つのまとまった機能を果たす。したがって，このルール群をメソッドに対応させて，メソッドに統一する。ルール群に対応するメソッドをルールメソッドと呼ぶ。

(2) 相補概念

表に示した相補概念について説明する。階層構造と継承機能については，本来，プロダクションシステムには無い機能であるが，データ構造を有するワーキングメモリやルール群を階層構造で表現し，継承機能を持たせるようにしても矛盾を生じないため，本機能を導入する。クラス・インスタンス機能についても，プロダクションシステムには無いが，矛盾を生じるものではないので導入する。手続き起動については，オブジェクト指向がメッセージ駆動型計算モデルであるのに対し，プロダクションシステムはデータ駆動型計算モデルであり，大きな違いがある。これらの方式は，10.3節で述べたように，双方共に専門家の問題解決のモデル化に欠かせないため，両方の機能を導入する。

(3) 対立概念

表に示した対立概念に関しては，次節で詳細に検討するが，次のような問題がある。オブジェクト指向では，オブジェクトの外からはメソッド名は見えるがスロット名（変数名）は見えないのが一般的である。一方，プロダクションシステムでは，ワーキングメモリはグローバルデータの役割を果たすため，それに対応するスロット名は外から見える必要がある。したがって，スロット名の有効範囲に関する規則の設定が難しい。

10.5.2　対立概念の融合

(1)　言語仕様の課題

オブジェクト指向とプロダクションシステムの対立概念として，スロット名の有効範囲があるが，これは見かけ上の課題である。言語仕様設計の観点では，次の1番目の課題がより本質的であり，2番目と3番目の課題は付随的である。

①メッセージ駆動型計算モデルの枠組みの中でのデータ駆動型計算モデルの実現

②スロット名の有効範囲とアクセス規則

③ルールメソッドの起動方式

(2)　オブジェクト指向を重視した仕様案（小部屋方式）

オブジェクト指向の枠組みを極力くずさない範囲でプロダクションシステムを実現する方法として，次の案が考えられる。

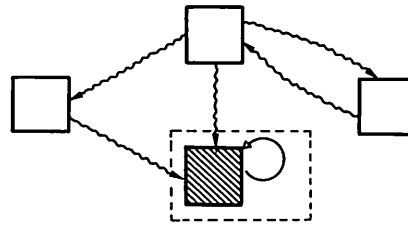
- 1つのオブジェクト内でのみデータ駆動型推論を許す。
- その推論に用いるルールおよびそのルールからアクセス可能なスロットは，そのオブジェクト内で定義されたルールメソッドとスロットおよび上位オブジェクトから継承したルールメソッドとスロットとする。

この方式は，全体の知識処理をオブジェクト間のメッセージ送受信で実行するという枠組みをくずさないで，特定の閉ざされた領域に限定してデータ駆動型推論を許しているという意味で，小部屋方式と呼ぶ。その概念図を**図10.7**(a)に示すが，この方式は，データ駆動型推論のためのルールの数が多くなった時に，ルールの管理が難かしくなる。それに伴って，ルール群の制御も難しい。

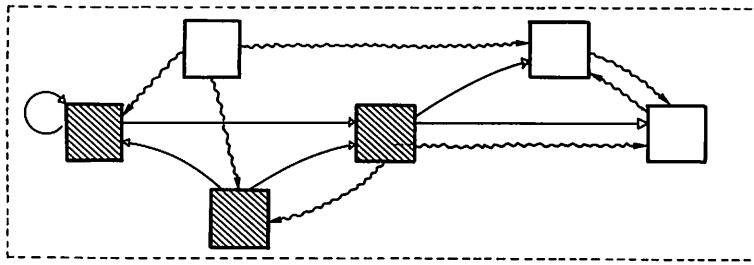
(3)　プロダクションシステムを重視した仕様案（大部屋方式）

大規模なプロダクションシステムを容易に構築するためには，オブジェクトをスロットあるいはルールを管理するモジュールと位置づけた次のような案が考えられる。

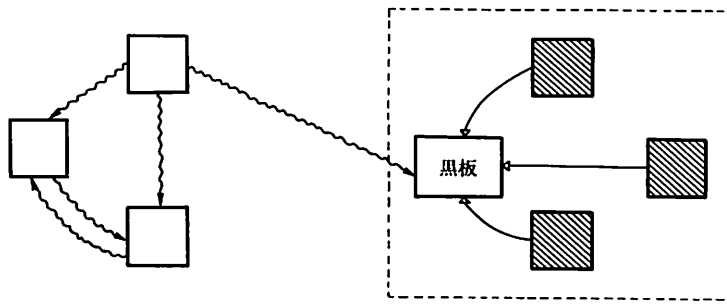
- スロットはすべて外から見えるものとする。
- すべての生成されたインスタンスのスロットおよびルールメソッドがデ



(a) 小部屋方式



(b) 大部屋方式



(c) 会議室方式

〔記法〕

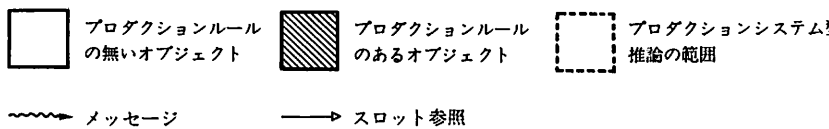


図 10.7 オブジェクト指向とプロダクションシステムの融合方式

ータ駆動型推論に参加する。

全体の知識処理が行われているオブジェクト指向の世界と全く同じ世界でデータ駆動型推論が実行されるという意味で，この方式を大部屋方式と呼ぶ。その概念図を図 10.7(b) に示すが，これは，大規模なシステムのモデル化に欠かせないオブジェクト指向のデータ抽象化機能を損ねている。

(4)　融合案（会議室方式）

小部屋方式による大規模なプロダクションシステム構築不可という欠点，および大部屋方式によるオブジェクト指向概念の崩壊という欠点はいずれもマルチパラダイム型言語にとっては致命的なものである。そこで，これらの欠点を回避しつつ，オブジェクト指向の枠組みの中にデータ駆動型計算モデルを組み込む方式として，次のような融合案を考えた。

「多数のオブジェクトから成るメッセージ駆動型計算モデルの中でデータ駆動型の機能を実現する必要が生じた時は，そのために必要な最小限のスロットとルールメソッド（ルール群）を指定し，その集合からなる小世界の中でのみデータ駆動型の推論を可能とする。」

この方式は，あたかも会議室に必要なオブジェクトを集めて，会議室内では自由に発言したり，どのデータにもアクセスできるようにしているという意味で，会議室方式と呼ぶ。すなわち，会議室の中はプロダクションシステムの世界であるが，廊下に出るとメッセージの飛びかうオブジェクト指向の世界である。概念図を図 10.7(c) に示す。

この会議室方式を実現するためには，オブジェクト指向の下でプロダクションシステムを制御する機構が必要である。そのために黒板モデルを導入し，データ駆動型推論を行う時には黒板にその起動条件となるキーワードを記入し，起動するようにした。本方式を具体化するための言語仕様は次のようなものである。

①データ駆動型推論に用いるスロットを明示する。
②データ駆動型推論に用いるルールメソッドおよびその起動条件となるキ

ーワードを明示する。

③黒板への起動条件キーワード書き込み機能とデータ駆動型推論の開始，終了機能を設ける。

まず，①については，外からのスロットアクセスを許すという意味で，10.4節で述べた公開指定と目的が同じであるため，ルールメソッドからアクセスするスロットは公開指定をすることとした。

次の②については，ルールメソッド定義時にその起動条件となるキーワードも同時に定義するようにした。たとえば，不動産評価エキスパートシステムの住宅評価オブジェクトにおける中古住宅評価のためのルールメソッドの定義例を図10.8に示す。この定義の中のon(住宅)が起動条件キーワードの指定である。データ駆動型推論の開始時に黒板に“住宅”というキーワードが記入されていれば，この中古住宅評価メソッドが起動される。このプロダクションルールの条件部で参照されているスロットはすべて公開スロットである。なお，行動部の＋(m, c)は黒板に中間結果mを書き込む組込み述語であり，cは確信度である。

```
class 住宅評価;
 inherit 評価;
 instance_part  {
  rule_method ;
    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10.8 プロダクションシステム用ルールメソッドの定義例

10.5.3　拡張機能：逐次型ルールメソッド

一般にプロダクションシステムは，ルール条件部の判定処理，競合解消処理，ルール行動部の実行という認知・行動サイクルの繰返し処理を行うため，ルール数が多くなると次のような問題が生じる。

①ルール条件部の判定処理に時間がかかる。
②適切な競合解消戦略の選択が難しい。
③推論結果の検証と不良箇所の検出が難しい。

本システムにおいても，第9章で述べたような競合解消戦略を用意しているが，各々の戦略の使い分け方が認知モデルとの関連で明確になっているわけではない。そこで，我々は，ルール形式の知識表現が持つ"簡易プログラミング"としての特徴に着目し，プロダクションシステム型ルールメソッドの他に，ルールを記述順に実行する逐次型ルールメソッドを新たに導入した。この主な仕様は次のようなものである。

①この逐次型ルールメソッドはメッセージ受信によって起動される。
②メソッド内のルール群は記述順に上から逐次実行される。

逐次実行の方式は4種類用意する。まず，ルール群を1回だけ実行する時はonce，繰返し実行する時はmultiを指定する。このmulti指定は手続き型言語のループに対応する。次にルール群の1回の実行の中で，条件を満たすルールが複数個あった時，最初の1つだけ実行する場合を基本とするが，すべて実行する場合はall指定をする。前者は手続き型言語におけるcase文に対応する。

図10.9 にonce指定の例を示す。図10.6(b)の会員数チェックメソッドもこの例である。このように記述順に実行される逐次型ルールメソッドは，実行方式が単純なため，一般に処理速度が早く，検証もしやすいという利点がある。したがって，ルール形式の知識のうち，本質的にデータ駆動方式のところだけは黒板型ルールメソッドで記述し，その他は逐次型ルールメソッドで記述するのがよい。

```
class 総合判定;
 inherit 業務;
 instance_part {
  rule_metods (中古住宅評価額設定);
    rules 中古住宅評価額設定 forward (once);
      if 報告書#中古指数 > 0.8 then 報告書#評価額 : = #新築評価額*0.2 ;
      if 報告書#中古指数 > 0.6 then 報告書#評価額 : = #新築評価額*0.4 ;
      if 報告書#中古指数 > 0.4 then 報告書#評価額 : = #新築評価額*0.6 ;
      if 報告書#中古指数 > 0.2 then 報告書#評価額 : = #新築評価額*0.8 ; } ;
end.
```

図 10.9 逐次実行型ルールメソッドの定義例

10.6 オブジェクト指向と述語論理の融合方式

10.6.1 概念の比較

ここでは述語論理のパラダイムとして，その代表的なプログラミング言語である Prolog を想定する。オブジェクト指向パラダイムがマクロなレベルの分散協調型問題解決モデルを与えるのに対し，Prolog は，すでに第 6 章で述べたように，論理学を基礎とした 3 段論法型の問題解決モデルを与えている。その結果，両者を言語として比較した時，類似点は少ない。主要な概念の比較を表 10.3 に示す。

表 10.3 オブジェクト指向と述語論理の概念比較

概念＼パラダイム		オブジェクト指向	述語論理
類似概念	知識表現の基本単位	オブジェクト	ホーン節集合
	データ	変数	変数とデータベース
	手続き	メソッド	ホーン節
	手続き起動	メッセージ駆動	ユニフィケーション
対立概念	変数の副作用	有	無
	階層構造と継承機能	有	無
	クラス・インスタンス	有	無
	並列実行性	有	無
	バックトラッキング	無	有

(1)　類似概念

表に示した類似概念について説明する。基本単位については，Prolog では，プログラム全体が事実と規則の集合から成る 1 つのデータベースとして扱われ，モジュール的概念はない。したがって，このデータベースをオブジェクトとみなすことにより，知識表現の基本単位をオブジェクトに統一する。

データについては，Prolog の変数は，各々の節の中だけで有効な一時的変数であり，オブジェクト指向における状態変数とは異なる。したがって，後者をスロットとすると共に，前者の機能は Prolog の変数の仕様のままで残す。なお，Prolog では，状態変数に対応する機能としては，データベースへの節の追加と削除の機能がある。

手続きについては，オブジェクト指向のメソッドに相当するものは Prolog では節である。実際には，左辺の述語名が同じ節がいくつか集まって 1 まとまりの機能を果たす場合が多いので，この節集合をメソッドに対応させ，述語メソッドと呼ぶことにする。図 10.6 の例では，入会メソッドや退会メソッドなどが述語メソッドである。

手続き起動については，オブジェクト指向ではメソッドの起動はメッセージ送信により行われるが，Prolog では節の起動はその節の左辺とのユニフィケーション（パターンマッチング）により行われる。詳細は後述するが，最終的には，述語メソッドをその定義を含むオブジェクトの外部から起動する時はメッセージ送信とユニフィケーションの組み合わせで行い，内部からはユニフィケーションのみとした。先の図 10.6 の例では，入会メソッドを外部から起動する時は，

会員登録：入会（田中，横浜市）

のようにメッセージ送信形式とするが，会員名薄登録メソッドのように内部から起動される時は，通常の Prolog と同じように，

会員名薄登録（田中，横浜市）

と記述すればよい。

(2) 対立概念

表に示した対立概念に関しては，次節で詳細に検討するが，以下に内容の説明をしておく。変数の副作用については，スロット変数の場合は，オブジェクト内部の全域で有効であり，その値を自由に書き換えられるため，バックトラッキングの処理が非常に複雑になる。

階層構造と継承機能については，Prologでは，節集合全体が1つのデータベースとみなされ，このデータベースに含まれる知識のみが真であるという閉世界仮説に基づいて推論が行われるため，このような世界の中にオブジェクトの階層構造と継承機能をどのように融合させるかという問題は，言語設計の大きな課題である。クラス・インスタンス関係については，オブジェクト指向では，クラスからインスタンスを動的に生成する機能があるが，Prologには無い。本機能もPrologの閉世界仮説との関係を明確にする必要がある。

並列実行性については，逐次実行と縦型探索を前提としたPrologの世界で生じるバックトラッキングを並列実行の世界で処理する場合，その対象が広範囲におよぶため，いわゆる分散バックトラッキング (distributed backtracking) という複雑な問題がある。バックトラッキングについては，オブジェクト指向の世界でこのバックトラッキングの対象をどの範囲に限定するかが重要であり，オブジェクト指向用に導入した組込み述語の成功／失敗の定義や副作用の回復処理などの課題がある。

10.6.2 対立概念の融合

(1) 言語仕様の課題

オブジェクト指向とPrologの融合方式については，本研究の基礎となった知識処理言語S-LONLIをマルチパラダイムの観点で発展させた。表10.3の5項目の対立概念は，言語仕様設計の観点では次の2項目の課題にまとめられる。

①メソッド名の有効範囲とアクセス規則

②バックトラッキング機能の制約条件

(2) オブジェクト指向を重視した仕様案

オブジェクト指向の枠組みを極力保持しながら述語メソッドの仕様を

Prologにする方法として，次の案が考えられる。

①あるオブジェクトが持つ述語メソッドデータベースには，自分のオブジェクトで定義したものと上位から継承したものが含まれる。
②バックトラッキングは同じ述語メソッドデータベース内でのみ可能とし，副作用を伴う可能性のあるメッセージ送信やスロットアクセスやインスタンス生成，削除などの組込み述語は常に成功とみなす。

上記①は，図10.10(a)に示すように，自分のオブジェクト内で定義された節と上位から継承した節を対等に扱うため，モジュール性が低下する。たとえば，

```
class A;                 predicate_methods ; p :- .. ,q, ..
                                             q :- ...  end.
class B; inherit A;      predicate_methods ; q :- ...  end.
```

という2つのクラスにおいて，クラスBに対し外部からメソッドpを起動するメッセージがきた時，クラスAで定義されたpが起動され，その右辺で参照したqとしてはクラスAのqではなく，クラスBのqが起動されることになる。

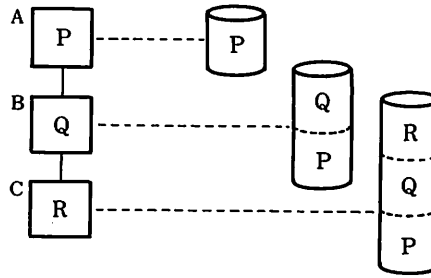
次に②は，バックトラッキング発生時に，述語メソッドの実行によって生じた種々の副作用をそのままにするため，バックトラッキングの使用が強く制約されてしまう。たとえば，他のオブジェクトのメソッドを利用した縦型探索はできなくなる。

(3)　述語論理を重視した仕様案

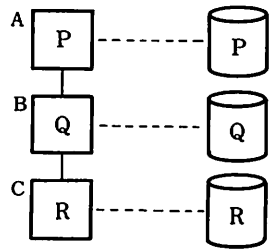
Prologが提供する述語論理型プログラミングパラダイムを極力保持する仕様として，次の案が考えられる。

①あるオブジェクトが持つ述語メソッドデータベースには，自分のオブジェクト内で定義した述語メソッドのみが含まれる。
②バックトラッキングは，プログラム全体を対象とし，入出力以外の副作用はすべて元に戻す。

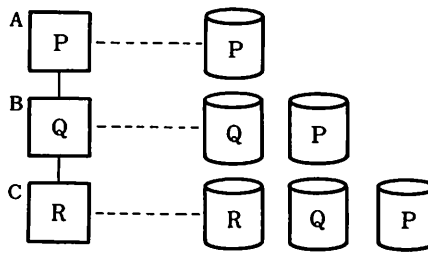
上記①の仕様では，図10.10(b)に示すように，オブジェクト指向やフレーム



(a) オブジェクト指向型プログラミング亜視案



(b) 論理型プログラミング亜視案



(c) 融合案

〔記法〕

図 10.10 オブジェクト指向のクラス階層概念と論理型知識との融合方式 (継承規則と述語メソッドの有効範囲規則の関係)

のパラダイムに特徴的な継承機能が妨げられるため，is-a 関係を基本とした知識の階層的分類ができなくなる。また，②を実現するためには，スロットの値の変更やインスタンスの生成，削除の履歴を管理する必要があるため，実行系が複雑になりすぎる。

(4) 融合案

上記2案はいずれに決めても，マルチパラダイム型言語としての機能に偏りが生じるため，両者の欠点を補う以下の融合案を考えた。

①あるオブジェクトが持つ述語メソッドデータベースは，そのオブジェクトで定義した述語メソッドのみからなるもの，および，上位の各々のオブジェクトから継承したメソッドのみからなるものを別々に有するものとし，外部から受信したメッセージに対しては下位優先で適用する。

②バックトラッキングは，プログラム全体を対象とするが，オブジェクト指向パラダイム用に導入した組込み述語の副作用は無視する。さらに，メッセージ送信に関して，メッセージの送信側はそのメッセージの実行終了まで待機する逐次実行型とする。

上記①の仕様は，図 10.10(c) に示すように，Prolog の閉世界仮説の概念とオブジェクト指向の継承機能を両立させるものである。すなわち，外部から受信したメッセージに対応する述語メソッドが自分のオブジェクト内で定義されていればそれを実行する。もし無ければ，そのメッセージの処理をすぐ上のオブジェクトに依頼すると考えればよい。この方式は，図 10.10(a) に示した方式のように自分の述語メソッドと継承した述語メソッドを1つにまとめることはしないため，モジュール性低下の問題は生じない。上記(2)で示した例において，述語メソッド p と同じオブジェクト内で定義された述語メソッド q を呼び出す場合は，Prolog の通常の形式で記述すればよい。一方，継承された述語メソッドであるクラス B の q を呼び出す場合は，send(self, q) または self：q のようにメッセージ送信の形式で記述する。

上記②の仕様は，Prolog に特徴的なバックトラッキングの機能を保存しながら，オブジェクト指向パラダイムの意味仕様および実現方式を複雑にしないための妥協案である。この方式では，副作用を無視したバックトラッキングを行

うため，その使い方次第では，かなり複雑な動きをするプログラムが作成される可能性がある。この問題を避けるためには，述語メソッドを以下のいずれの目的に用いるかを明確に意識して使用する必要がある。

①バックトラッキング機能を用いて，探索処理等の後向き推論を行う。

②通常の手続き処理を行い，バックトラッキング機能は用いない。

10.7 マルチパラダイム型協調推論の記述例

不動産評価エキスパートシステムのシステム設計において，2階層モデルに基づき，メッセージ駆動型とデータ駆動型の推論制御方式を組み合わせた協調型推論モデルの構築容易性を確認した。オフィス業務における指示／依頼型業務の処理をメッセージ駆動型オブジェクトで記述し，会議型業務の処理をデータ駆動型オブジェクトで記述することにより，自然なモデル化ができた。

(1) 複数の専門家の協調作業

ビジネス分野の業務（問題解決プロセス）の例として，金融機関などが不動産の価値を評価する例を取り上げてみよう。

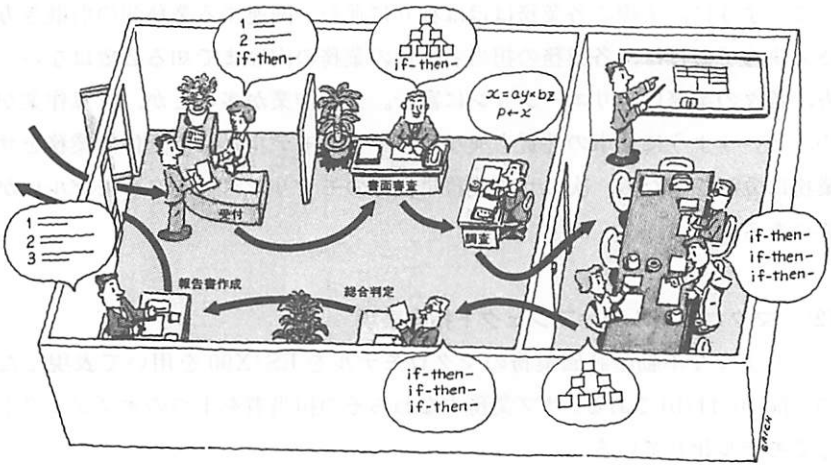
不動産評価には，図10.11(a)に示すように，受付係，書類係，各種評価係，報告書作成係など多数の人間が関わる。どの人もなんらかの専門的知識を有した専門家である。この業務は，大きく見て，部単位あるいは担当者単位のサブ業務に分割できる。個々のサブ業務が互いに協力しあい，不動産評価という1つの大きな業務を遂行するわけである。

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1つのモデルとして表すのは難しい。全体を複数のサブ業務の協調作業として考えるとわかりやすい。このような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。

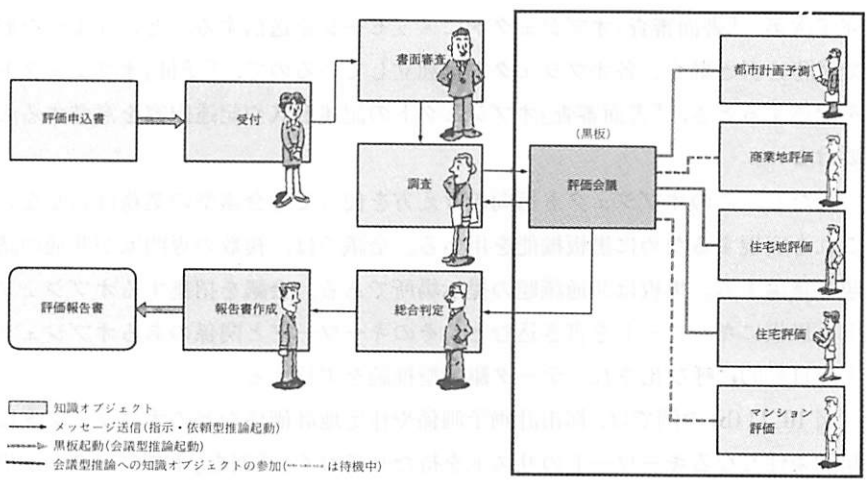
各サブ業務が協力しあうやり方には，2つのタイプがある。1つは，

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である。このような指示・依頼型のタイプでは普通，電話や書類の受け渡しによって業務を引き継ぐ。もう1つは，



(a)　複数の専門家の協調作業



(b)　不動産評価業務のマクロモデル

図10.11　マルチパラダイムを用いた不動産評価ESの例
(日経BP社の許可を得て文献(75)より転載)

複数の専門家が集まってそれぞれの意見を述べる会議型のタイプである。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に会議を開く。

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。分類作業が多いとか，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）が必要になる。

(2) マクロモデルのオブジェクト指向表現

このような不動産評価業務のマクロモデルを ES/X90 を用いて表現したものが図 10.11(b)である。サブ業務すなわちその担当者を 1 つのオブジェクトとしてモデル化している。

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」オブジェクトは受付業務が完了次第，「書面審査」オブジェクト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。各オブジェクトは独立しているので，「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や記述内容を意識する必要はない。

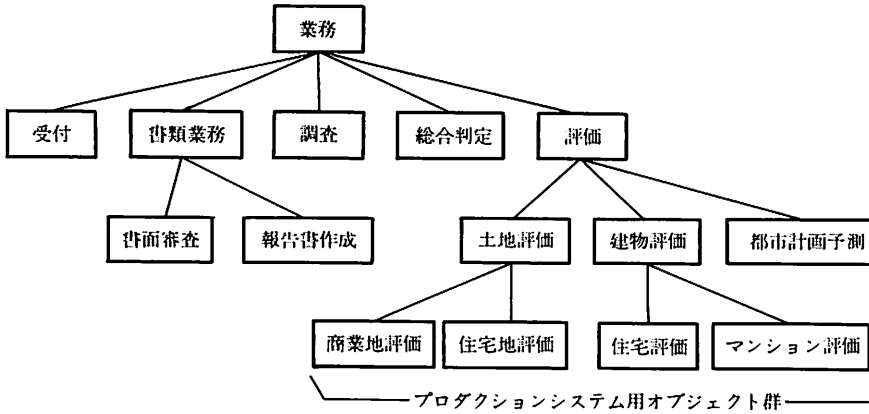
ただし，このオブジェクト指向の考え方を使っても会議型の業務は表せない。これを実現するために黒板機能を用いる。会議では，複数の専門家が共通の議題を議論する。黒板は共通議題の提示場所である。会議を招集するオブジェクトが黒板にキーワードを書き込むと，そのキーワードと関係のあるオブジェクトが自動的に呼び出され，データ駆動型推論を実行する。

図 10.11(b)の例では，都市計画予測係や住宅地評価係などのオブジェクトに起動条件となるキーワードのリストを持たせておく。「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」オブジェクトが呼び出され，不動産評価を開始する。

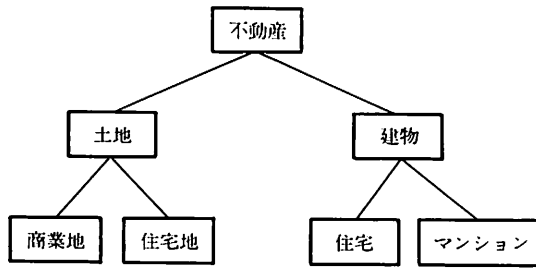
(3) ミクロモデルのマルチパラダイム型表現

このシステムのオブジェクト階層を図 10.12 に示す。図の(a)は業務の階層，図の(b)は業務の対象となる不動産の階層である。個々の記述例として，「住宅

評価」オブジェクトの「中古住宅評価」ルールメソッドについては図 10.8 で，「総合判定」オブジェクトの「中古住宅評価額設定」ルールメソッドについては図 10.9 でそれぞれ説明した。



(a)　業務担当者のオブジェクトの階層



(b)　不動産のオブジェクトの階層

図 10.12　不動産評価 ES のオブジェクト階層