段階的詳細化とデータ抽象化を支援する言語ＳＰＬの処理系と環境に関する研究

中所　武司

目　　次

第1章　序　論

第１章　序　　論

1.1　緒　　言

電子計算機システムは，その誕生以来，ハードウェア技術の飛躍的な進歩を背景に，応用分野の拡大を続けており，現代の情報化社会の中枢に位置するまでに発展した。その結果，電子計算機の利用形態はますます複雑化，多様化の傾向を強めており，この要求に応えるためのソフトウェアは大規模化の一途をたどっている。以来，この大規模ソフトウェアの開発は，中央処理装置の高速化や主記憶装置および補助記憶装置の大容量化などのハードウェア技術と仮想記憶方式のオペレーティングシステム，および大量の開発人員の投入によって実現してきた。しかしながら，このような方法だけでは信頼性の高い大規模ソフトウェアを効率良く作成することに限界があった。この問題は「ソフトウェアの危機」とも呼ばれ，１９７０年頃から，大規模ソフトウェアの信頼性と生産性向上のための技法の確立が急務とされた。

この研究課題の解決のために，次のような２つのアプローチがとられた。その第１は，「構造化プログラミング」という言葉に代表されるプログラミング方法論と言語に関する研究である。従来，ソフトウェア開発技術の中で最も大きな役割を果たしてきたのはプログラミング言語であり，１９５０年代後半から６０年代前半にかけて，Fortran，Cobol，Algol，ＰＬ／Ｉなどの手続き型言語が次々と開発され，記述量の削減によるプログラムの生産性向上をもたらした。しかしながら，７０年代には，ソフトウェアの大規模化とその保守，拡張費用の増大にともない，プログラムの書き易さよりも読み易さの方が重要になってきた。そこで，良い構造のプログラムを作成するための方法論の研究が行われ，それを反映した言語が開発された。

大規模ソフトウェアの信頼性と生産性向上のための第２のアプローチは，「ソフトウェア工学」[B8,Z2)]という言葉に代表される分野の研究であり，ソフトウェア開発工程のあらゆる局面，即ち，要求定義，設計，製造，保守運用を対象とした方法論，技法，ツールの開発が行われてきた。しかしながら，解決すべき課題は多種多様で，かつ広範囲に散在しており，その多くが８０年代に持ち越されている。

このようなソフトウェアの信頼性と生産性向上の問題は，汎用大型計算機システムに限られた問題ではない。筆者は，７０年代半ばより，日立の制御用計算機ＨＩＤＩＣ８０用の応用プログラムを対象として上記問題に取組み，プログラミング方法論，言語と処理系，プログラミング環境の研究を行ってきた。そして，段階的詳細化とデータ抽象化を支援する言語ＳＰＬ（Software Production Language）を開発した。

ＳＰＬコンパイラは，１９７７年に第１版を開発完了し，ＨＩＤＩＣ８０用応用プログラムへの適用を開始した。以後，今日までに３００を越えるシステムに使用され，その間にコンパイラも３度の改訂を行った。また１９８２年には，従来の１語長１６ビットを３２ビットに拡張した新機種ＨＩＤＩＣ Ｖ９０用のＳＰＬコンパイラを開発し，既に実用に付している。

ここに，これまでに得られた研究成果を報告し，批判を受けると共に，今後のこの分野の研究，開

発に携わる方々の参考としたい。なお，ここで述べる研究成果は，部分的に制御用計算機の分野に限られるものがあり，その旨を本文で明記するが，他の大部分については一般性があると信じる。

1.2 本研究の目的と範囲

1.2.1 本研究の目的

従来，制御用応用プログラムは，Fortranにリアルタイム処理機能を加えた言語で記述していたが，開発すべきプログラムの増加と大規模化に伴い，その信頼性と生産性の向上が重要な課題となってきた。

本研究の目的は，構造化プログラミングの概念を制御用応用プログラムに適するように具体化して実用化することにより，上記問題を解決することである。そのため，まず，新しい言語とその処理系を研究開発し，実用に付すと共に，その適用効果を増すための言語支援系の研究を行ってきた。そこで，本論文は，以下の項目に関する研究成果を明らかにすることを目的とする。

(1) 新言語の設計思想と実現方式

(2) その言語処理系の作成技法と性能解析

(3) その適用効果を高める言語支援系の機能

本節では，以下に各々についての概要を述べる。

1.2.2 従来方式の問題点

従来の制御用Fortranで記述されたプログラムには次のような問題があった。

(1) タスク間共通データへのアクセスエラーが多い。

(2) プログラムとその設計書の対応が不明確で，保守が難しい。

第1の問題は，相互に関連する複数のプログラム（タスクと呼ぶ）を同時に実行するマルチタスクシステムに一般的な問題である。通常，タスク間の情報の受け渡しやタスクの起動の制御などは，タスク間共通データを用いて行うため，共通データへのアクセスエラーはシステム全体に重大な障害を引き起す場合が多い。そして，共通データへアクセスするタスクはお互いに非同期実行されているため，エラーに再現性が無いなどの問題があり，この種の誤りの原因究明は困難なものとなっている。

第2の問題は，プログラムの機能分割によって生じた処理の階層構造をプログラム構造に表現できないことによる。即ち，プログラムの詳細設計書は3段階程度に階層化したＧＦＣ（General Flow Chart）で記述しているが，これから制御用Fortranプログラムを作成する時には，この階層構造を1階層に平面化する必要がある。そのため，この変換時に誤りが生じ易いことやプログラムの修正時や保守時に設計書との対応が不明確になり，処理内容の理解が難しいことなどの問題がある。

1.2.3 新言語の設計思想[C9)]

従来方式の第1の問題である，タスク間共通データへのアクセスエラーは，主に次の2種類が多い。

(1) そのデータにアクセスすべきでないタスクからのアクセス

(2) 複雑なデータ構造に対する処理方法の誤り

これらに対処する従来の言語技法として，(1)についてはデータアクセス権を階層的に制限するブロック構造，(2)については専用手続きを用いてデータアクセスを行うデータ抽象化がある。

ところが，これらの技法は，各々，異ったプログラム構造を導くため，両立が難しい。

そこで，本論文では，ブロック構造を変形して，次のような木構造形式のモジュール階層構造を導入することにより，上記問題を解決した。

(1) 環境モジュールと処理モジュールの2種類のモジュールを設ける。

(2) 環境モジュールは共通データの宣言に用い，複数の環境モジュールを木構造形式に結合可能とする。

(3) 処理モジュールは手続きの集りとし，適切な環境モジュールの下に結合する。そして，そこからアクセス可能な共通データをその上位方向に結合された環境モジュール群のものに限る。一方，手続きは他の処理モジュールから引用可能とする。

従来方式の第2の問題である，階層化された設計書とその階層構造を表現できない言語のプログラムとの対応関係の不明確さについては，段階的詳細化によるプログラム開発技法をいかにプログラミング技法に反映させるかという問題に一般化できる。従来，段階的に処理を記述する言語としてPDL[C1]（Program Description Language）があるが，これは設計用言語であり，最終的には人手によってプログラミング言語に変換するため，設計書とプログラムの対応不一致の問題は解決しない。

本論文では，段階的詳細化の各段階をそのままプログラムに記述できるようにするため，次のような機能を設けることにより，上記問題を解決した。

(1) 手続き名は複数の単語の並びで構成可能とし，自然語風の表現ができるようにする。

(2) 処理の詳細化に合せたデータ構造の詳細化の記述のために，ユーザ定義データ型機能を設けると共に，その型名は，手続き名と同じく自然語風の表現とする。

(3) 手続きのインライン展開指示機能を設け，段階的詳細化によって生じる，一度だけ呼ばれる手続きの実行効率向上をはかる。

1.2.4 言語処理系の作成技法[C11,C14]

ＳＰＬの処理系の開発に際し，従来の技法と異なる方式を幾つか採用したが，その主なものは次の3項目である。

(1) 分割コンパイル方式

従来のFortranやＰＬ／Ｉなどのコンパイラは，処理単位毎に個別にコンパイルする方式を採っているため，外部手続きのインターフェイス誤りなどは実行時まで検出できなかった。一方，すべてのプログラムを一括してコンパイルするPascalの方式では，大規模ソフトウェアの開発が難しい。

そこで，ＳＰＬでは，モジュール単位にコンパイルしながらモジュール間の整合性のチェックも行う分割コンパイル方式を採ることにより，信頼性の高い大規模ソフトウェアを効率良く開発できるようにした。そのため，モジュール単位のコンパイル情報を保存する共通ライブラリを導入し，関連モジュールのコンパイル時に参照できるようにした。本方式は後にＣＬＵ[A3]，Ｍｅｓａ[L2]，Ａｄａ[D4]などでも採用されている。

(2) ユーザ定義データ型チェック方式

従来，データ型の同一性をチェックする方法[W4]としては，型名の同一性をみる方式とデータ構造の同一性をみる方式があり，既存の言語ではいずれか一方の方式が採られている。しかしながら，データ抽象化の観点からは，抽象データ型の利用側ではその構造を知る必要がないので型名の同一性をみる方式が適するが，その操作手続きの定義側ではデータ構造に依存した処理を行うので，データ構造の同一性をみる方式が適する。

そこで，ＳＰＬでは，データ型を定義しているモジュールおよびその下位モジュールではデータ構造による方式，その他のモジュールでは型名による方式を採ることにより，上記問題を解決した。

(3) 手続きのインライン展開方式

従来，プログラムの一部分をインライン展開するものとして，アセンブリ言語のマクロ機能やＰＬ／Ｉのコンパイル時機能がある。ところが，これらはソースレベルでの展開方式のため，展開される部分の誤りはその定義時には検出できず，展開後に初めて検出される。また，展開されたものが文法に合っていれば，定義そのものは文法的に正しい必要はないため，展開される部分は機能的なまとまりでなくても良いことになってしまう。

そこで，ＳＰＬでは，プログラムの高信頼化のために，インライン展開される部分は外部手続きと同じ形式で定義をさせると共に，そのインライン展開処理は，構文解析と意味解析終了後の中間語形式の手続きに対して行うこととし，その効率の良い方式を考案した。

1.2.5 言語処理系の性能解析

ＳＰＬでは，前述したように，幾つかの新しい言語機能を有しており，それが言語処理系の性能に与える影響は少なくない。実際の処理性能の実測結果によれば，階層構造プログラムの分割コンパイルのために導入した共通ライブラリへの参照が頻繁に生じることから，補助記憶装置への入出力処理時間が大きいことが判明した。これは，特にＳＰＬコンパイラの稼動する制御用計算機ＨＩＤＩＣ 80が１６ビットマシンであるため，主記憶容量が最大でも６４Ｋ語と少なく，共通ライブラリ用の主記憶バッファサイズを十分には確保できないことによる。

そこで，本論文では，共通ライブラリの入出力処理に用いるページングアルゴリズムの分析[C8,C13]を行った。その結果，手続き部を対象とした従来の研究結果と異なり，最適ページサイズが小さいことを明らかにした。また，データ参照においても局所参照特性が存在することや，そのためにページングアル

ゴリズムとしてはＦＩＦＯ（ First In First Out ）よりもＬＲＵ（ Least Recently Used）方式が優れていること，およびその性能差がページフォールト率の低い所で大きくなる理由を明らかにした。

1.2.6　ＳＰＬプログラムの実行効率の分析

構造化プログラミング技法などを用いてプログラムのモジュール化を行った場合，一般に，コンパイラが出力するオブジェクトプログラムの実行速度とメモリ量に関して次のような欠点が生じる。

(1)　モジュール間リンケージのオーバーヘッドの増加

(2)　冗長な処理の増加

(3)　作業エリア用の局所変数の増加

これらの問題を最小限に抑えるために，ＳＰＬでは次のような対策をとっている。即ち，(1)に対しては，先に述べた手続きのインライン展開機能が有効である。(2)に対しては，コンパイル時実行機能を設け，冗長な処理を除くような記述を可能にしている。また，(3)に対しては，コンパイラの最適化処理によって作業エリアの共有化が可能である。その結果，記述実験などから，ＳＰＬによるオブジェクトプログラムの実行効率の低下は，制御用Fortranとの比較で１０％以下であるという結論を得た。

1.2.7　言語支援系の考察

大規模ソフトウェアの信頼性と生産性向上のための第２のアプローチとして最初に述べたソフトウェア工学の見地から，ソフトウェア開発工程の全体をみた場合，プログラミング言語とその処理系の果たす役割には限度がある[S12]。そこで，７０年代後半から８０年代にかけては，言語そのものと同時にその開発支援環境の重要性が認識されるようになった。本論文では，ＳＰＬの支援系として研究開発してきた次のような方法論とツールについて述べる。

(1)　２段階プログラミング法[C6,C7]

ＳＰＬの最初の適用分野は，応答性の厳しい制御用リアルタイムシステムであったこと，および慣れの問題から，利用者は当初，従来言語と類似の記述形式をとる傾向があった。そこで，新言語の本来の目的である構造化プログラミング技法の適用手段として，第１段階ではプログラムの構造化に徹し，その後に性能要求に応じて最適化を行う２段階プログラミング法を考案した。そして，この方法論を具現化する会話型システムに備えるべきコマンド機能を設計した。

(2)　構造エディタ[C10]

従来，プログラムの作成と修正にはテキストエディタが用いられてきたが，これは，プログラムを意味のない文字列とみなし，文字単位あるいは行単位のテキスト操作を行うものである。そのため，文法的な誤りを防げないことや操作効率が悪いなどの欠点があった。そこで，ＳＰＬのために，構文テンプレートの自動表示や構文要素単位の編集機能を有する構造エディタを研究開

発した。特にその中で，ＳＰＬの段階的詳細化機能を支援するようなプログラム生成機能を持たせることを重視した。

(3) 構造テストツール[C15)]

プログラムの検証作業は，その開発工数の約半分を占め，生産性と信頼性に重要であり，種々の支援ツールが開発されているが，テストの充分性の判断は難しい問題である。そこで，本論文では，プログラム内の全分岐方向のテスト実行の有無を測定するツールを研究開発すると共に，このようなパステストを効果的かつ効率的に行うための制御フローグラフ簡約法とそれに基づいて本質的な分岐にのみ着目するテスト網ら率尺度を考案した。

1.2.8 適用効果

最後に，制御用応用プログラムの信頼性と生産性向上を目的として，種々の新機能を導入したＳＰＬが，実際の適用にあたって，所期の目的を達しているか否かを確めるため，適用実験および実システムへの適用効果の分析を行った。そして，ＳＰＬは，従来言語（ＰＣＬ）に比べて，理解容易性に優れ，誤りの混入防止や早期検出に効果があること，および特に制御用応用プログラムで重要な役割を果たすタスク間共通データに関してその効果が大きいことを確めた。