ソフトウェア工学 22－10
（1982. 2. 10）

ソフトウェア・アーキテクチャー考察

山本正弘（日本アイ・ビー・エム 西日本第一営業所）

1. はじめに

ソフトウェアの複雑巨大化と，それに伴う開発・保守上の混乱を招いたのは，COBOL等の従来の言語そのものと，開発過程に於けるソフトウェア・アーキテクチャの欠如である。言語自体にソフト構造の設計機能が備っていないために，HIPOやSP等の出現を見たが，それらは方法論又は技法であって，ソフト構造を定めるものではなかった。ソフト構造は開発保守容易性と密接な関係があるが，その設計は使用者に委ねられている。こういう状況の中で，言語によってソフトウェア構造を与えようという試みを紹介することは意味のあることと考える。

当小論文で説明する言語は①代数②関数の二部から成り，作成されるシステムは①によって制御部分が②によって論理部分が記述される。両者は完全に分離された形で記述されるのである。制御と論理の分離というソフトウェア・アーキテクチャを当言語は呈示するのである。

2. 代 数

代数によってデータの流れを集合演算的に表現する。代数の演算子を以下に説明する。いづれも比較関数（f or g）付きの集合演算である。以下A·B·Cは集合，a·bはその項目名とする。

和　　：C←A［f(a)∨g(b)］B. ⟺ C＝A∪B
差　　：C←A［f(a)－g(b)］B. ⟺ C＝{Ax∈A | f(ax)≠g(by) ∀By∈B}
積　　：C←A［f(a)∧g(b)］B. ⟺ C＝{Ax∈A | f(ax)＝g(by) ∃By∈B}
ジョイン：C←A［f(a)＝g(b)］B. ⟺ C＝{(Ax,By) | f(ax)＝g(by)　Ax∈A, By∈B}
射　　影：C←F［f(a)］A.
　　Fは射影関数．f(a)－値が等しいレコード全体に作用して，一箇のレコードを作成する。
スタック：C1,------,Cn←（v1,-----,vn:F）［f(a)］A.
　　Fはスタック関数。f(a)－値が等しいレコード全体に作用して，スタック内の各レコードに比較値wjを与え，行先Ciを決める（wj＝vi）。
スタック：C1,------,Cn←（v1,-----,vn:F）［m］A.
　　Fはスタック関数。常時m⊐のレコードがスタックされ，1レコード到着の都度，関数が作用して出力レコードを決定し，行先を決める。
処　　理：C←F　A
　　Fは処理関数。レコード1件に対して1件のレコードを出力する。

右はコーディング例である。D,Eともに C から派生したものである。f(d)とf(e)がCに於けるレコード識別コードを表わすものであるならば，ジョイン演算はF1とF2の並行処理を同期させる働きがある。↑印は出力集合を表わす。入力集合を明示する必要はない。ステートメントの順序は意味がない。

```
C←A［f(a)－g(b)］B
D←F1　C
E←F2　C
↑Θ←D［f(d)＝f(e)］E
```

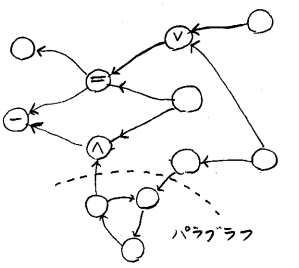
フォーマット・ステートメント．各集合の項目について属性を与える。

□集合名←項目名・属性,……,項目名・属性;

パラグラフ．上記の代数演算子で記述されるサブルーチンである。代数は予め決められたデータの流れを書くのに便利であるが，未決定の流れ（同一集合への再帰的なコール等）を記述することが出来ない。このためパラグラフ内では再記代入シンボル:=:を導入して，未決定のデータの流れをを記述するようにした。

paragraph名 B←A
C←A[f(a)=g(b)]D
B←A[f(a)-g(b)]D
A:=:C

3．代数の実行



代数ステートメントのオペランド集合に対応して待ち行列が作られ，FIFO方式（スタック演算を除く）でレコードが移動される。基本的にレコードが到着するのを待って処理が行われるが（順次方式），ジョイン・差・積の右オペランド集合は，その左オペランドのレコード内容に応じて直接読み込まれうる（直接方式）。（実際の入出力を行うのは，当待ち行列プロセサーではなく，アクセスメソッド毎に作られる補助プロセサーの役割である。この点についての詳細は割愛する）。直接読み込みの場合，比較関数を用いているので，読み込みアーギュメントが得られない。そこで比較値から読み込みアーギュメントを作成する逆比較関数g^{-1}を定義する必要がある。

各集合がどの方式で処理されるかをシステムに指示するため，以下の印を導入し，C←A[f(a)⊗-⊗g(b)]Bの様に記す。

Ⓘ-----順次・単調増大比較関数
Ⓥ-----順次　　Ⓓ-----直接

右図は両側オペランド形式の演算で許される処理方式の組合せである。

オペランド左 右	和	差	積	ジョイン
ⒾⒾ	○	○	○	○
ⒾⓋ	○	－	－	－
ⒾⒹ	－	○	○	○
ⓋⒾ	○	－	－	－
ⓋⓋ	○	－	－	－
ⓋⒹ	－	○	○	○

待ち行列処理方法．表－1に従う。

順次入力集合の選び方．処理可能な待ち行列がない場合，順次入力集合から1件のレコードを得る。入力集合の選び方は重要で，不急のレコードを読むことはシステムの資源を圧迫することになる。次の様な入力制御を行う。下図の様に，順次入力集合に対し，（フォーマットステートメント上に）比較関数を用いて入力のバランスをはかる定義式を指定する。右図の場合，例えばCが選ばれるのは，最後に読まれたレコードを比較してh(c)<f(a)の場合である。無指定のAが選ばれるのは，他の比較がすべて'≧'となった時である。この考えオは比較関数を重複して指定させているように見えるが，一つの集合が複数箇の比較関数によって評価されている場合，そのどちらを入力制御に用いるかを自

□A←----;
□B←----;g(b)=f(a)/A
□C←----;h(c)=f(a)/A
□D←----;i(d)=h'(c')/C

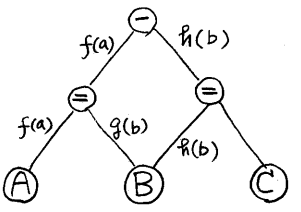
演算子	実行前の待行列の状態	条件	結果
和 Ⓘ Ⓘ	両オペランドにレコードあり。	$f(a_1) < g(b_1)$ $f(a_1) = g(b_1)$ $f(a_1) > g(b_1)$	A_1 A_1 B_1
差 Ⓘ Ⓘ	両オペランドにレコードあり。 あらかじめ $g(b_i) < f(a_1)$ なる B_i はパージされる。	$f(a_1) < g(b_1)$ $f(a_1) = g(b_1)$	A_1 擬似$-A_1$
積 Ⓘ Ⓘ		$f(a_1) < g(b_1)$ $f(a_1) = g(b_1)$	擬似$-A\,(f^{-1}g(b_1))$ 注1 A_1
ジョイン Ⓘ Ⓘ		$f(a_1) < g(b_1)$ $f(a_1) = g(b_i)\ i=1\cdots n$ $f(a_1) \neq g(b_{n+1})$	擬似$-A\,(f^{-1}g(b_1))$ 注1 $(A_1, B_i)\ i=1\cdots n$
その他の和	どちらかのオペランドにレコードあり。	—	全レコード
差 Ⓓ	左オペランドにレコードあり。	$\exists B\ \ g(b) = f(a_1)$ $\forall B\ \ g(b) \neq f(a_1)$	注2 A_1
積 Ⓓ		$\exists B\ \ g(b) = f(a_1)$ $\forall B\ \ g(b) \neq f(a_1)$	A_1 注2
ジョイン Ⓓ		$\exists B\ \ g(b) = f(a_1)$ $\forall B\ \ g(b) \neq f(a_1)$	$(A_1, B_x)\ g(b_x) = f(a_1)$ なるすべての B_x に対し 注2
射影	1レコード到着毎。	$f(a_1) = f(a_i)\ i=1\cdots n$ $f(a_1) \neq f(a_{n+1})$	函数が $\{A_1, \cdots, A_n\}$ に作用
スタック[f(a)]		同上	同上 注3
スタック[m]		—	函数が決めた A_x 注3

A_i は左オペランドヌはシングルオペランドのレコード。
A_1 が先頭。
B_i は右オペランドのレコード。
B_1 が先頭

注1--- $f(a_i) < g(b_1)$ なる A_i は除去される。
注2---Ⓘ Ⓓ型の場合，擬似レコード
注3---該当なし C_i には擬似レコード

(表－1)

動的に決定するのはむつかしい。誤った代数ステートメントはシステム内のレコード数を高める結果になるが（右図），入力制御を明示的に指定させることによって，使用者の問題に帰着させることができる。



以上の方法はバッチ型（入力データは常に読める状態にある）の処理がうまく制御できる事を示しているに過ぎない。機械効率や応答時間の観点にたてば，異なった入力制御になる。

擬似レコード．比較相手なしの理由でシステム内に滞留するレコードの数を減少するために，擬似レコードの考え方がある。比較値の意味で $A \cap B = \phi$ とすると，右図中Cは E⊖F までレコードは到着しないので，オニ式の比較演算はすすめられない。－オ集合B は入力制御によって読み込みがすすむが，オニ式のBでは相手待ちのレコードが滞留する。ⒾⒾ型の比較演算では，表－1にみるようにレコードが得られなくても，次に出力されるレコードの比較値の下限を求めることが出来る。比較値から項目値に変換する逆比較関数 f^{-1} を定義することによって，この下限値から擬似レコードが生まれる。後続の比較関数 f_1 のアーギュメントが，逆比較関数 f^{-1} の出力項目から成り立っている場合は，f_1 値は意味があり，代数の比較処理に使われる。擬似レコードが消滅するのは，①待ち行列内で他のレコードに追っかれた場合，②擬似レコードを処理出来ない関

C←A[f(a)Ⓘ∧Ⓘg(b)]B
D←C[f(a)Ⓘ∨Ⓘg(b)]B
□A←------;
□B←------; g(b)−f(a)/A

数を通過する場合，③比較演算の前後で除去される場合である。②の場合とは，擬似値の項目が比較関数のアーギュメントになっていない時等に代表されるが，関数の節で少し詳しく説明される。このような場合には，擬似レコードの意味がなくなる。

パラグラフ．パラグラフにとっての順次入力集合とは，パラグラフ見出しに書かれる集合のみである。パラグラフの実行は，この順次入力集合のレコード1件毎に実行される独立した待ち行列処理である。

4．代数実行例

```
      MAINT
↑ETRX←TRX[TR1-M1]OM
NN←OM[M1=TR1]TRX
TOM←OM[M1-TR1]TRX
NNM←PF1[M1]NN
↑NM←NNM[M1∨M1]TOM
□TRX←TR1,,TR3;TR1
□OM←M1,,,M4;M1
□NN←M1,,,,,,M7;M1
□NNM←M1,,,M4;M1
□TOM←M1,,,M4;M1
□ETRX←T1,,T3;T1
□NM←M1,,,M4;M1
```

```
      ∇PF1[□]∇
      ∇ A←PF1 B;I
[1]   I←0
[2]   A←4↑B[1;]
[3]  L:→((1↑ρB)<I←I+1)/0
[4]   A[1+B[I;6]]←A[1+B[I;6]]+B[I;7]
[5]   →L
      ∇
```

```
      OM
 1  76  86  62
 2  14  65  24
 3  89  88   6
 6   2  99  97
 7  76  12  26
 8  94  92  70
11  56  30  34
13  75  62  73
14  67  92  80
15  15  39 100

      TRX
 2  1 12
 2  2 30
10  1  2
13  3  7
13  3 24
15  2  1

      ⍙NPL MAINT

      ETRX←0 3ρ0

      NM←0 4ρ0

      ⍙NPROI

      ETRX
10  1  2

      NM
 1  76  86  62
 2  26  95  24
 3  89  88   6
 6   2  99  97
 7  76  12  26
 8  94  92  70
11  56  30  34
13  75  62 104
14  67  92  80
15  15  40 100
```

APL[2]によって代数の原型プロセサーを作った。上述のものと異なる点は①比較関数の代りに項目名を用いる②すべて①-①型式である③入力制御として比較項目を用いる，である。集合としてはAPL数値マトリックスを使用し，その各行をレコードと做した。

例1（左図）：集合OM（旧マスター）とTRX（取引き）から，NM（新マスター）とETRX（相手なし取引き）を作成する。処理関数の代りに射影関数PF1が用いられたのは，TRX中に同一キー値レコードが存在しうることを配慮したものである。

マスターファイルのレイアウトは次の通り。

キー	項目1	項目2	項目3

取り引きファイルは

キー	変更される項目#1	かえるべき値

例2（右図）：部門別集計をもとの集合に挿入して出力する。集計値を先頭に出す場合と最後に出す場合を示した。和演算のオペランドを入れかえのみでよい。関数部分には変更がない。

```
      ZKEI
↑C←B[B1∨A1]A
B←FUN1[A1]A
□A←A1,A2;A1
□B←B1,B2;B1
□C←C1,C2;C1


      ∇FUN1[□]∇
      ∇ Z←FUN1 X
[1]   Z←X[1;1],+/X[;2]
      ∇

      A
 1  25
 1  91
 1  40
 1  21
 2  83
 2  43
 2 100
 3  20
 3  48

      ⍙NPL ZKEI

      C←0 2ρ0

      ⍙NPROI

      C
① 177
 1  25
 1  91
 1  40
 1  21
② 226
 2  83
 2  43
 2 100
③  68
 3  20
 3  48
```

```
      ZKEI
↑C←A[A1∨B1]B
B←FUN1[A1]A
□A←A1,A2;A1
□B←B1,B2;B1
□C←C1,C2;C1

      ⍙NPL ZKEI

      C←0 2ρ0

      ⍙NPROI

      C
 1  25
 1  91
 1  40
 1  21
① 177
 2  83
 2  43
 2 100
② 226
 3  20
 3  48
③  68
```

5. 関数

代数で入出力等制御部分の記述がなされ，関数部に残されたのは，レコードの処理である。制御部分のない論理は比較的容易にトップダウンに記述できるので，それを紹介する。またここで説明される記述法は，射影関数が潜在的にもっているレコードを貯えるという性格に対処するためにも是非必要なものである。

特徴①：実行式と条件式がある。実行式は代入記号←をもっている。条件式が満足された時実行式は実行される。条件式の字下げは条件のネスティングである（右図）。ブランチ命令に相当するものはない。実行の順序付けはコンパイル時に於いて決定される。

(P1) A←f1　　(P1) A←f1
(P2) B←f2　⇔　(P1∧P2) B←f2
C←f3　　(P1∧P2) C←f3
(P3) D←f4　　(P3) D←f4

特徴②：変数は相対実行サイクルという手段で参照される。1実行サイクルとは並べ換えの終ったステートメントを上から下まで，一通り実行することを言う。例えばA←A[-1]+Bは Aの初期値が零の時Bの総和である。順序付けの都合上，相対実行サイクルは負整数，また代入される変数は相対実行サイクルを記述できない。

射影関数は同一タイプのレコード群に対する繰り返し演算である。この場合1レコードが1実行サイクルで処理される。相対実行サイクルで処理を記述すれば，全レコードの到着を待たずに実行可能である。また相対実行サイクル数を超えた（変数毎に異なる）過去の部分については，演算に用いられる事がないので不要である。この事をシステムが把握できるので，記憶域の減少に役立てることが可能である。

特徴③：実行式の並べ換えを行うには，変数の内容が破壊されないという保障が必要である。何故なら二度以上の代入を受ける事は順序が重要になるからである。このことを実行中にチェックしなければならない（右図）。一度も代入されないで他の実行ステートメントのアーギュメントになる変数についても同様のチェックが必要であるが，省略時値を与える余地がある。

特徴④：実行式の順序付けをコンパイル時に行う。実行式の右辺と関連条件式の変数が既値であれば，その式は実行可能である。そして新しい既知変数が生じる。このようにしてして次次と実行式を取りくずして，

```
      DUPLICATE
  ⁻2
  ⁻1(1)
  10Y←Y[⁻1]+X
  20(A>1)  X←X[⁻1]×A+C
  30(A<2)  X←X[⁻1]+∇FF A
  40(1)  A←A[⁻1]×1.1

      ∆COMPILE DUPLICATE
***FOLLOWING FIELD MUST BE PREDEFINED THRU ALL PERIOD
C
***FOLLOWING FIELD MUST BE PREDEFINED INITIAL VALUE
A  ,X  ,Y
***FOLLOWING ARE EXTERNAL CONSTANTS OR FUNCTION
FF

      C←0 1 2 3 4 5 6 7 8 9 10

      A←0.7

      X←0

      Y←0

      ∇FF[⎕]∇
      ∇ A←FF C
[1]   A←10×C
      ∇

      ∆GO 1 10
***FOLLOWING STATEMENTS ARE DUPLICATE EXECUTION IN CYCLE=   4
 20 30
*******END*******

      A
0.7 0.77 0.847 0.9317 1.02487 0 0 0 0 0 0

      X
0 7.7 16.17 25.487 35.7357 0 0 0 0 0 0

      Y
0 7.7 23.87 49.357 85.0927 0 0 0 0 0 0
```

実行順序を得る。(変数が既知であるとは，その変数に代入を行うすべての実行式が実行可能になった時である)。順序付け中に，新しく既知変数を得られない場合は，変数が互いにパラメータ関係に陥っている(右図－局大ループとして表現)。変数間のループはコーディング誤りによるものと，コンパイル時の順序付けによるものがある。右下図の場合はダミー変数によって解決する。

```
      LOOPSAMP
    -2
    -1(1)
    10  A←B+C
    20  B←D
    30  D←A+Y
    40  H←A+Y
    50  J←H+B
    60  I←J
    70  H←I

      ∆COMPILE LOOPSAMP
***FOLLOWING FIELD NAMES MAKE LOOPS
A,B,D
****
H,I,J
****
```

擬似レコードの扱い。逆比較関数が作成した擬似レコード中には値の入っている項目とそうでない項目がある。後者の値をＮＵＬＬと呼ぶ。関数中の計算式にＮＵＬＬがアーギュメントとして与えられた場合，結果はＮＵＬＬとする(参照ＡＰＬ)。この方法は，擬似レコードを通常のレコードと区別なく処理する考え方である。(勿論擬似レコードの出力は擬似である)。例えば擬似レコードが処理関数を経て有意な項目を得るのは，条件式が無条件であって，擬似値項目と定数のみの演算がその実行式でなされる場合に限られる。有意な項目をもたない擬似レコードは除去される。

```
(A>1)   X←Y+A
        Z←X+B
(A≤1)   Z←Y+A
        X←Z+C
```

この言語表記法が有用なのは射影関数，処理関数，比較関数である。処理関数と比較関数はレコード間にまたがる処理はありえないので，相対実行サイクルは不要(サブルーチンによるループ計算には有用)である。がトップダウン記述は可能。スタック関数は，レコードの組み検査，並べ換え等の「レコード間処理」を目的とした関数なので当表記法は役立たない。手順言語が主ルーチンとなる。(手順言語で書かれるスタック関数と擬似レコードの関係：全て他に追いつかれて除去されるので，言語上の考慮は不用)。

6．結　論

ソフトウェア・アーキテクチャとその基礎となるアルゴリズムについて述べた。このソフトウェア・アーキテクチャのオーの利点は，制御部分と論理部分の分割である。夫々が互いに影響を与えずに，定義的に記述できる特長を持つ。従ってソフト開発保守が極めて容易になるであろう。オニの利点は二つ以上のソフトウェアシステムが，ファイルを介して結ばれている時，それらを容易に一本化できることである。中間ファイルを労力なく減らすことが可能。オ三点は，代数実行が待ち行列方式であり，多頭プロセサーによる処理に向いている。プログラミング自体はその影響を受けない。オ四点はアクセスメソッドと適用業務の関連がないため，新しい型のアクセスメソッドを受け入れやすい。

[参考文献]

[1] Backus, J., "Can Programming be Liberated from the von Neuman Style?" CACM Vol21 no8, pp 613-641. AUG 1978. [2] Iverson, K., "A Programming Language", Wiley, New York. 1962.