ソフトウエア工学 16－1
（1980. 11. 17）

DF－COBOL

三井造船株式会社
明光　英樹　　石田　秀信　　大西　通雄

1. まえがき

コンピュータのソフトウエア，ハードウエアの両方において，最近注目されているものの一つにデータ・フロー（Data Flow）がある。データ・フローの注目される理由は，それがデータ処理の内容（仕様）の表現に優れた方法であることと同時に，複数のプロセッサを備えた並列演算コンピュータの論理的基礎を与えることにある[1][2]。

筆者は，ビジネス・システムのデータ処理内容の表現方法を検討した結果，データ・フローによる表現に到達し，簡潔で疑問の余地の少ないデータ処理内容の表現方法を開発した。さらに，その表現を用いると，汎用性が高く，かつ無駄のないプログラムを自動的にコンピュータで生成できることがわかったため，プログラム・ジェネレータを開発し，DF－COBOLと名づけた。

2. 従来の基本的な考え方

2.1 従来のコンピュータの枠組

フォン・ノイマンが与えた従来のコンピュータのハードウエアとソフトウエアの枠組では，ハードウエアは一つのCPUを備えており，目的とする計算結果を得るために，CPUが実行すべき命令の順序を規定した集まりとして，プログラムが与えられる。

その枠組の中では，CPUが一つであるため，ある時点においては一つの命令しか実行されない。このため，プログラムは必然的に命令が一本の帯状になった形式のものとなる。すなわち，CPUは，プログラムの先頭の命令から順次実行して行き，ブランチ命令があれば，帯の別の位置に制御が移動し，その位置から，また順次命令を実行して行く。

このような形式のプログラムにおける主要な関心事は，命令の実行順序（手順）である。この意味において，従来の考え方は，手順中心あるいはコントロール・フロー中心といえる。

2.2 手順型フローの問題点

手順型フローには，二つの大きな問題点がある。一つは，それが複数のプロセッサを備えた並列演算の処理ロジックに対応できないことである。もう一つは，それがデータ処理内容の表現には不十分であることである。

LSI技術等ハードウエアの進歩による処理スピードの向上は，目覚ましいものがあるが，それ以上に処理スピードを要求する計算処理のニーズは増大している。パターン認識，画像処理あるいは天気予報のための大規模な偏微分方程式を解くのに，ますます処理スピードの向上が要求されつつある。技術の進歩が新しいニーズを作り出しているといえる。

処理スピードの向上を実現する方法として，プロセッサを複数備えた並列演算コンピュータが考えられるのは，自然の流れであるが，従来の手順型フローがそれに対応できないのは，手順型フローが一つのCPUを対象としたものであるため，当然である。

手順型フローの第二の問題点は，それが目的とする計算を実現するために，多く存在する方法の一つを表現するに過ぎない点に由来する。例えば，$z=F_1(a,b)+F_2(c,d)$ の計算を手順型フローで表現すると，図1に示すとおり，三とおりの表現が可能である。手順フローにおいては手順が重要な意味を持つが，この例では，その手順が必ずしも必然ではないという矛盾を内包している。さらに，方法を表現するに過ぎないという点より，手順型フローは処理の内容（目的）の表現には必ずしも適さないといえる。最近のソフトウェアはますます大型化しており，また，品質の良いソフトウェアの開発には，その仕様を誤解なく適確に表現する方法が不可欠であるが，この点に関して手順型フローは満足できる方法とはいえない。

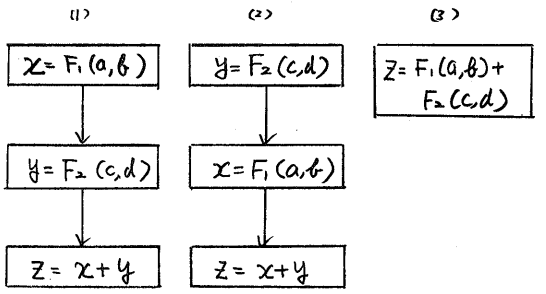

図1 $z=F_1(a,b)+F_2(c,d)$ の手順フローによる表現

3. データ・フロー

データ・フローは，従来の手順型フローの欠点を解決するものとして最近注目され，ハードウェア，ソフトウェアの両面より盛んに研究されている[1]。

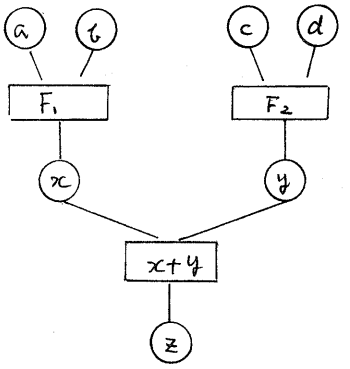

図2 $z=F_1(a,b)+F_2(c,d)$ のデータ・フローによる表現

3.1 データ中心の考え方

データ・フローの手順型フローとの最も重要な相異は，手順中心ではなく，データを中心に考える点にある。

図1の例をデータ・フローで表現すると，図2のようになる。両図の重要な相異は，図1は手順の流れを，図2はデータの流れをそれぞれ表現している点と，図2での表現はひとおりである点とにある。データの流れを表現するということは，データの相互関係を明確にすることである。

また，図1の手順の流れは，計算順序を固定しているのに対して，図2のデータの流れは，計算順序を規定しているが，固定はしていない。図2の表現では，F_1 と F_2 の計算は，どちらから先に行ってもよいし，同時に行ってもよい。

3.2 並列演算の基礎としてのデータ・フロー

データ・フローは，データの相互関係を示すものであり，計算順序を規定するが固定はしない。したがって，条件の揃った計算はいつ実行してもよいし，また同時（並列）に実行してもよい。

計算実行の条件は，ある計算の実行は，その入口変数（入力データ）がすべて揃っていて，出力変数（出力データ）がすべてなければよいという，ごく単純な表現も足りる。

このような単純な表現で計算条件を表現でき，条件を満たす計算は同時に並行

して実行してもよいということより，プロセッサを複数備えた並列演算コンピュータには，データ・フローがまことに好都合な考え方であるといえる。

すなわち，複数プロセッサに並列演算を行わせるには，条件を満たす演算を調べるか，あるいは割込みを受け，それを実行させるように，各プロセッサを起動させる機構（ソフトウェア，ハードウェアあるいは両方を組み合せたもの）を作ればよい。プロセッサが一つの場合には，並列演算をさせることはできないが，プログラムは単純な表現で記述することができる。

DF-COBOLは，CPUが一つの特別なケースで，ソフトウエアで上述の機構を組み込んだものである。

3.3 表現方法としてのデータ・フロー

データ処理は，データを組み合わせ，変換・加工・集約し，新しいデータを作り出すことである。データの相互関係を表現するデータ・フローは，その意味で，データ処理内容の最も素直な表現方法である。

事務処理アプリケーションにおいては，データ処理の概要を表現するために，データの流れを用いた表現（図3）が従来から用いられていたが，必ずしも十分なものではなかった。その理由は，従来の表現方法は必ずしもデータの相互関係を十分明確にしていなかった点にある。

筆者のいうデータ・フローによる表現（DFチャート）では，データの相互関係をあいまいさのなくなるまで分解して示すため，その表現を見れば，データ処理内容を一見して捉えることができる。

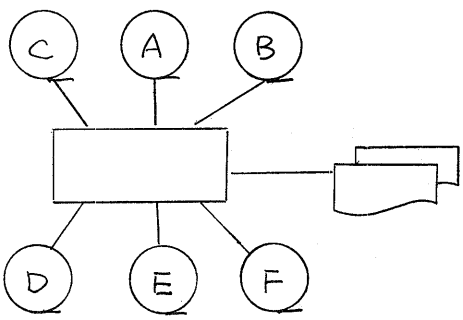

図3 従来のI/Oフロー

4. DF-COBOLの概要

DF-COBOLは，順次編成ファイル，索引順次ファイルを用いるバッチ処理向に開発されたが，その後VSAMファイル，データ・ベースも扱えるように拡張された。そして，現在オンライン機能の開発が着手された。

4.1 機能

大きく二つの機能を持っている。

(1) データ処理内容の表現

データ処理内容の表現のため開発されたDFチャートを用いて，データ処理内容を表現する。これにより，少量の簡潔な記述で，適確に処理内容を表現できる。

(2) DF-COBOL言語

DFチャートとオーサーの対応関係を持つ要素から成る言語であり，これで書かれたプログラムを入力し，COBOLプログラムを生成する。この言語を用いることにより，コーディング量は非常に少くなるとともに，理解しやすいソース・プログラムでシステムのメンテナンスが可能となる。

4.2 DFチャート

表現方法は重要である。的確な方法を用いれば，簡潔な記述により，良質のコミニケーションが得られる。

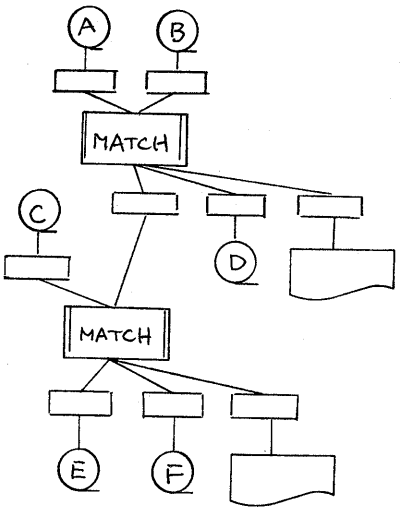

図4　DFチャート

データ処理の内容を簡潔に誤解の余地なく表現する方法は，どんな方法であろうか。

プログラムの処理内容の説明のために，図3のような表現が従来からよく用いられている。この例では，フロー・チャートは，単に入出力を示すのみで，その相互関係は別に文章で記述される。一般に，データの相互関係を文章で記述するのはやさしくはなく，また，書かれた文章を読んでも時間がかかるうえ，必ずしも正しく理解されるとは限らない。

もっと表現力を持つ方法はないであろうか。図4は，図3に対応するDFチャートによる表現である。DFチャートにおいては，データの相互関係を明確にするため，データの相互関係に関するあいまいさのなくなるまで，処理を分解して表現される。

DFチャートを書く上での基本的な規則は，次のとおりである。

- 直接関係を持つデータのみが一つの処理の入力になるまで処理を分解する。
- 入力が一つであっても，入力と出力が全く同じでない処理は一つの処理とする。例えば，サマリ，フォーマット変換，データの抜き出しがそれである。
- 同じデータが同時に別の処理の入力とならないようにするため，必要があれば，コピーをする。

このようにして，データの相互関係にあいまいさがなくなるまで処理を分解して，個々の処理に簡単な表題を付けることにより，作成されたDFチャートを一見するだけで，データ処理の概要は，ほとんど誤解の余地なく理解することができる。

DFチャート上のDA（□）は，後にDF-COBOLでのコーディングのために付けられているもので，メイン・メモリ中に取られるデータの変数名を示すとともに，個々の処理の境界でもある。さらに一つのプログラムのなかに含まれる種々のモジュールのインターフェイスとしての役割を持っている。

4.3　DF-COBOL言語

DF-COBOL言語は，考え方の基本にデータ・フローを応用したものであり，マクロ機能を利用してCOBOLプログラムを生成するものである。さらに，補助言語として部分的な処理の記述のために，COBOL言語を使えるようにしている。これが，この言語にDF-COBOLと名付けた理由である。

DF-COBOLは，COBOLプログラムを中間的に生成するものであるが，これを用いて開発されたシステムのメインテナンスは，DF-COBOLで行い，生成されたCOBOLプログラムは一時的なものである。

DF-COBOLは完全なモジュール方式となっており，モジュールの組み合わせや組み合せの数において，コンピュータの規模からの制約以外，なんらの制約もない。

DF-COBOLの要素を大別するとつぎのとおりである。（表1）

4.3.1 モジュール

DF4チャートの要素と一対一に対応するものである。入出力モジュールとして、¥INPUT, ¥OUTPUT, ¥LIST, ¥ISIN等多くのものが用意されているが、それぞれはいろいろのファイル編成に対応している。これらの入出力モジュールを用いることにより、通常の処理においては、ファイルのOPEN, CLOSE, READ, WRITE等の入出力命令をユーザは全く書く必要がない。これ等の命令は、DF-COBOLが必要なタイミングで実行されるように、自動的に生成する。

表1 DF-COBOLの主要な要素

要素		内容
定義	¥DA	データ・エリアの定義
	¥UA	ユーザ・エリアの定義
	¥TABLEA	ルック・アップ・テーブルの定義
モジュール	¥INPUT	順次ファイルの入力処理
	¥ISIN	ISAMファイルの入力処理
	¥VSIN	VSAMファイルの入力処理
	¥OUTPUT	順次ファイルの出力処理
	¥ISGEN	ISAMファイルの作成処理
	¥VSGEN	VSAMファイルの作成処理
	¥SUMMARY	サマリー処理
	¥MERGE	マージ処理
	¥MATCH	マッチング処理
	¥USER	ユーザ・ロジックの組み込みモジュール
ファンクション	#DO, #END	DO ループ
	#CLEAR	複数項目のクリア
	#DAYS	日数計算
	#WDATE	暦日計算
	#DATECK	日付のチェック
	#SETDA #RESETDA	DAのセット・リセット
	#ISGET #ISRPL #ISISRT #ISDLT	ISAMファイルのランダム・アクセス

種々のプログラムのDFチャートを作成すると、ほとんどのプログラムは限られた基本的なモジュールの組み合わせで実現できることがわかる。その基本的なモジュールは、先の入出力モジュールに加えて、¥SUMMARY, ¥MERGE, ¥MATCH等の数少ない処理モジュールである。

従来、マッチング処理、マージ処理などを手順型フローで表現しようとすると、微妙な処理のタイミングに注意する必要があったが、¥MERGEを使うと、このタイミングを全く気にする必要がないし、また¥MATCHモジュールでは、データの流れを制御することで代えられているので、誤りなく記述することができる。

4.3.2 ユーザ・ルーチン

これは、各モジュールの中で指定された部分的な処理、例えば、¥MATCHモジュールを使ったときに、マッチした場合の処理を、ユーザがCOBOL命令とDF-COBOLファンクションを使って、ユーディングするためのものである。

ここで使われるCOBOL命令は、加減乗除の四則演算やMOVE命令が主体であって、READ, WRITEの入出力命令を書く必要のあることはほとんどない。

4.3.3 定義

定義関係には、¥DA, ¥UA, ¥TABLEA等がある。¥DAは、メイン・メモリ中に取られるデータの格納領域(DA)を定義するものであり、データはこのDAを介してモジュールからモジュールへと流れて行く。¥UAは、ユー

ザ・ルーチンでワーク的に使われるエリアを定義するものであり，#TABLEAは，テーブル・ルック・アップ処理のためのテーブルを定義するものである。

4.3.4 ファンクション

ファンクションは，モジュールよりもっと基本的で，単純な処理を実行するものであり，ユーザ・ルーチンのなかで，COBOL命令と混合して使用される。

DF-COBOL特有のファンクションとしては#SETDA，#RESETDAがある。両者はデータの流れを制御するために用いられる。モジュールのなかには，自動的にデータの流れを制御しているものと，必要に応じてユーザに一部あるいは全部その制御をまかしているものがある。#MERGEは前者であり，#MATCHは，ユーザが一部データの流れを制御する例である。これの必要となるのは，部分処理の実行状態により，データの流れが異なる場合であり，例えば，部品マスタの残高の状態により，データの流れが少し異なる場合にな，#SETDAと#RESETDAを使う必要が生じる。

しかし，データの流れの制御は，手順型フローの場合におけるタイミングの制御と比べて，はるかに易しいものである。

4.4 生成されるプログラムのロジック

プログラム・ロジックは，メイン・ロジックと各モジュール別のロジックの二つに分かれる。

モジュールは，DAにより境界が定められていて，関係する変数は自己の内部の変数（会計のためのカウンタ等）と，境界となるDAのみである。また，モジュールが起動されるのは，入力のすべてのDAがデータを持ち，出力のDAがすべてデータを持たないときのみであり，ロジックはモジュール別に異なるが，比較的単純なものが多い。

メイン・ロジックはデータ・フローの応用であり，各モジュールの実行条件，すなわち，入出力のDAの状態を調べて，実行できるモジュールを実行させるだけである。そのロジックを要約すると，つぎのようになる。

各DAは二つの状態，"満"と"空"を与える。モジュールは，入力DAがすべて"空"であれば，何時実行してもよい。メイン・ロジックは，各DAの状態を調べ，実行可能なモジュールを実行させるだけで，モジュールの実行順序を気にする必要はない。モジュールの実行により関与したDAの状態が変化する。

このようなロジックの特長として，モジュールの数がいくら増加しても，プログラムのロジックは複雑にはならないし，誤りの起こる心配もない。

DF-COBOLが完全なモジュール方式となっているのは，この為である。

4.5 DF-COBOLによるシステムの開発

DF-COBOLを利用したシステムの開発は図5に示されるとおりである。DF-COBOLを利用することは，単なるプログラミング手段の変更だけではなく，データ処理設計手順の変更ももたらす。

DFチャート利用による表現力の向上により，設計段階からの質と生産性の向上が得られる。

DF-COBOLによるコーディングは，図6に示すように，DFチャートの要素と一対一の関係にあるため，プログラム知識をあまり持ちたなくとも，容易に行える。

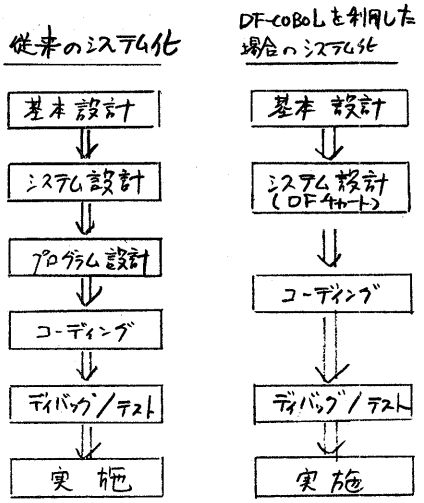

図5 従来のシステム開発とDF-COBOLを利用した場合のシステム開発との手順の違い

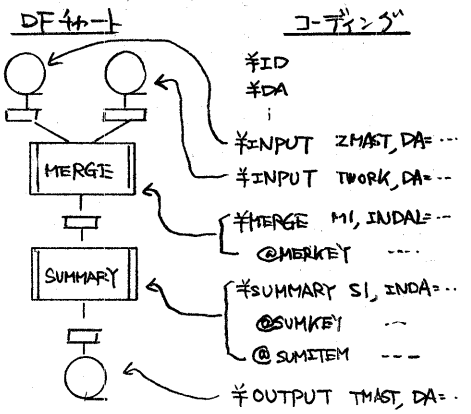

図6 DFチャートとコーディングの関係

4.6 DF-COBOLの設計方針

DF-COBOLは，アプリケーション・システムの開発とメインテナンスの質と生産性の向上を目指したものであるが，その仕様の設計において，次の事項を重視した。

(1)データ処理内容の表現力
(2)非専門家向
(3)既製ツールの活用
(4)完全なモジュール方式

4.6.1 データ処理内容の表現力

設計，開発，メインテナンスのすべてにわたり，簡潔で疑問の余地のない表現方法は，質と生産性の向上には不可欠である。

4.6.2 非専門家

現状では，良質のプログラムを作るためには，専門のプログラマが必要であるが，確保が難しくなりつつある。

非専門家でも利用可能にするために，パラメータの構造を，人間の思考にあった階層型式の表現にするとともに，過度の省略形にはしていない。

4.6.3 既製ツールの活用

我々の目的は，システムの設計から保守までの質の生産性の向上であるため，マクロ機能を使ったジェネレータ方式を採用した。

COBOLジェネレータを作ったのは，COBOLが事務計算の主流を占めていること，従来から登録されているCOPY句をそのまま使えるようにするためである。

4.6.4 完全なモジュール方式

従来のプログラム・ジェネレータの多くは，プログラム単位のパターン化が発想しているため，それを使うと，小さなプログラムに分解する必要があり，ファイルの入出力が増える傾向にあった。これを解決するため，各モジュールを容

易に組み合わせできる完全なモジュール方式とし，プログラム本数を減らすとともに，入出力の重複を排除している。

5. DF－COBOLの特長

以上DF－COBOLについて記述してきたが，ここにその特長を要約する。

(1) DFチャートは，データ処理内容の優れた表現方法である。

(2) DFチャートから直ちにプログラムのコーディングが可能で，手順に関する特別の知識を要しないため，非専門家でも習得しやすい。

(3) 完全なモジュール方式であるため，プログラム本数は少なくなる。

(4) パラメータは，理解しやすい階層型式になっているとともに，部分的な処理はユーザがCOBOLを使って自由に書く方式であるため，コーディング間違いは少なく，また処理の自由度が高い。

(5) 枚数が少なく，内容の把握のしやすいDF－COBOLのソース・プログラムで，プログラムの管理ができる。

(6) データのフォーマット定義がCOBOLのCOPY句でできるため，既存のシステムにすぐ適用できる。

6. 効果

DF－COBOLが，社内のシステム開発に適用され始めてから，すでた300本近くのプログラムが開発されている。これら使用実績を踏まえ，我々は，DF－COBOLを次のように評価している。

(1) コンパイル回数 1～2回

(2) デバッグ回数 1～2回

(3) 従来のCOBOLのステップ数に比較して 1/4～1/10

(4) 開発工数は，従来のCOBOLを利用する場合と比較して，システム設計からテストまでが 1/3 以下。

表2 DF-COBOLと従来のCOBOLを利用した場合の開発実績の比較

項目＼システム	従来のCOBOL		DF-COBOL	
	買掛金システム	給与システム	一般管理費システム	在庫発行システム
プログラム本数	55本	25本	25本	24本
ソース枚数	19,000枚	9,000枚	2,500枚	1,700枚
1本当りの平均ソース枚数	345本/枚	360枚/本	100枚/本	71枚/本
開発工数(決定設計以降)	30人月	10人月	3人月	2人月
1人月当たりの開発プログラム本数	1.8本/人月	2.5本/人月	8本/人月	12本/人月

7. あとがき

DF－COBOLの検討は，昭和53年12月から開始し，54年10月にバージョン1が完成した。その後データ・ベースのハンドリングができるようにするとともに55年に機能アップし 55年7月にバージョン2が完成した。

DF－COBOLとしてCOBOLジェネレータを開発したが，DF－PL/Iも同じ考え方で作ることが可能である。

データ・フローはDF－COBOLにおいて最も重要な役割を果たしており，価値のある考え方と考えている。

しかし，DF－COBOLができたのは，アプリケーション・システムの開発についての長い蓄積の上にデータ・フローが加わった結果である。