ソフトウェア工学 42－4
（１９８５・６・２６）

設計用言語（ＳＬ）の処理方式

渡辺　敏　　岡本　務　　稲田　満　　中村　雄三
（ＮＴＴ　ソフトウェア生産技術研究所）

１．はじめに

プログラミングにおける人間の思考過程を機械化することはソフトウェアの生産性向上に非常に有効である。そのため、プログラムの自動生成に向けて各種の研究がなされている。１つの分類として、ソフトウェア工学的なアプローチと人工知能的なアプローチとに分けることができる。後者は、研究段階のものが殆どである。前者は一般にジェネレータと呼ばれ、実用的なものが数多く作られているものの、適用性／柔軟性に欠けるという問題がある。既存のジェネレータは処理のモデル化（多くは固定化）によって、展開率を上げるものが多くこのような手法では適用性と展開率がトレードオフの関係になる。図１に、この関係を示す。ジェネレータの作成目的は、生産性向上にあるため、専用化される。結果として、ジェネレータは特定処理パターン，固定処理向きのものと位置付けられ、利用が広まらない原因となっている。ジェネレータ利用をより一般化させるためには、適用性／柔軟性を高めることが必須である。

本報告では、データ構造／データフローに着目した処理手順導出と共通処理の部品化を骨子とするプログラムの自動生成方式について述べる。これは、ジェネレータの本質的な問題点を解決する一つの新しい提案である。

２．自動生成方式の提案

ジェネレータの適用性を高めるためには、生成論理（処理アルゴリズム）に多様性を持たせること及び生成コード（部分的に固定化される処理）に柔軟性を持たせることが必要である。その解決法を、以下に提案する。

（１）生成論理の多様化

自動生成の着眼点としては、処理パターン，状態遷移，データ構造，データフロー，等が考えられ、何れを選択するかによって、適用域が定まる。事務計算のようなデータ操作主体の処理分野を対象とする場合、データ構造／データフローに着目するのが、以下の理由により最も有効である。

①プログラム設計の容易性

事務計算分野のプログラムは、データ構造に着目した設計が、分かりやすくかつ誤りを少なくできるという点で有効である（ジャクソン等）と言われており、ジェネレータの入力仕様を理解しやすいものにできる。

②記述範囲の広さ

事務計算分野の処理を対象とする限り、基本的なデータ構造の組み合わせにより、多様な処理を実現できる。

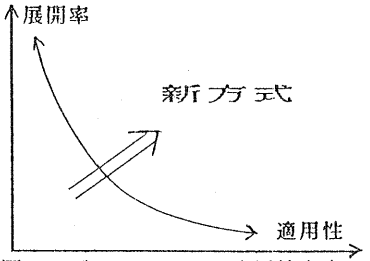


図１　ジェネレータの適用性向上

データ構造／データフローに着目した自動生成法も、各種の方式が考えられる。データ構造の記述能力を最大限に生かし、適用性を上げるためには、データ構造／データフローの情報を如何に自動生成に結びつけるかが重要である。本提案の骨子を以下に示す。

①プログラムを処理単位に分解し、それらをデータ構造／データフローの解析情報に基づき、組み立てる方式とする。この方式により、直接複数のデータ構造を合成して処理手順を導出する方法に比べ、各種の構造の組み合わせについて一々考慮する必要がないことから、処理手順の導出が容易になると共に、データ構造の組み合わせについて制約を少なくすることができる。

②データ構造を、ファイル等への格納形式及びＩ／Ｏ処理との関連を示す物理的なデータ構造とプログラムの処理対象となるデータを表わす論理的なデータ構造に分離する。本分離により、特定のＩ／Ｏ処理を前提とする必要が無くなり、多様なＩ／Ｏ処理を簡単に対象とすることができる。

③基本的なデータ構造（順列／選択／繰り返し）の他に、プロセス／同列／階層の３構造を追加する。本追加により、データ構造の表わす処理手順が明確になり、ジェネレータの入力仕様が分かりやすくなると共に手順導出が容易になる。

④プログラムの中には、データ構造／データフローでは、処理順序を決定できないデータ操作と直接関係のない処理もある。これらの処理については、処理と処理タイミングを利用者に指定させる。
（（2)との関連で、実現が可能となる。）

（２）生成コードの多様化

ジェネレータの問題である柔軟性の欠如は、本質的なものである。これは、生成オブジェクトの原型をジェネレータの中に持っておき、指定により展開

することによる。このまま、柔軟性を上げようとすれば、ジェネレータに保持すべき生成オブジェクト及びその展開機能が増え、非実用的なものになる。また、開発前から利用法が明確であれば、準備することができるが、開発後の要求については全て改造することになる。以上の問題は、生成オブジェクトのスケルトンをジェネレータ本体に持つことに起因する。つまり、スケルトンをジェネレータの外に出すことにより、解決できる。

具体的には、生成オブジェクトのスケルトン及び展開方法を部品として、ライブラリに登録しておき利用者の指定により、それを取り出し利用するという部品化機構を設ければよい。この方式により、ジェネレータの生成処理が利用者の要求仕様に合わなければ、部品のリファイン／再定義により簡単に修正でき、ジェネレータに柔軟性を与えることができる。

３．プログラム生成方式

本章では、プログラムを処理単位に分割し、それらをデータ構造／データフローの解析情報に基づき組み立てる方式を実現する上で重要な整理及び具体的な生成手順について述べる。

３．１　生成プログラムの構成法

本方式では、プログラムは基本的な処理単位（以後、機能要素と呼ぶ）と、繰り返し（REPEAT）／選択（CASE）／順列（並び順）の３制御要素との組合せで構成されていると捉える。この様な生成プログラムモデルを前提とした自動生成では、機能要素への分割法及び制御要素の組合せ方法が重要なポイントとなる。

（１）制御要素による機能要素の組合せ法とプログラムの枠組み

処理順序による機能要素の組合せは、単に機能要素をその順序に従って並べることにより実現できる。繰り返し処理における機能要素の組合せは、同一ループで処理されるものを集め、繰り返し処理のための付加処理（繰り返し終了条件等）を付与することにより実現できる。条件付きの機能要素の組合せは、単純に考えれば同一条件を持つものを集めれば良いが、ループ処理と異なり、各種条件が複雑に組み合わされることが多いため、同一条件の集約は容易ではない。そのため、本生成方式では、条件は全ての機能要素単位に付与することとした。図２に、データ構造／データフローに着目して生成される生成プログラムイメージ（本処理部と呼ぶ）を示す。又、生成プログラム全体の基本的な枠組みとしては、本処理部の実行に付帯して必要な他機能を加えたものとなる。表１に、その枠組みを示す。

（２）機能要素の分類

生成プログラム機能は機能要素の組合せで実現されるため、機能要素の機能／種類／大きさを如何に

表１　生成プログラムの枠組

枠組の分類	枠組の内容
データ宣言部	プログラムの処理対象であるデータの宣言，自動生成される制御データ／ワークデータの宣言及び部品で使用するデータの宣言
前処理部	本処理部に先立ち、実行されなければならない処理。部品の引用により展開され、自動生成の対象ではない。
本処理部	データ構造／データフロー情報に基づき、機能要素が組み合わされて生成されるメインの処理。
後処理部	本処理部の後に実行される点を除いて、前処理部と同じ。
例外処理部	本処理において、例外条件が発生した時の例外処理群

```
IF C1 THEN F1 ;
IF C2 THEN F2 ;
IF C3 THEN
  REPEAT WHILE( C4 );
    IF C5 THEN F4 ;
    IF C6 THEN
      REPEAT WHILE( C7 );
        IF C8 THEN F5 ;
        IF C9 THEN F6 ;
      ENDREP;
    IF C10 THEN F7 ;
  ENDREP;                          Cn：条件
IF C11 THEN F8 ;                   Fn：機能要素
```

図２　生成プログラムイメージ

するかが、重要である。自動生成を容易とするためには、次の条件を満たす必要がある。

①ジェネレータの入力仕様で規定される処理内容から、機能要素への分解あるいはマッピングが容易であること。

②機能要素の組合せにより表現される機能が広い範囲をカバーすること。

③自動生成時の組み立てが容易であること。

具体的には、機能要素の種類を少なくすること，機能要素を適切な大きさとすること，機能要素の組合せ時の制約を少なくすること，が必要である。特に、機能要素の大きさは、プログラム自動生成時の翻訳スピード、及び生成プログラム実行時の実行スピードに影響するため、大きくするほうが望ましい。更に、本方式では、利用者が生成プログラムを定義する部品化機構を合わせて実現するので、これとの整合性を考慮する必要がある。

データ構造に着目した自動生成では、データ構造の構成単位（レコード）対応にどのような処理が必要になるかを明らかにし、それらの処理を前述の条件に合った機能要素として分類整理すれば良い。モデル化した更新データ（レコード）に着目した機能要素を図３に示す。しかし、既に述べたように、機能要素は柔軟性が損なわれない範囲で、大きくする

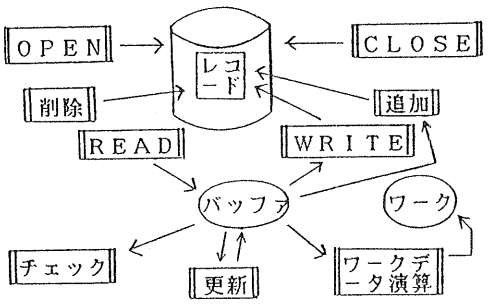


図３　更新データに着目した機能要素のモデル化

ことが望ましいので、最終的には、入力処理要素／相関チェック要素／出力処理要素／ワーク演算処理要素の４要素とした。レコード対応の入力仕様から機能要素への分解法を図５に示す。図５は一つのデータに着目したものであり、複数データを取り扱う処理では、これらが更に組み合わされることになる。実際のプログラムでは、更に例外処理やデータ操作（Ｉ／Ｏ，チェック，設定／更新）と直接関係のない処理（以後、付帯処理と呼ぶ）も必要である。それらを加え整理したものを表２に示す。

３．２　データ構造からの処理手順情報抽出

（１）データ構造の定義

データ構造をプログラムの自動生成に結びつけるためには、データ構造の定義が重要である。本方式での定義を以下に示す。

データ構造とは、構成要素（Ｉ／Ｏに於ける１アクセス単位　例えばレコード等）の並びを繰り返し，順列，選択等の構造を用いて示すものである。データ構造は、二つの意味付けを持つ。一つは、データの格納構造（以後、物理データ構造と呼ぶ）を示し、他は処理対象データの処理構造（以後、物理データ構造と呼ぶ）を示す。

物理データ構造とは、Ｉ／Ｏ機能をモデル化したものであり、構成要素（レコード等）のアクセス方法を示すデータ構造である。論理データ構造との対応でＩ／Ｏ処理と結びつけたり、処理可能性をチェックするのに用いる。

論理データ構造とは、処理対象となる構成要素の処理順序を示すデータ構造である。制御情報の抽出に用いる。

本方式では、データ構造をその意味付けで二つに分離した。この分離により、特定Ｉ／Ｏ機能を前提にして、論理データ構造へのマッピング可能性を判断しなくても、物理データ構造との対応でそれを判断できる。そのため、物理データ構造の定義変更で多様なＩ／Ｏ機能を処理対象にできる。

（２）論理データ構造と処理手順

論理データ構造から処理手順を導出する方式では、論理データ構造の表現能力が、適用範囲を直接左右する。そのため、基本構造である繰り返し、選択、

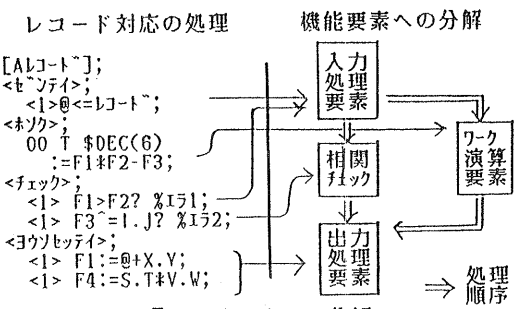


図５　機能要素への分解

表２　機能要素の種類

機能要素の種類	処理内容	部品化の処理対象
入力処理要素	制御情報の設定，入力Ｉ／Ｏ，Ｉ／Ｏチェック，固有チェック	Ｉ／Ｏ
相関ﾁｪｯｸ	データのチェック	関数
出力処理要素	レコード単位でのデータ設定／更新処理，出力Ｉ／Ｏ，Ｉ／Ｏチェック	Ｉ／Ｏ関数
ﾜｰｸﾃﾞｰﾀ演算処理	ワークデータへの代入処理	関数
自動生成ﾜｰｸﾃﾞｰﾀ演算要素	出力処理要素を一つの処理単位とすることによる他データ参照時の制約をなくすための仮演算機能	関数
例外処理要素	チェックで誤りを検出した時、実行する処理	一部の特殊機能
付帯処理	データ操作と直接関係しない処理	全て

順列構造に、更に３構造を追加して、生成対象の範囲を拡大すると共に、処理の意味付けを明確化した。

回列構造　：構成要素間での双方向の参照を可能とするため

プロセス構造　：同一データ／ファイルの複数回使用を可能とするため

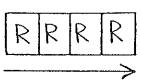
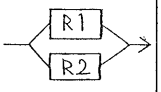
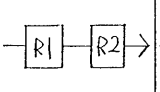
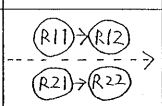
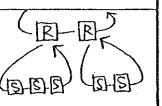
階層構造　：データベース等の様にデータ階層を持つものを対象とするため

論理データ構造とプログラムの処理手順との対応を表４に示す。制御情報としては、処理順序，ループ処理の必要性，機能要素対応に付与すべき実行条件が得られる。

３．３　プログラム処理手順の導出法

一つのデータ構造からプログラム処理手順を導出する場合は、構造種別対応に表４に示した処理手順との対応で簡単に導くことができる。図４に変換例を示す。実際のプログラムでは、複数の入出力情報を取り扱える必要がある。Ｎ入力－Ｌ更新－Ｍ出力

表３　論理データ構造と処理手順との対応

構造種別	繰り返し	選択	順列	同列	プロセス	階層
入力仕様の記述	００＊　Ｆ ０１　　Ｒ	００！　Ｆ ０１　Ｒ１ ０１　Ｒ２	００。　Ｆ ０１　Ｒ１ ０１　Ｒ２	００＋　Ｆ ０１　Ｒ１ ０１　Ｒ２	００．　Ｆ ０１　Ｒ１ ０１　Ｒ２	００＊　Ｆ ０１　Ｒ ０２＊　Ｇ ０３　Ｓ
並びの式図表現						

対応処理手順	REPEAT
	　Ｒの処理
	ENDREP
		IF C THEN
		　Ｒ１の処理
		IF ~C THEN
		　Ｒ２の処理
			Ｒ１の処理
			Ｒ２の処理
				機能要素レベルで自由に組合わせられる（他情報により、順序付けがなされる）	Ｒ(R1)の処理
					Ｒ(R2)の処理
					同一データを複数回処理
						REPEAT
						Ｒの入力処理
						　REPEAT
						　　Ｓの処理
						　ENDREP
						Ｒの出力処理
						ENDREP

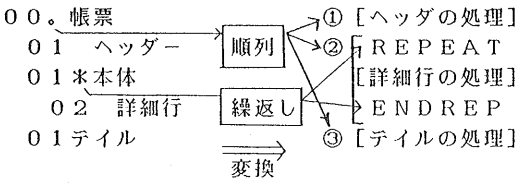


図４　単一データ構造からの導出

のデータを取り扱うプログラムを対象とする時は、複数の論理データ構造から、プログラム処理手順を導出するアルゴリズムが必要である。

本生成方式に於ける導出手順を以下に示す。

（１）繰り返し構造のみに着目したプログラム処理手順の導出

①論理データ構造を、繰り返し構造とその配下のデータ構造を除き機能要素と制御要素に分解する。この時、繰り返し構造とその配下のデータ構造はまとめて一つのノードに置き換えて機能要素と同じに取り扱う。制御情報は個々の機能要素対応に処理順序を示す順序リストと実行条件を示す条件リストとして持たせる。この様な操作で得られたものをプレートと呼ぶ。プレートは、当該プレートを構成する機能要素のリストとして表現される。

②ノードに置き換えられた繰り返し構造配下のデータ構造についても、①と同様な処理を行なう。又、新たに作成されたプレートにたいしてノードからチェインを行なう。この手順の繰り返しにより、論理データ構造は全てプレートの階層構造に変換される。

③論理データ構造対応に作成されたプレートの階層構造の最上位のプレートを全て無条件に合成する。（具体的には、単にリストをつなぐ処理）　これをシステムプレートと呼ぶ。

④システムプレート内のノードを調べ、同一ループで処理する必要のあるものについては、一つのノードに統合する。ノードの統合では、各ノードの順序リスト及び条件リストをそのまま新ノードに引き継がせる。又、それらのノードの下位に繋がれているプレートを合成して一つのプレートとし、新ノードに繋ぐ。（同一ループに於ける処理の指示は、入力仕様で与えられる。）

⑤下位プレートに対しても、④と同様な処理を行なう。この手順の繰り返しにより、複数の論理データ構造を合成したプレートの階層構造が得られる。このプレートの階層構造は、プログラム内の繰り返し処理を取り出したものと１対１に対応する。本処理の処理イメージを図６に示す。

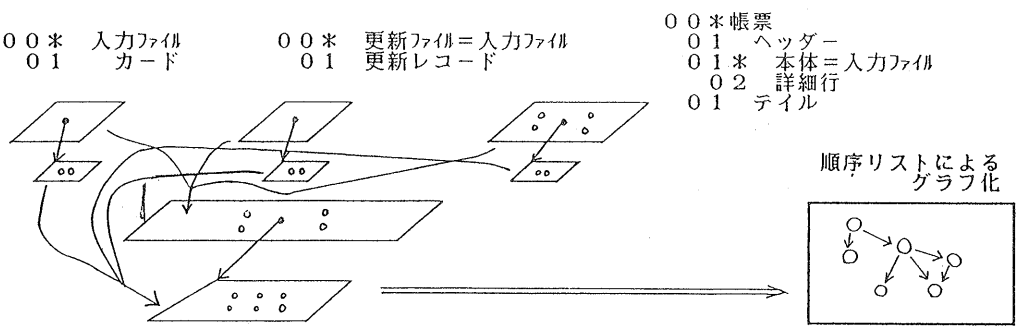


図６　繰り返し構造に着目した導出とグラフ化

（２）ブレート内の処理順序決定

ブレート内の機能要素に付与された制御情報は、一つの論理データ構造に関するものだけであり、論理データ構造間に渡る制御情報はない。これは、論理データ構造からは他の論理データ構造との処理順序に関する情報を収集できないことによる。本方式では、論理データ構造間に渡る処理順序の決定は、データの参照関係に基づき行なう。具体的には、ある機能要素が他機能要素の実行により得られるデータを参照するという関係がある場合、「他機能要素－＞ある機能要素」というデータフローに着目した処理順序を、個々の機能要素の順序リストに追加することにより、手順決定に反映する。ブレート内の処理順序決定の手順を以下に示す。

①入力仕様から求められた機能要素間のデータ参照関係を個々の機能要素の順序リストに追加する。

②最下位ブレートからシステムブレートに向けて、各ブレート内の機能要素の順序リストの内他ブレートを参照するリストをノードに集める処理を繰り返し行なう。これは、処理順序決定時に、ノードに下位ブレートの機能要素のデータ参照関係を代表させるために必要である。

③ブレート内の処理順序の決定は、機能要素及びノードをノード，順序リストをエッジ，とするグラフを作成し、処理可能なものを順次選択し、順序付けることにより行なう。これをシステムブレートから始めて全てのブレートに行なえば、プログラムの全ての処理順序を決定できる。

３．４　コード生成

以上の処理で導出された処理手順情報（ブレート階層構造，ブレート内の機能要素の処理順序、機能要素の実行条件）に基づき、ホスト言語ソースコードを展開する。

（１）システムブレートを最初の処理対象とする。

（２）ブレート内で使用する制御情報の初期化を行なうホスト言語ソースコードを展開する。

（３）ブレート内の機能要素の処理順序に従い機能要素あるいはノードを取り出し、以下の処理を繰り返し行なう。

機能要素対応の処理：

①条件リストの有無をチェックし、条件付なら条件に対応したＩＦ文を展開する。

②機能要素に対応したホスト言語ソースコードを展開する。この時、機能要素の内部処理に部品の引用あるいはＩ／Ｏタイミングでの部品の組み込み指示があれば、その部分に部品対応のコードを展開する。

ノード対応の処理：

①条件リストの有無をチェックし、条件付なら条件に対応したＩＦ文を展開する。

順序決定アルゴリズム

ａ．各ノードは、 ”ﾐｼｮﾘ， ”ｼｮﾘﾁｭｳ， ”ｼｮﾘｽﾞﾐ，のステータスを持ち、初期ステータスは ”ﾐｼｮﾘとする。ノード数をＮとする。

ｂ．Ｉはグローバル変数で、初期値１を持つ。

ｃ．ＯＲＤＬＩＳＴはノード対応の配列であり、初期値０を持つ。

ｄ．ＣＶはＯＲＤＬＩＳＴの番号からノードを得る関数

ｅ．ＲＣＶはノードからＯＲＤＬＩＳＴの番号を得る関数

ｆ．Ｓ（Ｘ）はエッジと結びつけられたノードのうち、未処理のノードを返す。

ｇ．アルゴリズムは、ＯＲＤＬＩＳＴ上で未処理ノードを検索するＭＡＩＮ手続きと順序決定を行なう再帰手続きＤＦＳとからなる。

```
MAIN:PROCEDURE;
  DO n=1 TO N;
    IF ORDLIST(n)=0 THEN CALL DFS(CV(n));
  END;
END MAIN;
DFS:RECURCIVE PROCEDURE(X);
  X="ｼｮﾘﾁｭｳ;
  DO WHILE (S(X)="ｱﾘ);
    CALL DFS(S(X));
  END;
  X="ｼｮﾘｽﾞﾐ;
  ORDLIST(RCV(X))=I;
  I=I+1;
END DFS;
```

②繰り返し開始文を展開する。（繰り返し条件は　END OF DATA ／利用者の指定）

③配下のブレートに対して、（２）（３）の処理を行なう。

④繰り返し終了文を展開する。

（４）以上で生成プログラムの枠組みの内、本処理部の生成が終わる。他処理部については、処理順序を特に導出する必要がなく、指定箇所に直接展開する。前処理部／後処理部については、当該箇所へ展開することを指示された部品を展開する。例外処理部には、入力仕様の記述内容から変換して得られたホスト言語ソースコードを、展開する。データ宣言部は、入力仕様で記述されたもの／処理順序の導出で必要性が明確になったもの（制御情報、演算用ワーク）／部品で展開が指示されたもの、を展開する。

４．部品化機構の実現法

部品化の着想及びプログラムの自動生成との関連については既に述べた。ここでは、部品化機構実現上の問題点及びその実現方法について述べる。利用者に生成プログラムの一部となるホスト言語ソースコード（以後、単にソースコードと呼ぶ）を直接定義させる機能を実現する処理系は、比較的簡単に作成できる。部品仕様（利用者定義）の記述に従い、ソースコードを展開するインタープリタとして実現できる。処理系のイメージを図７に示す。この様な機能を実現する場合、自動生成による展開処理で生成されたソースコード（以後、部品で展開されたものと区別するために、本体部分と呼ぶ）と部品の引用により展開されるソースコードを組み合わせて、一つの意味のある処理を構築できることが前提となる。つまり、その整合性を如何にとるかがポイントとなる。以下に、問題点と解決法を示す。

（１）部品化対象機能の限界

生成プログラムの全てを部品化の対象とした場合、部品と自動生成部分との整合をとるのは多種多様なケースが想定されることから、非常に難しい。部品化の目的である処理の多様性を吸収するという観点からは、全てを対象とするのでなく、多様性が必要な部分に限定しても問題とはならない。要は、何処に限定するかである。

本生成方式では、処理を構成する単位を機能要素と定義し表２に示した分類を行なっている。この中で、機能要素内の一部の機能を４つの範ちゅうに分類して部品化の対象とした。以下に、対象とした理由について述べる。

①Ｉ／Ｏ処理

プログラムの基本機能ではあるが、各種のＩ／Ｏ機能及び多様な使用法が存在することによる。又、必須機能であるため、未サポートの場合、そのジェネレータは使用されないことになる。

②付帯処理

データ構造／データフロー情報からは、処理順序が得られない処理（入力仕様での直接記述ができない）である，使用環境／処理方式によって、多種の機能／多様なインターフェースを必要とする，この種の処理をジェネレータ作成時から仕様を決めて準備することが難しい，及び他処理との独立性が比較的高い，等の理由により部品化の対象として最も適した機能と考えられる。ＯＡＰ（Online Application Program）を例にとれば、排他制御処理／ジャーナル収集処理等がこの範ちゅうに入る。

③例外処理

②と同様な理由による。但し、組み込み法（例外処理対応の入力仕様から直接引用される）が異なるため別分類とした。

④高度な演算処理

業務処理別に見た場合、当該業務の中では使用頻度が高い共通的な演算機能がある。それらを部品化の対象とすることにより、記述性向上に効果が期待される。又、標準演算機能の機能不足を補うというメリットも他にある。ＯＡＰを例にとれば、日数計算／利息計算等がある。

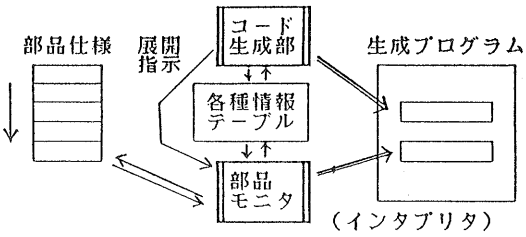


図７　部品化機構の処理イメージ

（２）オブジェクトのインターフェース整合

自動生成により生成されたソースコードと部品により展開されたソースコードは組み合わされて一つの処理となる。そのため、制御情報／データの授受に関して整合をとることが必須である。本方式では、以下の整理を行なった。

①制御方法

部品側での自由な制御移行を可能にすれば、移行先の決定法、自動生成された論理への影響、等の問題があるため、自動生成部分への制御移行を禁止した。（強制終了、他プログラムの呼び出しは含まない。）その代わり、部品仕様で処理結果（正常／異常等）が格納されているデータ名及びその判定方法を定義させておき、部品により展開されたソースコードの直後に、処理結果の判定及びそれに付随した制御移行を行なうソースコードを自動的に展開する方式とした。

②データの授受

部品では、直接ソースコードを記述するため、自由に処理を記述することができる。しかし、自動生成部分と直接関係しないという条件の基であり、実際には、組み合わされる場合が殆どである。そのため、部品での処理に必要なデータの、生成プログラム上での名標及び属性を取得する各種関数を設けると共に、部品側で定義したデータの名標及び属性を自動生成処理へ通知するインターフェースを設けた。

（３）多様なソースコードの展開

部品からは、固定的な処理しか展開できなければ、若干の処理差異により、別部品としなければならない。これでは、部品の作成がネックとなる。又、使用状況に応じて、展開処理を変更しなければならないケースも多くある。そのため、マクロ展開時機能と同様に条件によって展開ホスト言語のソースコードを変更できる機能を設けた。更に、使用方法及び使用状況を部品側から容易に把握できるように、自動生成時に得られた情報を参照する機能（関数）を併せて設けた。

（４）部品の誤使用防止

部品化方式の最も大きな問題点は、部品の誤使用／部品の作成誤りがあった場合、プログラムのテスト／デバグをホスト言語のソースプログラムレベルで行なわなければならない点である。

部品の使用誤りについては、ジェネレータの入力仕様に記述された参照形式と部品側の定義形式とによる形式チェックを行なうと共に、意味的な正しさについては、入力仕様での定義情報及び自動生成時に得られた情報を参照する機能（関数）を部品側に設け、部品作成者にソースコードの定義に加えて、部品使用法の妥当性チェックを行なわせる方式とした。

部品の作成誤りについては、十分テストされた品質の高い部品を使用することにすれば、問題はおきない。

部品化機構実現上の問題と解決法について述べて来たが、この処理イメージを図８に示す。

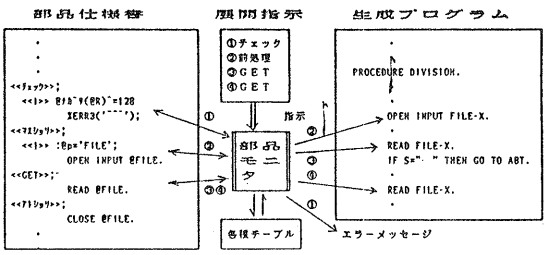


図８　部品展開処理のイメージ

５．本処理方式の試作評価

本処理方式は、設計用言語（ＳＬ）のインプリメントに適用し、ホスト言語をＣＯＢＯＬとして試作した。ＳＬは処理のバリエーションが多くかつ実行環境とのインタフェースが複雑であるため自動生成が難しいと考えられる大規模バンキングシステムのＯＡＰを記述対象としている。記述実験等を通じて、当該分野にもほぼ適用できる見通しを得た。適用性の向上に関する評価については、参考文献＜１＞で述べる。ここでは、現在評価中のため、サンプルプログラムから得た性能データ及びその要因を表４に示すに留める。

表４　本処理方式の評価

評価項目		評価値	要因
生成プログラム性能（手書ＣＯＢＯＬ比）	ＣＰＵ時間	１．２	制御（ＩＦ文の冗長性）
	使用メモリ量	１．３	バッファ管理の冗長性
コンパイル時間（対ＣＯＢＯＬコンパイラ比）		２	データフロー解析意味解析の複雑さ文字列操作
展開率		５	（生成ＣＯＢＯＬ／入力仕様）

６．おわりに

データ構造／データフローに着目したプログラムの処理手順導出機構と部品化機構を具備した適用域の広いジェネレータ実現方式について述べた。今後、フィールドテスト等の評価を通じて、本処理方式の有効性を明らかにして行く予定である。

今後の課題としては、以下がある。

①ジェネレータ／高級言語の共通の問題点として、仕様の形式化／高度化が進むことにより、誤りが生じた時、セマンティクスの複雑さに比例して、人間の対応が難しくなるという現象が派生する。入力支援／自動修正支援機能を設ける等により、利用容易性を改善して行くことが必要である。

②部品のテストは、ジェネレータのテストと同様に、テストプログラムを作成して、展開オブジェクトの確認及び実行テストを行なう必要がある。この一連の作業を、容易あるいは不要とする部品の検証支援方式について検討する必要がある。

［参考文献］

＜１＞稲田　他、「設計用言語（ＳＬ）の言語仕様」情処ソフトウェア工学研究会資料　４２－３

＜２＞J.W.Hughes、「A Formalization and Explication of the Michael Jackson Method of Program Design」 Software-Prac.&Exp Vol 9

＜３＞N.S.Prywes、「Automatic Generation of Computer Programs」 Advances in Computer,Vol16 (1977)

＜４＞ソフトウェア産業振興協会編、「知的プログラミング装置」

＜５＞佐藤　他、「データの構造および相関に基づくプログラム自動生成法の一提案」　昭和５６年度信学会全国大会

＜６＞岡本　他、「データ構造とオペレーションに基づくプログラム自動生成法の一提案」　情処学会昭和５８年度後期全国大会

＜７＞渡辺　他、「拡張可能な非手順型言語方式の一提案」　昭和５８年度信学会全国大会

＜８＞中村　他、「設計言語における部品構成法の検討」　情処学会昭和５９年度後期全国大会