Vol. 20 No. 4 情報処理学会論文誌 July 1979

高信頼性ソフトウェア開発のためのプログラミングシステム†

紫合 治†† 藤林信也†† 岩元莞二††

ソフトウェアが複雑化，大形化するにつれ，ソフトウェアの信頼性や保守性をいかに向上させるかが大きな問題になってくる．そこではモジュールのプログラム作成方法より，全体をモジュールに分割するための方法論や，分割されたモジュール間のインタフェースのとり方，モジュール間関連の明記の仕方などがシステム全体の成否に大きく影響する．理解し易いモジュール化のための方法論としてデータ抽象化や情報隠蔽の原則が注目されているが，これらの方法論にもとづく規則やインタフェース記述などをプログラミング言語の機能に埋め込めば，方法論にそった規律あるプログラミングを促し，設計の意図を素直にプログラム表現できる．

本論文では，以上の考察をもとに開発された，中大形ソフトウェアの信頼性向上と開発，保守の効率化を狙ったプログラミングシステム SPOT-6 について述べる．システムは，データ抽象化のためのモジュール化機構やモジュール間インタフェース記述機能をもつ新しいプログラミング言語とそのプロセッサを中心に，ソースリスト整形やモジュール概要書生成などの文書化ツール，インタフェースの自動解析ツール，および効率改善支援ツールからなる．本稿では，システムの設計思想，特徴的な言語機能，システム機能とシステム構成，さらにシステムの実際の使用経験にもとづく評価について述べる．

1. まえがき

ソフトウェアの信頼性や保守性向上のためにはソフトウェア開発の方法論や開発支援ツールの研究が重要な鍵を握っている．信頼性や保守性は多くのモジュールから構成される複雑な中大型システムにおいて特に大きな問題であり，そこでは，モジュールのプログラム作成方法よりも，全体をモジュールに分割するための方法論や分割されたモジュール間のインタフェースのとり方，モジュール間関連の明記の仕方などが，システム全体の成否に大きく影響する[1),2)]．

モジュール間インタフェースを単純化させ理解し易いシステム構造をもたらす新しいモジュール分割方法論として，データ抽象化[3)]や情報隠蔽[2)]の原則が注目されているが，従来の汎用プログラム言語 (FORTRAN, COBOL, PL/I など) でこれらの原則を適用するには，コーディング規則やプログラムリスト以外の文書に頼らざるを得ない．このため，規則を遵守させることの困難さや文書類の増大を理由に，これらの方法論の効果は理解できてもその実践をためらう傾向がみられる．

方法論にもとづく規則やインタフェース記述などをプログラミング言語の機能の中に埋め込めば，方法論にそった規律あるプログラミングを促し，設計とプログラミングのギャップを縮め，設計の意図を素直にプログラム表現できる．

以上の考察をもとに，信頼性の高い中大型ソフトウェア開発を支援するシステム SPOT-6* を NEAC ACOS-6 上に開発した[5)]．システムは，信頼性の高いプログラムを作成するための言語とそのプロセッサ，ソースリスト整形やモジュール概要書生成などの文書化ツール，インタフェースチェッカ，および効率改善支援ツールからなる．以下に，システムの設計思想，特徴的言語機能，システム機能/構成，およびシステムの評価について述べる．

2. 設計思想

システムは高品質のソフトウェアを高生産性で開発するために，新しい有効な方法論の導入支援と手作業の機械化の機能をもつ．このシステムを開発するにあたり以下の考え方で設計方針を定めた．

(1) 方法論の言語化

言語は考えを表現するだけでなく，思考法そのものに影響を及ぼす働きがある．このため，データ抽象化や情報隠蔽などの方法論を実践するには，これらの方法論に適した構造や機能をもつ実用的なプログラミング言語が必要である．

(2) モジュール間インタフェースの重視

モジュール数が数百に及ぶ中大型システムでは，モジュール単位の機能やモジュール関連を理解する上で特にインタフェースの明記が重要である．これによりインタフェース記述の無矛盾性の自動チェックができ，人手によるレビュー作業を効率化できる．既存言語はこの点の考慮があまりなされていないため，新た

† A Programming System for Reliable Software Development by OSAMU SHIGO, SHINYA FUJIBAYASHI, and KANJI IWAMOTO (Central Research Laboratories, Nippon Electric Co., Ltd.).

†† 日本電気(株)中央研究所

* SPOT-6 は，その前身である SPOT[4)] (Structured Programming Oriented Tools) の評価や反省をふまえて，新たに ACOS-6 上に開発されたものである．

なインタフェース記述機能やその自動チェック機構の追加が望まれる.

(3) プログラムリストの文書性向上

プログラムの設計書は人による理解のための文書であるが，擬似言語などを使った詳細設計書ではプログラムに近いレベルの表現が必要になる．このため，プログラム記述が十分に高水準で，さらにインタフェースの明記や作成者/作成日などの管理情報を含めば，少なくとも詳細設計書はプログラムリストとそれから自動生成される文書で置き換えることができ，プログラムと文書類の不一致を防ぐことができる.

(4) 効率の重視

コンピュータの利用は正確性と高速性にあるので，効率は決して無視すべきでない．(1)に上げた方法論をより実用的にするには，モジュール化に伴うオーバヘッドの解消のための機構が必要になる.

3. 言語機能

言語は実用性を考えて比較的構造的で中大型ソフトウェアの開発に広く使われている PL/I をベースとし，前述の設計思想にそった各種の機能をもたせた．ここでは，その特徴的機能について述べる.

3.1 データ抽象化のための機能[6]

言語の水準をあげるためにはデータ表現の高水準化が必須である．それぞれの応用分野に適した抽象的なデータをすべて言語にもつことは不可能であり，ユーザが目的に応じてデータを抽象化するための機能が必要になる．データやリソースを抽象化する場合，その実現のためのデータ構造とそれを操作する複数のオペレーション（例えばテーブルの全消去，追加，削除，検索など）が必要になるが，実現のためのデータ構造はその抽象データを使う側からは見えないようにすべきである．すなわち，抽象化する場合，それを使用する側にとって必要な情報を，それを実現するために必要な情報から明確に分離することが大切である．しかし，これらのオペレーションを従来の外部手続きで実現するとオペレーション間でのみ共有するデータが他の手続きからもアクセスできるようになり，情報隠蔽の原則に反した理解しにくい複雑な構造になり易い．このため，複数のオペレーションとその間でのみ共有するデータや内部ルーチンをまとめて1つのモジュールに閉じ込める (encapsulation) 機能が必要である.

データ抽象化のためのモジュール化機構としてグループを導入した．グループは1つの抽象データを定義するモジュールで，それを抽象レベルのまま使うのに必要な情報（機能概要，オペレーションとそのパラメータの宣言など）やオペレーション間共有データの宣言を記述する仕様節 (spec-section) と，複数のオペレーションや共通内部ルーチンの実現をまとめた形をしている．オペレーションとルーチンの実現は，それぞれの内部データの宣言からなるデータ節 (data-section) と，実行文をまとめたプログラム節(prog-section) からなる．これらの構文は次のとおりである．([ ] は選択，...はくり返しを表わす.)

group=g-header spec-section [routine]...operation...g-end

operation=o-header [data-section] [routine]... prog-section o-end

図1にグループ記述例の枠組みを示す．仕様節 (SPEC；から ENDSPEC；まで)には，グループ G1 が表わす抽象データの使用法（オペレーション①やパラメータ②の宣言）や，オペレーション間で共通に使われる内部データ（グループ変数③）の宣言が書かれる．オペレーションの実現は④に書かれる．グループの使用者にとっては，このうち使用法や機能概要のみが必要な情報になる.

グループを使ったプログラムは，図2のように記述される．グループを使う場合，そのモジュール（図2では手続きP）の仕様節のサブグループ宣言文①で，グループ名とそこで使うオペレーション名を宣言し，実行文中（prog-section 内）では，グループ名を修飾してオペレーションを呼び出す(②)．オペレーション名がそこで一意的な名前ならばグループ名の修飾は省略できる.

```
G1: GROUP
   SPEC;
      ---
      OPER                                  }
         OP1 (P1 IN, P2 OUT),               } ①
         OP2 (--) PETURNS (--),             }
         ---;
      PARM P1 --, P2 --; ————————②
      GROUPVAR ---; ————————③
      ---
   ENDSPEC;
   OP1: OPER (P1, P2);                       }
      ---                                    }
   ENDOPER OP1;                              } ④
   OP2: OPER (--) RETURNS (--);              }
      ---                                    }
   ENDOPER OP2;
   ---
ENDGROUP G1;
```

図1 グループの定義
Fig. 1 Group definition.

```
P: PROC;
   SPEC;
      ---
      SUBGROUP
         G1 (OP1 (---),
             OP2 (--) RETURNS (--));   } ①
      ---
   ENDSPEC;
   PROG;
      ---
      G1. OP1 (X, Y); ————————————②
      ---
   ENDPROG;
ENDPROC P;
```

図 2 グループの使用
Fig. 2 Usage of group.

グループは，CLU[3] の cluster や Alphard[7] の form のような抽象データ型ではなく，Modula[8] の module のように1つの抽象オブジェクトを表現する．現実のソフトウェア開発では1つの抽象データ型に対して1つのオブジェクトで間に合う場合が多く[8]，グループのような従来の汎用言語のモジュール(または，コンパイル単位）の概念にそった簡単な抽象化機構でも有効に利用できる．この制限により，データエリアの確保やその番地の参照が個々のグループのコンパイル時に解決できるので，比較的容易にインプリメントできる．さらに，グループではオペレーションやパラメータなどのその抽象データを使う側にとって必要な情報を仕様節にまとめて記述するので，使う側と実現する側の情報がより明確に分離されるという利点をもつ．

3.2 モジュール仕様の記述

コンパイル単位となる外部モジュールとして，前述のグループのほかに PL/I の外部手続きの機能をもつ手続きがあり，これらはいずれも仕様節をもつ．以下にモジュール仕様記述のための特徴的な機能を述べる．

（1） 鍵語付き注釈文

モジュール仕様書として最低必要と思われる機能概要説明，作成者/修正者，作成日/修正日の記述を，それぞれ，FUNCTION, AUTHOR, DATE で始まりセミコロン(;)で終わる自由書式文として言語要素にとり入れた．これらの鍵語の導入により，後述する自動段付けやモジュール概要リスト抽出などのテキスト編集処理を簡単に行うことができる．

（2） インタフェースの記述

設計思想でも述べたが，モジュール間インタフェースを明記する機能は，グループと共に言語の重要な特徴になっている．インタフェースは，①モジュール間の制御の推移を表わす呼び出し関係，②呼び出しに伴うパラメータの受け渡し，③その他の外部データのアクセス関係に分類される．①は当モジュールが呼び出す外部モジュール (subgroup, subproc) の宣言，②はパラメータの宣言，③は外部変数 (extvar) の宣言で記述される．

パラメータと外部変数については，さらに次の関係を明記する．

—当モジュールを呼び出すモジュールとの関係
—当モジュールが呼び出す下位モジュールとの関係
—直接の呼び出し関係はないが，データのアクセスを通して関連する**横のモジュール**との関係

ここで，ある外部変数に関して横の関係があるモジュールとは，①その外部変数に関し，当モジュールとの間に一方がセットし他方が参照する関係にあり，②ともにその直系上位（呼び出し関係）のモジュールにおいてその外部変数が現われていないモジュールである．横のモジュール（手続きだけ許す）はリモート手続き (remote proc) の宣言で記述する．

これらの外部モジュールとのデータの受け渡し方を示すために，アクセス属性（IN：参照だけ，OUT：入口では値が未定義で内部で値をセット，INOUT：入口で参照され内部で値をセット，INX：入口で参照され出口では未定義）がつけられる．また，上位（下位）モジュールから受けた値を直接参照変更することなく下位（上位）モジュールに渡す，いわゆる通り抜け効果 (conduit effect) を明示するために，パス属性 (PASS) が書かれる．

例えば，**図3**のモジュール構成において，モジュールPに着目する．図で，P, Q, R, S, T, U は外部モジュール，W, X, Y, Z は外部変数である．PはSからZを受けXを返す．ZはPで使われることなくQに渡される．RはPに X, Y を返す．さらに，Pは呼び出し関係のないTからYを受け，YをUに渡す．ここでYはSに現われてないとする（T, U はPの横のモジュール）．またPは不特定多数の横のモジュール

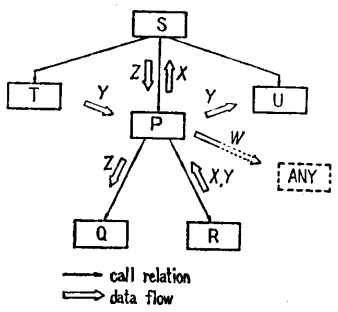


図 3 モジュール間インタフェースの例
Fig. 3 Illustration of intermodule interface.

```
P: PROC;
   SPEC;
      ---
      EXTVAR
         W BIN,
         X BIN OUT,
         Y CHAR (10),
         Z BIT (6) IN PASS;
      SUBPROC
         Q (--)
            EXTREF (Z IN),
         R (--)
            EXTREF (X OUT, Y OUT);
      REMPROC
         T EXTREF (Y OUT),
         U EXTREF (Y IN),
         ANY EXTREF (W IN);
         ---
   ENDSPEC;
   ---
ENDPROC P;
```

図 4 モジュールP (図3) のインタフェース記述
Fig. 4 Interface description for the module P.

(ANY) に W を渡す．このとき，モジュールPのインタフェース記述は**図4**のようになる．図4の EXTREF 指定は，そのモジュールとインタフェースになる外部変数とその使われ方を示す．

(3) その他の宣言文

文書性向上のため，内部変数 (INTVAR)，ファイル (EXTFILE, INTFILE)，組み込み関数 (BUILTIN)，定数 (CONST)，コンディション (COND) などの宣言文が，それぞれの鍵語に続けて書かれる．

3.3 プログラム記述

モジュール内のプログラム記述として，PL/I の実行文に加えて，ユーザ定義データ型や式マクロによる表現の高水準化，構造的コーディングのための新しい制御文を導入した．

(1) データ型と式マクロ

問題領域の概念に近い言葉でデータを扱うために，Pascal[9] の enumeration 型の機能をもたせた．例えば

```
TYPE COLOR=(WHITE, BLUE, YELLOW,
RED); INTVAR CAR_BODY_C COLOR;
```

と宣言すれば，実行文中で CAR_BODY_C に定数 WHITE, BLUE などを代入したり，それらとの比較ができる．ベース言語の PL/I のデータ属性の概念を大幅に変えずに実用的な便宜さを得るために，Pascal の型の概念を全面的に導入しないで enumeration 型に限った．

式マクロは，式や参照形式をまとめてパラメータ付きの意味のある名前で表現するための機能で，例えば次のように宣言する．

```
EMACRO    IS_COMMAND (CARD)=
                    SUBSTR (CARD, 1, 1)='$';
```

無規律なマクロの使用（文の途中から次の文の途中までの置き換えなど）を防ぐため，マクロは式マクロに制限した．

(2) 構造的制御文

構造的コーディング用として，PL/I の制御文に加えて次の文を導入した．

① DO FOREVER 文と UNTIL 文

無限ループと，ループの出口での終了判定を行う．

② EXIT 文と EPILOGUE オプション

ループの途中からの脱出 (EXIT 文) と，脱出時の後処理指定 (EPILOGUE オプション) を加えた．**図5**の(a)は(b)の構造の記述例を示す．

③ CASE 文

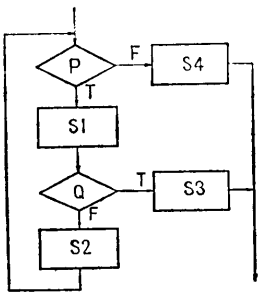
多岐選択構造文として CASE 文を加えた．選択条件の種類により3つの変形がある(**図6**)．図6(a)の RANGE 指定は，eのとりうる値の範囲を指定し，これにより効率的な目的コードが生成される．

4. 開発支援機能

ソフトウェア開発における詳細設計以後の文書は原則としてソースプログラムで一元管理するために，シ

```
DO WHILE (P);
   S1;
   EXIT [ABT] WHEN (Q);
   S2;
END EPILOGUE (
   [ABT]: S3;
   ENDED: S4; );
```

(a) プログラム例



(b) (a)の流れ図

図 5 EXIT 文と EPILOGUE オプションの例
Fig. 5 Illustration of EXIT statement and EPILOGUE option.

```
CASE e RANGE [lo: up]
   [C1, C2]: ---;
   [C3]:     ---;
   -----
   ELSECASE ---;
ENDCASE;

CASE;
   [e1, e2]: ---;
   [e3]      ---;
   -----
   ELSECASE ---;
ENDCASE;
```

e はスカラ式，C_i は e と比較可能な定数．
RANGE 指定はオプションで，l_o, up, C_i が 10 進定数か，enumeration type のとき指定できる ($l_o \leq C_i \leq up$).

e_i は論理式．

図 6 CASE 構造
Fig. 6 Case structure.

ステムはモジュール概要生成やソースリストの自動整形などの文書化機能，インタフェースの自動解析機能をもつ．さらに，効率改善支援ツールを加え，詳細設計以後の総合的な支援を狙った．

4.1 モジュール概要書生成ツール

プログラム管理の最小単位である外部モジュールごとに次の情報からなるモジュール概要書を生成する．

① モジュール名(管理上の名前)，手続きかグループの区別，手続きまたはグループ名，作成者/作成日．

② 機能概要 (FUNCTION 文から抽出)，インタフェース情報（当モジュールを使う側にとって必要な，パラメータ，オペレーション，外部変数などの情報)．

③ その他の特記事項（見出しのみ出力，内容は必要に応じて記入する)．

このツールは，既存のテキストエディタやテキストフォーマッタをコマンドファイル機能によって組み合せ，比較的簡単に作成できた．

4.2 リフォーマ

プログラムの構造(モジュール構造，ブロック構造，宣言文やデータの構造，実行文の制御構造など）に応じた段付けや改行/改頁を行うことによって見易いプログラムリストが得られるが，プログラム修正に伴う段付けの変更や TSS 端末からのプログラム入力を考えると，これをコーディング標準などの規則にするのはプログラマの負担を大きくしてしまう．

このため，プログラム自動整形ツールとしてリフォーマを備えた．リフォーマには，ソースファイルそのものを整形する S-リフォーマと，ファイルは変更せずに整形したリストを印刷するための I-リフォーマがあり，後者はプログラム構造を視覚化するために構造のレベルを縦線で示す(付録参照)．整形行の最大長や字下りの単位長はパラメータで指定できる．

4.3 インタフェースの自動解析[10]

システム全体でのモジュール間インタフェースの無矛盾性をチェックする作業は，1つのモジュール内のプログラムの正しさを調べる作業よりもはるかに工数を要する．しかし前者は後者に比べて機械的な作業（データ属性やアクセス属性の照合など）が多いので，その支援ツールの効果は大きいと思われる．このため，次のリストを出力するモジュール間インタフェースの解析支援ツールを開発した．

（1） インタフェースリスト

外部モジュールごとにインタフェース情報を清書出力し，関連モジュール宣言文（下位の／横のモジュールの宣言文）と，対応モジュール間のヘッダ文やインタフェース宣言文（パラメータ，外部変数，オペレーション宣言文など）の内容の無矛盾性（データ属性の一致，アクセス属性の無矛盾性など）を調べる．また使用されている未登録外部モジュールの一覧を出す．

（2） システム構成図

システム全体のモジュールの呼び出し関係を示す構成図を出力する．

（3） クロスリファレンスリスト

外部名(手続き名，グループ名，オペレーション名，外部変数名など）のシステム全体でのクロスリファレンス表を出力する．また，どこからも値が設定(または参照）されていない外部変数をリストする (IN, OUT などのアクセス属性の解析)．これらの変数は，システム全体の入力/出力変数とみなされる．

4.4 効率改善支援[11]

グループを使うと従来外部変数を直接参照していたところがオペレーション呼び出しになるため，呼び出しのオーバヘッドが無視できない場合が生ずる．このため，実行効率改善支援として，実行モニタとモジュール展開機能を用意した．

実行モニタは，システムのどのモジュールが効率低下の主要因となっているかを調べるために，被モニタシステムとリンクされ，実行時にモジュールの実行回数，処理時間を計測し，全体の処理時間に対する比率(%)と共に報告する．

モジュール展開機能は，呼び出しオーバヘッドをなくすため，もとの意味を変えずに呼び出しをインラインに展開する．展開は，展開後プロセッサの最適化処理を通すためにソースに近いレベルで行う．単純なマクロ展開と異なり，モジュールが呼び出される環境に従って展開するコードを変える必要がある．例えば，オペレーションの展開では，グループ変数をシステム全体で一意となる名前に変更してそのグループのオペレーション呼び出しを含むモジュールで共通に使う外部変数として宣言し，オペレーション本体の展開を行う．さらに RETURN 文を出口への GOTO 文になおすなどの処理も必要になる．

5. システム構成

図7に SPOT-6 のシステム構成を示す．ソースプログラムは（必要ならリフォーマを経て）トランスレ

ータによって PL/I プログラムに変換され，ユーザの介入なしに PL/I コンパイラを起動して目的コードに変換される．図の S-リフォーマと I-リフォーマは，入出力関係を除き同一モジュールで構成されている．これらの各機能は，選択指定ができる．

プログラムの診断は，新たに導入された構文以外は PL/I コンパイラにまかせ，コンパイラ出力のエラーメッセージともとのソースプログラムとの対応はライン番号でつけるようにした（トランスレータが出力する PL/I プログラムに，もとの対応するソースプログラムのライン番号がコメント形式でつけられる）．

6. システムの評価

システムは現在まで当システム自身の開発のほか，2, 3のプログラム開発に用いられた．これらの使用経験を通して得られたシステムの評価を，主として設計思想に対応した機能の有効性を中心に述べる．

（1） 言語の構文と意味

言語の構文と意味はベース言語の PL/I との一貫性を考慮したので PL/I 利用者には自然に受け入れられた反面，属性の省略時解釈の複雑さや手続き名変数に対する静的チェックの限界からインタフェースチェックの機能に制限を設けざるを得なかった．SPOT-6 独自の機能としては，①グループのオペレーション宣言(仕様節内)とオペレーション実現のヘッダ部でのパラメータ名や返す属性の記述で重複部分があること，②データ型名は外部名にできないため手続きの返す属性として使えないこと（PL/I のスコープ規則による），③仕様節に内部変数の宣言も書けることなど，一部構文/意味上の問題が残っている．

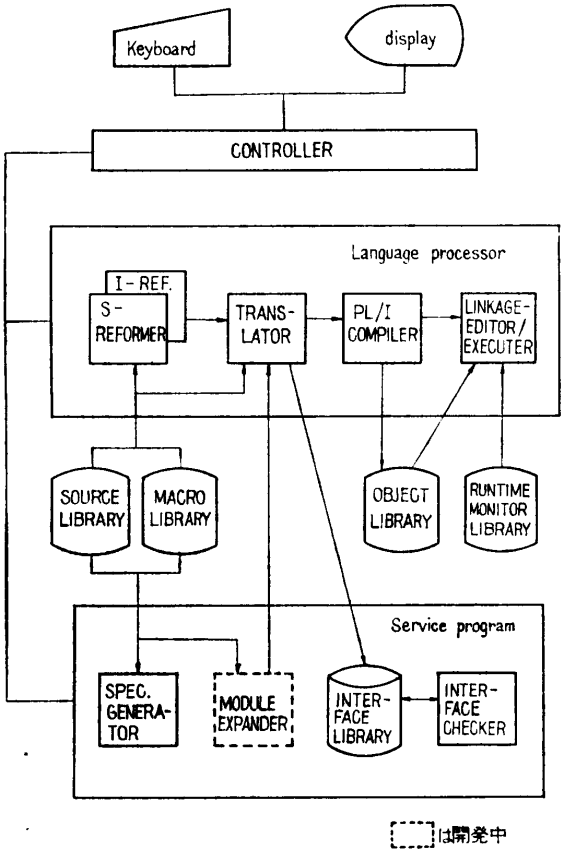


図 7 SPOT-6 プロセッサ構成図
Fig. 7 SPOT-6 processor organization.

（2） データ抽象化の機能

グループは，データ抽象化の概念を使ったモジュール化を促すだけでなく，その概念をプログラマに知らせる上でも効果があった．具体的なグループ使用の効果は外部変数の大幅な減少として現れた．例として SPOT-6 システムの規模とそこで使われた外部変数の個数を表 1 に示す．これらの外部変数はすべてサブシステムに局所的である．コントローラの 18 の外部変数のうち，7 個は既存サブルーチンとの結合で必要になったもので，残りの 11 個は，抽象レベルの高いグループの詳細化に伴い下位モジュールがいくつか生じ，これらの下位モジュールで上位グループ内の共有データ（グループ変数）をアクセス可能にするため，グループ変数を外部変数に変更する必要が生じたためのものである．

グループの使用はオペレーション呼び出しに制限したが，このほかにグループ変数を参照のみ許すように制限して直接アクセスできるようにすることも有効と

表 1 SPOT-6 システムの規模と使用された外部変数の個数
Table 1 SPOT-6 system size and the number of external variables in the system.

サブシステムの種類	SPOT-6 プログラムの規模（注3）				外部変数の使用			
	モジュール数(注2)			ソースカード枚数 (K行)(注1)	単純変数	配列	構造体(要素数)	計
	手続き	グループ	計					
コントローラ（含インタフェースモジュール）	24	11	35	4.6	15	2	1(2)	18
リフォーマ	17	4	21	3.1	2	0	1(4)	3
トランスレータ	36	9	45	5.5	0	0	1(2)	1
計	77	24	101	13.2	17	2	3(8)	22

（注1） ソースカード数のうち，約2割がコメントである．
（注2） 1モジュールあたりの大きさの平均は，手続きは96行，グループは242行．
（注3） SPOT-6 言語プロセッサはこのほか約 1.5k ステップのアセンブラプログラムを含む．

思われる（その場合，外部からのアクセスを許す変数を仕様節に明記する）．

（3） インタフェース記述

グループの導入により外部変数は大幅に減ったが，変数の概念がそれを使う論理レベルにふさわしい場合には外部変数が使われた．このような外部変数は呼び出し関係によるモジュール階層構造とは別の論理空間を形成することが多く，REMOTE PROC 宣言や EXTREF 指定の記述は外部変数によるインタフェースの理解に有効であった．インタフェースの自動解析は，無矛盾性のチェックよりむしろ文書化支援として役立ったようである．すなわち，インタフェースの明記そのものがインタフェースミスの防止を促進したものと思われる．

インタフェース情報は（主にモジュール単位でのコンパイル処理のため）すべての関連モジュールの仕様節に分散して記述する必要があり，記述の煩わしさや情報の一括管理という点から問題である．このため，今後の課題としてインタフェース情報をモジュールと分離して記述し，システム機能によって各モジュールに分配する方式や，既コンパイル済モジュール情報やインタフェース情報をデータベース化してコンパイル入力として利用する方式などが考えられる．

（4） 文 書 性

システムの目標の1つは，コメントを書かなくてもプログラムリスト自身が十分な文書性をもつことであった．言語のもつ文書性（仕様節とプログラム節の分離，鍵語付き注釈文，インタフェースの明記，構造的制御文など）とリフォーマの機能により，ソースリストの文書性は大きく向上し従来の詳細設計書を兼ねることができたと評価している．

詳細設計書をプログラムリストで兼ねる場合，モジュール概要書やインタフェース情報の自動生成機能は特に有効で必須と思われる．これらの機能によって，インタフェース宣言文の一部や鍵語付き注釈文のような，理解を助けるが目的コードの生成には直接関係しない記述を助長する．

（5） 効率改善支援

実行モニタでシステムのトランスレータを調べたところ，単語解析以下が全体の約5割の時間を占めていた．そこを中心に一部プログラムを修正して手作業でモジュール展開を行ったところ，トランスレータ全体で約3割の実行効率改善が得られた（メモリは約 5% 増加した）．

7. むすび

中大型のソフトウェア開発の詳細設計からプログラミングフェーズを主に支援するプログラミングシステムについて述べた．

規模の大きな，複雑なソフトウェアの開発では，モジュール間やサブシステム間のインタフェースのとり方と，文書類の増大が大きな問題である．SPOT-6 言語や，リフォーマ，モジュール概要生成，インタフェースチェッカなどのシステム機能によって，これらの問題は大きく改善されると思われる．特に，グループは実際的なソフトウェアにデータ抽象化の概念を適用させる有効な機能を提供する．

文書性を増すために，言語上の考慮や，システム機能が検討され，導入されたが，これによって従来の詳細設計以降の文書（量的には最も多い）は，プログラムリストと，それから自動生成されたレポートにより置き換えることができた．

今後に残された主な問題として，以下のものが上げられる．

- 静的インタフェースチェックの限界を見きわめ，効率的な動的チェックの導入を検討すること．
- 文書性の向上に欠くことのできない，日本語の導入（カナだけでは不便？）を検討すること．
- インタフェースの明記を進めると，外部変数などのインタフェース記述量が増大するので，各モジュールごとに内容を書かなくてもよい方式を検討すること．
- グループの仕様記述については，ここでは機能概要（叙述）やオペレーション，パラメータの宣言によったが，さらにオペレーションの妥当な使い方や抽象データの仕様などの形式的な記述法の検討が必要である．ただし，現在活発に研究が進められている形式的仕様記述法そのままでなく，実用性を考慮した，通常のプログラマが無理なく使える方式にすることが望まれる．

SPOT-6 では有効と思われるプログラミング方法論の具現化を試みたが，特に抽象データの概念やそれを支援するプログラミング言語の有効性については，今後さらに実際的な経験を通して評価する必要がある．

謝辞 本システムの開発を手伝っていただいた当社中研コンピュータシステム研究部の西村氏，伊東氏，宮下氏，福田嬢，長谷川嬢に謝意を表する．また，同研究部藤野部長（現在ソフトウェア開発本部長）には，

終始励ましと有益な助言をいただいた．ここに深く感謝する次第である．

参考文献

1) DeRemer, F. and Kron, H.: Programming-in-the-Large versus Programming-in-the-Small, Proc., Int. Conf. on Reliable Software, pp. 114–121 (1975).
2) Parnas, D. L.: On the Criteria to be used in decomposing systems into modules, Commun. ACM, Vol. 15, No. 12 (1972).
3) Liskov, B. H. and Zilles, S.: Programming with abstract data types, SIGPLAN Notices, Vol. 9, No. 4 (1974).
4) Shimomura, T., et al.: SPOT; a structured system development system, Proc. Annual Conf. ACM, pp. 387-394 (1974).
5) 岩元，紫合，藤林：SPOT-6；高信頼性ソフトウェア開発のための言語システム，情報処理学会ソフトウェア工学研究会資料 3 (1977).
6) Shigo, O., et al.: Implementing the abstraction technique in software development, Proc. 2nd USA-Japan Computer Conf., pp. 517-522 (1975).
7) Wulf, W. A., et al.: An introduction to the construction and verification of Alphard programs, IEEE Trans. Softw. Eng., Vol. SE-2, No. 4 (1976).
8) Wirth, N.: Modula; A language for modular multiprogramming, Software-Practice and Experience, Vol. 7, No. 1 (1977).
9) Wirth, N.: The programming language Pascal, Acta Informatica, 1, pp. 35-63 (1971).
10) 西村，藤林：モジュール間インタフェースの自動解析，昭和52年度通信学会情報部門全国大会，367 (1977).
11) 宮下，紫合：モジュールのうめ込みについて，昭和53年度電子通信学会総合全国大会，1367 (1978).

付　録　SPOT-6 プログラムの例

コンパイラなどで用いられる名標テーブルを，抽象データとしてグループで表現した例を示す．リストはI-リフォーマによって出力されたものである．ただし，ページヘッダやラインナンバなどは除いてある．

```
ID_TABLE:GROUP;

SPEC;
   FUNCTION  A TABLE WHICH CONTAINS ID NAMES
             AND THEIR ATTRIBUTES;
   AUTHOR  O. SHIGO;
   DATE    78/02/10;
   OPERATION
      INITIALIZE          ,
      SEARCH(ID_NAME IN)  RETURNS(BIT(1)),
      ADD_NEW(ID_NAME IN,ID_ATTR IN),
      SEARCHED_ATTR       RETURNS(BIT(6));
      /* 'SEARCHED_ATTR' CAN BE USED ONLY AFTER  */
      /* 'SEARCH' IS CALLED  AND ITS RETURNED    */
      /* VALUE IS TRUE('1'B).                    */
   PARAMETER
      ID_NAME             CHAR(31) IN,
      ID_ATTR             BIT(6) IN;
   GROUPVAR
      HASH_TBL(0:63)      POINTER,
      CURR_POS            POINTER,
      1 ENTRY             BASED(CURR_POS),
         2 CHAIN          POINTER,
         2 NAME           CHAR(31),
         2 ATTR           BIT(6);
   INTVAR
      (N,INDEX)           FIXED BIN;
ENDSPEC;

GET_HASH_CODE:ROUTINE(NAME,HASH_CODE);
   DATA;
      PARAMETER
         NAME             CHAR(31) IN,
         HASH_CODE        FIXED BIN OUT;
      INTVAR
         C                CHAR(1);
   ENDDATA;
   PROG;
   !  HASH_CODE=0;
   !  DO N=1 TO 31 BY 3;
   !  !  C=SUBSTR(NAME,N,1);
   !  !  EXIT WHEN(C=' ');
   !  !  HASH_CODE=HASH_CODE+BINARY(UNSPEC(C));
   !  END;
   !  HASH_CODE=MOD(HASH_CODE,64);
   ENDPROG;
ENDROUTINE GET_HASH_CODE;

INITIALIZE:OPERATION;
   PROG;
   !  DO N=0 TO 63;
   !  !  HASH_TBL(N)=NULL;
   !  END;
   ENDPROG;
ENDOPER INITIALIZE;

SEARCH:OPERATION(ID_NAME) RETURNS(BIT(1));
   PROG;
   !  GET_HASH_CODE(ID_NAME,INDEX);
   !  CURR_POS=HASH_TBL(INDEX);
   !  DO WHILE(CURR_POS^=NULL);
   !  !  IF CURR_POS->ENTRY.NAME=ID_NAME THEN
   !  !  !  RETURN('1'B);
   !  !  CURR_POS=CURR_POS->ENTRY.CHAIN;
   !  END;
   !  RETURN('0'B);
   ENDPROG;
ENDOPER SEARCH;

ADD_NEW:OPERATION(ID_NAME,ID_ATTR);
   PROG;
   !  GET_HASH_CODE(ID_NAME,INDEX);
   !  ALLOCATE ENTRY;
   !  ENTRY.CHAIN=HASH_TBL(INDEX);
   !  ENTRY.NAME=ID_NAME;
   !  ENTRY.ATTR=ID_ATTR;
   !  HASH_TBL(INDEX)=CURR_POS;
   ENDPROG;
ENDOPER ADD_NEW;

SEARCHED_ATTR:OPERATION RETURNS(BIT(6));
   PROG;
   !  RETURN(CURR_POS->ENTRY.ATTR);
   ENDPROG;
ENDOPER SEARCHED_ATTR;

ENDGROUP ID_TABLE;
```

(昭和53年7月10日受付)
(昭和54年2月13日採録)