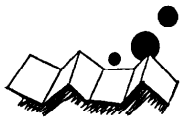
解　説

2. 方　式



2.5 部品合成による自動プログラミング†

古宮誠一††　原田　実†††

1. はじめに

人間向きの高度な仕様記述から，そこに記述されたユーザの要求を満足するプログラムを自動的に作り出す技術を自動プログラミング (automatic programming) という．類似の言葉にプログラム自動生成 (automatic program generation) とプログラム自動合成 (automatic program synthesis) があり，これらの言葉は若干異なった響きをもって使われている．自動生成という言葉は，主にソフトウェア工学の分野で使われ，なんらかの仕掛があり，所望のプログラムが，人間の指導性を生かして半自動的に作り出されるというイメージがある．これに対して，自動合成という言葉には，何も仕掛のない（と思われる）ところから所望のプログラムが自動的に作り出されるというイメージがあり，主に人工知能の分野で使われる．

自動合成などによって得られるプログラムを目標プログラム (target program) といい，目標プログラムの記述言語を目標言語 (target language) という．目標プログラムの満たすべき要件や作り方をまとめたものを要求仕様 (requirements specification) といい，要求仕様を与えることを要求定義 (requirement definition) という．ここで重要なことは，ユーザが要求定義のために与える情報は，原則として「プログラムをいかに動作させるか（＝how)」ではなく，「プログラムに何をさせるか（＝what)」でなければならないということである．なぜなら，要求定義のために how の情報を与えることは，ユーザが自分でプログラムするのと本質的には等価だからである．このため，要求定義のために how の情報を与えるシステムを特に「自動化環境でのプログラミング」と呼び，自動合成システムとは区別することもある．

これに対して，自動プログラミングという言葉は，プログラム自動合成という言葉よりも意味が広く，非手続き型言語や抽象データ型言語によるプログラミングも含まれる．そして，ときには自動化環境でのプログラミングをも含めて考えることもある．

自動プログラミングの方法は，本特集号の分類によれば①知識処理的手法による方法，②演繹推論による方法，③帰納推論による方法，④プログラム変換による方法，⑤部品合成による方法，⑥実行可能な仕様記述による方法，の6つに分類できる．本稿では，部品合成による自動プログラミングシステムを実現する上での考え方を明らかにするとともに，実現のための要素技術を分析し分類して解説することにより，研究の現状と動向を明らかにする．このため，本稿で事例として掲げた個々のシステムの全体像について，分かりにくいものがあるかも知れない．その場合には，稿末に掲げた参考文献や本特集号の解説「事務処理分野における自動プログラミング」などを参照されたい．

2. 部品合成による自動プログラミングの考え方

2.1 プログラムの生成過程

複雑な問題を単純な問題の集まりに分解して捉え，後でこれらを組み合わせていくのが複雑な問題に対する常套手段である．この考え方は自動プログラミングにも適用できる．すなわち，ユーザの要求仕様を単純な要求仕様の集まりに分解して捉え，個々の要求仕様を満足するプログラムを生成してから，後でこれらを組み合わせていくというものである．この考えを指向したものが部品合成による方法であり，この方法は図-1 に示すように，

（a） 与えられた要求仕様を理解して，単純な要求

† Automatic Programming by Fabrication of Reusable Program Components by Seiichi KOMIYA (Information-technology Promotion Agency, Japan) and Minoru Harada (Central Research Institute of Electric Power Industry).

†† 情報処理振興事業協会（略称 IPA）技術センター
††† (財)電力中央研究所知識処理研究室

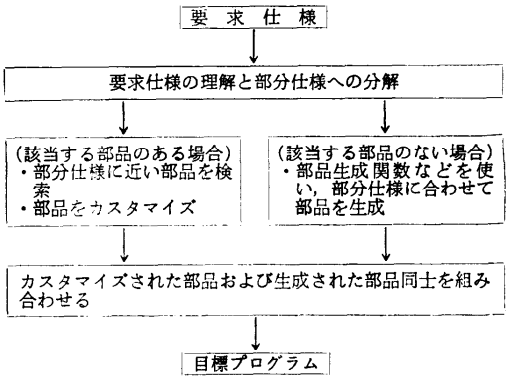


図-1 部品合成による自動プログラミングにおけるプログラムの生成過程

仕様の集まりとなるように分解する過程

(b) 分解後における個々の要求仕様に近い仕様の部品の有無により，次のいずれかを行う過程

- 該当する部品のある場合には，部品を検索して個々の要求仕様に合うようにカスタマイズする．
- 該当する部品のない場合には，部品の生成知識を用いて，個々の要求仕様に合うような部品を生成する．

(c) 生成ならびにカスタマイズされた部品同士を組み合わせていく過程

の3つに分けて考えることができる．そして，部品合成による自動プログラミングの実現は，この3つの過程を自動化することにより達成される．なお，部品合成による方法は，参照の透過性*を必要としないので，手続き型言語のようなこの性質をもたない言語で記述されたプログラムの合成に適した方法でもある．

2.2 分解後の仕様に合わせて部品を生成する方法

上記のように分解され単純化された要求仕様でも，これを満足するプログラムを何も仕掛のない（と思われる）ところから自動的に生成することは容易ではない．そこで，次に示す方式のいずれかを採るのが普通である．

(1) 関数などを用いて部品を生成する方法

分解後の仕様に合わせて部品（ソースコード）を生成するのに，部品化されたソースコードを使用するのではなく，部品生成のための知識を使う方式である．

* 1回の実行の間に同じ変数に異なる値を代入することが禁じられている場合，プログラムは参照の透過性 (referential transparency) があるといい，この変数をもったプログラムは，プログラム変換の際に変数を定数のように扱うことができるので，変数の値による場合分けをすることなく，字面だけで変換の正当性を検証できる．このことがプログラム変換を容易にしている．

部品生成のための知識は，関数やサブルーチンの形をしており，この場合の要求仕様は，関数やサブルーチンに渡される引数が対応する．この方法は，要求仕様を十分に細分化すれば，部品をもたなくても関数やサブルーチンにより，要求仕様に合わせてソースコードを生成できるという事実を拠所にしている．

(2) コード化された部品の検索・修正による方法

単純化された要求仕様に対応するソースコードをあらかじめ部品として登録しておき，ユーザの要求に合う部品を検索して，これを組み合わせていくのが実用的である．そして，用意された部品が汎用的で，かつ，ユーザのあらゆる要求に対応する部品が網羅されていればさらに強力である．ところが，プログラムというものは，ある状況を想定しそこで使われる機能を具体的に記述するものであり，この具体性ゆえに意味をもつものだから，プログラムまたはその一部をそのまま部品として切り出すだけでは，部品に汎用性をもたせることはできない．したがって，これをそのまま再利用しようとすれば，再利用のための工数がかかり過ぎて使いものにならない．そこで，プログラム部品に汎用性をもたせるために，

(a) 部品用に切り出されたソースコードから，特定の状況を想定した部分の記述を汎化したものを部品として作成する（部品の抽象化）段階と，

(b) ユーザの要求に合うように，必要なソースコードを(a)に追加（部品のカスタマイズ）して部品を利用する段階に分ける．

ここで(a)のような部品から(b)を実現することは柔軟性（汎用性の一種）ある部品の実現にほかならない．そして，(a)のようにして作られた部品を一般にプログラムスケルトン (program skeleton) と呼んでいる．

2.3 システムの構築方式

自動プログラミングシステムの構築方式は**図-2**に示すように2つある．コンパイラ方式は，システムが生成するすべてのプログラムに共通なプログラムのモデル（これをプログラムモデルと呼ぶ）を想定し，対応するプログラムの生成に必要な部品生成のための関数群またはサブルーチン群（以後，これを部品生成関数群と呼ぶ）と，その起動を制御する機構（この機構はプログラムモデルに対応して作成される）を用意しておき，要求仕様に合わせ，必要な部品生成関数群をプログラムモデルに従って起動することにより，求めるプログラムを得るという方式である．この方法は，

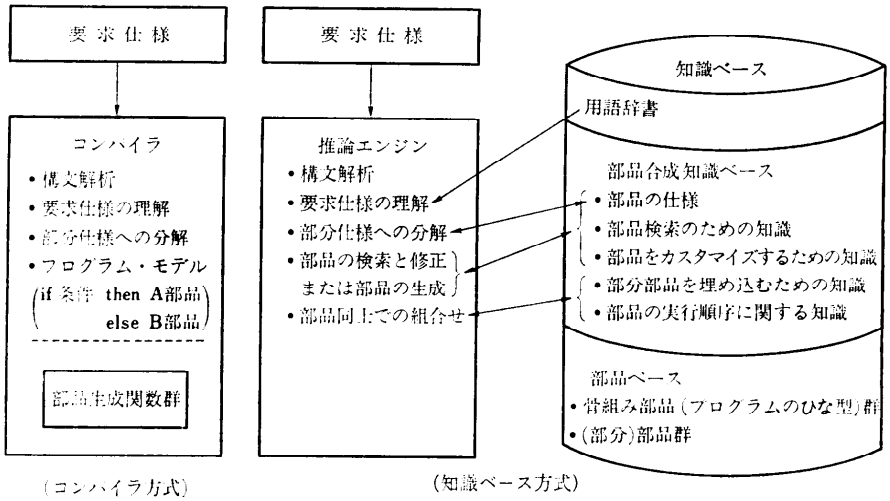


図-2 部品合成による自動プログラミング・システムにおける2つの構築方式

部品の代わりに，部品生成のための知識と，部品を利用するための知識がコンパイラとして一体になっているところに特長がある．

知識ベース方式は，システムが生成するプログラムのパターン（レコードを処理単位とする入出力処理の制御ロジックのパターン）ごとにプログラムのひな型と部品群を用意しておき，プログラムのひな型と部品群を要求仕様に合うように組み合わせることによって求めるプログラムを生成する方式である．この方式は，プログラムのひな型と部品は部品ベースとして，プログラム合成のための知識は知識ベースとして用意され，推論エンジンの外にもっているところに特長がある．

3. 要求仕様の与え方と理解の仕方

「システムが要求を理解するということは，次の知的な処理のために必要な情報を抽出し，処理しやすいように，構造をもった形式性の高い表現にまとめることである」ということができる．このため，部品合成による自動プログラミングにおいて，要求仕様として与えなければならない情報は，①使用する部品を指示または暗示するもの，②どのような部品を生成するか，または検索した部品をユーザの要求にどのように適合（カスタマイズ）させるかを指示するもの，③適合させた部品をどのように組み立てるかを指示するもの，の3つである．

すでに提案されている，自動プログラミングにおける要求仕様の与え方は，①形式的な仕様記述による方法，②例題記述による方法，③（制限された）自然言語による方法，④（階層化された）メニュの利用による方法，の4つに分類できる．このうち，②の方法は，入出力対やアルゴリズムの例をユーザが自分で考え作成して与えるもので，「帰納推論による方法」専用の要求仕様記述法と考えてよい．したがって，部品合成による方法で使用されるのは，①③④の3つの方法である．

3.1 形式的な仕様記述によるもの

部品合成による方法における形式的な仕様記述は，使用する部品の選択と組合せを指定するもので，その代表例をいくつか紹介する．第1は，述語論理や様相論理に基づく形式論理によって部品のカスタマイズと適切な組合せを導くものである．典型的な例としては，大阪府立大学の西田と藤田らによる SAPRE[12] がある．第2は，どのような部品を生成して，どのように組み合わせるかをテーブル論理を使って指示することにより要求を仕様化するものである．典型的な例としては，電力中研の原田らによる SPACE[3),4)] や富士通の BAGLES[19] がある．第3は，部品の使用方法を部品名とパラメータで指示するもので，いわゆる，第4世代言語（4GL）とか超高水準言語（VHLL：Very High Level Language）と呼ばれるものがこれに当たる．DFCOBOL[19]（三井造船），HYPER COBOL[10]（富士通），IDL[10]（日本電気），EASYTREIVE[10]（Pansophic System 社），JASPOL[10]，JASMAC[11]（以上，日本システムサイエンス社）などの例がある．

ユーザ：関連分野の名前を入力
↓
システム：関連するモジュールを列挙する．
↓
ユーザ：必要なモジュールを選び出し，モジュールの手続き的表現を処理の順に並べる．
↓
ユーザ：隙間があれば，入出力表現やプログラム表現を用いて隙間を埋める．
↓
システム：モジュールの手順名のチェックや因果関係のチェックを行い，ユーザに不完全な部分の修正を要求する．仕様の中の入出力表現と適合する入出力表現をもつモジュールがあれば，その手続き的表現で置き換える．
↓
ユーザ指定の言語によるプログラム

図-3 SAPRE におけるプログラム生成過程

（1） 形式論理に基づく仕様記述

SAPRE は，登録済みの階層的なライブラリモジュールと呼ばれる部品群を使って，仕様からプログラムを半自動的に生成するシステムである．ライブラリモジュールは，従来のサブルーチンを一般化し統合化したもので，見出し部(手続き的表現＋入出力表現)，ENTITYTYPE 部，OP 部の3つから構成されている．SAPRE のプログラム生成過程は **図-3** のとおりである．

SAPRE では，要求仕様の記述方法として，手続き的表現，入出力表現，プログラム表現の3種類を適宜に使い分けている．

（a） 手続き的表現は次の形をしている．

処理名(格名：ターム，…，格名：ターム)

ターム：変数，配列，集合，関数，論理式など

処理名：タームを処理する手順名

格名：OBJect, SOurce, CONDition, GOAL, KEY, PARTICipant など

（b） 入出力表現は次の形をしている．

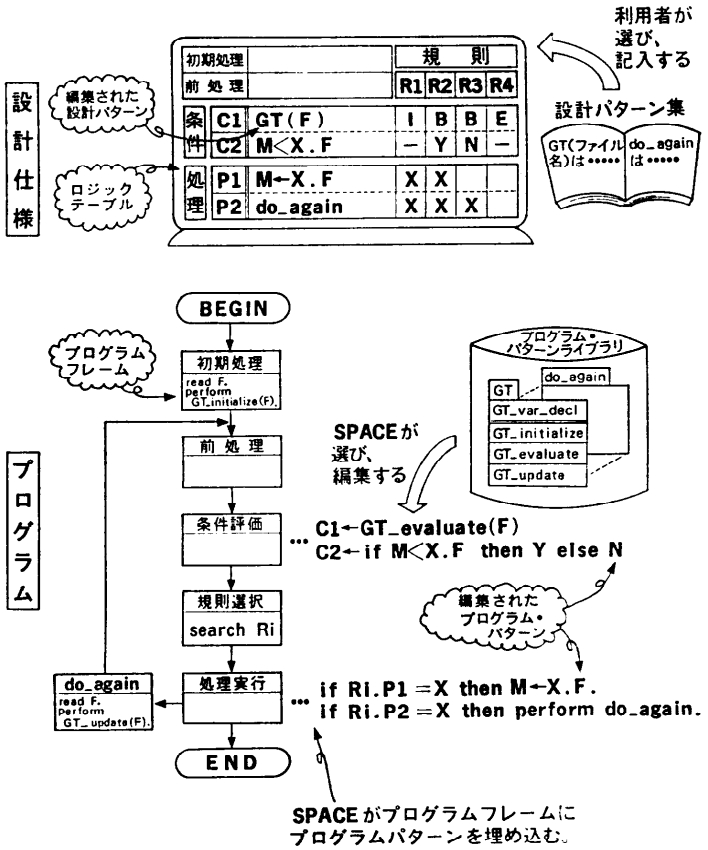
(IN : $P(x)$, OUT : $Q(x,z)$)

ここで，$P(x)$, $Q(x,z)$ は，通常，連言標準形か $\bigwedge_i(\bigwedge_j P_{ij}\rightarrow Q_i)$ の形をとるものとする．

（c） プログラム表現は，演算制御をプログラムレベルで指示するために目標言語の制御ステートメントなどで直接表現するか，または一般的なプログラム表現のステートメントで表現する方法である．

（2） テーブル論理に基づく仕様記述

SPACE では，ディスプレイ上に表示された設計画面に，設計パターンと呼ばれる仕様要素を組み込むことにより，要求仕様を与える．設計パターンには，図形パターンと文パターンとがある．図形パターンは名前や属性を与え，これらを線で結ぶことによりプログ



注) SPACE は『どの設計画面の，どこに，どんな設計パターンがあるか』に従って，適切なプログラムパターンを自動的に検索・編集し，これらをプログラムフレームの適切な所に埋め込んで1本のプログラムを作成する．

図-4 SPACE による自動プログラミング[4]

ラム構造を記述する．一方，文パターンは，4GLやVHLLと同様に文法に従う文であり，事務処理に現れる一群のプログラム制御を1パターンで表せるようになっている．これらを，図-4の上半分に示したロジックテーブル（デシジョンテーブルを改良したもの）と呼ばれる画面に適切に配置することで，プログラムの機能を簡潔に記述することができる．パターンの業務への適合化はパラメータによって行う．これらの設計パターンの使用方法や組合せ方法を2図5表からなる設計画面に埋め込み方式で入力するだけで，COBOLによるプログラムを自動生成できるようになっている．SPACEは，高精細ディスプレイの機能を駆使したビジュアルなユーザインタフェースを提供することにより，形式的な仕様記述がもつ記述しにくいという欠点を克服している．

仕様記述に文法上の誤りや論理的な誤りがある場合には，これをチェックして誤りの個所を表示する機能が必要である．SPACEでは，仕様作成作業を終えた後でチェックメニュを選択すれば，表現方法は正しいか，仕様は完全か（記述に漏れや矛盾がないか）などを検査して誤りの個所を通知するようになっている．

一方，BAGLESは，金融向け総合開発支援ツールで，モジュールの機能を処理，論理，条件からなる表形式の設計書に辞書に登録された業務用語で記述すると，IF-THEN型のプログラム（目標言語はCOBOL）に展開される仕組みになっている．

3.2 （制限された）自然言語で与える方法

（制限された）自然言語を用いて要求仕様を与えるシステムの典型的な例としては，電力中研の原田と篠原らによるARIES/I[2]，富士通の杉山らによるKIPS[20]，Olson Research Associates社のJ.G. Riceらによる ASGS[17]，阪大の上原らによるAutoPro[25]，阪大の今中らによるAPSS[6]などがある．

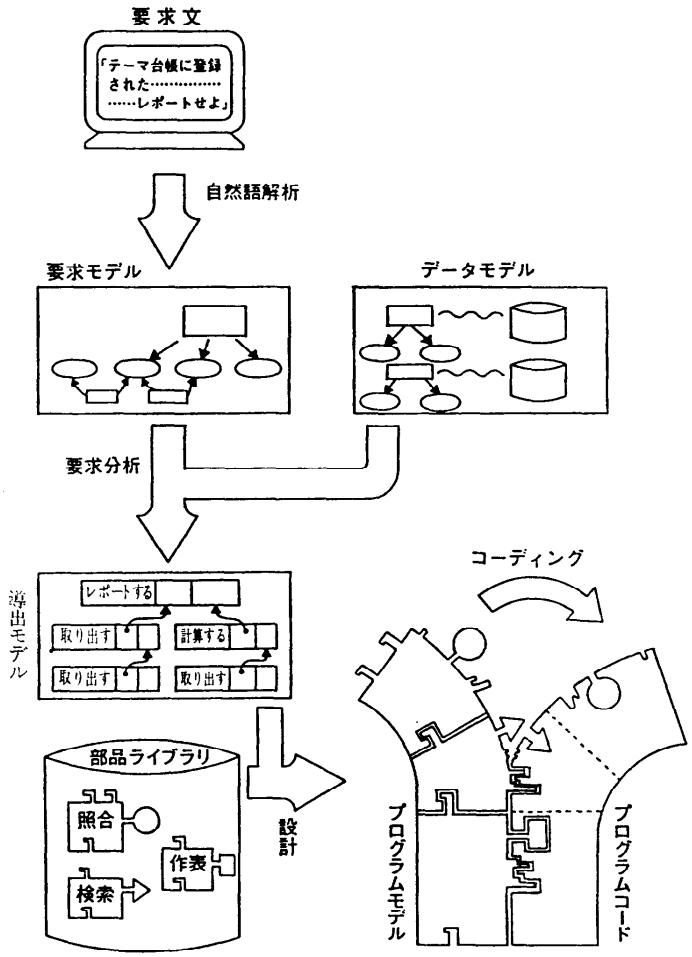


図-5 ARIES/Iにおけるプログラム生成過程[3]

ARIES/Iは，日本語文で入力された要求仕様からファイル処理プログラム（目標言語はHYPER COBOL）を自動生成する自動プログラミングシステムであり，要求に合わせて必要な部品を検索・編集し統合するまでの過程をすべて自動化している．ARIES/Iのプログラム生成過程は図-5のとおりである．

KIPSは，日本語文から情報を抽出して断片的な情報をまとめ上げることにより，HYPER COBOLのプログラムを自動生成する対話型自然言語プログラミングシステムである．

ASGSは，要求定義言語RSLで記述された自然言語に近い表現の要求仕様から，COBOLまたはFORTRANで記述された事務処理プログラムを自動生成するシステムである．ASGSでのプログラム生成

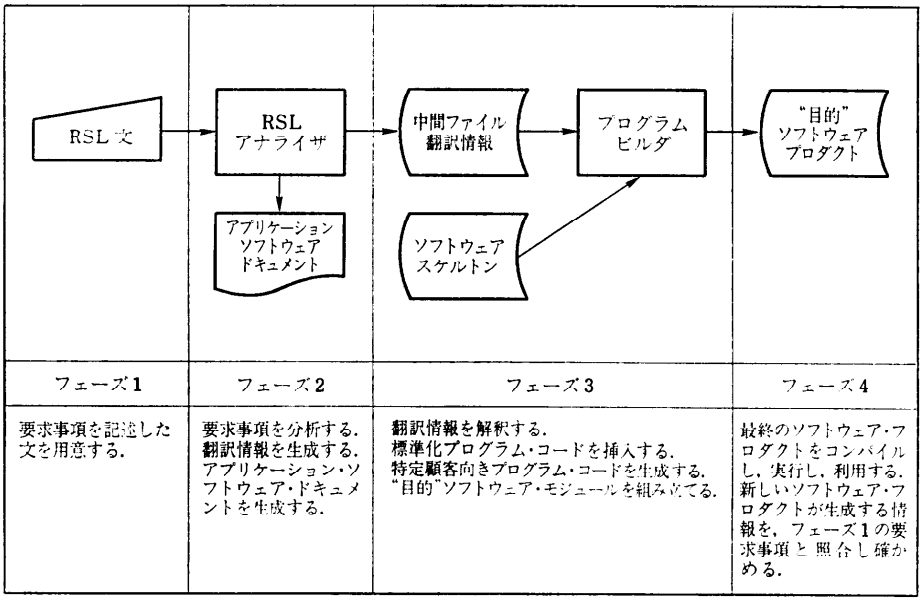


図-6 ASGS における4つのフェーズ[17]

過程は，図-6のような4つのフェーズで構成されており，その自動化は，要求仕様→内部表現，内部表現＋プログラムスケルトン→目標プログラムという2つの変換を自動化することにより実現されている．

AutoPro や APSS は，ゴールとプランの概念に基づいて，自然言語で記述された要求仕様からゴールを推論し，ゴールからプランを，プランからよりプリミティブなプランを，といった段階的詳細化によりプログラムを合成するシステムである．AutoPro では，要求仕様の中にプログラムの制御に関する記述を含まないものと仮定しているので，プログラム自身が制御構造をもたなければならない手続き型言語によるプログラムの生成は困難である．一方，Prolog は後戻り機構などの高度な制御構造を言語の処理系がもっているので，プログラム自身が制御構造をもつ必要がない．このため，Prolog を目標言語にしている．また，APSS では，同様の理由でデータベース操作言語を目標言語としている．

ところで，自然言語で何の制約もなく記述された要求仕様をシステムが正しく理解するのは不可能に近い．そこで，要求仕様を自然言語で記述する場合，記述分野，記述内容，構文，記述順序などに関する制約を設けるのが普通である．

自然言語で記述された要求仕様は，記述内容の詳細さにおいて目標プログラムとの差が大きいので，言葉の変換だけではプログラムの合成は不可能であり，さまざまな情報の補充が必要である．また，日本語のもつ曖昧さを避けるために，要求仕様の入力がシステムとの対話で行われるのが普通である．実際，理解を確実にするために入力の単位を1センテンスずつに制限し，その中で意味の曖昧な点や矛盾点があれば，システムから質問を出してこれを正すという方法が多く用いられている．さらに，ユーザとの対話を自然なものとするためには，システムが理解した内容をプログラム図などの分かりやすい形式でユーザに提示してやる必要があるとして，ユーザとの対話をこの上で可能にしているシステムも少なくない．KIPS はこの例である．

自然言語で記述された要求仕様をシステムが理解するということは，次に行う知的な処理（＝部品合成のための種々な処理）で必要となる情報を抽出して，後の処理がやりやすいように，構造をもった形式性豊かな表現にまとめることである．そのためには，対象領域の概念をモデル化して，対象領域における用語や述語の意味を記述することが必要である．たとえば，KIPS は，対象世界のモデル☆として，意味モデル（se-

☆ ここでいう「対象」とは，オブジェクト指向でいう広義の対象 (object) ではなく，知識ベースシステムにおける問題解決の対象としての対象物を指している．このとき，この表現モデルを対象モデル (object model) と呼ぶ．ただし，対象物そのものだけに限定せず，それが存在する世界のモデル (world model) をも含めて考えることも多く，両者を区別できないこともある．そこで，対象世界のモデルという言葉を用いた．したがって対象モデルおよび対象世界のモデルは深層の意味表現のモデルと理解してよい．

mantic model), プログラムモデル (program model), コードモデル (code model) の3つをもっている. 意味モデルは単語や文の意味を, プログラムモデルは, ユーザが作り出そうとしているプログラムを, コードモデルは, 自動生成するプログラムを, それぞれモデル化したものである. この中で, 要求理解のために使われるのは意味モデルであり, プログラムモデルの完成をもって要求理解の処理が終了する. また, ARIES/I では, 図-5 に示したように, 要求理解のためのモデルとして述語ネット (要求モデルと呼んでいる) が使われ, データモデルとの照合による, 要求モデルの完成をもって要求理解の処理は終了する.

モデル化のための知識表現としては, フレーム型知識表現の上に意味ネットワーク的な表現を可能にするためのスロットを追加したもの (述語ネット, 格フレーム, 概念依存表現など) が多く使われている. たとえば, ARIES/I では, 対象世界のモデルとして, フレームに述語の意味をスロットとして追加したものを採用している. そして, モデルを記述するための言語として Prolog を採用している. また, KIPS では, モデルの動的な知識を表現する手段としてオブジェクト指向モデルにデータ指向プログラミングの機能を追加したものを, 静的な構造を表現する手段としてクラスとインスタンスおよび上位・下位概念の考え方を採用している. そして, モデルを記述するための言語として, フレーム型知識表現言語 FRL にメッセージ転送関数を追加したもの (わかりやすく言えば, デモンとメッセージ転送が使えるフレーム型知識表現言語) を採用している.

これらのモデルを解釈するための知識については, ルールやアルゴリズムとしてモデルと独立に作成する方法と, モデル中に現れるオブジェクト (またはフレーム) のメソッド (またはデモン) としてもたせる方法の2つがある. ARIES/I は前者であり, KIPS は後者でメソッドとデモンの両方を採用している.

3.3 (階層化された) メニュの利用による方法

一般に, システムが合成できるプログラムの範囲は, システムごとに決まっているのだから, 要求定義の範囲を規定しても不都合はない. この考えを押し進めたものがメニュ利用による方法だということができる. 自然言語入力と比較した場合の, メニュ利用による方法の利点は次のとおりである.

(a) 自然言語入力における構文解析の処理が不要.

(b) 同義語を用意しなくてもよい.

(c) 対話の主導権がシステム側にあるので, 要求に対する誤解はない.

(d) キーボード入力におけるユーザの負担が少ない.

メニュ利用による方法は, プログラムのひな型をカスタマイズするために使われるのが普通であることからも分かるように, 要求のバリエーションが絞れる場合に有効である. したがって, このような場合における要求仕様入力の理想的な方法の1つだと言える. しかし, この方法をより実用的なものにするには, さらに次のような配慮が必要である.

- 要求のバリエーションを絞りにくい部分に対しては, 部品を利用するよりも部品生成知識を使うほうが得策である. このような部分の要求仕様については, デシジョンテーブルを使用するか, オウンコーディングする.
- 要求の段階的指定を可能にするために, メニュの構成を階層的にする. しかし, メニュの階層があまり深くなると煩わしいので, そうならないようにメニュを工夫する.
- メニュの中で使われている用語やアイコンの意味を知るための HELP 機能を用意する.

メニュを利用したシステムの典型的な例としては, 情報処理振興事業協会 (IPA) の古宮らによる PAPS[7] (目標言語は COBOL) などがある.

また, 自動プログラミングシステムではないが, 日本電気の SEA/I[10], 日立の EAGLE[10], キャノンソフトウェアの CANO-AID[10] などの, プログラミングを支援する統合化されたソフトウェア開発環境でも, メニュ利用による方法が主として用いられている.

4. 部品のもち方

自動プログラミングの処理過程において, 目標プログラムの構成要素として利用され, 原型のままか, または形を変えて目標プログラムの一部となるものを「プログラム部品」と呼ぶ. この定義に従えば, プログラム部品は①ソースコードを部品化したもの, ②設計仕様を部品化したもの, ③設計情報を部品化したもの, の3つに分類できる. なお, ここでは, パッケージソフトウェアの改造母体としてのプログラムやソフトウェアツールなどは部品とは呼ばない. なお, 混乱を避けるために, 以後, 単に「部品」と呼んだときには, 一般常識に従って①のみを指すことにする.

（1） ソースコードを部品化したもの

ソースコードを部品化する場合，部品の切り出し方には2つの戦略がある．1つは，部品の切り出しを容易にすることに重点を置くという戦略であり，もう1つは，切り出された部品の組立を容易にすることに重点を置くという戦略である．前者は部品の収集に重点を置いた戦略であり，後者は部品の合成過程に重点を置いた戦略である．したがって，自動プログラミングシステムでは後者の戦略を採るのが普通である．後者の例としては，PAPS などがある．そして，自動プログラミングシステムではないが，SEA/I, EAGLE, CANO-AID なども後者の戦略を採っている．

後者の戦略として具体化されたものは2つある．1つは，骨組み部品（＝プログラムのひな型）とその部分部品（＝合成の過程で骨組み部品の中に埋め込まれて使用される部品）という組で部品を切り出すという方法である．これは，事務計算のバッチ処理については，幾つかの骨組み部品（SEA/I や EAGLE の例では，それぞれ11種，22種）を用意するだけで，プログラムの基本ロジックをほとんどカバーできるということが経験的に分かっている．

もう1つは，データフローの概念に基づき，先行して実行する部品の処理結果が，次に実行する部品の入力データとなるように部品を切り出す方法である．このような部品はユニット部品などと呼ばれ，DF-COBOL, HYPER COBOL (DF-COBOL を改良して製品化したもの)，IDL などのシステムで使用されている．

（2） 設計仕様を部品化したもの

設計仕様レベルでの部品とは，目標言語とは異なる言語で部品の仕様を記述したもので，言わば，仕様部品とでも言うべきものである．ソースコードを部品化したものに比べて，目標言語を固定しない分だけ部品の抽象度が高いということができる．抽象化された部品の仕様の記法としては，次の4つの方法がある．

（a） 抽象化したアルゴリズムとして記述する方法

ソースコードを部品化したものから，ある特定の状況を想定して記述された（特化された仕様の）部分を汎化することにより，抽象化したアルゴリズムを表す汎用的な部品を作ることができる．この方法を用いたシステムの典型的な例としては，ARIES/I や電総研の大石らによるFURNACE[14),15)]がある．FURNACE は，ALGOL 流の言語 Metal で記述された部品を使って，PASCAL で記述されたプログラムを生成するシステムである．ARIES/I は，Prolog で記述された部品を使って，HYPER COBOL で記述されたプログラムを生成するシステムである．

（b） オブジェクト指向モデルとして記述する方法

この方法は，オブジェクト指向プログラミングにおけるオブジェクトを部品とするものであり，目標言語をオブジェクト指向言語とするものと，手続き型言語とするものがある．前者は，オブジェクトのもつ切り出しやすさを拠所にして部品を収集し，部品の検索や再利用を研究の対象にしているものが多い．例としては，東芝の内平らによる MENDEL[22)~24)] や京大の垂水らによる MOMOCLI[13)] などがあるが，MENDEL のみは並列処理プログラムにおける部品結合をも研究の対象にしている．後者は，オブジェクト指向プログラミングの仕組みを利用して，COBOL で記述された事務処理プログラムを生成するシステムで，例としてはKIPSや日本電子計算(株)の大木らによるREUSE-I/II[18)]などがある．

（c） 対象世界のモデルとして記述する方法

ソースコードを部品化したものを利用してプログラムを生成する方式では，部品ごとに部品の意味や構造などをモデル化し，このモデル（＝部品モデル）に従って骨組み部品と部分部品群を組み合わせることにより，ユーザが要求するプログラムを組み立てる方法が有効である．ところが，部品をモデル化しただけでは，生成されるプログラムは骨組み部品の規模を越えることができない．そこで，ユーザの作り出すプログラムをモデル化したもの（＝プログラムモデル）をもち，このモデルに従って必要な骨組み部品を埋め込んでいくことにより，複数の骨組み部品からなるプログラムをも生成できるようにしている．このような方法を用いたシステムの典型的な例としては，PAPS などがある．

なお，事務処理分野では，プログラムを骨組み部品を単位とするジョブステップに分け，ジョブステップごとの処理結果をファイルで繋いでいくという運用方法が普通である．この場合には，プログラムモデルは骨組み部品をモデル化したもので代用できる．

PAPS では，部品の意味や構造などを表現するために，対象世界のモデルとして，デモン機能をもつフレームに PART-OF, PRE-CONDITION, POST-CONDITION などの概念を追加したものを採用している．骨組み部品のモデルと部品合成に必要な知識との関係を 図-7 に示す．ここで，骨組み部品1とか骨組み部

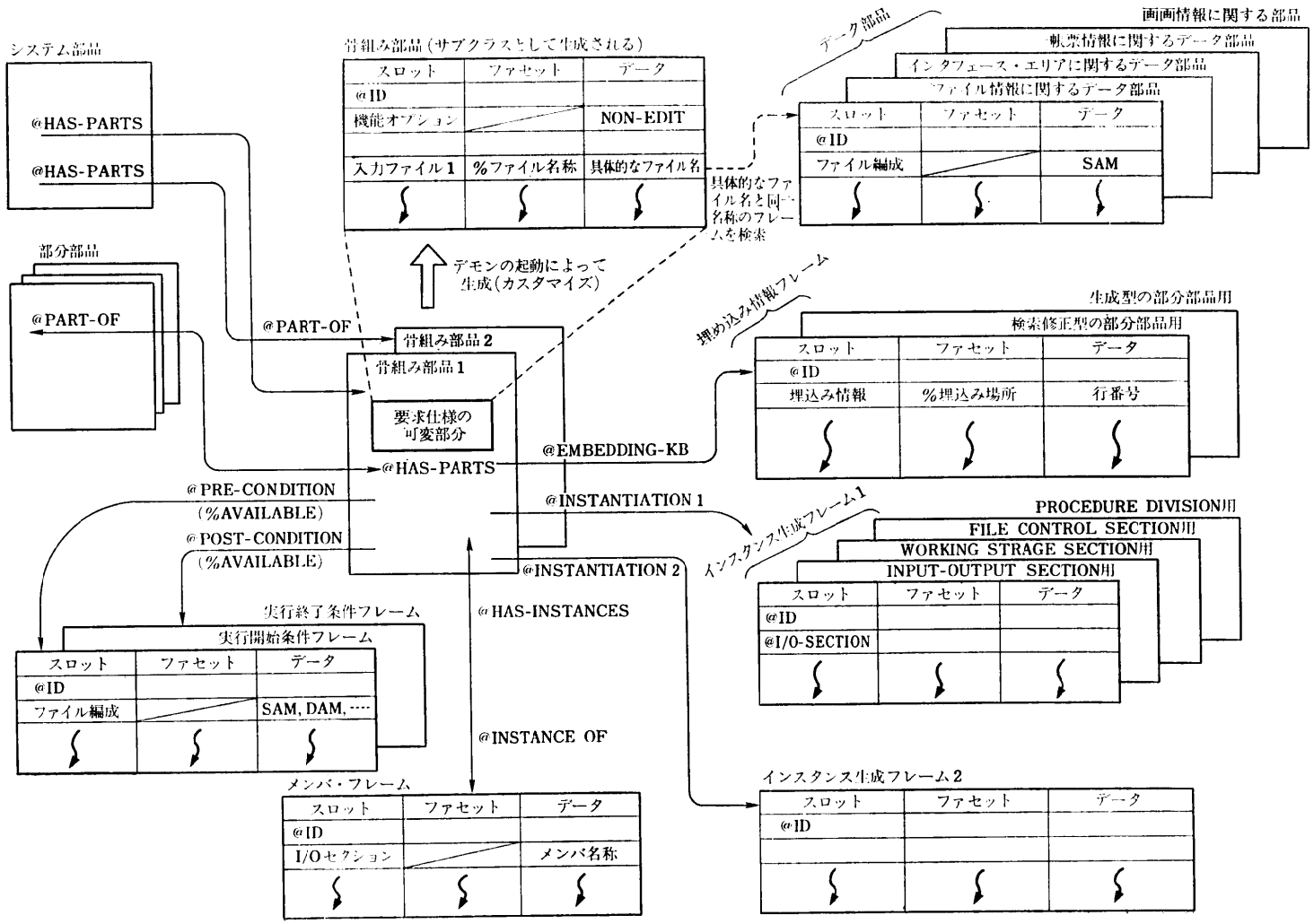


図-7 PAPS における部品表現のモデルと合成に必要な知識との関係[7]

品2とか書かれているのが，骨組み部品を表現したモデルで，@で始まる各種のスロットをもっており，メニュの利用により要求仕様が入力されると，デモンが起動されて特化された仕様がサブクラスとして生成される．ソースコードを部品化したものがあらかじめ用意されており，メンバフレームにより手繰れるようになっている．生成されたサブクラスの情報を基に，このソースコードがカスタマイズされ，必要な部分部品が検索される．このとき，各種のフレームに記述された知識が部品合成のために利用される．

（d） コンパイラにおける意味解釈として記述するもの

SPACE などのように，仕様コンパイラとして構築されるシステムでは，ソースコードを部品化したものはもたないが，要求仕様の仕様要素に対応し，関数やサブルーチンによって生成されるソースコードが部品（以後，これを生成型部品と呼ぶことにする）に相当する．この種のシステムでは，生成型部品のソースコード同士が組合せやすいように関数やサブルーチンの仕様を定めている．

（3） 設計情報を部品化したもの

部品化された設計情報は，自動プログラミングというよりも，設計情報の再利用を支援するシステムで主に使用されている．そのようなシステムの典型はカリフォルニア大学アーバイン分校の J. M. Neighbors らによる Draco[9]と日立の千吉良らによる ICAS-REUSE[1] である．本稿の範囲外になるが，自動プログラミングシステムの技術と共通するところがあるので以下に例を示す．

Draco は，ドメインアナリストとドメインデザイナが作成した分析情報と設計情報の再利用を支援するツールである．分析情報とは，類似した多くのシステムの中から必要性と要求事項を調べ（これをドメイン分析という），その業務（これを domain と呼んでいる）で必要となる対象 (object) と操作 (operation) を記述したものである．設計情報は，ドメイン分析で得られた個々の対象や操作に対する種々の実現方法を明らかにし（これをドメイン設計という），Draco に登録済みの他のドメインの表現を使って記述したものである．ドメイン分析とドメイン設計の結果の記述をドメイン記述といい，①パーサ，②プリティプリンタ，③トランスフォーメイション，④コンポーネント，⑤プロシジャ，の5つの部分から構成されている．①は，ドメインの言語を定義して，ドメインの言語で記述されたプログラムを Draco の内部形式に変換する方法を記述したものである．②は，Draco の内部形式に変換されたプログラムを，読みやすいように変換する方法を記述したものである．③は，ドメインにおける対象や操作の記述を，プログラム変換*により他の表現に書き換えるための規則を記述したものである．④は，ドメインの意味 (semantics) を記述したものであり，ドメイン中の個々の対象と操作について，それぞれ1つずつ存在する．各コンポーネントは，1つ以上のリファインメントを含んでおり，各リファインメントは，そのコンポーネントに対応する対象や操作の意味を，Draco に登録済みの他のドメインの表現で言い替えたものである．⑤は，特殊な分野で使われる翻訳規則などを記述したものである．

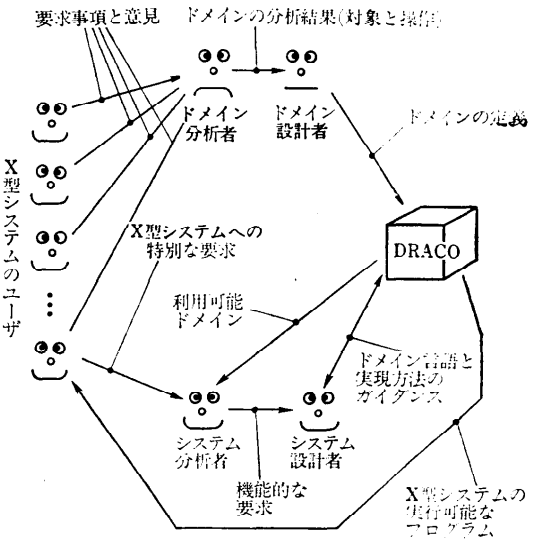


図-8　DRACO の位置付け[9]

Draco における再利用の方式は 図-8 のとおりであり，再利用までのプロセスは次のとおりである．④に書かれたドメインにおける対象や操作を，リファインメントを使って登録済みの他のドメインの表現に書き替える．③を用いて，これをプログラム変換することにより，ドメイン内での最適化を図る．①で Draco の内部形式に変換され，②で読みやすい形でプリティプリントされる．これは，④→③→①→②のプロセス

* プログラム変換 (transformation) とはあらかじめ用意した変換規則に従って，1つのプログラムを同じ意味をもつ別の形式に機械的に書き換えることである．プログラムの記述言語がプログラム変換の前後で変わらないものを狭義のプログラム変換といい，変わるものを広義のプログラム変換という．

により，ラピッドプロトタイピングにも適用できることを意味している．

なお，④のリファインメントは広義のプログラム変換に相当し，③は狭義のプログラム変換に相当するが，これらにおける変換規則の適用は，人間の指導によって行われる．このための煩雑さを避けるために，変換規則の適用範囲を限定する機能 (locale) や意思決定のための一般的な規則を用意しておく機能 (tactics) ももっている．

ICAS-REUSE は，自然言語で記述された曖昧な要求から定型的なシステム仕様書の記述へ変換する過程と，ユーザの要求する機能を含む仕様を既開発システムの仕様の中から抽出して再利用する過程を支援することにより，システムの詳細設計を効率化することを狙ったシステムである．

5. 部品の検索方法

部品合成による自動プログラミングシステムには，部品検索の処理を全く必要としないシステムと，必要な部品を自動的に検索するシステムの二つがある．前者のシステム（たとえば，SPACE や KIPS など）では，ソースコードを部品化したものはもたず，部品はすべて生成するようになっている．後者のシステム（たとえば，ARIES/I や PAPS など）は，要求仕様あるいは設計仕様の中から，部品検索のためのキーワードを自動抽出する機構をもち，これにより必要な部品を自動検索するようになっている．このことは，SEA/I や EAGLE などが部品検索のためにキーワードを与える操作が必須となっているのと対照的である．

（1） 知識ベース利用による検索

検索のための知識ベースをもつシステムである．知識ベースの形態としては，意味ネットワーク，フレーム，プロダクションルールなどが用いられている．

REUSE-I/II では，メッセージ・パターンを基に対応する部品のクラスを検索する．知識ベースには，部品のクラスの呼び出し文，属性のキーワードとその値などを蓄えておき，プロダクションルールを用いて検索するようになっている．

PAPS では，要求仕様からの条件の絞り込みにフレーム（具体的には，図-7 における埋め込み情報フレーム）を使用し，絞り込まれた条件から必要な部品を検索するのにプロダクションルールを用いている．これにより，合成に必要な全部品に対して，要求仕様からの自動検索を実現している．

（2） 部品の体系化とその方法

また，部品の検索を自動化するには，部品を体系化して管理することが必要である．部品の体系化には2とおりあって，1つは，部品の集合全体をなんらかの基準によって体系化する方法である．体系化の基準としてよく用いられるのは，MOMOCLI や東芝の IMAP[6] で用いられているような対象名と操作による分類である．もう1つは，事務計算分野の完全自動を目指すシステムで用いられる方法で，骨組み部品ごとに，そこでの合成に必要となる部分部品を体系化するという方法である．なぜなら，後続する過程では，骨組み部品1つに複数の部分部品を埋め込むことにより，部品組立を行う仕組みになっているからである．このため，骨組み部品の1つを想定することにより，初めて，そこでの合成にどのような部分部品が必要となるかが見えてくるのであり，部品の全体が見えてくることにより，初めてその体系化が可能となるからである．ここでもう1つ大切なことは，将来，部品の追加により現在の分類体系が崩れても，部品検索のメカニズムが変わらないような方式を採用することである．PAPS では，後者の方法で部分部品を体系化し，検索処理にプロダクションルールを採用することにより，これを実現している．

（3） 類似部品の扱い方

自動プログラミングを指向するシステムでは，部品を上手に選ぶことにより，類似の部品を作らないようにするのが普通である．複数の類似部品があると，その中の1つを特定するために，要求仕様に最も近いのはどの部品かを判定するための尺度を導入する必要がある．また，カスタマイズの箇所も部品ごとに異なるので，カスタマイズの自動化が課題となる．

与えられた要求仕様を，複数の部品の組合せなら満足できるという場合がある．このような複数個の部品の組を検索するために，MENDEL では，入出力属性（仕様）で部品（＝オブジェクト）を表現する方法を用いている．そして，部品の類似度を判定するために，入出力属性に is-a と has-a の概念による全順序関係を導入して構造化し，意味ネットワークを構築する．この意味ネットワークを辿ることにより，要求仕様に最も近い部品の検索を可能にしている．

APSS では，部品間に類似性を定義して，類似性に基づいた類推型の推論を用いることにより，数多くある部品から，修正すれば要求に合う部品を選択する機

能を実現している．ここで，部品AとBが類似しているとは，部品Aに簡単な修正を施せば，別の部品Bと同等の機能を果たすプログラムになる場合を言う．

6. 部品のカスタマイズとその組立方法

プログラム部品のあり方は，前述したように①ソースコードを部品化したもの，②設計仕様を部品化したもの，③設計情報を部品化したもの，の3つのタイプに分類できる．このうち，タイプ③の部品からプログラムを生成する方法については **4.** で詳述したので，本章では，タイプ①と②からプログラムを生成する方法について詳述する．

6.1 部品のカスタマイズとインスタンス生成

「部品のカスタマイズ」という言葉は，ここでは部品の仕様をユーザの要求に合うように修正するという意味で使用する．タイプ②の場合には，部品が設計仕様の形で与えられるから，設計仕様をユーザの要求に合うように修正し，修正後の仕様に基づいて，これに対応するソースコードを生成するので，「部品のカスタマイズ」という言葉よりも，「部品のインスタンス生成」と呼ぶほうがふさわしい．

（1） ソースコードを部品化した場合

（a） 専用画面を使ってカスタマイズする方法

SEA/I では，ファイルの数が最大構成となるような標準処理パターンと呼ばれる骨組み部品を用意しておき，ファイルの要否などをユーザに指定させることにより，求める仕様に合うようにこの骨組み部品をカスタマイズする方式を採っている．このとき，ユーザがあらかじめ定義しておいたファイル，共通エリア，画面などの情報もカスタマイズに利用される．また，EAGLE や CANO-AID も部品のカスタマイズに専用の画面を使っている．

（b） ユニフィケーションによるもの

SAPRE では，モジュールを与えられた仕様のプログラムに変形するための主な処理は，各変項に仕様中の具体的な値や記号を代入する処理（ユニフィケーション）と見なしている．1階のユニフィケーションも用いられているが，ライブラリモジュールは，一般に関数や述語が変項化されたものを含むので，2階のユニフィケーションを導入している．

MENDEL も部品のカスタマイズをユニフィケーションと見なしている．

（c） カスタマイズのための情報を部品の中にもつ方法

日本電子計算(株)の PRECOBOL[16] では，カスタマイズのための情報を部品の中にもち，メタ言語 GCL で記述している．

なお，(a)～(c)のカスタマイズとは目的が異なるが，効率の良い目標プログラムを得るために，部分計算により部品をカスタマイズする研究[21]もある．

（2） 仕様部品に対応するソースコードをもち，要求に合わせて部品をカスタマイズする方法

仕様部品から，これに対応するソースコードを生成する過程は，オブジェクト指向プログラミングにおけるインスタンスの生成に相当する．インスタンス生成機能は，オブジェクト指向言語では基本機能として実現されているが，他の言語はこのような機能をもたないので，仕様部品からこれに対応するソースコードを直接生成するのは容易でない．

そこで，仕様部品に対応するプログラムスケルトンをもち，ユーザの要求仕様との差分だけをこれに追加するのが実用的な方法である．このようなシステムの例としては ARIES/I, PAPS, REUSE-I/II などがある．

ARIES/I では，部品の仕様を記述するのに，フレーム表現を使っており，この中に，部品検索のための見出し，カスタマイズするときの条件や具体的なカスタマイズの方法（適合化ルール）などを記述しておくことにより，システムがこれを解釈して部品をカスタマイズする仕組みになっている．

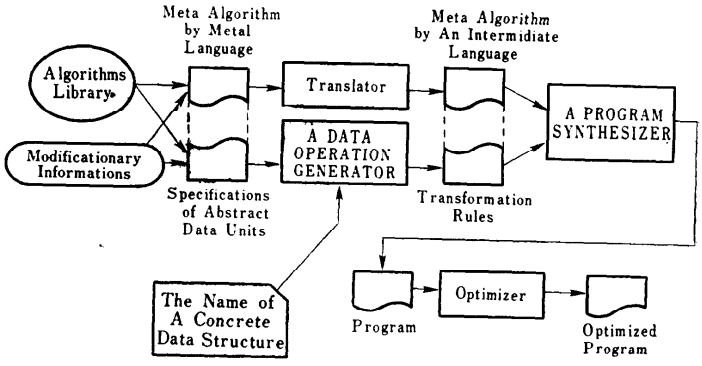


図-9 FURNACE システムのプログラム生成過程[14]

PAPS では，骨組み部品とそこに埋め込まれる部分部品のすべてについて，部品をカスタマイズするための知識をフレーム（具体的には，図-7 におけるインスタンス生成フレーム1と2）でもっている．骨組み部品のカスタマイズに関する知識は，要求仕様との係わりで整理されており，部分部品の生成やカスタマイズに関する知識は，要求仕様や骨組み部品との係わりで整理されている．

（3） 抽象アルゴリズムの具体化による方法

実現に関する記述が一切含まれていない抽象アルゴリズムから，目標プログラムを生成する研究もある．FURNACE がその例である．FURNACE のプログラム合成過程を**図-9** に示す．

抽象アルゴリズム（＝抽象基本関数を主体として記述されたもの）は，選択や修正がなされて後，トランスレータに入力され，Syntax Weak な中間言語で記述されたアルゴリズムに変換され，プログラム合成器の入力の1つとなる．一方，抽象単位データの仕様と具体データ構造名の指定の2つがデータ操作生成器の入力となる．データ操作生成器は，4種類の抽象基本関数について，抽象関数と1対1に対応する具体データ構造操作関数を対の形式で自動生成する．この対集合を抽象・具体変換規則と呼ぶ．これがプログラム合成器への残りの入力となる．プログラム合成器では，入力された抽象アルゴリズムを走査して，構文中に出現する抽象基本関数を変換規則に従って具体基本関数に置き換える．

（4） モデル化して部品を生成する方法

KIPS は，部品生成のためのモデルとして，コードモデルと呼ばれる対象世界のモデルをもっている．モデル化のための知識表現としては，意味モデルと同じく，オブジェクト指向＋データ指向が使われている．

（5） 関数などを用いて部品を生成する方法

SPACE などの仕様コンパイラとして構築するシステムにおける部品生成がこれに該当する．また，知識ベース利用によるシステムにおいても，ユーザのすべての要求に対応して部品を用意することは不可能なので，ユーザの要求に合わせてその部分のソースコードを生成する機能が必要である．

6.2 部品の組立方法

（1） プログラムモデルに基づく方式

SPACE では，図-4 で示したように，デシジョンテーブルを解釈実行する制御ロジックをプログラムモデル（これをプログラムフレームと呼んでいる）としてもっていて，要求定義で用いられた仕様部品（これを設計パターンと呼んでいる）に対応するプログラムコード（これをプログラムパターンと呼んでいる）をプログラムモデルに従って自動的に組み合わせていく仕組みになっている．

（2） 骨組み部品に部分部品を埋め込む方式

事務処理計算のバッチ処理では，骨組み部品に部分部品を埋め込むことにより，要求のほとんどをカバーできることが経験的に確かめられている．このことは，プログラム合成のための戦略的知識とも言うべきもので，強力さにおいて，これに匹敵する戦略的知識は他のプログラム合成法には見当たらない．（このことも他のプログラム合成法に比べて，部品合成による方法を有利にしている．）

この方式に基づくものは，次の3つに分類できる．

（a） 編集・複写機能ユティリティの利用によるもの

SEA/I, EAGLE, CANO-AID などでは，標準処理パターンと呼ばれる骨組み部品を指定した後，そこに埋め込まれる部分部品の名称と埋め込み位置（EAGLE では，骨組み部品の最後尾の位置をデフォルトとしている）を画面を使って指定すると，ユティリティが起動して部分部品を指定の場所に埋め込む仕組みになっている．

（b） オブジェクト指向言語とメタ言語の利用によるもの

REUSE-I/II では，クラスオブジェクトの中に 11 個のセクションを設け，この中に COBOL のソースコードや部品の生成手続き，部品間の接続関係，部品の組立手続きなど，プログラム生成に必要なすべての情報を記述している．このうち，プログラム生成のための手続きは，メタ言語 GDL で記述している．目標プログラムの組立は，部品間の接続関係の情報をオブジェクトに記述しておき，この情報を辿ることにより行われる．インスタンスが存在しないときには，新しいインスタンスを生成して，これにメッセージを送るようになっている．このようなことを可能にするために，インスタンスは，メッセージのみによって生成されるのではなく，新たなインスタンスを他のクラスから生成させる手続きが書けるようになっており，このことがインスタンス間に階層的な構造を与えて，最終的なプログラムを生成することを可能にしている．プログラム合成過程を**図-10** に示す．

（c） PART-OF の概念に基づくもの

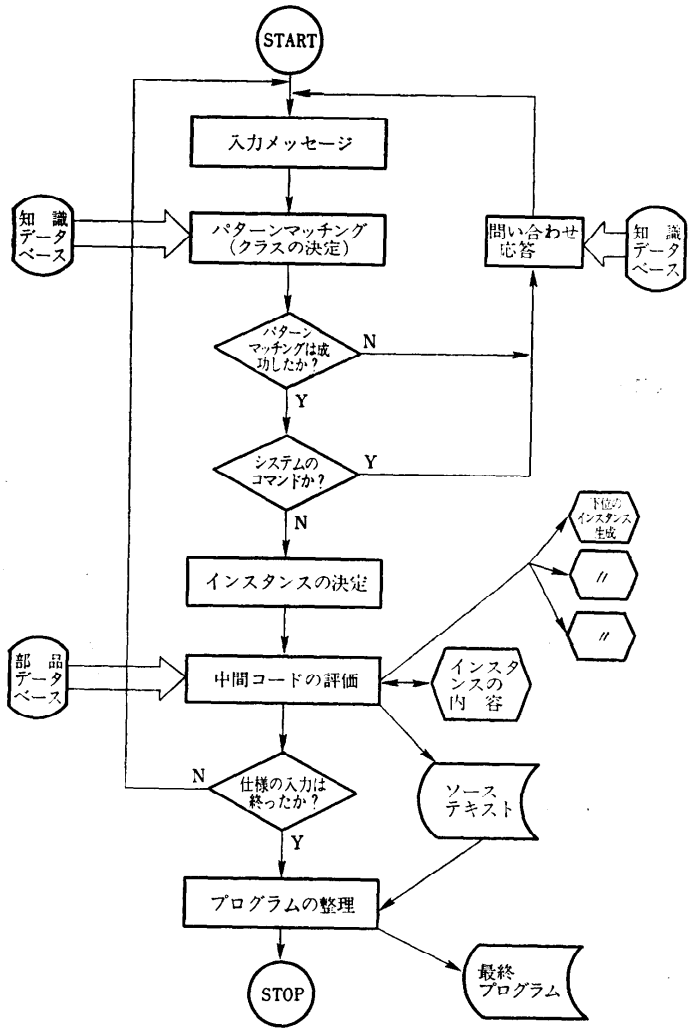


図-10 REUSE-I/II における合成フェーズの動作フロー[10]

PAPS では，仕様部品の中に，骨組み部品ごとに，そこでの合成に必要となる部分部品の候補を HAS-PARTSスロットで記述しておき，図-7 の埋め込み情報フレームと呼ばれるフレームに記述された，部品検索用プロダクションルール群の名称と部分部品埋め込みのための知識を参照することにより，骨組み部品中の必要な場所に，必要な部品を自動的に埋め込んでいる．

また，図-7 に示すように，ユーザが作り出すプログラムのモデル（PAPS では，システム部品と呼ばれる目標言語の構造を意識して作られた部品を利用して作成される）に必要な骨組み部品（複数個）を埋め込むときに，part-of の概念を利用している．

（3） パイピング機能の考え方に基づく方式

あるコマンドの標準入力を，別のコマンドの標準出力から直接入力することができる．逆に，あるコマンドの標準出力を，次のコマンドの標準入力に直接注ぐことができる．これを UNIX のパイピング機能と呼ぶ．このパイピング機能の考え方を部品合成の場面に当てはめると，1つの部品の処理結果がそのまま別の部品の入力情報となるように，一方の出力条件と他方の入力条件を調整することになる．手続き型プログラムの合成においては，骨組み部品に部分部品を埋め込む方式に匹敵する強力な戦略である．この方式に基づくものは，次の 4 つに分類できる．

（a） データフローの概念に基づくもの

ある部品の出力エリアが次の部品の入力エリアとなるとき，このエリアを共通エリアという．このような共通エリア（ファイルのこともある）を介して，レコード単位でデータの受渡しを行うような部品の集合を想定する．このとき，入力エリアのすべてにデータがあり，出力エリアにデータがない部品があれば，その部品にプログラムの制御を移す．その結果，その部品のある出力エリア（＝次の部品の入力エリア）に，処理結果に従ってデータが満ちるので，次の部品の実行が可能となる．このような考え方に基づき部品群を繋いでゆけば，1つの実行可能なプログラムを構成することができる．これをデータフローの概念という．

この方式を採用したシステムの例としては，DF-COBOL, HYPER COBOL (DF-COBOL を改良して製品化したもの)，IDL などがある．

類似の方法を用いているシステムとして BAGLES がある．BAGLES では，部品をサブルーチンの形で切り出し，部品の呼び出しには関数形式を用いてお

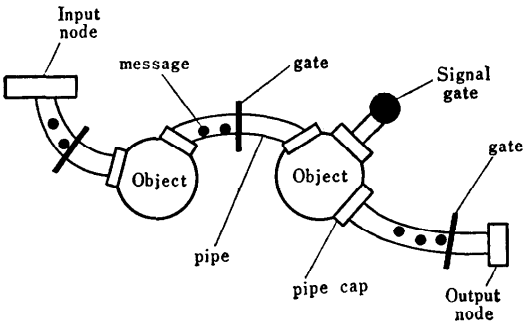


図-11 MENDEL におけるオブジェクトとパイプとメッセージ[24]

り，モジュール間の入出力データをパケットと呼ばれる規格化された領域で受け渡す方法を用いている．

ARIES/I（目標言語は HYPER COBOL）では，自然言語で記述された要求仕様を基に，最終出力のデータ仕様をまず明らかにする．そして，そのデータを出力する部品とその入力データを決定する．次に，この入力データを出力データとする部品とその入力データを決定する．（入力データレコードと出力データレコードがうまく対応しないときには，中間的なデータレコードを想定して，入力データレコードから出力データレコードを出力する部品と，中間データレコードから出力データレコードを出力する部品を生成する．）要求仕様により与えられた入力データを入力とする部品に辿り着くまで，このような動作を繰り返す．このようにして導かれた入出力データの導出関係を表現するモデルを導出モデルといい，このモデルを基に部品の自動組立を実現している．

（b） 属性ネットワークによる自動パイピング

MENDEL では，図-11 のように，オブジェクト（＝部品）の内部と外部はパイプ栓を通じてしか情報交換できないようにして，パイプ栓にメッセージの入出力属性を規定した属性名を与える．2つのオブジェクトの結合は，両者の入出力属性を同一化させることにより実現できると考え，同一化のために参照する知識として入出力属性を意味ネットワークで表現する．この属性ネットワークに基づき，属性の記述形式を定義し，さらに，オブジェクト間の類似に関する全順序関係を与える．これをメッセージ・パターンに導入することにより，自動パイピングを行う．

なお，MENDEL は，並列処理プログラムの合成を研究対象にしているので，同期部分のプログラムが必要である．同期部分のプログラム合成には Manna と Wolper の方法[8]を用いている．この方法は，時制命題論理とその拡張論理を用いて記述した仕様から，並列処理プログラムの同期部分を自動生成するものである．

（c） ゴールとプランの概念によるもの

AutoPro や APSS では，プログラムを合成するのに物語理解の分野で使われているゴールとプランの概念を適用している．すなわち，与えられた要求仕様からプログラムを合成するための目標（ゴール）を抽出し，その目標を手続き（プラン）列に展開して（ゴール・プラン展開），手続きに対応してあらかじめ準備している部品を用いてプログラムを合成する．ここでは，部品の適用条件をルールで表現しており，ルールの条件部には入力に対する制限を割り当て，ルールの帰結部には出力を割り当てている．これにより，帰結部で得られた出力結果が他のルールの条件部にある入力の制約を満たせば，さらに新しい出力結果が帰着されることになり，両者の入出力を繋げることが可能となる．

（d） pre-condition と post-condition の概念に基づくもの

PAPS では，仕様部品の中に対応する部品の実行開始条件と終了条件を 図-7 の実行開始条件フレームと実行終了条件フレームに記述しておき，先に実行する部品の終了条件と，次に実行する部品の開始条件とが一致するように，入出力処理部品を自動選択することにより，隣接して実行する骨組み部品同士の結合を実現している．このとき，両方の条件が一致するような入出力部品がない場合には，%VIOLATION ファセットで手繰られるプロダクションルールが起動されて，2つの部品の間を繋ぐような部品を自動的に生成するようになっている．

KIPS では，目標プログラムの構造や組み立て方に関する知識をモデル化したもの（＝プログラムモデル）をオブジェクトの形でもっている．プログラムモデルを表現するための知識表現としては，オブジェクト指向＋データ指向が使われている．プログラムモデルは，処理記述のモデルとファイルのモデルに大別され，上位・下位の概念に基づいて表現される．ここでは，クラスレベルのオブジェクトによって各クラスに共通な一般的属性が定義され，そのインスタンスとして個々の具体的プログラムが表現される．したがって，要求仕様の理解処理の結果は，プログラムモデルのインスタンスとして保持される．これが仕様獲得で

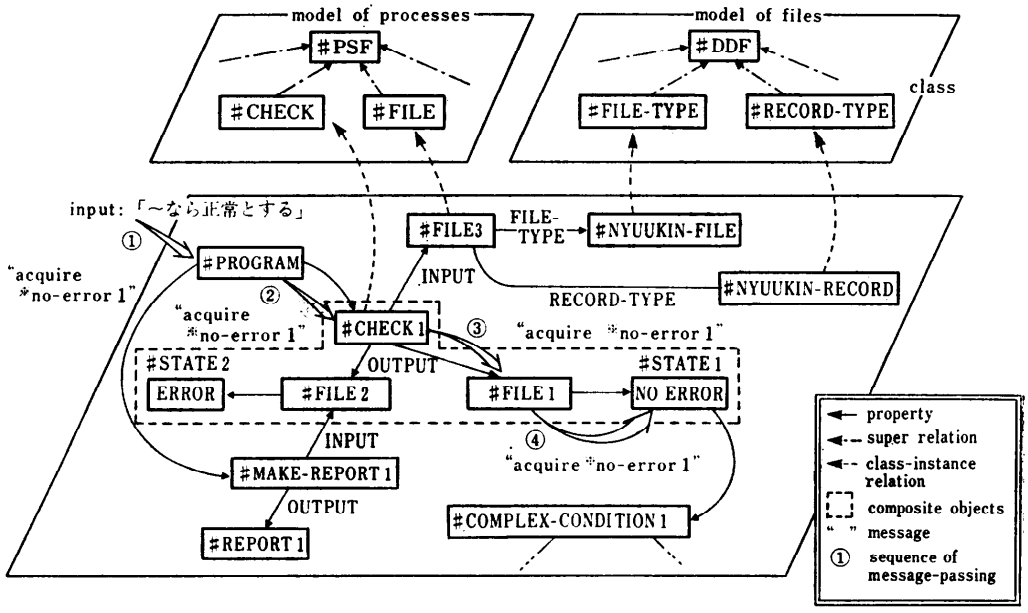


図-12 KIPS におけるプログラムモデルと仕様獲得[10]

ある．プログラムモデルは，オペレーションを中心として，入出力が pre-condition と post-condition で示され，条件は出力ファイルの処理状態として示される．たとえば，**図-12** では，#CHECK1（オペレーション“チェック”を意味するインスタンスオブジェクト）の条件である #COMPLLEX-CONDITION1 が，出力ファイル #FILE1 の状態を示すように構成される．ここで使われている pre-condition と post-condition の概念は，目標言語である HYPER COBOL の基礎となっているデータフローの概念と同種のものである．このプログラムモデルに従って，コードモデルの部品生成知識（メソッド）を起動することにより，部品の自動組立を実現している．

7. おわりに

自動プログラミングを実現する方法は種々あるが，その中で最も実用性が高いのは部品合成による方法である．それだけに，多くの企業で各種の試みがなされているが，その実現方法が公開されているシステムは少ない．また，実用性を指向したシステムは多いが，完全な自動化を目指したシステムは少ない．そこで，自動プログラミングシステムを作成した経験から，自動プログラミングの考え方とこれを実現するための要素技術を余すところなく採り上げ，紹介したつもりである．しかし，丹念に調査したつもりではあるが，少ない資料と浅学のため，書き残した事例や誤った記述もあるかも知れない．ご指摘いただければ，何かの折りに，改めて公表したい．

参考文献

1) 千吉良英毅，永松祐嗣，小林正和：システム仕様書の再利用によるソフトウェアの開発技法，日立評論，Vol. 69, No. 3, pp. 47-52 (1987).
2) 原田 実，篠原靖志，鈴木道夫：プログラム自動生成システム ARIES/I の開発，情報処理研究 No. 13, 電力中央研究所，pp. 59-88 (1985).
3) 原田 実：パターン指向型プログラム開発技法，研究報告：584006, 電力中央研究所 (1985).
4) Harada, M.: Specification Compiler: SPACE, Proc. of IEEE COMPSAC '87, Tokyo (Oct. 1987).
5) 今中 武，上原邦昭，豊田順一：類似したプログラムの再利用による自動プログラム合成，ロジック・プログラミング・コンファレンス，pp. 47-56 (1987).
6) 金子信一，石川和彦，福田由紀雄，蓮田広保：ソフトウェア部品検索ツール，第 31 回全国大会講演論文集，情報処理学会，pp. 493-494 (1985).
7) 古宮誠一：部品合成によるプログラム自動合成システム PAPS～知識工学的アプローチを用いたその実現方式～，情報処理学会研究報告，87-SF-21, Vol. 87, No. 40, pp. 5-12 (1987).
8) Manna, Z. and Wolper, P.: Synthesis of Communicating Processes from Temporal Logic

Specification, ACM Trans. on Lang. and Sys., Vol. 6, No. 1, pp. 68-93 (1984).
9) Neighbors, J. M.: The Draco Approach to Constructing Software from Reusable Components, Vol. SF-10, No. 5, pp. 564-574 (Sep. 1984).
10) 日経データプロソフト「ソフトウェア開発支援ツールの現状」NS 2.101・001～210 (1987).
11) 日経データプロソフト「ソフトウェア開発支援ツールの現状」NS 2・151・151～505 (1987).
12) 西田富士夫，藤田米春：ライブラリモジュールを用いたプログラムの半自動的詳細化，情報処理学会論文誌，Vol. 25, No. 5, pp. 785-793 (1984).
13) 岡村和男，垂水浩幸，阿草清滋，大野 豊：MOMOCLI におけるクラス再利用のための知的検索，日本ソフトウェア科学会第2回大会論文集，pp. 97-100 (1985).
14) 大石東作：抽象アルゴリズムの記述と具体プログラムへの変換，情報処理，Vol. 19, No. 11, pp. 1042-1049 (1978).
15) 大石東作：手続きの抽象化・蓄積・再利用，情報処理学会研究報告，86-SW-46, Vol. 86, No. 7, pp. 73-80 (1986).
16) 大木幹雄：部品再利用支援ツール PRECOBOL, SOFTWARE TOOLS SYMPOSIUM '87 予稿集，情報サービス産業協会，pp. 217-230 (1987).
17) Rice, J, G. (著)，山口圭一，原田悦男，志村順(訳)：自動プログラム構築技法の実際，近代科学社 (1983).
18) ソフトウェア・エンジニアリングに関する調査研究「ソフトウェア部品データベースに基づく問題向き言語開発支援システム」(社)ソフトウェア産業振興協会，昭和 57 年度と 58 年度.
19) ソフトウェア合成技術に関する調査研究報告書，(財)日本情報処理開発協会 (1984).
20) 杉山健司，秋山幸司，亀田雅之，牧之内顕文：対話型自然言語プログラミングシステムの試作，電子通信学会論文誌，Vol. J 67-D, No. 3, pp. 297-304 (1978).
21) 竹村 司，鯵坂恒夫，阿草清滋，大野 豊：部分計算によるプログラム部品のカスタマイズ，日本ソフトウェア科学会第2回大会論文集，pp. 49-52 (1985).
22) 内平直志，関 俊文，粕谷利明，本位田真一：MENDEL における並列プログラムの部品結合，情報処理学会研究報告，86-SW-46, Vol. 86, No. 7, pp. 57-64 (1986).
23) 内平直志，本位田真一：時制命題論理を用いた部品からの並列プログラム合成，日本ソフトウェア科学会第3回大会論文集，pp. 145-148 (1986).
24) Uchihira, N., Kasuya, T., Matsumoto, K. and Honiden, S.: Concurrent Program Synthesis with Reusable Components Using Temporal Logic, The Logic Programming Conference '87, pp. 37-45 (1987).
25) 上原邦昭，藤井邦和，豊田順一：自然言語による仕様からの自動プログラム合成，コンピュータソフトウェア，Vol. 3, No. 4, pp. 55-64 (1986).

(昭和 62 年 8 月 17 日受付)