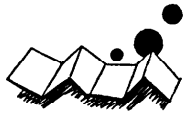
解　説



ストリームを扱う言語 Stella による在庫管理システムの記述†

久　世　和　資††

1. はじめに

近年，ソフトウェアの生産性・保守性が問題になっている．プログラムの設計技法としては，ジャクソン法[1]，ワーニエ法[2]，複合設計[3]などが知られている．これらの設計技法は，プログラミング言語とは独立に考えられており，それぞれの設計技法に向いた言語は，特に用意されていない．しかし，ソフトウェアの生産性・保守性の向上のためには，設計技法と共に，その技法を用いて設計されたプログラムを的確に表現し，記述できるプログラミング言語があることが望ましい．われわれは，このような立場から，新たなプログラミングの捉え方および手法を提案し，同時にその手法に従って記述できるプログラミング言語を提供する．

まず，プログラミング手法であるが，従来のプログラムは，「データ」よりも「処理」の方に着目して，処理の連続として設計される．しかし，プログラムを処理の流れではなく，データの流れに沿って設計した方が，自然に表現できる問題も多いと思われる．そこで，プログラミングに，データの流れ（ストリーム）を取り入れることを提案した[4]．種々の問題について考察した結果，多くのプログラムが複数のストリーム (stream) とそれらのストリームに対して簡単な操作をするモジュール (module) 群とによって表現できることが，わかってきた．この手法によれば，それぞれのモジュールは単純であるが，それらをストリームで結合することによって，より複雑な作業が記述できる．また，各モジュール間の通信は，ストリームによるものに限定しているので，モジュラリティが高く，モジュール単位での作成，デバッグが容易にできるし，プログラムの仕様を変更する際にも変更に関係するモジュールのみを付け替えればよく，その作業も手軽にできる．

以上に述べたようなストリームの操作とモジュールの結合を自由にできる機能を持つプログラミング言語として，Pascal をベースに設計されたのが Stella である．現在の処理系は，Stella から Pascal へのトランスレータであり，Pascal 上で疑似並列処理を行っている[6]．

以下，ストリームによるプログラミングと Stella について，「在庫管理システム」[9]のプログラミングを例に取り上げて解説する．

2. 在庫管理システムの解釈

在庫管理システムの仕様には，あいまいな点が，いくつかあるが，特に気になるのは，問題文「在庫が無いか数量が不足の場合には，そのむね，依頼者に電話連絡し，在庫不足リストに記入する」中の「在庫が無いか数量が不足」の部分である．在庫がある銘柄に対して在庫量以上の出庫依頼があれば，在庫不足リストに記入するのだが，続いて同じ銘柄に対して在庫量以下の出庫依頼があった場合，在庫が無いとするのか，あるとして出庫するのかが，明確でない．そこで，「同じ銘柄について出庫依頼があると，出庫指示は依頼の順番にする」という前提条件を設ける．したがって，この場合には，第2の出庫依頼に対しては在庫がないものとして在庫不足リストに記入し，出庫されない．

3. ストリームとストリームを扱う言語 Stella

3.1 ストリームによるプログラミング

Stella で記述されたプログラムは，複数のモジュールと，それらを結ぶ複数のストリームで構成される．ストリームは，一定の型を持つデータの列である．ストリームの流れる方向は，一方向であり，モジュールへ入ってくるストリームを入力ストリーム，モジュールから出て行くストリームを出力ストリームと呼ぶ．

† Programming an Inventory Control System using Stella -a Programming Language with Streams by Kazushi KUSE (Doctoral Program in Engineering, University of Tsukuba).

†† 筑波大学工学研究科

モジュールが扱えるストリームの数に制限はなく，モジュールは，一般にネットワーク状に結合される．各モジュールでは，入力ストリームからデータを1つずつ取り出し（入力），そのデータを使って処理をして，出力ストリームにデータを送り出す（出力）．プログラムを実行すると，各モジュールは並列に動作し，モジュール間をストリームを通してデータが流れる．

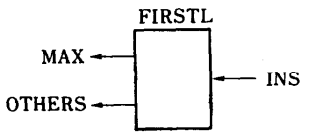
Stella では，結合したモジュール群も新たに1つのモジュールとして扱える．複数のモジュール群の結合によって得られるモジュールを合成モジュールと呼び，単体のモジュールを基本モジュールと呼ぶ．このモジュールの階層化の方向によってトップダウン開発とボトムアップ開発の2種類の開発方式がある．

トップダウン開発は，問題が与えられると，外側の仕様から内側に向かってモジュールを構成して行く方式である．最初は，問題全体を1つのモジュール，問題の入出力をストリームとする．次にそのモジュールを構成するのに必要なモジュールとそれらを結合するストリームを決定する．各モジュールに対して同様の操作を繰り返す．モジュールがこれ以上，分割できないならば，これを基本モジュールとして，ストリームの入出力とデータに対する処理を記述する．

ボトムアップ開発は，トップダウン開発とは逆に複数の基本モジュールまたは合成モジュールを，最初に用意しておき，これらのモジュールを組み合わせて新たな合成モジュールを順次作成して行く方式である．この方式は，プログラムの部品化・再利用に適している．

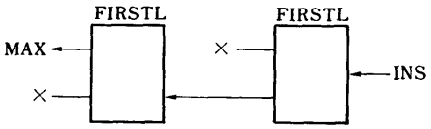
プログラム設計技法とともに，プログラム仕様の適切な文書化が問題になっているが，仕様書は文書よりも図式で表現した方が理解しやすい場合が多い．Stella でも，合成モジュールの構成を表現するのに図式を用い，これをストリーム図式と呼ぶ．ストリーム図式は，各階層ごとにストリームを矢印，モジュールを箱で表わしたものでプログラムの仕様書としても利用できる．

簡単な例として，整数型の入力ストリームから2番目に大きい数を求めるプログラム SECONDL をストリームを使って作成する．部品として整数型の入力ストリーム INS 中の最大値 MAX とそれ以外の数 OTHERS を出力するモジュール FIRSTL が使えるものとする．FIRSTL はストリーム図式では，**図-1** のように表現できる．2番目に大きい数を求めるプログラム SECONDL は，2つの FIRSTL を1本のストリームで結合することによって得られる（**図-2**）．図中の記号×は，出力を他のモジュールに結合しないことを表わし，非結合子と呼ぶ．



整数型ストリーム INS の中で最大値を MAX に，それ以外の値を OTHERS に出力する．

図-1 モジュール FIRSTL のストリーム図式



INS の中で2番目に大きい数を MAX に求める．

図-2 プログラム SECONDL のストリーム図式

Stella では，動的に同じモジュールを複数個，生成して，それらをストリームで結合することもできる．FIRSTL に，この機能を適用すると，ソーティングのプログラムが記述できる．

3.2 言語 Stella

Stella の言語仕様は，Pascal の仕様を基本としており，Pascal の文はすべて使用できる．ここでは，上の SECONDL の例を用いて言語仕様の概要を説明する．SECONDL を Stella で記述したのが，**図-3** のプログラムである．Stella で記述されたプログラムは，（1）ストリーム定義・宣言部，（2）モジュール宣言部，（3）結合・実行部の3つの部分に分類できる．以下，それぞれの部分について説明する．

（1） ストリーム定義・宣言部

ストリーム定義・宣言部は，Pascal の型定義部と変数宣言部に対応する．ストリーム定義部では，ストリームの型を定義し（**図-3** ①），定義された型は，各モジュールの入力または，出力ストリームの型指定に使用する．ストリーム宣言部は，結合に用いるストリームを宣言する部分である（**図-3** ②）．ストリームの宣言は，ストリーム定義部で定義した型名を使用してもよいし，直接，

stream of 要素型

の形で記述してもよい．

（2） モジュール宣言部

モジュール宣言部は，プログラムで使用するモジュールを宣言する部分である．モジュールの頭部では，

```
program SECONDL (INPUT, OUTPUT);
    type STRINT=stream of INTEGER; ……①
    var  S1, S2, S3: STRINT; ……②                    {ストリーム定義・宣言部}

  module FIRSTL (INS: STRINT) MAX, OTHERS: STRINT;……③
    var I, LASTMAX: INTEGER;

    procedure SWAP (var A, B: INTEGER);
      var C: INTEGER;
    begin C:=A; A:=B; B:=C end;

  begin
    LASTMAX:=next INS ……④                           {モジュール宣言部}
      ≪WRITELN ('empty stream')≫; ……⑤
    loop
      I:=next INS ≪next MAX:=LASTMAX≫;
      if I > LASTMAX then SWAP (I, LASTMAX);
      next OTHERS:= I ……⑥
    end
  end;

begin
  connect
    READS (INPUT) S1; ……⑦
    FIRSTL (S1) free, S2; ……⑧                       {結合・実行部}
    FIRSTL (S2) S3, free;
    WRITES (OUTPUT, S3)……⑨
  end
end.
```

図-3　プログラム SECONDL の Stella による記述

module モジュール名

の後の（ ）内に入力ストリームを宣言し，（ ）の後に出力ストリームを宣言する（図-3 ③）．

入力ストリームから要素を1つ入力するには，式の中に

next 入力ストリーム名

の形で記述する（図-3④）．図-3④は，入力ストリーム INS から，要素を1つ入力し，変数 LASTMAX に代入する文である．入力ストリームを生成するモジュールが終了し，要素が尽きた状態を eos (end of stream) と呼ぶ．eos状態のストリームから要素を入力しようとすると，eos 例外処理として，通常は，そのモジュール自身も終了する．終了する直前に何らかの処理をしたい時には ≪ ≫ 内に記述する（図-3⑤）．

逆に出力ストリームに要素を1つ出力するには，

next 出力ストリーム名:=式

の形で記述する（図-3 ⑥）．出力ストリームを消費するモジュールが終了した状態を blocked (blocked stream) と呼ぶ．eos と同様に blocked 状態のストリームへ要素を出力しようとすると，blocked 例外処理として，通常は，そのモジュール自身も終了する．

（3） 結合・実行部

結合・実行部は，モジュール宣言部で宣言したモジュールの間を，ストリーム宣言部で宣言したストリームを用いて結合し，実行する部分で，**connect** と **end** の間に結合の仕方を記述する．2つのモジュールを結合するには，一方のモジュールの入力ストリーム部と，他方のモジュールの出力ストリーム部に，同じストリーム名を記述する（図-3 ⑧）．出力ストリームをどのモジュールにも結合しない時には，非結合子として **free** を記述する．

また，ストリームの要素を，標準入力や外部ファイルから入力する標準モジュール READS（図-3⑦）と，逆に標準出力や外部ファイルへ出力する標準モジュール WRITES（図-3 ⑨）がある．

3.3 Stella 処理系

現在，Stella で記述したプログラムを実行するのに Stella から Pascal へのトランスレータを使って，Pascal プログラムへ変換して，Pascal 上で疑似並列処理をしている．トランスレータは，Pascal-P 4 コンパイラ（約 4000 行）を改良したもので，Pascal で約 6000 行ある．トランスレータ自体が Pascal で記述されているので，Pascal の処理系さえあれば，Stella で

記述したプログラムを走らせることができる.

4. 在庫管理システムの作成法

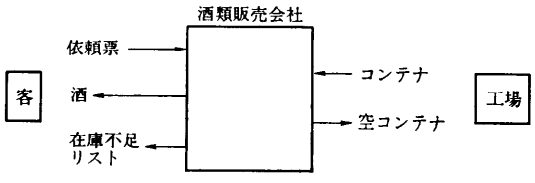
「在庫管理システム」を Stella で記述する. この種の問題はトップダウンに作成できる. 以下, 各レベルごとにストリーム図式を使って説明する. 上の方のレベルは, 直接, プログラムに関係しないが, 問題全体を理解するのに重要である.

4.1 問　題

まず, 問題を整理すると, 酒類販売会社が1つのモジュールと考えられる. 客は, 酒類販売会社に対して依頼票を送り, 在庫の状況に応じて, 依頼した酒または在庫不足リストを受け取る. 工場は, 酒類販売会社にコンテナを送り, 空コンテナがあれば受け取る. ここで, 酒やコンテナの連続列を, ストリームと考えれば, この問題は図-4 のストリーム図式で表わせる.

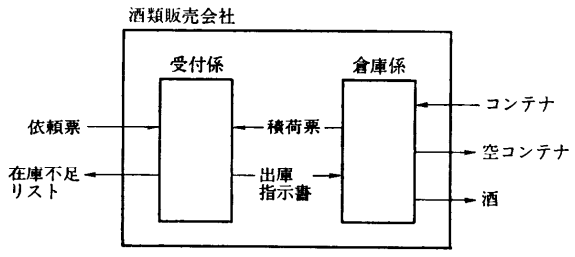
4.2 酒類販売会社

次に酒類販売会社の中身を整理してみると, 受付係と倉庫係の2つのモジュールと, それらを結ぶ積荷票と出庫指示書の2つのストリームで構成される (図-5). 積荷票は, 倉庫に運ばれたコンテナの内蔵品の状



依　頼　票: 品名, 数量, 送り先名 (1銘柄のみ)
酒　　: 約200銘柄
コンテナ: 10銘柄まで混載できる
在庫不足リスト: 送り先名, 品名, 数量

図-4 問題 (在庫管理システム) のストリーム図式



積荷票: コンテナ番号 (5桁), 搬入年月, 日時,
(品名, 数量) の繰り返し
出庫指示書: 注文番号, 送り先名,
(コンテナ番号, 品名, 数量, 空コンテナマーク)
の繰り返し

図-5 酒類販売会社のストリーム図式

品　名	コンテナ番号	数　量
純	12345	50
レモン	12345	100
炭酸水	12345	200
純	54321	80
レモン	54321	100
⋮	⋮	⋮

図-6 積荷管理表の例

況を倉庫係から受付係へ知らせる書類であり, 出庫指示書は, 依頼品の出庫を受付係から倉庫係へ知らせる書類である.

4.3 受 付 係

受付係の業務をプログラミングするのが「在庫管理システム」の本題である. 受付係の業務は, 次の3つに分類できる.

① 依頼票に対して在庫がある場合は, 依頼票を受け, そのつど, 倉庫係へ出庫指示書を出す.

② 在庫が無いか, 数量が不足の場合には, 在庫不足リストに記入する. そして, 当該品の積荷が必要量に達した時点で, 不足品を出庫する.

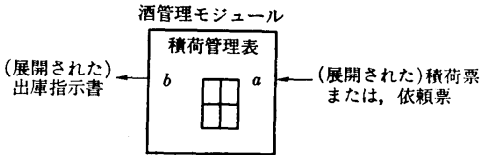
③ 空になる予定のコンテナを倉庫係に知らせる.

以上の①から③までの業務に対して, それぞれ, 酒管理モジュール, 注文管理モジュール, コンテナ管理モジュールを用意する. 以下, 各モジュールについて説明する.

(1) 酒管理モジュール

酒管理モジュールでは, (出庫可能な) 依頼票を受け, そのつど, 出庫指示書を出す. このために, 酒の在庫状況を管理する積荷管理表 (図-6) が必要である. この表は, 品名とコンテナ番号の組に対する数量を登録しておく. 積荷票がくれば, この表に登録し, 依頼票がくれば, 依頼数量が満足されるまで, この表を検索して, 出庫指示書を出す.

次に酒管理モジュールが扱うストリームであるが, 出力ストリームは, 出庫指示書について1本用意すればよい. 入力ストリームは, 積荷票と依頼票の2種類が考えられるが, 積荷管理表を操作する順番は, 在庫管理システムへの積荷票および依頼票の入力順に従わなければならないので, この2種類の書類については, 1本の入力ストリーム上を流すことにする. また, 積荷票と出庫指示書には, 繰り返しの構造があるが, ストリームの要素を1枚の書類単位とす



入力 a が積荷票の時　積荷管理表に登録する．
入力 a が依頼票の時　積荷管理表を検索し，依頼量を満足するまで，出庫指示書を b に出力する．

図-7　酒管理モジュールのストリーム図式

るよりも，この繰り返しの１回について１要素とする方が，先に述べた表の操作に適合する．つまり，積荷票の１回の繰り返しが，積荷管理表の１構成要素に一致するので，更新の操作は，入力ストリームの１要素に対して行えばよい．依頼票に対しても，積荷管理表を検索して適当な積荷が，見つかるごとに，出力ストリームの１要素として出力すればよい．酒管理モジュールのストリーム図式を，**図-7** に示す．

（２）　注文管理モジュール

注文管理モジュールは，依頼票に対して在庫が不足の場合には，在庫不足リストに記入し，当該品の積荷が必要量に達すると，不足品を出庫する．このためには，酒品目別に内蔵量を管理する酒別内蔵量表（**図-8**）と在庫不足で出庫できなかった依頼を管理する在庫不足注文表（**図-9**）の２つが必要である．

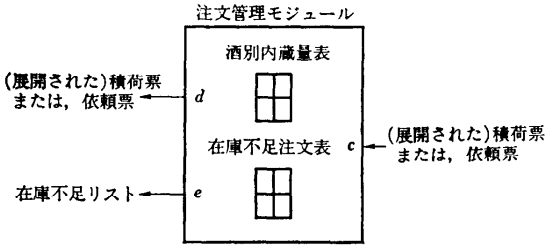
依頼票がくると，依頼数量と酒別内蔵量を比較して出庫可能であれば，酒別内蔵量から依頼数量を差し引いて，依頼票を次のモジュールへ送る．出庫不可能ならば，在庫不足リストを出し，在庫不足注文表に登録する．

品　名	内蔵量
純	130
レモン	200
炭酸水	200
⋮	⋮

図-8　酒別内蔵量表の例

品　名	送り先名	数　量
純	鈴木	100
純	高山	80
レモン	坂元	200
⋮	⋮	⋮

図-9　在庫不足注文表の例



入力 c が積荷票の時　入庫量を酒別内蔵量表に加えて，d に出力する．新たな酒別内蔵量で在庫不足注文表を検索して出庫できる依頼があれば，依頼票を d に出力する．
入力 c が依頼票の時　依頼数量を酒別内蔵量表で調べて
出庫可能な時　依頼数量を酒別内蔵量表から差し引いて依頼票を d に出力する．
出庫不可能な時　在庫不足リストを e に出力し，在庫不足注文表に登録する．

図-10　注文管理モジュールのストリーム図式

積荷票がくると，酒別内蔵量に入庫数量を加えて，積荷票を次のモジュールへ送る．新たな内蔵量で出庫可能になった依頼を在庫不足注文表から検索し，次のモジュールへ送る．

注文管理モジュールが扱うストリームについては，先ほどと同じ理由で，積荷票と依頼票は，同じストリームを共有することとし，その入出力用の各１本と，在庫不足リストの出力用の１本の計３本が必要である（**図-10**）．なお，積荷票については，（１）と同様に繰り返しの１回をストリームの１要素とする．

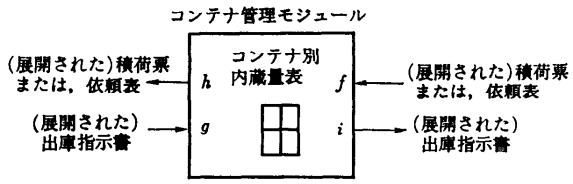
（３）　コンテナ管理モジュール

コンテナ管理モジュールは，空になったコンテナを搬出するために，出庫指示書に印を付けるモジュールである．このために，コンテナ別に内蔵量を管理するコンテナ別内蔵量表（**図-11**）が必要である．

積荷票がくるとコンテナ別内蔵量に入庫数量を加え，次のモジュールへ積荷票をそのまま送る．出庫指示書がくるとコンテナ別内蔵量から出庫数量を差し引いて，出庫指示書を次のモジュールへ送る．この際に内蔵量が０になると，出庫指示書に空コンテナ搬出

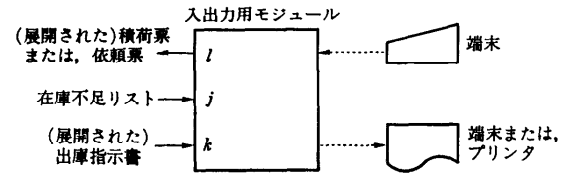
コンテナ番号	内蔵量
12345	350
54321	180
⋮	⋮

図-11　コンテナ別内蔵量表の例



入力 f が積荷票の時　コンテナ別内蔵量表に入庫数量を加えて，h に出力する．
入力 f が依頼票の時　そのまま h に出力する．
g から出庫指示書を入力した時　コンテナ別内蔵量表から出庫数量を引き，i に出力する．その際，コンテナ別内蔵量が0ならば，空コンテナ搬出マークを付ける．

図-12　コンテナ管理モジュールのストリーム図式



端末から積荷票を読み込んだ時　展開して l に出力する．
端末から依頼票を読み込んだ時　l に出力する．
j から在庫不足リストを入力した時　端末または，プリンタに書き出す．
k から出庫指示書を入力した時　端末または，プリンタに書き出す．

図-13　入出力用モジュールのストリーム図式

マークを付ける．

ストリームは，積荷票について入出力用が，各1本，出庫指示書についても入出力用が，各1本で，計4本用意する（図-12）．なお，積荷票と出庫指示書は，(1)，(2)と同様に繰り返しを，ストリームの1要素とする．また，積荷票は，他のモジュールでは，依頼票と同じストリームを共有するので，ここでも，操作には無関係であるが依頼票も入力し，そのまま何もせずに出力することにする．

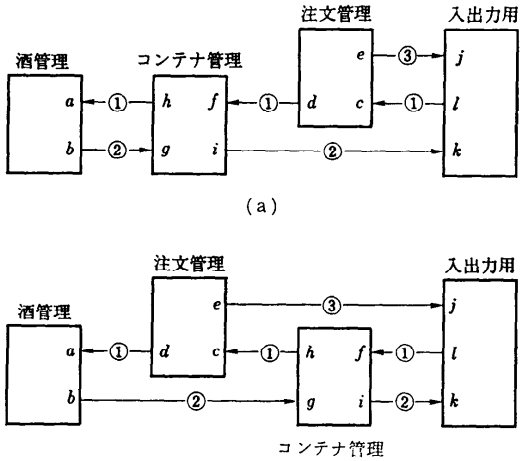
これらのモジュールのほかに端末から積荷票および依頼票を読み込んで，ストリームを作り出し，逆にストリームから，出庫指示書および在庫不足リストを端末またはプリンタに書き出す入出力用モジュールが必要である（図-13）．

以上のモジュールを結合して，受付係を作成するが，結合に用いるストリームは，積荷票または依頼票用，出庫指示書用，および在庫不足リスト用の3種類が必要である（図-14）．

モジュールの結合の仕方であるが，酒管理モジュール，注文管理モジュール，および入出力用モジュールは，この順に結合でき，これだけでも③の業務以外は，実行できる．③の業務を行うコンテナ管理モジュールは，酒管理モジュールと注文管理モジュールの間にも入れられるし（図-15の(a)），注文管理モジュールと入出力用モジュールの間にも入れられ（図-15の(b)），どちらでも，実行結果は同じである．図-16は，図-15の(a)を Stella で記述したものである．また，モジュールの記述例として，酒管理モジュールを図-17に示す．なお，図-18は実行例である．

```
type
    積荷票_依頼票要素=record
                        品名：STRING;
                        数量：INTEGER;
                        case　種類：積荷票_依頼票種類 of
                            積荷票：(コンテナ番号，搬入年月，搬入日時：INTEGER);
                            依頼票：(送り先名：STRING)
                      end;
    出庫指示書要素=record
                      品名，送り先名：STRING;
                      注文番号，コンテナ番号，数量：INTEGER;
                      空コンテナ搬出マーク：BOOLEAN
                    end;
    在庫不足リスト要素=record
                          品名，送り先名：STRING;
                          数量：INTEGER
                       end;
    積荷票_依頼票 =stream of 積荷票_依頼票要素;
    出庫指示書　　=stream of 出庫指示書要素;
    在庫不足リスト=stream of 在庫不足リスト要素;
```

図-14　ストリーム型の定義



ただし，①は，積荷票_依頼票，②は，出庫指示書，③は，在庫不足リストである．

図-15 受付係のストリーム図式

5. 使用方式の評価

在庫管理システムの，設計からコーディング，デバッグ，実現にあたって，Stella の使い勝手をそれぞれの段階について考察する．

（1） 設計，コーディング

まず，ストリーム図式の利用であるが，前述したように問題をストリーム図式で表現し，形式化することによって，問題を理解する手助けとなり，プログラムの仕様書としても役に立った．

次に与えられたデータのストリーム化であるが，各書類に対してそれぞれ1つのストリームを用意するのが自然である．しかし，今回は積荷票と依頼票については，1つのストリームを可変レコード型にして共有した．これは，問題の性質にもよるが，現在，Stella に，1つのモジュール内で同時に別々のストリームの入力を待つ機能がないことも原因している．将来，この機能を導入することによって，より幅の広いプログラミングが可能になると思われる．

さて，在庫管理システムの主な仕事は，各種の表を管理し，入出力データに応じて，これらの表を更新することである．Stella で記述したプログラムでは，それらの表を各モジュールで独立に管理させ，表を更新するデータをストリームとして流した．これは，データを主体としたプログラムであり，従来の処理を主体としたプログラムに比べて，データに対する操作が明確に記述でき，必要なデータさえ流しておけば，正しく動作する．

ストリームについては，ストリームに対して行う操作を，あらかじめ十分検討して，ストリームの仕様を決定する必要がある．特に異なる型のデータが流れるストリームの場合には，それらのデータの流れ方についても，規則を定めておく必要がある．

この解説で示した在庫管理システムにおいてストリームを流れるデータは，入出力用モジュールから酒管理モジュールまで行って，再び入出力用モジュールまで戻ってくる(図-15)．このモジュールとストリームの構造は，入力に対してすぐに何らかの出力を返す在庫管理システムの仕様によるものである．入力と出力を交互に会話的にする必要がないのなら，入出力用モジュールの入力部と出力部は分離でき，並列度も上がる．

（2） デバッグ

Stella では，デバッグは，モジュール単位でモジュールが入出力するストリームとモジュールの内部変数（在庫管理システムの場合は各種の表）をモニタして行う．モジュールが機能単位に比較的小さく分割されていれば，流れるデータを見ながらのデバッグは，従来の処理を追いかけるデバッグより容易である．

（3） 仕様変更

仕様の変更は，変更に関係するモジュールのみ書き換えるか，付け替えればよく，容易に行える．たとえば，「在庫不足であった依頼に対して，積荷が必要量に達した時点で，不足品を出庫する」というのは，前回までの問題[9]からの仕様変更部分であるが，この場合は，注文管理モジュールを作り直せばよい．しかし，ストリームの仕様に変更が必要になった場合には，そのストリームを扱っているすべてのモジュール

```
connect
    入出力用      (出庫指示書1，在庫不足リスト1)    積荷票_依頼票1；
    注文管理      (積荷票_依頼票1)                  積荷票_依頼票2，在庫不足リスト1；
    コンテナ管理  (積荷票_依頼票2，出庫指示書2)     積荷票_依頼票3，出庫指示書1；
    酒管理        (積荷票_依頼票3)                  出庫指示書2
end
```

図-16 結合・実行部（図-15(a)）の Stella による記述

```
module 酒管理 (INS: 積荷票_依頼票) OUTS: 出庫指示書;
    var SHIP_REQ: 積荷票_依頼票要素;
        INSTRUCT: 出庫指示書要素;
        積荷管理表: array [1..100] of record
                                        品名: STRING;
                                        コンテナ番号, 数量: INTEGER
                                    end;
        INDEX, LAST: 0..100;
begin
    LAST:=0;
    loop
        SHIP_REQ:=next INS;          {積荷票または依頼票の入力}
        if SHIP_REQ. 種類=積荷票 then
            begin                                                          }
                LAST:=LAST+1;                                              }
                積荷管理表 [LAST]. 品名:=SHIP_REQ. 品名;                    } {積荷票の登録}
                積荷管理表 [LAST]. 数量:=SHIP_REQ. 数量;                    }
                積荷管理表 [LAST]. コンテナ番号:=SHIP_REQ. コンテナ番号      }
            end
        else        SHIP_REQ. 種類=依頼票
            begin                                                                          }
                INSTRUCT. 品名:=SHIP_REQ. 品名;                                            }
                INSTRUCT. 送り先名:=SHIP_REQ. 送り先名;                                    }
                INDEX:=0;                                                                  }
                while SHIP_REQ. 数量≠0 do                                                  }
                    begin                                                                  }
                        repeat INDEX:=INDEX+1                                              }
                            until (積荷管理表 [INDEX]. 数量≠0) and                         } {依頼票に
                                  (積荷管理表 [INDEX]. 品名=SHIP_REQ. 品名);               } {対する出
                        INSTRUCT. コンテナ番号:=積荷管理表 [INDEX]. コンテナ番号;          } {庫指示
                        if SHIP_REQ. 数量≧積荷管理表 [INDEX]. 数量                         }
                            then INSTRUCT. 数量:=積荷管理表 [INDEX]. 数量                  }
                            else INSTRUCT. 数量:=SHIP_REQ. 数量;                           }
                        積荷管理表 [INDEX]. 数量:=積荷管理表 [INDEX]. 数量-INSTRUCT. 数量; }
                        SHIP_REQ. 数量:=SHIP_REQ. 数量-INSTRUCT. 数量;                     }
                        next OUTS:=INSTRUCT          {出庫指示書の出力}                    }
                    end                                                                    }
            end
    end
end;
```

図-17 酒管理モジュールの Stella による記述

を変更しなければならない.

(4) 実 行 効 率

Stella の実現方式は，疑似並列処理であるので，最適な実行効率とは言えないが，在庫管理システム程度の会話型システムであると十分，実用に耐える．実際には，Pascal 上で処理するので，効率は，Pascal 処理系に依存する．しかし，より実行効率を必要とするプログラムの場合には，この方式は作成・開発時に利用して，実行時には，Stella で記述された並列プログラムを逐次プログラムに変換して実行するのが望ましい．この並列プログラムから逐次プログラムへの変換をインライン展開と呼び，いくつかの展開系を開発してきた[4),5),8)]．すべての Stella プログラムが，インライン展開できるわけではないが，経験的に，ほとんどのプログラムが，展開可能である．

6. おわりに

Stella は，VAX-11/750 の UNIX 4.2 BSD 上と VMS 上で稼働している．しかし，Stella→Pascal トランスレータは，Pascal で記述してあるので，Pascal 処理系さえあれば，使用できる．

現在，Stella の開発支援環境について検討，実験中である[7)]．この支援環境は，端末の画面上で図式表示を利用してモジュールを自由に結合できるシステムで，結合したモジュール群を実行したり，動作解析[6)]をしたり，新たなモジュールとして登録したりできる．

```
積荷票／依頼票／終了 ？                          積荷票
コンテナ番号，搬入年月，搬入日時 ？              12345      8501      1209
コンテナ内の銘柄数 ？                            2
品名，数量 ？                                    純         100
品名，数量 ？                                    レモン     200
積荷票／依頼票／終了 ？                          依頼票
品名，数量，送り先名 ？                          純         50        鈴木
出庫指示書
    1          鈴木
  12345        純         50
積荷票／依頼票／終了 ？                          依頼票
品名，数量，送り先名 ？                          純         150       高山
在庫不足リスト
    高山       純         100
積荷票／依頼票／終了 ？                          依頼票
品名，数量，送り先名 ？                          レモン     300       坂元
在庫不足リスト
    坂元       レモン     100
積荷票／依頼票／終了 ？                          積荷票
コンテナ番号，搬入年月，搬入日時 ？              54321      8501      1214
コンテナ内の銘柄数 ？                            2
品名，数量 ？                                    純         100
品名，数量 ？                                    レモン     100
出庫指示書
    2          高山
  12345        純         50
  54321        純         100
出庫指示書
    3          坂元
  12345        レモン     200       空
  54321        レモン     100       空
積荷票／依頼票／終了 ？                          終了
```

{システムの出力}　　{ユーザの入力}

図-18　在庫管理システムの実行例

また，ストリームのモニタによるデバッグもできるし，先に述べたインライン展開も行えるものである．

謝辞　本論文の執筆についてご指導下さった筑波大学電子・情報工学系中田育男教授，および同佐々政孝講師に感謝いたします．

参考文献

1) Jackson, M. A.: Principles of Program Design, Academic Press (1975).
2) ワーニエ，J. D.（鈴木訳）：ワーニエ・プログラミング法則集，日本能率協会 (1975).
3) Myers, G. J.（久保，国友訳）：高信頼性ソフトウェアー複合設計，近代科学社 (1976).
4) 中田，佐々：ストリームによるプログラミング，プログラミングシンポジウム，情報処理学会 (1984).
5) Kuse, K., Sassa, M. and Nakata, I.: Analysis and Transformation of Concurrent Processes Connected by Streams,「ソフトウェア科学，工学の数理的方法」研究会，京都大学数理解析研究所講究録 511, pp. 120-143 (1984).
6) 久世，中田，佐々：ストリームを扱う言語のトランスレータ方式による実現，第29回情報処理学会全国大会論文集 3P-10 (1984).
7) 久世，鳥谷，佐々，中田：ストリームを扱う言語 Stella におけるモジュール分割管理機能の実現，第 30 回情報処理学会全国大会論文集 1R-8 (1985).
8) 鳥谷，久世，中田：ストリームを扱う言語 Stella-インライン展開方式の実現，第30回情報処理学会全国大会論文集 1R-9 (1985).
9) 山崎利治：共通問題によるプログラム設計技法解説，情報処理，Vol. 25, No. 9, p. 934 (Sep. 1984).

（昭和 60 年 2 月 6 日受付）