設計情報を付加した構造化仕様記述法

1G－8

奥山哲史，井上藤男，川崎健史，荻本浩三
株式会社　島津製作所

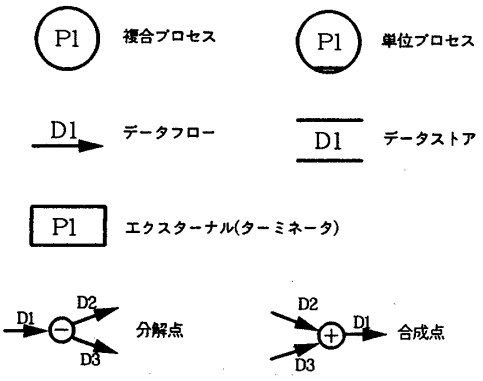
１．はじめに

現在、ソフトウエア開発において構造化分析と呼ばれる手法が注目されている。これはデータの流れを中心にしたデータフローダイアグラムを描き、その図をもとにして要求の正しさを確認する方法である[1]。この手法を用いることにより、システムの要求分析段階での誤りを少なくできることが認められており、最近ではこれをコンピュータ上で支援するＣＡＳＥツールも普及しつつある。しかしこれらのツールが支援できる範囲はまだ限られており、要求分析から設計への自動化も実現していない。

本稿では、データフローダイアグラムの形式性を高めるとともに、制御として必要な情報を加えることによって、分析から設計を連続して行う方法について述べる。

２．設計を含めた要求定義

構造化分析では、システムの機能を処理とデータに分けて段階的に細分化していくことで、階層を作りながら要求の定義を行う。しかし、この階層は要求の定義が完了して設計に移る際に、一度分解して改めて組み直す必要がある。そこで、データフローダイアグラムから直接処理手順を抽出することで、分析時に得られた階層構造をそのまま設計に利用する方法を提案する。そのためにまず、各シンボルが実際のプログラムにおいてどの様な実体に対応するかを次のように定める。



（図１）　データフローダイアグラムのシンボル

（１）プロセス

データを処理する機能単位であるプロセスは、下位のデータフローダイアグラムへ展開される「複合プロセス」と、それ以上展開できない「単位プロセス」とに分けて扱う。プロセスはプログラム上ではサブルーチンに対応させ、プロセスの内部と外部をつなぐデータフロー線は引数とみなす。

（２）データフロー

データフローは処理されるデータが流れている間のみ有効な変数として扱う。

（３）データストア

データストアは、集合としてのデータを蓄え、その要素を出し入れすることができる変数に割り付ける。この変数は属するデータフローダイアグラム内で局所的であり、その図を出ると値は無効となる。また、データストアには初期値を設定することができる。

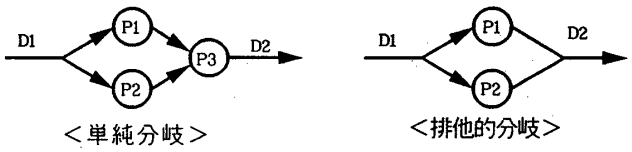
（４）エクスターナル（ターミネータ）

構造化分析では、エクスターナル(ターミネータ)はシステム外部の実体を表すが、ここではシステム外部とのインターフェイスを含む単位プロセスとして扱う。

（５）データの分岐と合流

データが分岐する場合、単純に2方向へ分かれる場合と、条件によって排他的に分けられる場合の2種類の意味がありえる。この区別は、分岐したデータがどの様に合流するかによって判断する。また、集合型のデータから一部の要素を分解したり、逆

（図２）　集合データからの要素の取り出し



（図３）　単純分岐と排他的分岐

A method of Structured specification with design information
Tetsushi OKUYAMA, Fujio INOUE, Takefumi KAWASAKI, Kouzou OGIMOTO
SHIMADZU CORPORATION

に合成したりする場合には、新しいシンボルである「分解点」、「合成点」を用いる。

（６）データ定義

データの定義は、データ構造の表現に優れているジャクソン法を基本とし、その中で連接と反復のみを用いる。連接と反復はそれぞれレコード型と配列型として扱う。データ定義は型定義であるが、データフロー線そのものが変数となるので、設計者は変数定義を意識しなくてもよい。

３．プロセスの動作順序

以上のようにシンボルの意味を限定すると、十分な正確さでデータの流れを指示することができるが、さらにプロセスの動作順序を決定するため、次のような規則を定める。

１）　プロセスは逐次的に実行される。

２）　プロセスは入力データが揃うと実行可能となる。

３）　実行後は出力データがすべて生成されている。

この規則により、データフロー線は前後のプロセスの実行順序を決める要素となり、プロセス間の半順序が定まる。この半順序をソートとすることによって、何通りかの実行可能な順序列が求まる。プログラムコードへの変換時には、この順序列の中から選択して適当なものを用いる。

データストアなどによって順序関係が分断される場合があるが、このような実行順序が不定となる部分については図5のようにプロセスフローダイアグラムによって制御情報を加える。このように、制御情報をデータフローダイアグラムに追加せずに別に記述することで、構造化プログラミングにおける制御はすべて表すことができる。

４．仕様の検証

次のような仕様の文法誤りは、コンピュータによってチェックできるため仕様のバグを減少させることができる。

１）　定義もれ、２重定義、名前のミス。

２）　プロセスへ入出力するデータの整合性。

３）　データフロー線の接続の誤り。

４）　実行不可能な順序列の有無。

検証機能を仕様記述エディタに組み込むことにより、効率良く設計作業を進められる。

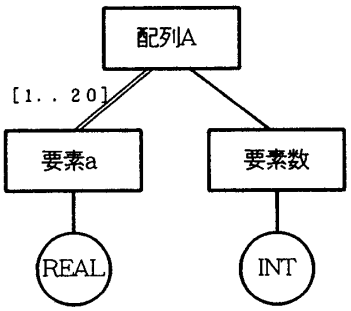
５．まとめ

以上のような手法に基づけば、要求定義から設計までを連続した作業で行うことができ、さらに作成した仕様を機械的にプログラムコードへ変換することが可能となる。

現在この手法に基づいた構造化設計支援システムを開発中である。

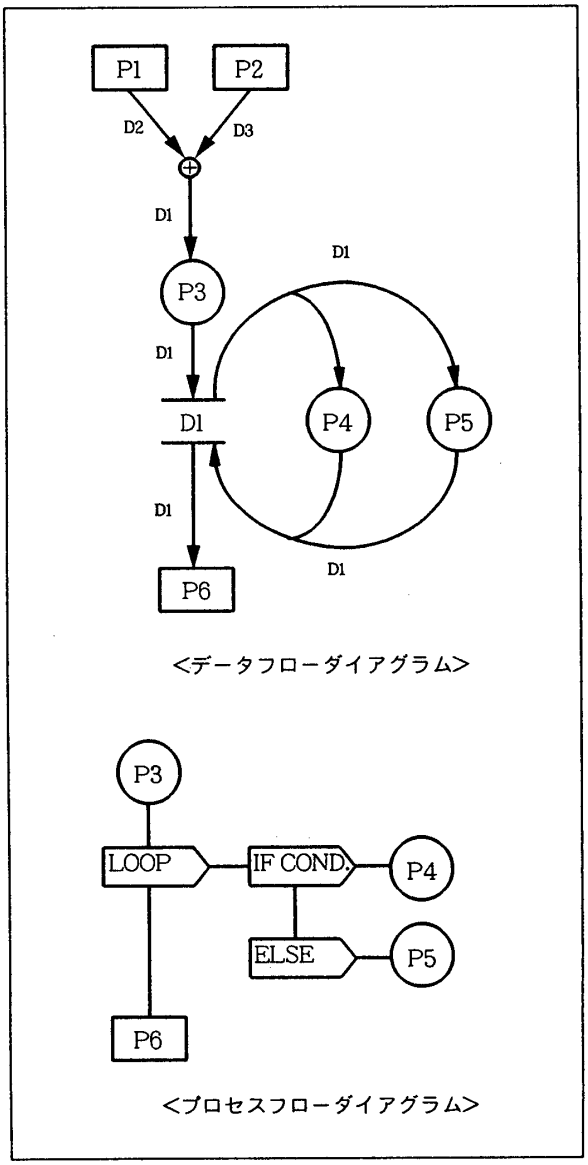
[参考文献]

[1] Tom DeMarco著、　高梨、黒田監訳
「構造化分析とシステム仕様」日経マグロウヒル社、1987



配列Ａ：：要素ａ［１．．２０］＋要素数；
要素ａ：：ＲＥＡＬ；
要素数：：ＩＮＴ；

（図４）　データ定義の例



（図５）　仕様記述の例