ソフトウェア工学　52－14
（１９８７・２・１４）

ソフトウェア設計法の味見

峰尾欽二

日本ユニバック株式会社

あらまし

ソフトウェア設計法の有用性を示し，普及を計るために小さな課題に対して２つの設計法の味見を行った．ひとつは，表示的意味記述の枠組みによる仕様からCobol コードの作成法であり．もうひとつは，ＣＳＰによる仕様からModula-2コードの作成法である．これらの味見の結果，産業界においても十分実用に耐える設計法が存在する感触を得た．

A Sampler of Software Design Methods

Kinji MINEO

Nippon UNIVAC Kaisha, LTD

2-17-51 Akasaka Minatoku Tokyo, Japan

Abstract

To demonstrate the applicability and usefulness of software design methods and to promote them, we "tasted" them in applying to a small exercie. One of the design methods we tasted is one that generates Cobol code from a specification in the frame of denotational semantics. The other is the method to generate Modula-2 code from specification in CSP. As a result of the trail testing, we got the feeling that above two methods are applicable and also effective for the applications in the business fields.

1．はじめに

企業でのソフトウェア開発に活用するためにソフトウェア工学の味見を行なってきた．その第一報は「仕様記述の味見」[1] である．これは第二報である．

ソフトウェア工学を開発の現場で活用するためには，ある種の条件を整える必要がある．たとえば，

1）現場担当者の教育訓練，

2）集団作業を行なうから，技法に関連するいろいろな標準の設定，

3）現行技法との共存方法と移行手順，

4）技法の推進調整のための機関の設立，

などである．また，利用を勧告するためには，具体的な課題に対して試用しその成果を明示しなければならない．このような理由から，ソフトウェア工学全般にわたっての方法を試用しその味見を数人の仲間とともに行なっているわけである．この報告は設計方法についてである．

設計に関して，設計対象，設計方法，設計支援環境などについて議論できる．対象についてのそれは，対象に関する理論を作り，それによって設計自体を算法化しようというものである．このような設計法は対象主導型といっていい．LCP[2]や JSP[3] はこの代表例である．設計方法の議論は対象を特定しないで一般のソフトウェア作成法を問題にする．別途に解法は得られるものとして，その解法に依存した解法主導型の方法である．分割統治や抽象の概念を中心に，抽象データ型とその段階的詳細化(HISP[4], IOTA[5] , HDM[6], VDM[7]) , 後件から述語変換系によって最弱前件を求めることによってプログラムを導出する Dijkstra[8], Gries[9]の方法，その並行プロセス版である相互通信型逐次プロセス(CSP[10]) などがこの例である．

支援環境は方法を支援する環境で，その方法の成否によって大きく影響するものであるがここでは触れない．

対象主導型の設計法は，それを修得すれば，その種の課題に対して直ちに活用できる利点をもっている．解法主導型の設計法は一般的な方法であるだけに，対象に対する優れた見識を必要とし，多くの職業プログラマにとって直ちに活用できるとはいえない．そこで小さな課題を設定し，勉強のなかから浮上した二つの方法を試用してみることにした．設計法１として，プログラム言語の仕様記述を見習って，表示的意味論[11] の枠組で仕様を作成し，それから自然にプログラムが作成できることを確かめた．これは対象主導型設計法になっている．設計法２として，CSP を取上げ，JSD[12] 流に考えてみた．これは解法主導型の方法に対象主導型の考えを埋め込んだ，いわば混合型のものになっている．

2．例題

例題として書店の売掛管理を選んだ．

ある書店では，特定の顧客に対して書籍の掛売を行なっている．販売係は，書籍とともに納品伝票を顧客に渡し，すぐにその写し（売上伝票）を会計係に送る．

会計係では，月末になるとその月の売上伝票を集計し売掛残高報告書を作成すると同時に，各客先に請求書を送ることになっている．

顧客は，現金や銀行振込などによって請求額を支払うが，一度に全額を支払うとは限らない．入金額は財務係から随時入金伝票として会計係に回送されている．会計係の仕事をプログラムしたい．

売上伝票：顧客名；［書籍名；単価；冊数；金額］
（の繰り返し）
入金伝票：顧客名；入金金額
売掛残高報告書：前月末総未済残高；当月総売上高
；当月総入金高；当月末総未済残高
請求書：顧客名；前月末ご請求額；当月お買上額；
；当月お支払額；当月ご請求額

3．設計法１

3．1　表示的意味論による仕様

表示的意味論では構文の領域と意味の領域を帰納的に方程式系で定義し，意味を構文領域から意味領域への関数として把える．例題の仕様を書くと次のようになる．なお，仕様で使用する表記について補足する．

①下線部は領域名を表わす．

②【　】は構文部分を表わす．

③関数ｆ，ｇに対し　ｆ；ｇは関数合成ｇｏｆと同じである．

④σ／［ｓ］は状態σの変化を示し，ｓは変化部分を表わす．

構文領域

客名 ∈ 客名 = 文字列
金額 ∈ 金額 = 整数
残高 ∈ 残高 = 整数
売上 ∈ 売上 = 客名 × 金額
入金 ∈ 入金 = 客名 × 金額
顧客 ∈ 顧客 = 客名 × 残高

入力 ∈ 入力
客データ ∈ 客データ
取引データ ∈ 取引データ
取引 ∈ 取引

構文

入力 ::= 客データ*
客データ ::= 顧客 取引データ
取引データ ::= 取引*
取引 ::= 売上 | 入金

意味領域

```
客名  ∈  客名   =  文字列
前残  ∈  前残   =  整数
買上  ∈  買上   =  整数
支払  ∈  支払   =  整数
請求  ∈  請求   =  整数
残高  ∈  残高   =  整数
総売  ∈  総売   =  整数
総入  ∈  総入   =  整数
新残  ∈  新残   =  整数
残    ∈  残     =  整数
売    ∈  売     =  整数
入    ∈  入     =  整数

出力ファイル名  =  ( invf, newcf, rep )
出力領域名１    =  出力ファイル名∪(sumarea1)
出力領域名２    =  出力領域名１∪(sumarea2)
請求書  =  客名×前残×買上×支払×請求
新顧客  =  客名×残高
報告書  =  前残×総売×総入×新残
σ   ∈  Σ   =  [出力ファイル名  →  請求書+新顧客+報告書]
σ1  ∈  Σ1  =  [出力領域名１  →  請求書+新顧客+報告書+SUM1]
σ2  ∈  Σ2  =  [出力領域名２  →  請求書+新顧客+報告書+SUM1+SUM2]

sum1 ∈  SUM1  =  前残×総売×総入
sum2 ∈  SUM2  =  客名×残×売×入
```

意味関数

```
billing : 入力  →  Σ
billing【入力】= auxbilling【入力】(initialstate); report
  ここで
    initialstate = [ invf → <>, newcf  → <>, rep  → <>,
                     sumarea1 → <0,0,0,> ]
    auxbilling : 入力→ [Σ1 →Σ1 ]
    auxbilling【客データ  入力】
          = cdataf【客データ】 ;  auxbilling【入力】
    report :Σ1 →Σ
    report( σ1) = [ invf →σ1(invf), newc →σ1(newc),
                    rep  → reporteditting( σ1(sumarea1) ) ]
      ここで
        reporteditting :SUM1→報告書
        reporteditting(sum1) =  <  sum1.前残,  sum1.総売,  sum1.総入,
                                   sum1.前残+ sum1.総売− sum1.総入  >

cdataf :客データ→ [Σ1 →Σ1 ]
cdataf【客データ】 = auxcdataf【客データ】σ1  ; sumlupdate ; invoicing  ; newc
  ここで
    auxcdataf: 客データ→ [Σ1 →Σ2 ]
    auxcdataf【顧客  取引】 (σ1) =
         custf【顧客】 (σ1) ; tloop【取引】
    sumlupdate:Σ2 →Σ2
    sumlupdate(σ2) =σ2 ／[sumarea1 → <σ2(sumarea1).前残+ σ2(sumarea1).残,
                                      σ2(sumarea1).総売+ σ2(sumarea1).売,
                                      σ2(sumarea1).総入+ σ2(sumarea1).入 >]
    invoicing :Σ2 →Σ2
    invoicing( σ2) =σ2 ／[ invf→<sum2.客名,  sum2.残,  sum2.売,  sum2.入,
                                              sum2.残+ sum2.売− sum2.入 >]
    newc : Σ2 →Σ1
    newc(σ2) = [ invf →σ2(invf),
          newc → < sum2.客名,  sum2.残+ sum2.売− sum2.入 >,
          rep  →σ2(rep),
          sumarea1 →σ2(sumarea1) ]

custf: 顧客→ [Σ1 →Σ2 ]
custf【顧客】( σ1) =  σ1 +[ sumarea2→< 顧客. 客名, 顧客. 前残 , 0, 0 > ]

tloop: 取引データ→ [Σ2 →Σ2 ]
tloop【取引  取引データ】=
   tranf【取引】 ; tloop【取引データ】

tranf: 取引→ [Σ2 →Σ2 ]
tranf【取引】( σ2) =
  ( isSales(取引) →
      σ2 ／[ sumarea2→ <σ2(sumarea2).客名, σ2(sumarea2).残,
                        σ2(sumarea2).売+売上. 金額, σ2(sumarea2).入>],
    isReceipt(取引) →
      σ2 ／[ sumarea2→ <σ2(sumarea2).客名, σ2(sumarea2).残,
                        σ2(sumarea2).売, σ2(sumarea2).入+入金. 金額>])
```

意味関数の定義域と値域をＪＳＰ流に図示すると図１のようになる．

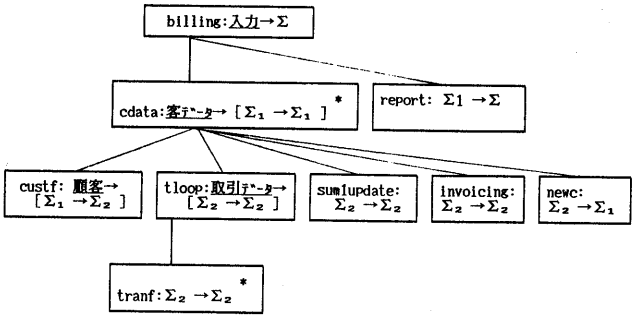

図１　意味関数の定義域と値域

３．２　実現

仕様を直接変換することで実現できる．命令型言語によって実現するときは，帰納的に定義した関数を反復構造に変換した方が効率的である．現場で普及しているプログラム言語COBOL でコード化すると以下のようになる．この変換は直接的である．

```
identification division.
program-id. billing.
environment division.
configuration section.
source-computer. univac-1100.
object-computer. univac-1100.
input-output section.
file-control.
    select sales assign to disc.
    select receipts assign to disc.
    select customers assign to disc.
    select invoices assign to disc.
    select balreport assign to disc.
    select newcustomers assign to disc.
data division.
file section.
fd  sales label record standard.
01  srec pic x(18).
*
fd  receipts label record standard.
01  rcprec pic x(18).
*
fd  customers label record standard.
01  crec pic x(18).
*
fd  invoices label record standard.
01  invrec.
    02 cnames pic x(10).
    02 bal pic 9(8).
    02 sal pic 9(8).
    02 receipt pic 9(8).
    02 newbal pic 9(8).
*
fd  balreport label record standard.
01  breprec.
    02 bal pic 9(12).
    02 sal pic 9(12).
    02 receipt pic 9(12).
    02 newbal pic 9(12).
```

```
fd  newcustomers label record standard.
01  ncusrec.
    02 cnames pic x(10).
    02 newbal pic 9(8).
working-storage section.
01  sum1.
    02 bal pic 9(12).
    02 sal pic 9(12).
    02 receipt pic 9(12).
01  sum2.
    02 cnames pic x(10).
    02 bal pic 9(8).
    02 sal pic 9(8).
    02 receipt pic 9(8).
*
01  inputfile.
    02 inputeof pic x value space.
       88 is-inputeof value 'e'.
    02 inputrec.
       03 rectype pic x.
          88 is-sales value 's'.
          88 is-receipts value 'r'.
          88 is-customer value 'c'.
       03 inputitems.
          04 cnames pic x(10).
          04 amounts pic 9(8).
          04 balances pic 9(8).
    02 mincode pic x(10).
01  inputrecordarea.
    02 salesrec.
       03 cnames pic x(10).
       03 amounts pic 9(8).
    02 receiptrec.
       03 cnames pic x(10).
       03 amounts pic 9(8).
    02 cusrec.
       03 cnames pic x(10).
       03 balances pic 9(8).
procedure division.
billing.
    perform openinf.
    perform openoutf.
    perform initialstate.
    perform readinputfile.
    perform auxbilling until is-inputeof.
    perform makereport.
    perform closeinf.
    perform closeoutf.
    stop run.
initialstate.
    move 0 to bal of sum1,
              sal of sum1,
              receipt of sum1.
auxbilling.
    perform cdataf.
makereport.
    move corr sum1 to breprec.
    compute newbal of breprec =
       bal of sum1 + sal of sum1 - receipt of sum1.
    write breprec.
cdataf.
    perform auxcdataf.
    perform sum1update.
    perform invoicing.
    perform newc.
auxcdataf.
    perform custf.
    perform readinputfile.
    perform tloop until not (is-sales or is-receipt)
                          or is-inputeof.
sum1update.
    add bal of sum2 to bal of sum1.
    add sal of sum2 to bal of sum1.
    add receipt of sum2 to receipt of sum1.
invoicing.
    move corr sum2 to invrec.
    compute newbal of invrec =
       bal of sum2 + sal of sum2 - receipt of sum2.
    write invrec.

newc.
    move cnames of sum2 to cnames of ncusrec.
    compute newbal of ncusrec =
       bal of sum2 + sal of sum2 - receipt of sum2.
    write ncusrec.
custf.
    move cnames of inputitems to cnames of sum2.
    move balances of inputitems to bal of sum2.
    move 0 to sal of sum2,
              receipt of sum2.
tloop.
    perform tranf.
    perform readinputfile.
tranf.
    if      is-sales
            add amounts of inputrec to sal of sum2
    else if is-receipts
         add amounts of inputrec to receipt of sum2.
openinf.
    open input sales, receipts, customers.
    perform readsales.
    perform readreceipts.
    perform readcustomers.
closeinf.
    close sales,
          receipts,
          customers.
openoutf.
    open output invoices,
                balreport,
                newcustomers.
closeoutf.
    close invoices,
          balreport,
          newcustomers.
readinputfile.
    if not (cnames of cusrec = high-value and
            cnames of salesrec = high-value and
            cnames of receiptrec = high-value)
        perform getminrec
    else
        move 'e' to inputeof.
getminrec.
    move cnames of cusrec to mincode.
    if mincode > cnames of salesrec
        move cnames of salesrec to mincode.
    if mincode > cnames of receiptrec
        move cnames of receiptrec to mincode.
    if      mincode = cnames of cusrec
              move 'c' to rectype
              move corr cusrec to inputitems
              perform readcustomers
    else if mincode = cnames of salesrec
              move 's' to rectype
              move corr salesrec to inputitems
              perform readsales
    else if mincode = cnames of receiptrec
              move 'r' to rectype
              move corr receiptrec to inputitems
              perform readreceipts.
```

```
readcustomers.
    read customers into cusrec at end
     move high-value to cnames of cusrec.
readsales.
    read sales into salesrec at end
     move high-value to cnames of salesrec.
readreceipts.
    read receipts into receiptrec at end
     move high-value to cnames of receiptrec.
```

4．設計法2

4．1　ＣＳＰの概要

CSP の中心概念はプロセスである．対象つまりプロセスP の事象全体を Pの事象集合(alphabet)といい，αP と書く．プロセスが生まれた最初の状態からある時点までに生起した事象の系列を考え，これを足跡(trace) という．プロセスP の生起できるすべての足跡の全体を足跡集合(traces)といいτP と書く．ここでプロセスP をその事象集合αP との足跡集合τP との組；

$$P = (\alpha P, \tau P)$$

として定義する．ここで，$\tau P \subseteq \alpha P^*$ は次の性質を満たすものとする（αP^*はαP の有限列の全体）．

① $\langle\rangle \in \tau P$ （$\langle\rangle$ はαP^*の空列）

② $s\hat{}t \in \tau P$ ならば $s \in \tau P$

（$\hat{}$ はαP^*の結合演算）

①は，P の生誕したままの状態，つまりなんの事象も起こっていないときを示し，②はある時点までの足跡が $s\hat{}t$ であれば，それ以前のある時点で，それまでの足跡がちょうど s になっていることを示している．

プロセスの全体Pの上でいくつかの演算を定義する．

(1) STOP = (A, {<>})

プロセス STOP は事象集合として A をもっているが，なんの事象も生起しない（死に詰まり）ものである．

(2) $(a \to P) = (\alpha P \cup \{a\}, \{\langle\rangle\} \cup \{\langle a\rangle\hat{}t \mid t \in \tau P\})$

プロセス(a→P)はまず事象 a が生起して，それ以降はちょうどプロセス P と同様に挙動するプロセスである（a それから P）．

(3) $(a \to P | b \to Q) = (\alpha P \cup \alpha Q \cup \{a,b\}, \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid s \in \tau P\} \cup \{\langle b\rangle\hat{}t \mid t \in \tau Q\})$ $(a \neq b)$

$a \neq b$ として，a が生起すればその後は Pと同様，b が生起すればその後は Qと同様に挙動するプロセスである（a それから Pまたは b それから Q） (choice)．

(4) $(x:B \to P(x)) = ((\bigcup_{x \in B} \alpha P(x)) \cup B, \{\langle\rangle\} \cup \{\langle x\rangle\hat{}s \mid x \in B \wedge s \in \tau P(x)\})$

(5) $P/s = (\alpha P, \{t \mid s\hat{}t \in \tau P\})$

プロセス Pの足跡がちょうど sであったとき，それ以降の Pの挙動をするプロセスである（s のあとの P）．

(6) $P \| Q = (\alpha P \cup \alpha Q, \{s \mid s \in (\alpha P \cup \alpha Q)^* \wedge s \in (s \uparrow \alpha P) \in \tau P \wedge (s \uparrow \alpha Q) \in \tau Q\})$

プロセス Pとプロセス Qが並行に動いているようなプロセスのことである（P は Qと並行）．

$s \uparrow \alpha P$ は足跡 sを事象集合αP に制限したものを表している．

(7) $(\mu X:A.F(X)) = (A, \bigcup_{n \geq 0} \tau(F^n(STOP)))$

事象集合 Aをもつプロセスでプロセス変数 Xに対する方程式 X=F(X) の一意解となるもの．

4．2　INVSYSTEM の仕様

例題の仕様を書くと次のようになる

```
Id = [0..maxid]  (* The set of Customer
                          Id Numbers *)
Money = NonNegativeInteger
Balance = Integer
INVSYSTEM = Customers || DataBase || Invoice
          || Report
Customers = ||(i∈Id) i:Customer
Customer = (openacc → C)
C = (buybook→ u:Money→ book!u →  C
    | makepay → v:Money→ pay!v  →  C)
DataBase = ||(i∈Id) i:D
D = (openacc  → D(0,0,0))
D(balance,sale,receipt)
    = (book?u → D(balance,sale+u,receipt)
      | pay?v → D(balance,sale,receipt+v)
      | mkinv → state!<balance,sale,receipt>
              → D(balance+sale-receipt,0,0))
Invoice = (start→ Inv(0)(0,0,0))
i < maxid ⇒ inv(i)(tb,ts,tr)
    = (i.mkinv→i.state?x →left.mkinv!<i,x>
      →Inv(i+1)(tb+π1(x),ts+π2(x),tr+π3(x)))
i = maxid ⇒Inv(i)(tb,ts,tr)
    = (i.mkinv→i.state?x →left.mkinv!<i,x>
      →left.mkrep!<tb,ts,tr> →Invoice)
Report = (left.mkinv?x  →  right!ε1(x)
         → left.mkrep?x → rightε2(x))
```

上の仕様で、π_1、π_2、π_3 は次の射影

$$\pi_x(\langle e_1, e_2, e_3\rangle) = e_x \quad (x=1,2,3)$$

のことで，ε_1 はinvoice の編集を，ε_2 はreportの編集をする関数を意味する．

編集の詳細については省略．

4．3　Modula-2による実現

この仕様を Modula-2 で実現した． Modula-2 の並行処理機能を利用すれば素直に実現できるからである．

仕様に見えるＣＳＰプロセスのうち DataBase, Invoice, Report はそのまま Modula-2 のプロセスとして実現した．従って DataBase のプロセスは顧客の数だけある．Customerは，単に入力データを受け取り，そのまま Database に引き渡すだけなので入力処理中に吸収されてしまった．

相互通信については，SENDと WAIT で実現した．

その一部は次の通り．

```
MODULE INVSYSTEM;
 FROM DataBase  IMPORT  D;
 FROM Invoice   IMPORT  INV;
 FROM Processes         IMPORT  Alpha, SEND;
 FROM InOut     IMPORT  Read, ReadInt, Write, WriteString, WriteLn;
 FROM PC9801    IMPORT  StartScroll, EnableLastline, DisableLastline,
                        SaveCursor, SetCursor, ClearScreen, ClearLine,
                        WriteInvert;
CONST   BEL = 07C;  ESC = 33C;
VAR     process, action: CHAR;  id: INTEGER;
BEGIN
  EnableLastline;  ClearScreen;
  WriteInvert ("INVSYSTEM = CS || DB || INV.");
  WriteLn;  WriteLn;
  WriteString ("  >Customer id ( OPENACC | BOOK u | PAY v )");
  WriteLn;  WriteLn;
  WriteString ("  >Invoice");  WriteLn;   WriteLn;
  StartScroll (7);
  LOOP
    SaveCursor;  WriteString (">");  Read (process);
    CASE  process  OF
      ESC:  EXIT
    | "c", "C":  WriteString ("Customer id: ");  ReadInt (id);
                 WriteString ("  ");  Read (action);
                 CASE  action  OF
                   "o", "O":  SEND (D, id, openacc);  WriteLn
                 | "b", "B":  SEND (D, id, book);  WriteLn
                 | "p", "P":  SEND (D, id, pay);  WriteLn
                 ELSE
                   Write (BEL);  SetCursor;  ClearLine
                 END
    | "i", "I":  WriteString ("Invoice Start");  WriteLn;  WriteLn;
                 SEND (INV, 0, start);  WriteLn
    ELSE
      Write (BEL);  SetCursor;  ClearLine
    END
  END;
  DisableLastline
END  INVSYSTEM.

IMPLEMENTATION MODULE DataBase;
  FROM Invoice IMPORT  INV;
  FROM Processes IMPORT
    Alpha, MessPtr, Message, StartProcess, WAIT, WAITS, OUTPUT;
  FROM InOut IMPORT  ReadInt, Write, WriteString;
  FROM SYSTEM IMPORT  ADR;
VAR     i: INTEGER;
PROCEDURE  D;
  VAR   action: Alpha;  rec: Message;  recptr: MessPtr; u, v: INTEGER;
BEGIN
  rec.i := i;  recptr := ADR (rec);
  WAIT (openacc);  INCL (openids, rec.i);  WriteString ("OPENACC");
  WITH  rec  DO
    b := 0;  s := 0;  p := 0
  END;
  LOOP
    WAITS (action);
    CASE  action  OF
      book :    WriteString ("BOOK u: ");  ReadInt (u);
                WITH  rec  DO
                  s := s + u
                END
    | pay  :    WriteString (" PAY v: ");  ReadInt (v);
                WITH  rec  DO
                  p := p + v
                END
    | mkinv:    OUTPUT (INV, 0, state, recptr);
                WITH  rec  DO
                  b := b+s-p;  s := 0;  p := 0
                END
      ELSE      Write (07C)     (* Bell *)
    END
  END
END  D;
BEGIN
  openids := {};
  FOR  i := 0  TO  15  DO  StartProcess (D, i, 512)  END
END  DataBase.

IMPLEMENTATION MODULE  Invoice;
 FROM DataBase  IMPORT  D, openids;
 FROM Report    IMPORT  R;
 FROM Processes         IMPORT
   Alpha, MessPtr, Message, StartProcess, SEND, WAIT, INPUT, OUTPUT;
 FROM SYSTEM    IMPORT  ADR;
PROCEDURE  INV;
  VAR  id: INTEGER;  rec, sum: Message;  recptr, sumptr: MessPtr;
BEGIN
  recptr := ADR (rec);  sumptr := ADR (sum);
  LOOP
    WAIT (start);
    WITH  sum  DO
      b := 0;  s := 0;  p := 0
    END;
    FOR  id := 0  TO  15  DO
      IF  CARDINAL(id) IN openids  THEN
        SEND (D, id, mkinv);  INPUT (D, id, state, recptr);
        SEND (R, 0, mkinv);   OUTPUT (R, 0, state, recptr);
        WITH  sum  DO
          b := b + rec.b;
          s := s + rec.s;
          p := p + rec.p
        END
      END
    END;
    SEND (R, 0, mkrep);  OUTPUT (R, 0, state, sumptr)
  END
END  INV;
BEGIN
  StartProcess (INV, 0, 512)
END  Invoice.

IMPLEMENTATION MODULE Report;
 FROM Invoice   IMPORT  INV;
 FROM Processes         IMPORT
   Alpha, MessPtr, Message, StartProcess, WAITS, INPUT;
 FROM InOut     IMPORT  WriteString, WriteLn, WriteInt;
 FROM SYSTEM    IMPORT  ADR;
PROCEDURE  R;
  VAR  action: Alpha;  rec, sum: Message;  recptr, sumptr: MessPtr;
BEGIN
  recptr := ADR (rec);  sumptr := ADR (sum);
  LOOP
    WAITS (action);
    CASE  action  OF
      mkinv:  INPUT (INV, 0, state, recptr);
              WITH  rec  DO
                WriteString ("Customer ");      WriteInt (i, 1);
                WriteString (",  Balance ");    WriteInt (b, 1);
                WriteString (",  Book ");       WriteInt (s, 1);
                WriteString (",  Pay ");        WriteInt (p, 1)
              END
    | mkrep:  INPUT (INV, 0, state, sumptr);
              WITH  sum  DO
                WriteString ("Total Balance "); WriteInt (b, 1);
                WriteString (",  Book ");       WriteInt (s, 1);
                WriteString (",  Pay ");        WriteInt (p, 1)
              END
    END;
    WriteLn
  END
END  R;
BEGIN
  StartProcess (R, 0, 512)
END  Report.

DEFINITION MODULE Processes;
  EXPORT QUALIFIED
    Alpha, MessPtr, Message, StartProcess, SEND, WAIT, WAITS, INPUT, OUTPUT;
  TYPE  Alpha = (null, openacc, book, pay, start, mkinv, mkrep, state);
        MessPtr = POINTER TO Message;
        Message = RECORD
                    i, b, s, p: INTEGER
                  END;
  PROCEDURE StartProcess (P: PROC;  i: INTEGER;  n: CARDINAL);
  PROCEDURE SEND (P: PROC;  i: INTEGER;  e: Alpha);
  PROCEDURE WAIT (e: Alpha);
  PROCEDURE WAITS (VAR e: Alpha);
  PROCEDURE INPUT (P: PROC;  i: INTEGER;  c: Alpha;  recptr: MessPtr);
  PROCEDURE OUTPUT (P: PROC;  i: INTEGER;  c: Alpha;  recptr: MessPtr);
END Processes.

DEFINITION MODULE DataBase;
  EXPORT QUALIFIED  D, openids;
  VAR  openids: BITSET;
  PROCEDURE  D;
END  DataBase.

DEFINITION MODULE  Invoice;
  EXPORT QUALIFIED  INV;
  PROCEDURE  INV;
END  Invoice.

DEFINITION MODULE Report;
  EXPORT QUALIFIED R;
  PROCEDURE  R;
END  Report.
```

5．まとめ

設計法１は表示的意味論の枠組といっても単に関数型言語でプログラムを書いたにすぎない．continuation など厄介なことは現れないので簡単であり，企業内で採用できる方法である．普及のためには，枠組をさらに明確にし，具体構文，抽象構文，文脈条件などと，記述項目を分離して書くといいようである．

設計法２はＣＳＰの記法で ad hoc に仕様を書いたようにみえるが，これはＪＳＤの考えをＣＳＰで記述したものといえる．したがってＪＳＤの普及をまず計る必要がある．

設計法１，２ともに，結局，対象主導型の設計法で，仕様を作成すると実現方法が自然にえられるので，仕様から実現を導くプログラム生成系ができる可能性がある．仕様作成，設計，文書化，保守などのための支援系も考えることができ，これらのためにも，両者の記述言語を設定し，保守し発展させる必要を感じる．

本発表は，ソフトウェア工学味見会の成果の一部である．ソフトウェア工学味見会のメンバーは次の通りである．

板倉教，加藤潤三，峰尾欽二，森澤好臣，宗像清治，澤田晟司，染谷誠，田端福雄，山崎利治

参考文献

1）加藤潤三，染谷誠，板倉教，田端福雄，森沢好臣，山崎利治：仕様記述の味見、情報処理学会ソフトウェア工学研究会，40-12(1985.2.7)

2）J.D.ワーニエ,B.M.フラナガン 著，鈴木君子訳：ワーニエ方式によるプログラミング学習（上，下），日本能率協会（1973）

3）M.A.ジャクソン著，鳥居宏次訳：構造的プログラム設計の原理，日本コンピュータ協会(1980)

4）K.Futatugi,K.Okada: A hierarchical structuring method for functional software systems, 6th ICSE, IEEE, pp.393-402(1982)

5）R.Nakajima,T.Yuasa(eds) : The IOTA programming system, LNCS No.160, Springer-Verlag(1983)

6）B.A.Silverberg,et al.:The HDM Handbook 3 vols, SRI Internationl(1983)

7）C.B.Jones : Systematic software development using VDM, Prentice/Hall(1986)

8）E.W.ダイクストラ 著，浦昭二，土居範久，原田賢一訳：プログラミング原論，サイエンス社(1985)

9）D.Gries: The Science of programming, Springer-Verlag(1981)

10）C.A.R.Hoare:Communicating sequential processes , Prentice/Hall(1985)

11）中島玲二：数理情報学入門，朝倉書店(1982)

12）M.A.Jackson: System development, Prentice/Hall (1983)