プログラミング言語 20－3
（１９８９．２．１０）

雇用化の概念を持つ図式モジュラー言語 ：ＤＩＡＬ／Ｍ

田中　旭　　堀川　英明　　田山　典男
岩手大学　情報工学科

本論文では、部品化・再利用によるソフトウェア開発の立場から、雇用化という新しい概念を持った図式モジュール言語DIAL/Mを提案している。雇用化とは、抽象データ型を使用せずに、抽象データのような内部保存データを持った１個の履歴依存モジュールを、多数のユーザがそれぞれ自分に専用のモジュールとして使用できるようにする、モジュール共同再利用の簡便な方式である。DIAL/Mでは、初心者向きのわかり易い手続き型の言語を目指しており、モジュールの種類を２種類だけにして単純にし、図式構造エディタと対話型コンパイラにより能率よく支援する。本論文では、DIAL/Mの図式言語仕様、図式構造エディタと対話型コンパイラの結合試験結果、雇用化の仕組を述べている。

SIGPL 20-3　"Diagrammatic Modular Language supporting the Employing Method : DIAL/M"

Akira TANAKA, Hideaki HORIKAWA and Norio TAYAMA
Iwate University
Morioka-shi, 020, Japan

In this paper, we propose a new modular language capable of visual programming. The language DIAL/M supports a novel simple method for enabling each of plural users to reuse exclusively one history-dependent module in common by no means of the abstract data type. We call it "employing method".

As the DIAL/M aims at learning easy, it has only two kinds of module and makes use of the diagrammatic descriptions of data structure, control structure and functional hierachical structure. We also give an outline of the implementation on the DIAL/M system.

1．まえがき

近年、ソフトウェアの巨大化、複雑化が進むにつれて、いかにして品質の良いソフトウェアを能率よく作成し保守するかという問題が深刻になっている。[1]これに対処するために、プログラム構造化[2]や、段階的詳細化[3]、情報隠蔽[4]、データ抽象化[5]、オブジェクト指向[6]、その他多くの方法が提案されている[7][8]。ソフトウェアをモジュールに分割して、部品化・再利用する方法は、その内部をブラックボックスにするのでシステム全体の複雑度を軽減し、ソフトウェアの生産効率を上げるので、有力な方法として期待されている[8]~[11]。又、データ抽象化の概念も、モジュールを扱い易く整理するので、今後益々重要になると思われる。

従来の標準サブルーチンや共有サブルーチンのような既製のモジュール部品に対してこれを各ユーザが自分の専有のモジュールとして共同再利用する方法は、ソフトウェアの生産性や信頼性を向上させる上で極めて有効である。しかしながら、抽象データのような内部保存データを持った履歴依存モジュールに対しては、それを各ユーザが自分の専有のモジュールとして共同再利用しようとすると副作用が生じてしまう。そのために、通常は1個の履歴依存モジュールを多数のユーザがそれぞれ自分に専有のモジュールとして個別に利用するということはできない。しかしながらもし、このような内部保存データを持つ履歴依存モジュールに対しても、1個のモジュールを多数のユーザがそれぞれ自分に専有のモジュールとして個別に使用できるようにする共同再利用の簡便な方法が実現するならば、データ抽象化の概念に基づくモジュールの共同再利用が可能になるので非常に有効であろうと思われる。

オブジェクト指向におけるクラス（抽象データ型からインスタンス（実体）を生成する概念[12]~[16]はこの共同再利用を可能にするものであるが、本論文では、この抽象データ型の概念を用いないで、従来の局所性の概念を発展させた簡便な方法を検討している。

本論文では、このような観点から、抽象データのような内部保存データを持つ1個の履歴依存モジュールを、複数個の別々の各ユーザに専有のモジュールとして使用できるようにする共同再利用の簡便な方式を提案しており、それを支援する対話型図式モジュール言語ＤＩＡＬ／Ｍの言語仕様を提示し、試作した図式構造エディタの簡単な応答試験結果と、雇用化を実現する為の1つの仕組みを明示している。

この方式により、抽象データやその他の内部保存データを持つ履歴依存モジュールも、従来の標準サブルーチンや共有サブルーチンのように共同利用の部品として再利用することが可能になる。各ユーザがこの共同利用のモジュールを自分専属のモジュールであるかのように専有して使用できるようになるので、この概念をモジュールの「雇用化」と呼ぶ[21]。以下では、モジュール雇用化、雇用化再帰呼び出し、複製雇用化、切り出し合成雇用化の4種類の雇用化の形態を提示する。

本論文で提案する図式モジュール言語ＤＩＡＬ／Ｍは、初心者向きのわかり易い手続き型の言語を目指しているので、モジュールの種類を2つだけに単純にしている。又図式記述を採り入れて、制御構造の他にデータ構造やモジュール階層構造の図式化を図っている。更に図式構造エディタと対話型のコンパイラによる対話型処理系の構築を試みている。

2．モジュールと雇用化の概念

2．1　モジュールの設定

モジュールの種類として、例えばModula-2言語[15]では、定義モジュール、実現モジュール、局所モジュール、プログラムモジュール、プロシジャの5種類を設定しているが、初心者には必ずしも取扱い易いものではない。そこでDIAL/Mでは、システムを機能階層化によりモジュール分割設計でき、しかもプログラム単位の取扱いを統一化して容易にする観点から、次に2種類のモジュールだけを設定する。

① プログラムモジュール

PASCALのprogramやprocedure，functionと同じ目的で使用する1入口のモジュールであり、プログラム実行の詳細を記述する実行部を持つ。このモジュールは次に述べる可視制御性と存在制御性を持っている。内部保存データを持つことはできない。

② グループモジュール

抽象データを表現する場合や、ある機能的な共通性を持つ幾つかのモジュールをまとめる場合に使用する多入口または1入口の中間的なモジュールであり、実行部を持たない。このモジュールは可視制御性を持っているが、存在制御性を持っていない。内部保存データを持つこともできる。

ここで、モジュールの可視制御性とは、モジュールが、その中で局所的に宣言されている局所的オブジェクト(変数やモジュール)を外側に見せるか見せないかを制御することである。これは、コンパイルの仕方に影響を与える。これに対して、存在制御

性とは、モジュールが、その中で宣言されている局所的オブジェクトの生成および消滅を制御することである。これは、プログラムの実行に影響を与える。

プログラムモジュールの可視制御性は、内側のすべての局所的オブジェクトを外側に対して不可視にするという制御を行う。さらに、プログラムモジュールの存在制御性は、プログラムモジュールに実行が移った時に内側のすべての局所的オブジェクトを生成して、実行が離れた時にすべての局所的オブジェクトを消滅させるという制御を行う。従って内部保存データを持てないので履歴依存モジュールとはならない。それに対して、グループモジュールは可視制御性だけを持っており、内側の特定の局所的オブジェクトだけを外側に見せるという制御を行う。グループモジュールは局所的オブジェクトの存在（つまり生成と消滅）については制御しない。従って内部保存データを持てるので履歴依存モジュールと成りうる。このグループモジュールの存在を制御するのは、それを含む一番内側のプログラムモジュールである。

これらのモジュールは、分割コンパイルの単位であり、従ってオブジェクトコードのレベルで再利用ができる部品として扱われるものとする。

２．２　雇用化の概念

「雇用化」とは、１つのシステムを分担して開発する何人かの作成者の複数のモジュールMi(i=1,2,...,n)がある１個のモジュールR自体を又はそのモジュールを構成している種々の対象Oを共同で再利用する場合に、それらの各モジュールMiがモジュールR又は対象Oをそれぞれ自分に専有のものとして使用できるようにすること（専属化すること）である。ここで対象Oとは、モジュールRの中に含まれている個々のモジュールや、定数、型、変数、モジュールへのコール関係、変数データへのアクセス関係のことである。次にこの雇用化の形態を列記する。

①モジュール雇用化の概念

モジュール雇用化とは、モジュールの機能的包含関係において、あるモジュールXを実現するために既製のモジュールRを再利用する時、モジュールXがモジュールRを自分に専有のものとして使用するということであり、他のモジュールが同じRを共同で再利用していても全く別個なモジュールとして扱われるという概念である。

②雇用化再帰呼出し

モジュール雇用化の概念により、雇用化したプログラムモジュールの再帰呼出しが可能になる。これは、PASCAL言語のprocedureにおける再帰呼出しよりも柔軟な再帰呼出しとなっている。つまり、再帰的に呼出すモジュールを雇用化しているモジュールが、それぞれ専属に再帰呼び出しするモジュールの外部関係を設定して使用することができるようになる。

③複製雇用化

これは、同じ機能のモジュールを幾つか用いてプログラミングする場合に、そのモジュールを複数個複製してそれぞれを雇用化するという概念である。

④切出し合成雇用化の概念

これは、（実現する処理系とも関連するが）ディスプレイに描かれたモジュール関係図において任意の部分を切出して、切出した部分を雇用化し、他のモジュールの部分と組立て合成して使用するという概念である。これは、そのモジュールを構成している種々の対象（そのモジュールに含まれている個々のモジュール、又は定数、型、変数、モジュールへのコール関係、変数データへのアクセス関係）が雇用化されるということに起因している。又、モジュールを雇用化する時に、周囲との接続作業を容易にする為に、モジュールのコール関係やアクセス関係には、仮想のインタフェースを設定する。このようにして、切出し部分と外部との接続容易に行えるようになり、既製モジュールからの組立て合成の作業が効率的に行える。

３．ＤＩＡＬ／Ｍの図式言語仕様の概要

３．１　DIAL/Mにおけるモジュール機能階層図

DIAL/Mでのプログラム開発は、モジュールをいかに作成するかに関わる。DIAL/Mによるモジュール設計では、大きなシステムを段階的に分解していく通常の"機能階層化の概念'(8×9)'"に基づいて行うことができる。対象とするシステムは、中間的要素であるグループモジュールと、最終的要素で実行部を持つプログラムモジュールとに分解される。

DIAL/Mにおける２種類のモジュールは、この分解されたモジュールの間でのコール関係や共用変数データへのアクセス関係を図式で記述する関係設定部を持っている。この関係設定図をモジュールの分解に沿って継ぎ合わせると、図１のようになる。この図は、モジュールRの分解の過程を表わすとともに、Rの機能階層の構造をも表わしているので、Rのモジュール編成が理解し易い図となっている。この図をモジュール機能階層図と呼ぶ。

３．２　モジュールの構成

どのモジュールも、定数・型宣言部、変数宣言部、

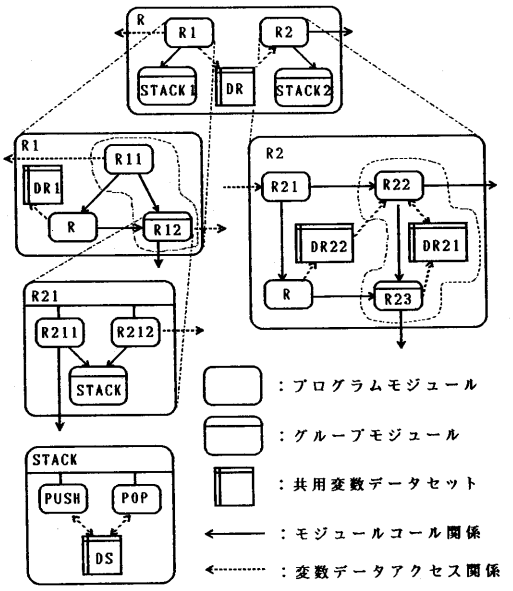
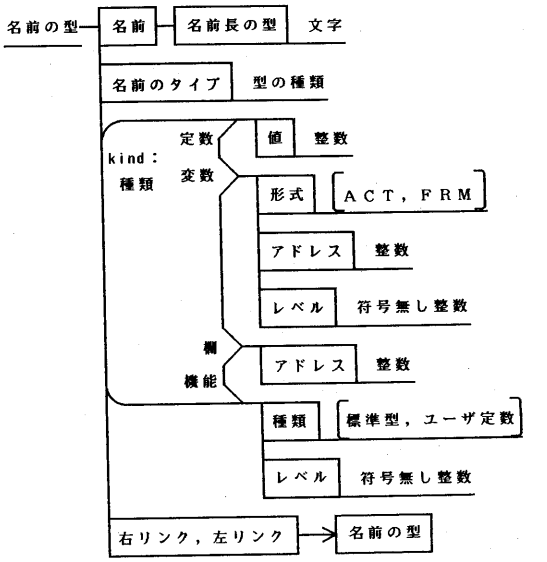


図１．モジュールＲの機能階層構造



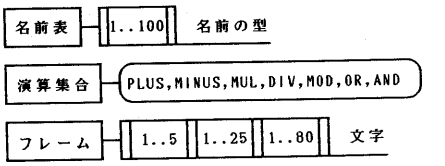


図２．データ構造の例

関係設定部を持つ。プログラムモジュールについてはさらに実行部も持つ。各部の記述順序は任意でよい。以下では、各部について簡単に説明し、図式表現を示す。

① 定数・型宣言部

この宣言部では、定数や型を宣言する。基本データ型としては、整数型、短数型、長整数型、符号なし整数型、符号なし短整数型、符号なし長整数型、実数型、長実数、文字型、論理型がある。基本データ型以外には、列挙型、部分範囲型、配列型、集合型、構造体型、可変構造体型、指示型がある。特に、基本データ型以外のデータ型については、表１に示す図式表現を用意しており、図２にその例を示す。この図をデータ構造図と呼ぶ。

② 変数宣言部

この宣言部では、変数を宣言する。変数宣言も型宣言と同様に図式表現を用意している。また、図式表現として変数データセットがあり、これはなんらかの共通性を持った幾つかの変数をまとめて扱う為に設けている。変数や定数に対する演算子としては、PASCAL言語と同様な演算子を設定している。さらに、実数と長実数以外の基本データ型に対して、C言語と同様なビット演算子（AND,OR,排他的OR,左シフト,右シフト,1の補数)を用意している。

③ 関係設定部

ここでは、3.1で述べたように分解したモジュールの間でのコール関係や変数データセットへのアクセス関係について図式表現を設定している。

④ 実行部

これは、PASCAL言語の実行部分に当たる部分で、処理の詳細を記述するところである。制御構造としては、連接、多重分岐選択、CASE分岐選択、前判定反復、中判定反復、後判定反復、指定回反復、反復脱出、非常脱出の９種類を設定する。これらに対して表１に示す図式表現を用意しおり、図３にその例を示す。この図を制御構造図と呼ぶ。

図式言語DIAL/Mの図式表現は、以上のモジュール機能階層図と、モジュール内のデータ構造図、制御構造図から成っている。先駆的なPAD[17]やHCP[18]，SPD[18]，YAC[20] 等の多くの木構造チャートと同様に次の特徴を持つ。

a) プログラムの流れを理解する時には、図の周辺をたどっていけばツリーウォークの順になぞっていけるし、不必要と思ったら途中から枝へのプログラム流れを無視して理解を進めてゆくこともできる。従来の文字表記によるプログラムでは、制御構造を文字で表現しているので、解りにくかった。（特に、選択や反復が何重にもネストしているときには、ネ

処理箱名	図式	意味
プログラムモジュール		プログラムモジュールを表わす。
グループモジュール		グループモジュールを表わす。
変数データセット		変数データセットを表わす。
関係		モジュールコール関係及び変数データアクセス関係を表わす。

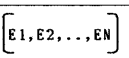
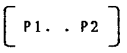
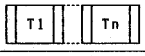
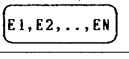
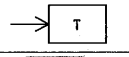
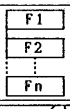
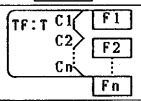
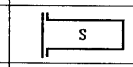
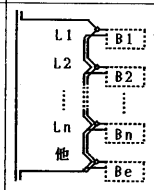
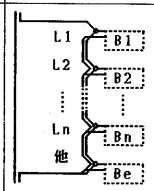
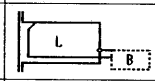
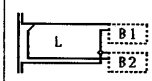
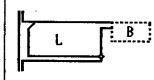
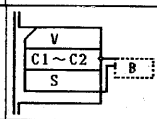
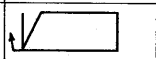
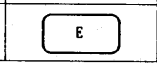
列挙型		要素がEiの列挙型を表わす。
部分範囲型		P1からP2までの部分範囲型を表わす。
配列型		Tiを添字の型とするN次元配列を表わす。
集合型		要素がEiの集合型を表わす。
指示型		T型の変数を示す指示型を表わす。
構造体型		n個のフィールドFnを持つ構造体を表わす。
可変構造体型		タグフィールドTFの可変構造体を表わす。 T:型　Cn:定数

表１．DIAL/Mの図式シンボル

処理箱名	図式	意味
連接		代入や呼び出し文の並びを逐次実行する。
多重分岐選択		論理式Liが真のところのブロックBiを実行する。全てが偽の場合はBeを実行する。
ＣＡＳＥ分岐選択		式Eの値に等しい定数Ciに対応するブロックBiを実行する。対応する定数がない場合は、Beを実行する。
前判定反復		論理式Lが真の間、ブロックBを実行する。
中判定反復		ブロックB1を実行してから論理式Lの評価を行ない、真ならばB2を実行しB1から同様に繰り返し実行する。
後判定反復		ブロックBを実行してから論理式Lの評価を行い、その後Lが真の間、Bを実行する。
指定回数反復		制御変数Vを定数C1からC2まで、増分Sで変化させながらブロックBの実行を行なう。
反復脱出		反復からの脱出を表わす。
復帰		式Eの値とともに呼び出し元へ戻る。

ストの範囲が解りづらかった） 図式表現ではより明確になり直感的で理解し易い。
b） 書き方による個人差が出にくくなる。

４．図式構造エディタと対話型コンパイラの試作

DIAL/Mの対話型処理系を実現するための第一段階として、DIAL/M言語仕様のサブセットを設定し、プログラムモジュールの実行部の図式構造エディタと対話型コンパイラの試作を行った。本章では、処理系の特長である図式構造エディタと対話型コンパイラについてその概要を述べ、DIAL/M処理系の対話性を調べる為に行った結合試験について述べる。

４．１　図式構造エディタ

試作した図式構造エディタは、DIAL/Mの制御構造図を図式編集するものであり、DIAL/Mの制御構造に沿った編集機能を提供する。プログラムモジュールの実行部は、処理箱をフロー線で結んだ処理フロー部と、実行部に局所的な変数を宣言する宣言部とで構成される。処理箱は処理の種類を表す箱枠と内部の処理テキストから構成される。但し今回の試作では、宣言部は図式ではなくPASCAL風のテキスト形式で入力を行う。従って試作した図式構造エディタの編集モードは、処理箱編集モード、処理テキスト編集モード、宣言テキスト編集モードから構成されている。

処理箱編集モードでは、制御構造図に対して図４

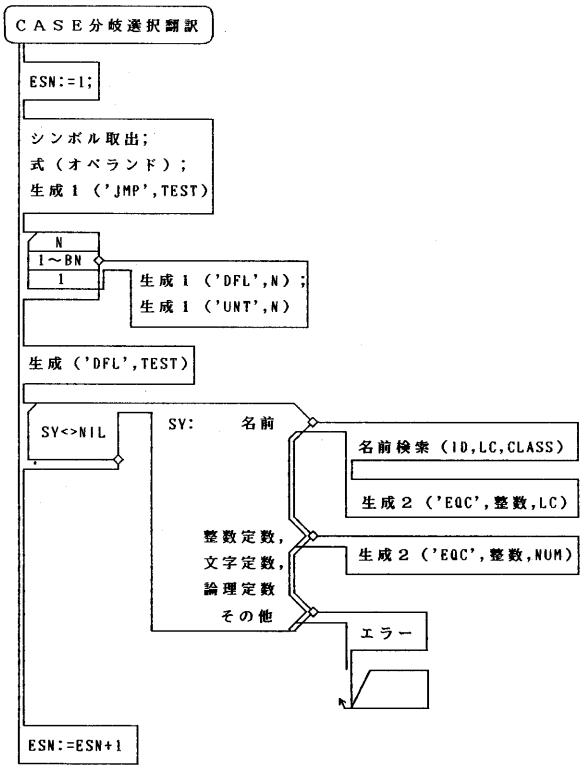


図３．制御構造の例

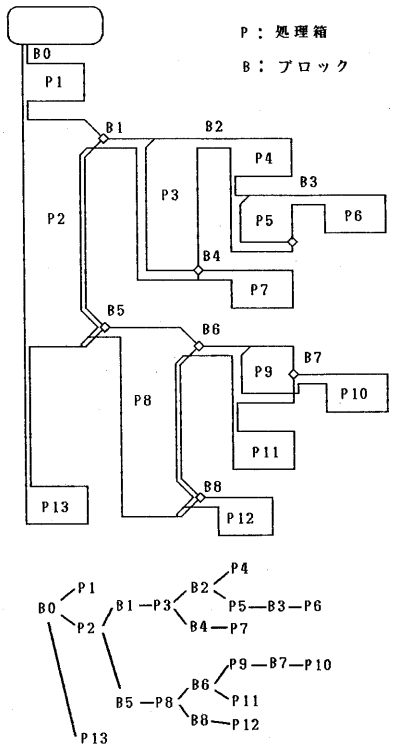


図４．制御構造木の例

に示すような処理箱とブロックの「制御構造木」を定義している。編集作業は、この制御構造木に基づいて進められる。例えば、処理箱やブロックの移動、コピーなどの編集処理は、制御構造木上の処理箱やブロックをルートとする部分木を単位として実行される。また、エディタが常に処理箱やブロックを編集の対象とするように設定するので、処理箱やブロックの不正な連結が起こらない。従ってユーザからは、常に論理的に意味のある処理箱やブロックが対象となり、扱いやすい。制御構造図の内部データ構造は、図５に示すような情報を持つ処理箱のセルとブロックのセルを制御構造木における関係に従ってポインタでつないだリスト構造となっている。

処理テキストと宣言テキストの編集モードでは、文法に則した案内メッセージを表示してユーザの編集を誘導したり、式や代入文、呼出し文の文法チェックや意味チェックするので、処理テキストや宣言テキストの文法的誤りが、排除される。

図式編集を円滑に行うために、各々の編集操作おいて制御構造図を変更するときには、処理箱の生成や再配置を自動的に行う機能を持たせており、ユーザに対して図面を描く負担を極力少なくするように配慮している。　図６に、この図式構造エディタにおける編集中の画面表示例を示す。

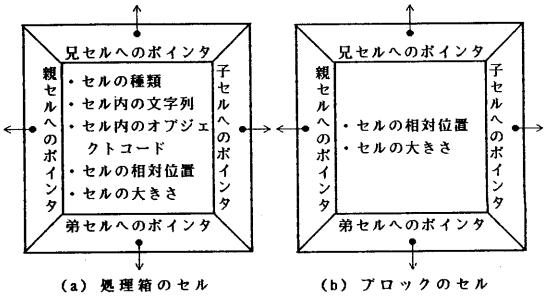


図５．内部データ構造のセル

４．２　対話型コンパイラ

処理テキスト編集モードでは、式や代入文、呼出し文に対して文法チェックや意味チェックを行うのであるが、ここでは更に、機械コードの生成も行うことを考えた。つまり、DIAL/Mにおける対話型コン

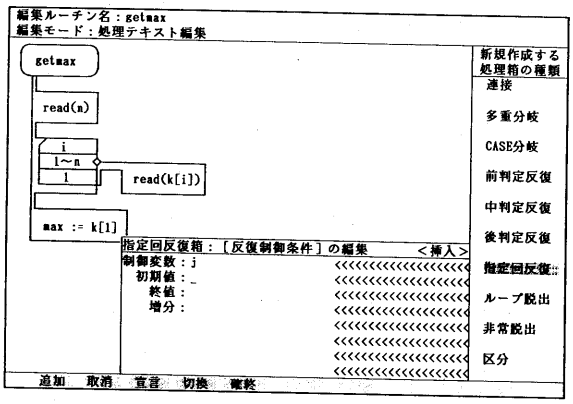


図６．画面表示例

パイル方式とは、処理箱単位に図式編集とコンパイルとを対話型で逐次的に交互に行おうとする、インクリメンタルなコンパイル方式である。このような対話型方式にすると、プログラムモジュールの実行部を対話型で入力し終えた時点で、各処理箱の機械コードが生成されている。従って制御構造木をツリーウォーク上になぞりながらそれらのコードを寄せ集めると実行形式のオブジェクトモジュールが出来るので従来の編集後にコンパイルを行うという作業が省ける。

しかし、この方式の問題点は、処理箱のコンパイル時にその時点での分岐先や変数のアドレスを決定したとすると、編集時には処理箱やブロックや変数が変更される可能性があるので、変更時にアドレスを修正しなければならなくなるということである。そこでDIAL/M処理系では、処理箱のコンパイル時には分岐先や変数のアドレスを未決定のままにして置いて、プログラムモジュールの実行部を入力し終えた時点で、まず変数のアドレスを決定し、次に処理箱のコードを寄せ集めながら、分岐先のアドレスを決定し、求まったアドレスを未決定になっているコードに埋め込む処理を行うことによりオブジェクトモジュールを生成している。この処理をギャザーと呼ぶ。

４．３　結合試験

この図式構造エディタと対話型コンパイラとは、パーソナルコンピュータＮＥＣ－ＰＣ９８０１ＶＭ２１のＭＳ－ＤＯＳ上で開発している。システム記述言語には、Microsoft社のMS-C言語を使用している。以下に図式制御構造エディタと対話型コンパイラの大まかな応答時間とギャザーの処理時間を示す。測定は、ＶＭ２１をクロック周波数10MHZで動作させて行った。

① 図式構造エディタの応答時間として、ペーストの処理時間を示す。

・連接箱　40個のペースト処理時間＝約2.4秒
・多重分岐箱　20個のペースト処理時間＝約2.3秒
・処理箱　47個（連接箱 29個、多重分岐箱 16個、後判定反復箱 2個）のペースト処理時間＝約2.7秒

② 対話型コンパイラの応答時間

次の代入文 50行のコンパイル時間＝約4.3秒

```
v:=((a1[10*e1[6]]-b)/(c1[c*d]+d)+(e-f)*(g+h*i))      /(j*k-1*m);
```

③ ギャザー処理時間

処理箱 46個（連接箱 28個、多重分岐箱 16個、後判定反復箱 2個）のギャザー処理時間＝約2.5秒、出力ステップ数＝694ステップ

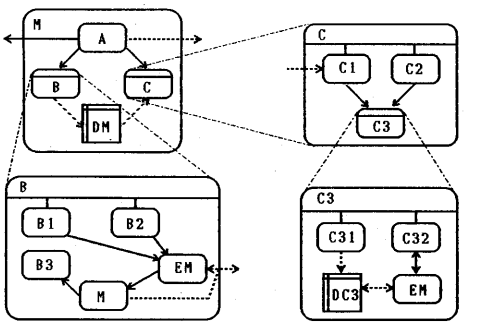
処理箱の処理テキストの大きさや編集中に再配置される処理箱の数などにより、応答時間は変動するが、大まかな目安としては、例えば、5行の代入文をもつ連接箱を新たに入力し、その連接箱の追加により30個の処理箱の配置替えが生じた場合には、コンパイルから表示までの応答時間を測定してみると2.1秒であった。プログラム制御構造を編集している時の処理系の応答時間として、まず十分であると思われる。

５．　雇用化のインプリメント

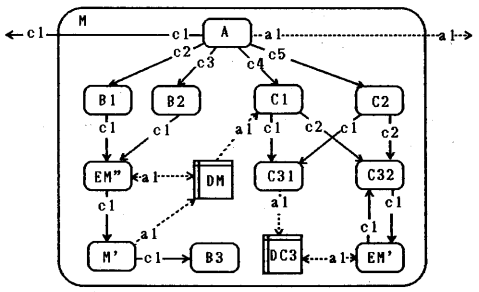
ここでは、DIAL/M処理系によってユーザプログラムがどのようにインプリメントされるのかを、具体例により示す。その前にモジュールの可視制御性と存在制御性について若干の考察をする。

５．１　グループモジュール展開

前述のように、プログラムモジュールが実行部を持ち、グループモジュールは実行部を持たない。つまり実際にコンパイル後にオブジェクトコードとして実現されるのは、プログラムモジュールだけである。さらに、プログラムモジュールだけが存在制御性を持ち、グループモジュールは持たない。従って、文法的に誤りのないユーザプログラムでは、グループモジュールの可視制御性が満たされているので、そのオブジェクトコード生成においては、グループモジュールを考慮せずにプログラムモジュールだけを考えれば良い。これは、例えば図７(a)のような



(a) モジュールMの機能階層構造



(b) グループモジュール展開

図7．モジュールMのグループモジュール展開

モジュール関係図を持つプログラムモジュールのMをインプリメントする時には、図7(a)に示すBやCのような雇用化されているグループモジュールや、さらにそのグループモジュールの中のC3でのように雇用化されているグループモジュールを、図7(b)のようにマクロ的に展開して考えればよいということである。このとき、同一のモジュールが複数個現れたり、展開の対象であるプログラムモジュールと同一のモジュールが現れたときには、それぞれを別々のモジュールとして扱わねばならない。例えば図7(a)のモジュールBやC3で雇用化しているモジュールEMは2個あるので、図7(b)では モジュールEM'やEM"として区別している。また、図7(a)のモジュールBでは展開の対象であるモジュールMを再帰的に呼出しているので、図7(b)ではモジュールM'として表している。さらに図7(b)では、説明上、モジュールコール関係や変数データアクセス関係を表す矢印をc1,c2,...やa1,a2,..で表している。これは後で述べる図9におけるc1,c2,..やa1,a2,..と対応するものである。そこで以下では、対象とするプログラムモジュールにおいて雇用化しているグループモジュールをすべて展開しプログラムモジュールだけにすることをグループモジュール展開と呼び、図7(b)をプログラムモジュールMのグループモジュール展開図と呼ぶ。

5．2　雇用化を実現する手法

さて、プログラムモジュールをインプリメントする時の大きな問題点は、雇用化の実現にある。雇用化されるモジュールは、雇用化する各モジュールの元でそれぞれに専有して使用されるので、その雇用化されるモジュールが呼出すべき外部モジュールの呼出し先や、そのモジュールがアクセスすべき外部変数のアクセス先が、一定にはならないことに留意しなければならない。(ここで、外部モジュールとはモジュールコール関係により指定されるモジュールのことで、外部変数とは、変数データアクセス関係により指定される変数データセット内の変数のことである) 例えば、図7(a)のモジュールBおよびC3の中でのモジュールEMは、外部変数へのアクセス先や外部モジュールの呼び出し先がそれぞれ異なっている。また、前掲の図1では、モジュールR1,R2がモジュールRを雇用化再帰呼出しを行っている。図1においてモジュールR1が雇用化しているモジュールRは、変数データセットDR1内の変数にアクセスしている。ここで、モジュールR1内のモジュールRが再帰的に呼出され、さらに、モジュールR1が呼出されると、新たにDR1内の変数が生成される。この時、モジュールRは新たに生成されたDR1内の変数にアクセスしなければならない。つまり、モジュールRの外部変数のアクセス先を再帰呼出しの度に動的に変えなければならないのである。そこでこの雇用化を実現する為に本論文では、プログラムモジュールの呼出し時に、引数と一緒に、そのモジュールが呼出すべき外部モジュールの呼出し先アドレスや、そのモジュールがアクセスすべき外部変数のアクセス先アドレスの情報をも引き渡すという手法を明らかにする。

5．3　インプリメントの具体例

それでは、プログラムモジュールをインプリメントする具体例を図8、9により示す。図8は、図7(b)のモジュールMについてのグループモジュール展開図に対して生成されるオブジェクトコードの構成をC言語風の記述で表している。図8の変数データセット領域部は、グループモジュール展開図に現れる変数データセットDM,DC3の為の領域である。モジュール外部関係領域部は、グループモジュール展開図に現れる各プログラムモジュール毎に領域が取られる。その領域には各プログラムモジュールが呼出すべき外部モジュールのアドレスやアクセスすべき外部変数のアドレスが格納される。次の、モジュ

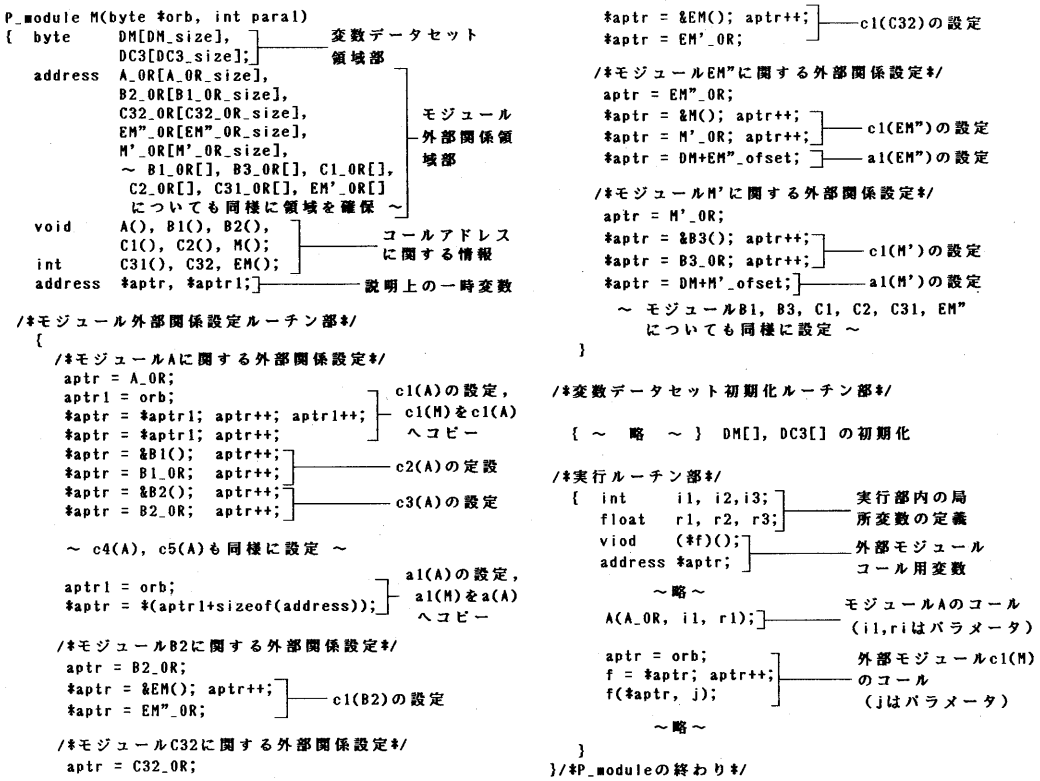


図８．モジュールMのオブジェクトコードの構成

ールのコールアドレスに関する情報は、処理系のリンカーに引き渡されて、各アドレスが確定する。以下のモジュール外部関係設定ルーチン部では、各プログラムモジュール毎に取られている外部関係領域の各内容を設定する。例えば、モジュールAについて述べると、図８のモジュール外部関係領域部のモジュールAに関する外部関係領域A_OR[ ]は、図９のスタックにおけるAに関する外部関係領域と対応している。そして、図８のモジュール外部関係設定ルーチン部におけるモジュールAに関する外部関係の設定操作により、図９のスタックに示すようにAに関する外部関係領域の内容が決定される。次の変数データセット初期化ルーチン部では、各変数データセット内の変数を初期化する。最後に実行ルーチン部では、プログラムモジュールの実行部をインプリメントする。

図９は、モジュールMが雇用化しているモジュールAを呼出したときの実行時のスタック状態を表している。図８の実行ルーチン部におけるモジュールAの呼び出しで、i1,r1という引数の他にAの外部関係領域のベースアドレスA_OR をも引き渡していることに注目されたい。これは、プログラムモジュールAを実行する時に、モジュールAの外部関係領域の情報を使用するためである。さらに、モジュールAからモジュールB1を呼出すときには、図９のスタックにおいて、モジュールAの外部関係領域のc2(A)の情報を使用する。このモジュールコール関係c2(A)には、モジュールB1の呼出 し先アドレス&B1() とB1の外部関係領域のベースアドレスB1_OR とが格納されている。また、モジュールB2やC32に関する外部関係領域の内容から解るように、BやC32からモジュールEM'やEM"を呼出す時には、それぞれ別個の外部関係領域EM'_ORやEM"_ORをモジュールEMに引き渡す。

同様にモジュールEM"からモジュールM'を呼出す時には、M'の外部関係領域M'_ORをモジュールMに引き渡す。

以上のようにこの論文では、雇用化の概念をインプリメントする為に、次のような手法を提示している。それは、まずグループモジュール展開により現れる各プログラムモジュール毎に外部関係領域を確

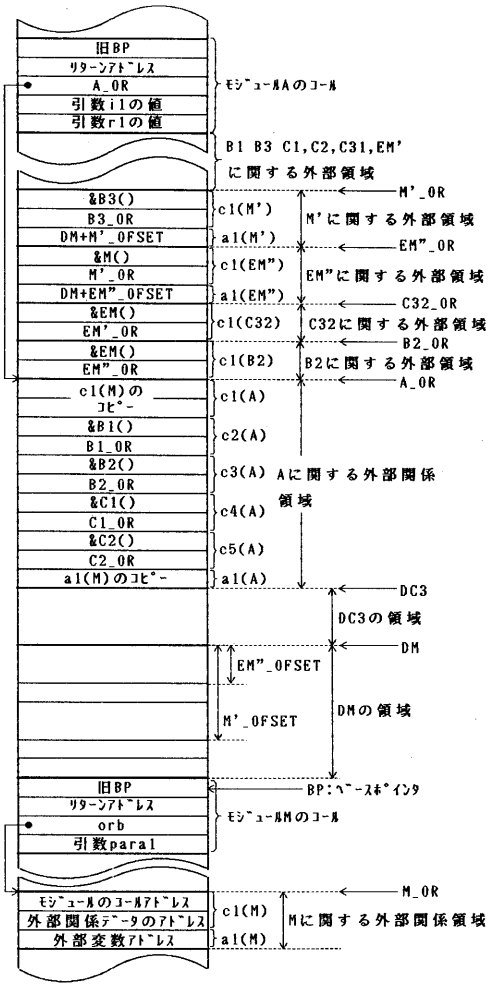


図9．モジュールMのスタックの状態

保して、各プログラムモジュールが呼出すべき外部モジュールの呼出し先アドレスやアクセスすべき外部変数のアクセス先アドレスをそれらの各領域に設定する。そしてそのモジュールを呼出す時にその外部関係領域のベースアドレスを引き渡す、というものである。

このモジュール呼出し時に外部対象のアドレス情報を引き渡す手法により、雇用化の概念や柔軟な切出し合成等のインプリメントが可能になっている。これを外部アドレス情報の呼出し時渡しと呼ぶ。

6．むすび

本論文では、既製のモジュールを共同で再利用する為の「雇用化」と呼ぶ概念を持った対話型図式モジュール言語DIAL/Mを提案しており、その図式言語仕様と、対話型処理系の結合試験の結果と、雇用化を実現する1つの仕組みを明示した。

雇用化は、従来の標準サブルーチンを共同再利用するように、抽象データやその他の内部保存データを持つ履歴依存モジュールをも多数のユーザが共同再利用できるようにする簡便な方法である。雇用化の形態として、モジュール雇用化、雇用化再帰呼出し、複製雇用化、切出し合成雇用化の4種類があり、特に切出し合成雇用化は、モジュール切出し部分の外部との接続を容易にするので既製モジュールからの切出し合成に有効と思われる。

図式言語DIAL/Mでは、図式構造エディタと対話型コンパイラから成る対話型処理系を構成しており、その対話性を調べる結合試験を行ったところ、応答時間は十分であることがわかった。

本文で示した雇用化の実現手法は、プログラムモジュールの呼出し時にその外部関係領域を設定する動的な手法であるが、オーバーヘッドが大きいという欠点を持っている。今後の課題として、実行効率の良い雇用化の実現手法を開発することが重要である。

文　献

(1) 特集："ソフトウェア工学の現状と動向"，情報処理，Vol.28，No.7，PP.844-938(1987).
(2) O.J.dahl，E.W.Dijkstra，C.A.R.Hoare："Structured Programming，Academic press，London(1972)
(3) N.Wirth：Program development by stepwise refinement，Commun.ACM，14，4，P.221(1971).
(4) D.L.Parnas："On the criteria to be used in decomposing systems into modules"，Commun.ACM，15，12，P.1053(1972).
(5) B.H.Liskov，et al.："Abstraction mechanisms in CLU"，Commun.ACM，20，8，P.564(1977).
(6) 大特集："オブジェクト指向プログラミング"，情報処理，Vol.29，No.4(1988).
(7) 大野　編："新しい時代のソフトウェア"，bit臨時増刊，No.5，共立出版(昭59).
(8) 藤野，花田："ソフトウェア生産技術"，電子通信学会，P.67-75(昭60).
(9) 紫合，岩元，藤林："統一的設計方法論に基づくソフトウェア設計システム"，情報処理，Vol.

21, No.5, PP.528-538(1980).
(10) 中所 : プログラムのモジュール化技法, 信学会誌, Vol.62, No.1, PP.91-93(1979).
(11) 片岡 : "ソフトウェア・モデリング", 日科技連, PP.3-17(1988).
(12) B.H.Liskov, et al. : "CLU Reference Manual", Comput. Structures Group Memo 161, Lab. for Comuter Science, Massachusetts Inst. of Tech.(1978).
(13) R.Nakajima, T.Yuasa, et al. : "The IOTA Programming System", Springer Lecture Notes in Computer Science, Vol.160(1983).
(14) A.Goldberg, D.Robson : "Smalltalk-80, The Language and Its Implementation", Addison Wesley(1983).
(15) N.wirth : "Programming in Modula-2", SpringerVerlag(1985).
(16) U.S.Department of Defense : "Reference manual for the Ada programming Language", ANSI/MILSTD-1815A-1983(1983).
(17) 二村, 他 : "PADによるプログラムの設計および作成", 情報処理学会論文誌, Vol.21, No.4 (昭55-7).
(18) 花田, 他 : "コンパクトチャートを用いたプログラム設計法", 情報処理学会論文誌, Vol. 22, No.1,pp.44-50(1981)
(19) M.Azuma et al. : SPD: A Humanized Documentation Technolory, IEEE Trans. Softw. Eng., Vol.SE-11, No.9, pp.945-953(1985).
(20) 村上, 他 : "YACⅡによる設計仕様のOA化と表記法考察", 情報処理学会第27回全国大会予稿集(Ⅰ), pp.487-488, 情報処理学会, 1983.
(21) 田中, 田山 : "抽象データの雇用化による共同再利用", 情報処理学会東北支部研究会, 63-2-8(昭63)