ショートノート

ストリーム・プログラミングのための図式表示を利用した開発支援環境について†

久世和資†† 佐々政孝††† 中田育男†††

プログラムに，データの流れであるストリームを導入することによって，プログラムの記述性，保守性の向上が期待できる．ストリームによるプログラミング言語として設計，開発されたのが Stella である．Stella は，プログラムをデータの流れにそって設計でき，図式的に表現できるといった特徴がある．これらの特徴をより一層，活用し，端末の画面上で，会話的に図式表示を利用して，プログラムの作成支援を行う開発環境について，従来のストリームによるプログラミングと比較して述べる．

1. はじめに

近年，ソフトウェアの生産性，保守性の向上が問題になっている．生産性，保守性の向上のために種々のプログラミング・パラダイムが提案されているが，我我は，プログラムにデータの流れであるストリームを導入することによって，プログラムの生産性，保守性の向上および，プログラムの部品化，再利用の促進を目指している[1]．

ストリームを扱う機能を持ったプログラミング言語として開発したのが Stella[2] である．Stella プログラムは，複数のモジュールとそれらを結ぶストリームで表現され，プログラムを実行すると，各モジュールは，ストリームを通してデータをやりとりしながら，並列に動作する．モジュール間の通信はストリームによるものに限定されているので，モジュラリティが高く，テストが容易で，プログラムの部品化や再利用にも適している．

Stella による記述例には，アルゴリズムを素直に記述できる問題として，ハミング問題，ライニング問題，素数列，フィボナッチ数列の計算などが挙げられる[3]．また，より実用的な問題としては，在庫管理システム[2]，プリティプリンタ，コンパイラのフロントエンドとバックエンド，様々な論理回路のシミュレーションなどがある．

Stella は，通常のプログラミング言語に比べて，プログラムを図式で表現しやすく，データの流れに従ってプログラミングできるといった特徴がある．これらの特徴をより一層，活用すべく，我々は，プログラムの作成，デバッグ，実行，最適化までを総合的に行えるプログラミング開発支援環境を設計した．これを SPRING (Stream PRogrammING environment) と呼び，VAX-11/750，Unix 4.2 bsd 上に C 言語で実現した．

SPRING では，端末の画面上で，図式表示を利用して，図式を会話的に操作することにより，プログラムを作成，修正する．この図式によるプログラミングは，通常のテキストによるプログラミングよりも，容易に行え，生産性，保守性の向上にもつながる．本稿では，従来の Stella によるプログラミングと比較して，SPRING によるプログラミングについて，その特徴を述べる．

2. Stella による従来のプログラミング

Stella が扱うストリームを構成する要素としては，単純な整数型，実数型から複雑な構造を持ったデータまで任意にその型が指定できる．モジュールはそれらのストリームからデータを取り出して処理をし，結果を別のストリームへ送り出す．一つのモジュールが扱えるストリームの本数には制限がなく，プログラムは一般に各モジュールをいくつかのストリームで結合したネットワーク状になる．また，複数の結合されたモジュール群を新たに一つのモジュール（合成モジュール）として扱うこともできる．

Stella によるプログラミングの手順は，まず，モジ

† On Stream Programming Environment Using Diagram Representations by KAZUSHI KUSE (Doctoral Program in Engineering, University of Tsukuba), MASATAKA SASSA and IKUO NAKATA (Institute of Information Sciences and Electronics, University of Tsukuba).
†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系

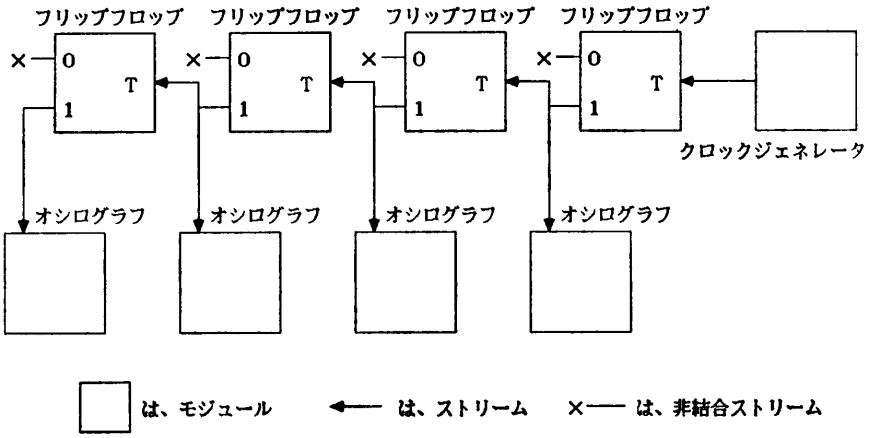

図 1 4段2進カウンタのストリーム図式
Fig. 1 A stream diagram for a ripple counter.

ュールの結合関係を示すストリーム図式を描き，次に，そのストリーム図式に従って，Stella でコーディングし，最後に，コーディングしたプログラムを Stella 処理系で実行する．ストリーム図式において，モジュールとストリームは，それぞれ箱と矢印で表現される．

図 1 は，四つのフリップフロップを用いた4段2進カウンタのシミュレーションを行うプログラムのストリーム図式である．ストリーム図式において，複数に分岐している出力ストリームは，同一のストリームがコピーされることを表し，出力ストリームの×印は，他のモジュールに結合しない非結合ストリームを表す．

次にこのストリーム図式に従って，Stella 言語により，各モジュール本体と結合関係を記述すると**図 2** のようになる．Stella プログラムでは，まず，ストリーム clock の型を定義し（図2 a），次に各モジュールを宣言し(b)，最後にこれらのモジュールの結合，実行部を記述する(c)．モジュールは分割コンパイルも可能であり，外部にあるモジュールは，**external** で参照される．各モジュールでは，モジュール頭部で入出力ストリームを宣言する (b-1)．実際にそれらのストリームからデータを取り出すには，式の中に **next** を使って記述し，逆にストリームへデータを送り出すには，代入文の左辺に **next** を使って記述す

```
program ripple( input, output);
    type    signal = 0 .. 1;
            clock = stream of signal; ------------------------------ (a)
    var     n: integer;

    module  flipflop( t: clock) out0, out1: clock; ------ (b-1)
        var  s, lasts: signal;
             state: boolean;
    begin
        lasts := 0;
        state := false;
        loop
            s := next t;
            if (lasts = 1)and(s = 0) then state := not state;     (b)
            lasts := s;
            next out0 := ord( not state );
            next out1 := ord( state )
        end
    end;

    external module clock_gen;
    external modeul oscillo;

begin
    read( n );
    connect
        clock_gen( n ) #c0;
        flipflop( #c0 ) free, #c1;
        flipflop( #c1 ) free, #c2;
        flipflop( #c2 ) free, #c3;                               (c)
        flipflop( #c3 ) free, #c4;
        oscillo( 1, #c1 );
        oscillo( 2, #c2 );
        oscillo( 3, #c3 );
        oscillo( 4, #c4 )
    end
end.
```

図 2 4段2進カウンタの Stella による記述
Fig. 2 A stella program for a ripple counter.

る．結合，実行部(c)では，**connect** と **end** の間に結合するモジュールを記述し，ストリームで結合する．結合のためのストリームは，一般の引数と区別するために＃をつける．複数の入力ストリーム部に同一のストリーム名が出現する場合は，ストリームのコピーを表し，出力ストリーム部の **free** は非結合ストリームを表す．

最後に，コーディングした Stella プログラムを Stella 処理系で実行する．Stella 処理系は Stella から Pascal へのトランスレータであり，変換した Pascal 上で，擬似的に並列処理する[4]．

3. SPRING を使ったプログラミング

先に述べたプログラミング手順のうち，Stella によるコーディング過程において，モジュール数が多数の際には，結合関係の把握や（単体，全体の）テストが容易でない．開発支援環境 SPRING では，プログラム・テキスト（図2）を意識することなく，図式表示(図1)を用いてプログラミングすることができ，コーディング作業が軽減される．

SPRING では，関連部品モジュールが既存である場合，プログラマが端末上で，前述のストリーム図式を作成すれば，それに従って，Stella プログラムが内部で自動生成される．また，端末の図式上で，モジュールをつなぎ替えたり，削除すれば，Stella プログラムも自動的に変更されるため，プログラムの修正も容易に行える．さらに，生成された Stella プログラムを実行し，ストリーム中を流れるデータをモニタすることによってデバッグもできる．プログラマは，SPRING を用いることにより，単純な部品（モジュール）を端末上で，自由に結合して新たな部品を作成し，それを再び結合の対象として利用するといった操作の繰り返しで，より複雑な処理をする部品を段階的に構築することが可能である．

SPRING の使用の様子を，先の4段2進カウンタのシミュレーション・プログラムの作成を例にして説明する．まず，ライブラリの中から，ハードウェア・シミュレーションのセットを選ぶと，いくつかの部品モジュールがリストされるが，フリップフロップ用のモジュールがないので，モジュール作成コマンドを選択し，画面エディタで作成する（**図3**）．モジュールには，その外部仕様を示すコメントを付加することができる．

次に実行コマンドにより，作成したモジュール flip flop の単体をテストしてみる．入力ストリームを要求してくるので，それに従って，データを入力すると処理され，出力ストリームにデータが出力される（**図4**）．正しい動作が確認されれば，新たな部品として登録し，動作が誤っていれば，モジュールを修正する．このようなモジュール単位での実行機能は，作成中のモジュールの動作確認や，すでに作成されたモジュールを部品として再利用する際の動作の理解に役立つ．

実行コマンドにより，モジュール flipflop が正しく動作することが確認されたら，結合コマンドにより四つのモジュール flipflop を適当な位置に配置し，ストリームを指定することによってそれらを結合する．対象としているモジュールについては，ストリーム名，コメントなどが表示されるので，それらを参考に結合していく．結合が終了すると先ほどと同様，結合したモジュール群を実行し，動作が誤っていれば，つなぎ替えなどの修正を行う（**図5**）．

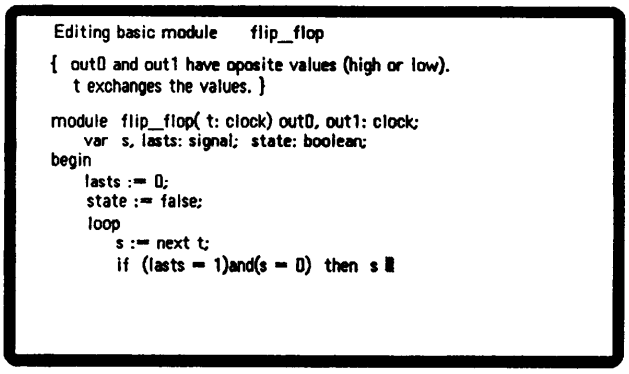

図3 作成コマンドによるモジュール flipflop の作成（途中）
Fig. 3 Making a module flipflop by make-command.

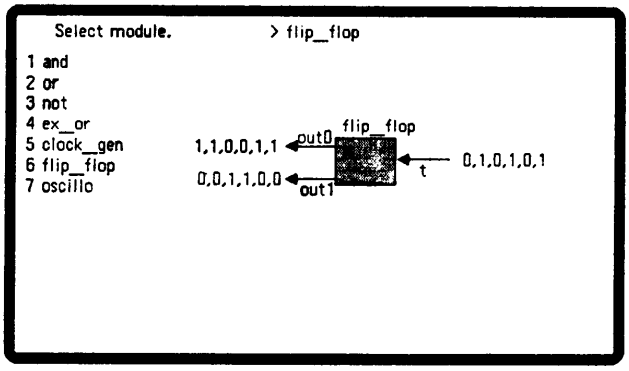

図4 実行コマンドによるモジュール flipflop のテスト
Fig. 4 Testing a module flipflop by execution-command.

Stella では，結合した複数のモジュールをまとめて，一つのモジュールとして，再び，部品として利用できる階層化の機能があるが，SPRING では，この階層化も支援し，段階的なプログラムの構築を容易にしている．ここでは，先ほどの四つの flipflop を合成して，一つのモジュール counter として登録し，モジュール名，ストリーム名，コメントなどを指定する．これ以降，モジュール counter も一つの部品として利用できる．counter と clock_gen と四つの oscillo を結合したのが**図 6** であり，実行結果は**図 7** となる．

Stella の処理系は，Pascal 上での擬似並列処理のため実行効率が良くない．そこで，実行効率を良くするために，ストリームによる通信部分を取り除き，並列プログラムを逐次プログラムに変換するインライン展開[5),6)] による最適化の手法がある．この機能も SPRING に用意されており，必要なら合成したモジュール群について自動的にインライン展開を行い，実行効率の高い部品として登録できる．インライン展開により実行効率は，2倍から 10 倍に向上する．なお，展開コードは，一般に元のプログラムより長くなり，goto 文とラベルによる制御構造が増え，読みにくくなるが，ユーザが直接，これを取り扱うことはないので，差し支えない．

以上の例によって説明したストリーム・プログラム開発支援環境 SPRING の特徴についてまとめる．

① 端末上で，図式表示を利用してモジュールを結合することによりプログラムを作成し，プログラムの修正も，図式表示上で行える．

② モジュールは単体でも，結合したものでも自由に実行し，テストすることができる．

③ 実行時には，ストリーム中を流れるデータを観察することによりデバッグできる．

④ モジュールの階層化の支援を行う．

⑤ インライン展開により，実行効率の良い部品モジュールを得ることができる．

4. おわりに

SPRING は，ストリームによるプログラムが，図式表現しやすく，データの流れにそって記述される特徴を利用した開発支援環境であり，端末の画面上で図式表示を利用して，プログラムの作成や修正を行うものである．実際に SPRING を使用してみると，すでに作成されたプログラムについては，その構成を一目で理解することができる．また，その動作も，ストリームを流れるデータが観察できるので，通常のテキストによるプログラムに比べて，格段に理解しやすいものとなっている．プログラムの作成，修正も，一般には，

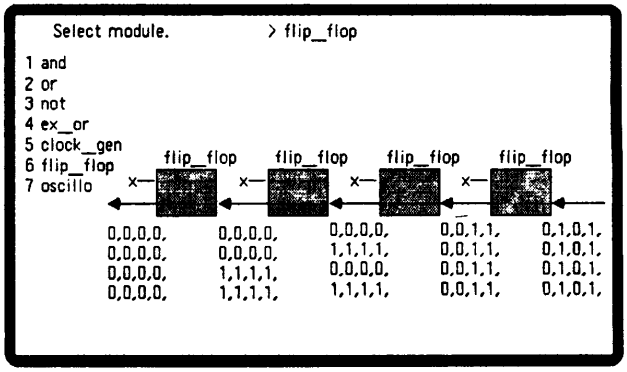

図 5 結合コマンドによるモジュール flipflop の結合と実行
Fig. 5 Execution and composition of flipflop modules by compound-command.

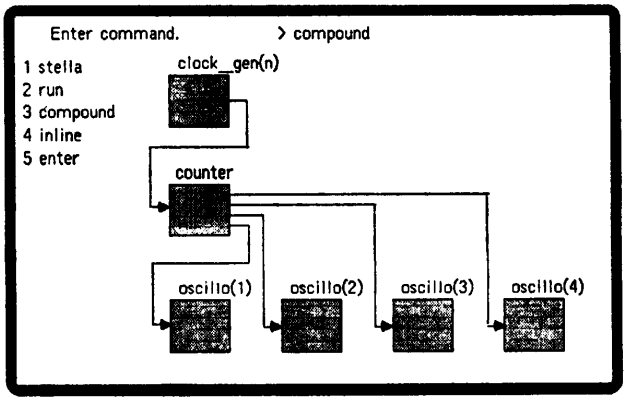

図 6 合成したモジュール counter を使った結合
Fig. 6 Composition using a compound module counter.

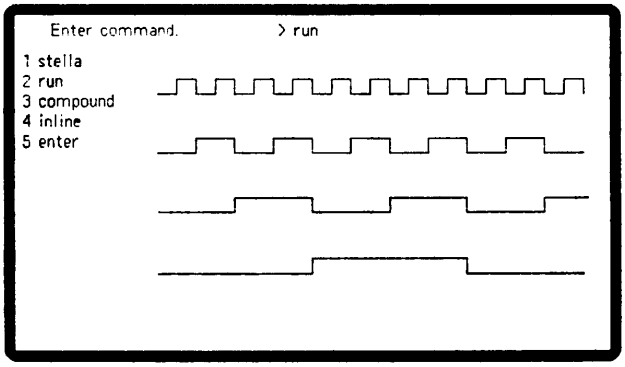

図 7 4段2進カウンタの実行結果
Fig. 7 Execution of a ripple counter.

モジュールの付け替えだけで行え，簡単である．

今後の課題は，基本的な部品モジュールの充実と，それらの系統的な管理および効率の良い検索手法である．現在のシステムでは，あらかじめ各モジュールに付加したコメント中の単語をキーワードとして，必要なモジュールを検索することができる．基本モジュールの充実とモジュールの系統的な管理によって，SPRING は，さらに強力な開発支援システムとなることが期待できる．

参考文献

1) 中田，佐々：ストリームによるプログラミング，第25回プログラミングシンポジウム報告集，pp. 124-135 (1984).
2) 久世：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，情報処理，Vol. 26, No. 5, pp. 497-505 (1985).
3) Nakata, I. and Sassa, N.: Programming with Streams, IBM Research Reports, RJ 3751 (43317)(Jan. 1983).
4) 久世，佐々，中田：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).
5) Kuse, K., Sassa, M. and Nakata, I.: Analysis and Transformation of Concurrent Processes Connected by Streams, 京都大学数理解析研究所講究録 511, pp. 120-143 (1984).
6) Kuse, K., Sassa, M. and Nakata, I.: Modelling and Analysis of Concurrent Processes Connected by Streams, *J. Inf. Process.*, Vol. 9, No. 3 (1986).

(昭和61年6月18日受付)
(昭和61年9月10日採録)

久世 和資（正会員）

1959年9月15日神戸に生まれる．1982年筑波大学第三学群情報学類卒業．現在，同大学院博士課程工学研究科（電子・情報工学専攻）在学中．工学修士．ストリームを扱うプログラミング言語の処理系，解析系，支援系の設計，開発に従事．並列プログラミング言語，プログラム開発支援環境に興味を持つ．

佐々 政孝（正会員）

1948年生．1970年東京大学理学部物理学科卒業．1974年同理学系研究科博士課程中退，東京工業大学理学部情報科学科助手となる．1981年より筑波大学電子・情報工学系講師．理学博士．プログラミング言語，属性文法，コンパイラ生成系に興味を持っている．1981年本学会論文賞受賞．ソフトウェア科学会，ACM, IEEE 各会員．

中田 育男（正会員）

1935年生．1958年東京大学理学部数学科卒業．1960年同大学院修士課程修了．1960～1979年(株)日立製作所中央研究所，同システム開発研究所勤務．1979年4月より筑波大学電子・情報工学系教授．理学博士．プログラム言語，言語処理系，ソフトウェア工学などに興味を持っている．著書「コンパイラ」(産業図書)．