特　集　「知的プログラミングの基礎」

人工知能とプログラミングの接点

Contact Points between Artificial Intelligence and Programming

畝見達夫*
Tatsuo Unemi

*　東京工業大学大学院総合理工学研究科システム科学専攻
Dept. of Systems Science, Grad. School of Sci. & Eng. at Nagatsuda, Tokyo Inst. of Tech., Yokohama 227, Japan.

1987年5月28日　受理

Keywords: artificial intelligence, knowledge engineering, programming support, programming paradigm.

1. はじめに

プログラミングは人工知能の手段である．なぜなら，コンピュータは，知能を人工とするための強力な道具であり，プログラミングは，コンピュータに特定の機能を備えさせる手段だからである．知能のメカニズムが，人間の場合，脳を中心に構成される柔軟な情報処理能力に依存しているという主張を受け入れるならば，知能の人工物による実現のためにコンピュータを使うことは，きわめて自然である．

また，人工知能は，あらゆる知的活動を研究の対象とする学問である．一口に知的活動といっても，連想・記憶・推論・学習など，さまざまな局面があり，日常会話やおもちゃのパズル解きから，高度な研究活動，あるいは芸術の創作まで，実にさまざまな典型を捜すことができる．プログラミングは，そのような知的活動の1つであって，しかも，プログラミングそのものが人工知能研究者にとって馴染み深い作業である．つまり，プログラミングは，人工知能の応用問題としては格好の対象となる．

以下では，筆者の経験もふまえ，2つの観点「人工知能の対象領域としてのプログラミング」および「人工知能研究の手段としてのプログラミング」について，概略的に考察する．

2. 人工知能の対象領域としてのプログラミング

まず，人工知能の技術が，プログラミング，大きく言えばソフトウェア生産にどのように応用され得るかという問題である．プログラミング作業の効率化に対する人工知能の応用の試みは，「自動プログラミング」と呼ばれ，すでに多くの研究がなされてきた[1]．ここでは，人工知能の応用例題としてプログラミングのどのような場面に注目すればよいか，という視点に立って考えてみることにする．

2・1　プログラミングのフェーズ

そもそも，プログラミングとはどのような性質をもつ作業なのであろうか．プログラミングを部分的な作業に分解し，それらのつながりを考えると，作業の流れはおおよそ図1のようになろう．すなわち，最初に，

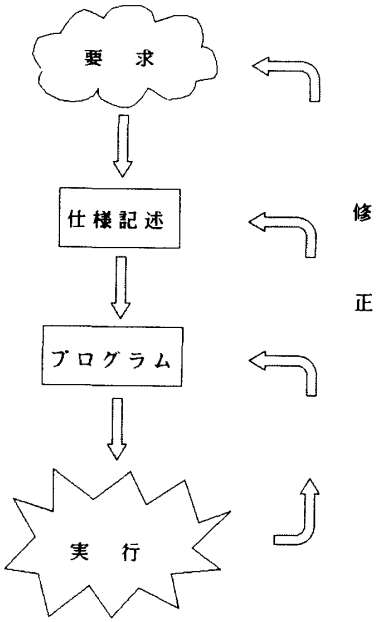

図1　プログラミングの作業過程 I

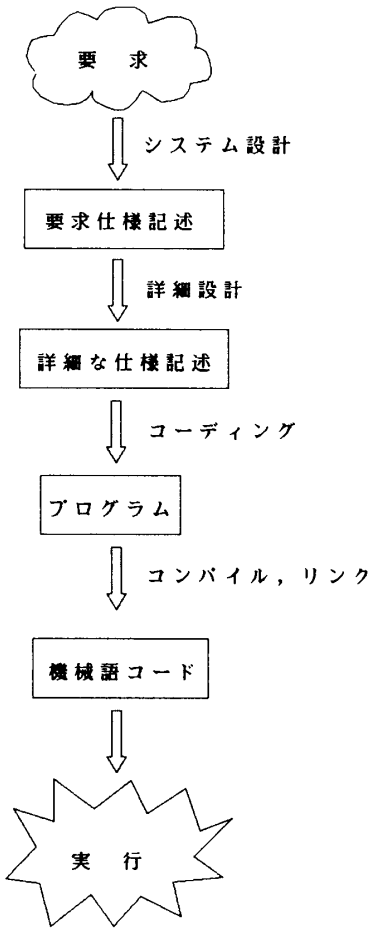

図 2　プログラミングの作業過程 II

コンピュータの動作（入出力）についての人間側からの要求があり，最終的にコンピュータを使ったシステムの上にそれが実現されれば完成となる．

各フェーズの詳細は，プログラミングのスタイルによって，あるいはソフトウェアの規模や，実験的なシステムか，実用的なシステムかといったシステムの性質によっても異なってくる．

図2は，世の中に存在するおそらく最も多くのプログラムの開発形式と思われるものである．すなわち，自然言語や図による要求仕様記述が与えられ，大まかなシステム設計，詳細設計，コーディング，コンパイル，テスト実行，デバッグといった過程を経る．多くのエンジニアによる分業によって，1つのシステムを組み上げるような場合には，各フェーズで，できるだけ完璧なものを念入りに作り上げたほうが作業効率の向上という点では有利である．

図3は，Lisp や Basic のようなインタプリタ言語での実験的なシステムを作る場合の開発過程である．小規模なプログラムでは，プログラマの知識の整理や，発想の手助けのためにメモを書くことはあっても，いちいち仕様記述を詳細に文書化する必要はない．また，必要な文書は，プログラム中に注釈として挿入してお

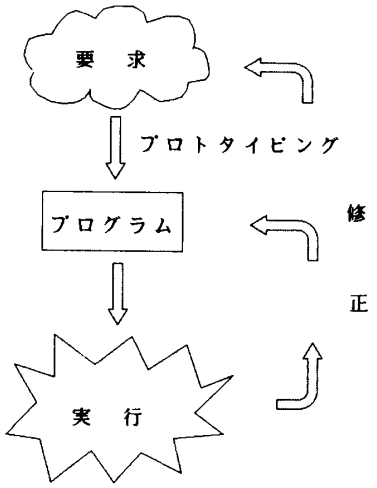

図 3　プログラミングの作業過程 III

くくらいで十分な場合も多い．このようなプログラミング・スタイルでは，要求仕様をコーディングの前に詳細に記述する必要もないし，プログラムの修正から実行までの手間がかからず，ソースコード中のバグの同定も比較的容易なため，プログラム記述のレベルにおいても，多少の不完全性が作業効率を致命的に悪くするようなことは少ない．

プログラミングの過程では，図1にあるように，プログラムの完成に至るまでの要求，仕様，プログラムなどのあらゆるレベルの修正，変更が起こり得る．要求に近いほうを上，プログラムに近いほうを下と呼ぶならば，下のレベルの記述は，上のレベルの条件を満足するよう修正されるべきであり，下のレベルが実現困難，あるいは上のレベルの本来の要求を満たす別のよりよい方法が見つかった場合には，上のレベルでの条件を修正すべきである．

以下，このようなフェーズへの分割を元に，考え得るプログラミングにおける困難さ，つまり，知能を駆使すべき点について考察する．

〔1〕 仕様として何を表現するか

プログラムは，コンピュータに解釈可能なだけ具体的に表現された仕様記述である．もちろん，応答のタイミングなど最初の要求に含まれる，いくつかの仕様は，プログラムの記述の中では隠れてしまい，かわりにハードウェアの制約に由来する表示方法や，入力方法に関する余計な仕様が付加されているかもしれない．ある意味では，プログラムは，本来の要求仕様と，コンピュータの能力との間の妥協の産物なのである．

プログラムが，コンピュータ側の都合を反映した記述であるなら，反対に人間の都合に合わせた，本来の要求を直接的に，あるいは人間が説明しやすいように表現するための手段が必要となるのは当然である．こ

のような手段として，述語論理などの形式的表現に基づくものと，自然言語に基づくものが研究されている．

人間は，要求の全体を直接的に表現するよりも，やってほしい動作の具体例を挙げるほうが得意である．動作例からのプログラム合成の研究は，帰納学習の応用問題としても興味深い課題である．仕様の記述として適切な典型的入出力例を人間から引き出すこと，およびその記述を元にした手続きの生成は，まさに知的作業である．

〔2〕 **要求仕様の言語化**

さて，表現手段が与えられたとしても，果たして，要求仕様自体を表現し尽くせるものなのであろうか．日常において，思ったことをそのとおりに他人に伝えることに，困難を感じたことのある人は多いであろう．原因は2つある．1つは言葉にならないもやもやしたものを伝えたい．つまり，話し手に思ったことを表現する能力が足りない場合であり，もう1つは，聞き手の能力が話し手の期待するものから掛け離れている場合である．

現状では，聞き手としてのコンピュータの能力は，きわめて貧弱なものであり，普通のプログラマならば，みなその能力のほどを知っている．知られているという点では，ある表現が話者の意図を正しく伝達するものかどうかを，容易に判断できるという意味では都合がよい．しかし，その代償として，話し手は，わずらわしいほど懇切丁寧な表現をしなければならないのである．

また，表現手段を用意することと，実際の意図を明確にすることとは別である．われわれは，自分自身の本当の欲求について，それほど厳密に意識しているわけではないし，いったい，どうなれば満足できるのかが具体的にわかっているわけでもない．もやもやとしたイメージをはっきりとした形の表現に移行する手助けが必要なのである．

〔3〕 **仕様の作成と修正**

そもそも，元の要求自体があいまいである場合が多いのだから，仕様の記述が最初から完全であることなど稀である．実際のプログラミングでは，まずは不完全であっても要求をある程度反映した何らかの記述を作り出し，下のレベルの記述へ翻訳することになる．

記述自身には，翻訳に対するあいまい性を含まないほうが望ましい．もちろん，たとえば，変数の値を格納するアドレスのように，下のレベルでは，いずれかに決定しなければならない事柄であっても，本来の要求ではどうでもよい場合もあるから，要求として必要な部分について，あいまい性を少なくせよということである．

実際に翻訳あるいは実行した結果が，作者の意図にそぐわなければ記述の修正が必要となる．下のレベルでの誤りの原因は，それより上のレベルの記述の中にある．たとえば，あるテストデータに対して期待した動作が得られなかった場合，その誤りの所在によって修正の難しさは質的に異なってくる．すなわち，テストデータに誤りがあるなら最もたやすく，プログラムならば若干困難であり，仕様記述の場合には，さらに困難になる．最初の要求に矛盾があるような場合には，プログラミングの作業をほぼ全面的にやりなおさなければならなくなる．

できるだけ明確な要求を引き出すことが重要であるとはいっても，人間に負担をかけすぎてはならない．細部にわたって要求を具体化するには，人間の発想を広げるための援助が必要である．事実，類似のシステムあるいは試作システムの実行結果を観察することは，発想を広げるためにかなり役立つ．試作システムの実行を容易にするためには，実行結果を見た時点で，本当の要求に合わないと判断された場合に，スムーズに要求仕様あるいはプログラムを修正できることが大切である．

〔4〕 **要求の実現可能性**

コンピュータを使って何かをしようと考える以前に，コンピュータで何ができるかということを，ある程度知らなければならない．いかに要求が明確であっても，実現困難な内容ではどうしようもない．また，人間側がコンピュータの能力を過小評価し，本当の要求をあきらめている場合もあろう．類似のプログラムを作成した経験があれば，だいたいの予測はできるが，未経験の分野では，最初の要求を結果的にかなり縮小しなければならなくなるような事態が起こりやすい．実現可能性を左右する要素として，利用可能なハードウェア環境，サブルーチン・ライブラリなどの利用可能なソフトウェア，プログラミングあるいはデータの量的な入力能力などが考えられる．プログラミングを成功させるには，これらの知識を総合的に利用することによって，できるだけ早い時期に，的確な仕様の修正を行う必要がある．当然，これを支援するには総合的な知識の利用が要請される．

2・2 プログラミングのための知的支援システム

人工知能の応用技術の1つである知識工学的なアプローチによって，プログラミングを支援するシステムを設計することを考えてみよう．

エキスパートシステムは，そのタスクの性質によっ

て，①データ解釈型，②診断型，③監視・制御型，④計画・設計型の４つに分類できる[2]．先に述べたプログラミングの各フェーズのどの部分に，これらのタイプのタスクが存在するか，ということについて考えてみよう．

〔1〕 データ解釈型

いくらか雑音を含んだ，比較的大量のデータに対して，定性的な分析あるいは変換を与えるようなタイプのタスクである．アセンブラやコンパイラなどのように，プログラミングの各段階で上のレベルの記述を，より機械語に近い下のレベルの記述に翻訳するタスクは，この範疇に入る．

記述の翻訳に際して，知識や推論が必要となるのは，たとえば，手続きの最適化である．メモリの配置，高速なメモリの有効利用，冗長な手続きの排除などは，コンパイラの基本的な技術として伝統的に練り上げられた分野であるが，型推論，部分計算，プログラム変換などは，人工知能の技術を応用する格好の材料である．

また，プログラム記述の明らかな誤りやあいまいさの解消も，知識を要する処理である．

〔2〕 診　断　型

診断とは，対象システムの動作やテスト結果から，対象システムの構造や，動作機構に関する知識を元に異常の原因を同定する作業である．要求仕様のとおりに動作するのが正常とすれば，バグを含んだプログラムの動作は異常であり，バグはその原因である．

プログラムの場合，生体や機械とは違い，その構造のすべてが，コンピュータに読める形，すなわち機械語あるいはソースコードの形で，すでに表現されているわけだが，バグの同定という推論に使用するには，そのままでは都合が悪い．プログラムの構造を何らかの知識の形式に表現し直す，あるいは，特徴的な要素，たとえば変数やサブルーチンの参照関係などを抽出することが必要となる．

また，プログラミングの途中では，正常状態がどのようなものであるかということについては，完全に記述できない場合もある．正常状態とはすなわち要求仕様であり，推論に利用可能な形式に，要求仕様を記述することを考えなければならない．

他の種類のシステムにおける診断と大きく異なる点は，対象システムが少量多品種であることであろう．機械ならば，同じ機種については，同じエキスパートシステムを使うことができるが，プログラムの場合，それぞれの製品に専用の知識が必要となる．であるから，専門家から得られる診断のための知識だけでは，個々の対象システムに対処することは困難であり，可能な限り知識の抽出を自動化することが要請される．

〔3〕 監視・制御型

システムの状態の変化を監視し，意図どおりに動作するよう制御するのが，このタイプに当たる．プログラミングの場面においては，人間が行う記述の作成・編集過程を監視の対象とすることが考えられる．これは，一般的に人間が機械を操作する過程に当てはまるものである．具体的には，プログラム編集時の対話的な操作に対する支援などが考えられる．

また，プログラムが，オンライン実時間制御のためのものであれば，当然，診断の項で述べた動作のテストが，この種のタスクを含むことになる．

〔4〕 計画・設計型

プログラミングを，機能記述つまり要求仕様から，構造記述つまりプログラムへの変換であると考えれば，まさにこれは設計問題の１つである．過去のプログラミングを通して蓄積されたサブルーチン・ライブラリは，プログラムの部品カタログを提供する．また，プログラム例やマクロは，定型的なプログラム形式に関する構成のためのガイドとなる．

一般に設計問題では，要求されたシステムの特性を実現するために，構成要素つまり個々の部品の構造と，システムとしての全体的構造を決定しなければならず，これらの決定の困難さによって，問題が特徴づけられる．診断の項で述べたように，プログラムは，個々の製品によって性質がかなり異なる．あるものは，要求仕様の段階ですでに全体的構造が明確に与えられるかもしれない．あるものは，あらかじめ用意された少数の部品の組合せで実現可能かもしれない．また，あるものは，過去の類似のプログラムからの類推が有効に使えるかもしれない．

機械語の命令セットを部品と考えれば，すべての利用可能な部品は既知である，とも言えるが，よほど小さなプログラムでない限り部品としては細かすぎる．大きなシステムでは，構成要素に関する何段階もの部分全体階層を考える必要があろう．

プログラムに共通な性質といえば，機械設計における形状寸法のような数値的なパラメータは存在せず，すべて離散的であるという点である．つまり，構成要素の構造決定に際して，数理計画あるいは統計的な手法が役に立つような場面はほとんどない．

統計的な情報を利用できる箇所としては，入出力データあるいは手続きの実行頻度に関する統計的な性質を用いた手続きの最適化の問題であろう．実時間システムなどの実行速度に対して厳しい制約が課せられる領域では，このような手法は重要である．

3．人工知能研究の手段としてのプログラミング

人工知能研究者の少なくとも半分以上は，プログラマである．もちろん，知能を実現するための構成要素はプログラムだけではないが，主要な部分を占めていることは確かである．現時点のコンピュータ・ハードウェアは，多少の不満はあるものの，それなりに柔軟性のある環境を提供してくれる．そのお陰で，多くの人工知能研究者は，ハードウェアの設計に手をつけずとも，知能のサブシステムと考えられるものをプログラミングによって実現し，実証することができる．

3・1　人工知能研究とプログラミング

ある意味では，人工知能の研究は，それ自体が壮大なプログラミング・プロジェクトである．要求仕様は，「知能を実現すること」である．知能とは何かということを直接的に記述することは困難である．たとえ記述したとしても，それが多くの人に受け入れられるかどうかはさらに困難である．

現実にとられている接近法は，特徴的な知的活動の例を１つずつ征服していくことである．人間と対戦するゲームや，パズルを解くこと，自然言語の理解，自然言語での人間との対話，また，複雑な画像や，音声などに対するパターン認識などが，そのような具体例として試みられてきた．例題を通して発想をさらに広げることによって，本来の要求仕様が徐々に具体化されていくはずである．

もちろん，人工知能を狭い意味での科学としてとらえるならば，現実に動作する機械として知能を実現する必要はなく，知能のメカニズムに対する理論的なモデルが提示され，多くの人を納得させられればそれで十分とも言えるが，理論を実証する根拠として，さらに，工学的応用の可能性を示す根拠として，最終的に動くシステムを作り上げることは研究戦略上きわめて重要である．

3・2　人工知能が産み出したプログラミング・パラダイム

人工知能の研究では，ほんのおもちゃのようなプログラムを作るにも，人間に都合の良い表現言語，表現方式が強く要請される．というのは，ほとんどの人工知能モデルが人間を手本としているからである．すなわち，人間が実際に行っている知的活動にかかわる，あらゆる情報をより自然に表現する手段が必要なのである．このような要請から，人工知能研究は，プログラミングのためのさまざまな規範を提出してきた．そのような規範，すなわちパラダイムは，知能に対する認識論を考えるうえでもきわめて重要である．

以下では，伝統的な枠組みである手続き指向も含め，いくつかの代表的なパラダイム[3]について考察する．

〔1〕 手 続 き 指 向

現在，最も一般的となったノイマン型の計算機構では，プログラムとして，データ操作の手順によって動作の指示を与えるという形式が最も自然である．

この方式では，「何を解くか」ではなく「どのようにして解くか」を記述することになるので，内部のメカニズムについての知見に乏しく，入出力の現象のみがわかっているような要求の記述には不向きである．

手続き指向の特徴は，動作の時間的順序づけを容易に記述できる点であろう．非決定性や複雑な条件判断を含まないような動作系列の記述に適している．

〔2〕 対 象 指 向

知識を「もの」中心にとらえる考え方である．プログラムは，概念範疇の階層構造に従った概念定義の集合である．概念そのもの，および概念の具体化されたものが「対象」である．個々の対象は，状態と動作の定義を持つことができる．動作は，ある対象にメッセージを送るという操作の組合せによって実現される．メッセージ自身も対象である．概念の中にはリストや配列などのデータ構造も含まれ，さまざまな構造のデータを扱うことも容易である．

このような枠組みは，モジュール性と構造化記述に優れており，要素と全体の双方の構造を明確に記述することができる．さまざまな構成要素からなる世界の表現に適しており，複雑な事象のシミュレーションなどには，たいへん都合が良い．

〔3〕 事象・アクセス指向

「もの」ではなく「こと」中心にとらえる考え方である．何かが起こったとき，それに連動して何をすべきかを記述する．事象を起こす側の記述と，事象によって起動される手続きの記述は完全に分離され，動作の監視に基づく柔軟な制御機構を実現できる．コンピュータの動作では，メモリのアクセス，すなわち変数の値の参照・書き換え，および外部装置との入出力がこれに相当する．オンライン実時間システムのように入出力に同期した処理を行う必要のある場合，あるいはデバッグのためのプログラム動作の監視のように，監視される側のプログラムを変更すべきでない場合には，この枠組みが適している．

〔4〕 ルール指向

手続き指向では，システムの全体的な構造を記述する必要があるが，知能のように内部のメカニズムが明らかでない場合には，そのような全体的構造を決定することは困難である．ただし，断片的な手続きあるいは事象の動きについてはある程度推測が可能である．このような断片的な知識の集積によって，全体を構成することを可能にしたのが，プロダクション・システムによる記述の枠組みである．個々の断片的知識は，条件部と行動部の対からなるルールで表現される．

当然ながら，大規模なルール集合では，全体の構造が見えにくく，ルール間の相互作用によるバグが生じやすくなる．

〔5〕 論理・関係指向

この枠組みでは，要求は「この命題は真か？」あるいは「この命題を満足する具体例は何か？」という形式になる．「どのように計算するか」ではなく「何を得たいか」を表現できるという意味で，より直接的に要求を記述できる．ただし，どのように計算すれば，それが得られるかという情報を陽に記述するのは難しく，巧妙な制御構造を組み込むには適さない．

すなわち，この枠組みは，機械よりも人間側の要求記述に都合のよいものである．よって，要求を書き出すためには適しているが，コンピュータ上で動くコードへの変換に困難が生じる場合がある．

〔6〕 複数パラダイムの統合的プログラミング環境

知的活動は，実にさまざまな側面を持っており，単一のプログラミング・パラダイムで記述するのは好ましくない．構成要素のおのおのの様相に従ってふさわしいパラダイムに則るべきであり，全体としては複数のパラダイムを統合した表現を採用すべきであろう．

ここで問題となるのは，表現の均質性が犠牲になることである．さらに，このことからパラダイムごとの記述法を習得する，わずらわしさも派生してくる．もちろん，どのパラダイムであっても，書こうと思えばどのようなプログラムでも書けないことはないが，記述の容易さ，変更の簡便さ，読みやすさ，さらに，発想の種としての有効性などを考えることも重要である．

論理プログラミングはProlog，対象指向はSmalltalkというように，それぞれのパラダイムに従った記述を中心に設計されたプログラミング言語は存在するが，複数パラダイムの統合は，どのような形態で実現されるべきであろうか．ある言語を用いて，その中心パラダイムとは異なる別のパラダイムに基づいて，プログラミングを行うための方法論を示すというだけでは，言語自身がプログラマをそのパラダイムの発想に誘導する契機となることはできない．

2種類のパラダイム，特に対象指向と何かという組合せは数多く試みられているが，多数のパラダイムの統合の試みとしてはLOOPS[4]が代表的である．LOOPSは，Lispマシン上に構築されたプログラミングシステムであり，上で述べた5つのパラダイムのうち論理指向を除く4つを統合したものである．基本的にはInterlisp-Dの環境の中に埋め込まれており，各パラダイムのプログラミングを実現するLisp関数の使用によって，異なるパラダイム間のプログラム上での結合を実現している．

以下，具体的なLOOPSの統合方式について紹介する．

対象指向では各クラス中のメソッドは，Lisp関数として定義される．すなわち，メソッドとしてLispプログラムによる手続きを記述できる．また，クラス変数，インスタンス変数のアクセス，およびメッセージ通信のための関数やマクロが用意されており，メソッドの定義の中での使用と同時に，Lispプログラムの中からオブジェクトを起動することもできる．

アクセス指向は，アクティブバリューと呼ばれる機能によって実現されている．アクティブバリューは，値の格納場所，読み出し関数，書き込み関数の3つの組からなっており，この値をアクセスする時点でこれらの関数が起動される．クラス変数あるいはインスタンス変数の値としてアクティブバリューを設定しておくことにより，対象指向の環境の中でアクセス指向のプログラミングを実現できる．

LOOPSには，ルールセットと呼ばれるクラスが組み込まれており，そのインスタンスとしてルール集合を実現する．ワーキングメモリは，任意のクラスのインスタンスである．このときインスタンス変数がワーキングメモリの要素となる．ルールの実行制御は前向きのみであり，競合解消戦略としては，単純な4つの方式のうちの1つを設定することができる．ルールセットの実行は，ルールセットとワーキングメモリを引数とする関数の評価によって起動される．すなわち，メソッドの中からでも，アクティブバリューの中からでも起動できる．また，各ルールの左辺および右辺の要素は，LispのEVALによって評価されるので，個々のルールの中からオブジェクトへのメッセージ通信や，別のルールセットの起動を行うこともできる．

このように，LOOPSでは，4つのパラダイムに基づくそれぞれのプログラムから，それとは異なるパラダイムに基づく任意のプログラムを起動することが可能であり，要求仕様のさまざまな側面に応じたパラダイムを選択し，それらのプログラムを統合化した形で

記述することができる.

〔7〕 **その他のパラダイム**

上に挙げた5つのパラダイム以外に，イメージ情報や，神経回路モデルに基づくプログラミング・パラダイムも考えられ得る．たとえば，コネクショニスト・モデル[5]と呼ばれる計算機構に基づくプログラミングである．知能の中でも低レベルの連想や認識の部分は，このようなパラダイムに基づくべきかもしれない．ただし，これらと他のパラダイムとの統合の方式については，困難な局面が少なくない．というのは，これらの方式がLOOPSで統合のベースとなったLispなどとは対象的に，本質的に分散的かつアナログ的であるためである．この問題は，今後の人工知能研究において，重要な課題の1つと思われる．

4. おわりに

以上2つの視点から人工知能とプログラミングの接点について述べてきたが，プログラミング教育におけるICAIや，人工知能研究の副産物であるプログラミング環境，ユーザ・インタフェースやデバッグ・ツールについてもふれるべきだったかもしれない．

いずれにせよ，人間の要求を自然な形で引き出し，コンピュータ上に記述するための技術は，人工知能の得意とするところである．自動支援システムを構築するにせよ，表現の前提となる認識論を論ずるにせよ，人工知能が提示してきたさまざまな概念や手法は，プログラミングの方法論を整理，発展させるに十分有益であろう．

また，人工知能研究の側からすれば，知能を記述するプログラミングの方法論は，遠大なる目標に向かうにはまだまだ不十分である．個別の知的システムの研究とともに，既存の知識表現研究の成果にとらわれず，思い描くままに表現するための方法論について，思慮をめぐらすことも必要ではなかろうか．

◇参考文献◇

(1) Barr, A. and Feigenbaum, E. A. (eds.) : The Handbook of Artificial Intelligence Vol. II, Chapter X, William Kaufmann (1982) (田中幸吉，淵　一博監訳：人工知能ハンドブック第II巻，第X章，共立出版 (1983)).

(2) 小林重信：知識工学，昭晃堂 (1986).

(3) Bobrow, D. G. : If Prolog is the Answer, What is the Question?, Proc. of the Int. Conf. on Fifth Generation Computer Systems, pp. 138-145, ICOT, Tokyo (1984).

(4) Bobrow, D. G. and Stefik, M. : The LOOPS Manual, Xerox PARC (1983) (日本語版：LOOPS マニュアル，富士ゼロックス (1987)).

(5) Feldman, J. E and Waltz, D. L. (eds.) : Special Issue : Connectionist models and their applications introduction, Cognitive Science, Vol. 9, No. 1, (1985).

著者紹介

臿見　達夫（正会員）

昭和53年東京工業大学工学部制御工学科卒業．昭和55年同大学院総合理工学システム科学修士課程修了．昭和56年同博士後期課程中退，同年より同助手，現在に至る．n進木 LINGOL，MELCOM-COSMO上のLISP 1.9，分散型プロダクションシステム，マルチウインドウシステムなどの開発，および学習モデルの研究に従事．情報処理学会，電子情報通信学会，日本認知科学会，日本ソフトウェア科学会会員．