5U−9

プログラム設計言語の視点から見たソフトウェア開発支援とinfer言語による実現

伊藤　昌夫

MHIエアロスペースシステムズ(株)

1. 緒言

プログラム設計言語（以下ＰＤＬ)は、時に疑似言語と呼ばれ、様々な場面で使われている。しかし、ソフトウェア開発環境を考え、その一部としてＰＤＬの役割を積極的に考える時、ＰＤＬはより大きな役割を演じることが出来ることが分かる。

プログラム言語は、ソフトウェア工学の成果をとり込むことによって優れた言語となり得る。一方、対象領域の多様性、或は実行時の効率を考える時、現実的には"古い"言語を使わざるを得ない。しかし、ＰＤＬをその前段階で使用する、つまりＰＤＬを用いて設計し、最終的に対象言語に変換することにより、対象領域に適し、効率が良く、より安全なプログラムを作成することが可能となる。

ＰＤＬに関する標準としてはＡｄａを用いる方法（準則）等が知られている[1]。しかし、本論では、別の視点から、ＰＤＬの可能性について検討を加えた。次に、これら可能性を実現する為に、今回新たに設計したＰＤＬ"infer"について説明する。

2. ＰＤＬと設計モデル

2.1 ＰＤＬの役割と新たな可能性

ここでは、一般的なＰＤＬの役割と、新しい可能性について考える。尚、以下の議論に於ける誤解を避ける為に、ここでのＰＤＬは"モジュール分割後に、プログラムの論理的構造を具現化するもの"と定義しておく。ここでの論理的構造とは、プログラム言語の持つ文法を意識しないと云う意味である。

(1) 図式との関連

図式との関連で考えると、ＰＤＬを用いず直接画面に描くやり方が一般的である（ＹＰＳ、ＳＥＷＢ等)。しかし、ここではアルゴリズムの設計及びデータの設計をＰＤＬで行った後、ＨＩＰＯ、ＰＡＤ等の図式表現に置換する方法について考える。

図式エディタを用いるか否かは、最初に用いる表現手段だけの問題であり、本論の主旨と直接には関係なく、またこれらは相互に補完しあっていると考える。ここでは、あくまでＰＤＬを中心として考える。ただ、一般的な特性の比較を行うと次の様に云える。

図式エディタを用いる方法では、構造を直接操作すると云う面で直接的に理解を助ける。しかし、ＰＤＬを用いるテキスト処理に比べ、描画領域が大きくなる他、修正等での操作の手間も一般的に大きい。

重要なのは、ＰＤＬから図式への変換に於て基本的に問題がないと云うことである。（筆者は、代表的な図式、ＨＩＰＯ、ＰＡＤ、ＨＣＰ等に関してＰＤＬからの変換ルールを定めることが出来た。）

(2) ソースコードの標準化・均一化

ＰＤＬからソースコードを生成する時に、以下に示す様な操作を行うことにより、ローカルな標準に準拠させたり、ソースコードの可読性等のメンバー間の均一化、属人性の排除を図ることが可能となる。

（ア）ヘッダ、コメントの自動付加
（イ）コーディング規約等に対する違反の検出
（ウ）構文のチェック

また、複雑度（例えばMcCabe、Hallsteadによる方法）等のプログラムのメトリックスが品質に及ぼす影響も知られており（例えば[2])、この時点で複雑度の計量も可能となる。

(3) 早期に於ける検証

これは、新たな可能性の一つである。

ソフトウェア開発が困難である一つの理由として、設計や製造の妥当性を最終段階、つまり検証フェーズでしか評価出来ないことが挙げられる。より早期に於ける妥当性の検査は、製品の生産性や品質を向上させる。

プログラムの正しさを評価する方法としては、仕様記述言語に用いられている厳格な、"代数的・関数的方法で書き、検証系により確認する"と云う方法があるが、最終的なプログラム言語への写像が困難であり、完全な適用は困難である。

次に考えるのは、実際的な最も良く用いられる方法としての、ウォークスルーないしはレビューである。しかし、問題のあるプロジェクト程、これらを実行する時間がないと云うパラドックスがあり、可能な限り自動化することが望まれる。

その中間的な方法として、ＰＤＬに表明[3]機構を加えることにより、或はモジュール定義とのつき合わせを通じて、（実際の処理系なしに、）モジュールの挙動や、その正しさを確認すると云う方法が考えられる。

更に付加的には、ＰＤＬで書かれたプログラムを自然語（日本語）に変換することで、可読性が向上しウォークスルー等での効率向上を図ることが出来る。

(4) 再利用性

再利用はソフトウェア工学に於ける重要な課題の一つである。現実的には、数値演算ライブラリ、画面制御ライブラリと云った特定の部分でしか実現されていない。その理由としては、検索が困難であること、モジュール結合度が様々で切り離しが困難であること等が挙げられる。

ＰＤＬで書かれたプログラムは、最終プログラムの骨格となり、短時間でのプログラムの大筋の理解に役立つ。また、このＰＤＬと云う単一の言語で書かれたプログラムは、最終的な詳細を含まず、再利用時の検索に於けるキーとして十分に役立つ。

つまり、ＰＤＬレベルで書かれたものをそのまま再利用部品としては利用出来ないが、ＰＤＬに表現されている知識を容易に利用出来る。

2.2 ＰＤＬを中心とした設計モデル

ここまでの議論を基に、図１に設計がＰＤＬを中心としてどの様に進展していくかを示す。

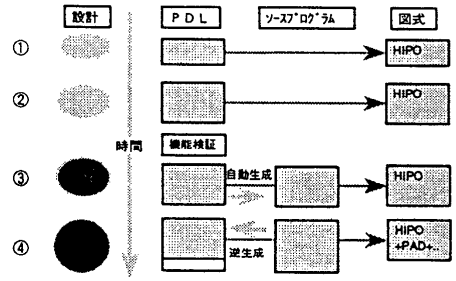
①から②にかけては、アルゴリズムやデータ構造の設計に伴い、ＰＤＬが書き進められ、即時に参照可能な図式により、支援される。（必要があれば、ライブラリから類似部品を検索する。）

②終了時点、即ち設計者が設計終了を判断した時点で、機能検証がなされプログラムの正しさが検証される。不十分であれば再設計される。また、ＰＤＬレベルでの静的解析を行う。

これらに問題ない場合③、ソースコード（のスケルトン）を生成する。対象プログラム言語でしか記述出来ない部分をソースコード中に付加する。

コーディング終了時点④で、リバースを行い必要な修正をＰＤＬに対して自動的に行う。この時点で、要すればコミュニケーションの為に複数の図式で表現する。

Software development support from the view of PDL, and the language Infer
Masao Ito, MHI Aerospace Systems Co.,LTD



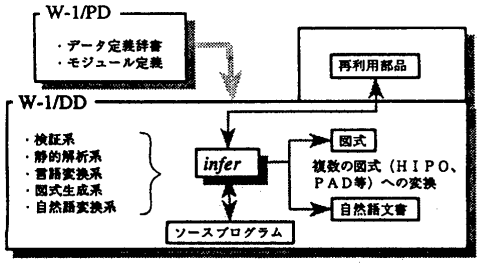
（注）設計に於ける濃淡は、設計に於ける思考の明確さ、及び設計に対する確信度を疑似的に表現したものである。

図１．ＰＤＬを中心とした設計思考モデル

3. infer言語

3.1 動作環境

infer言語は、ＰＤＬであるが単独でも動作可能な言語仕様となっている。ここでは、本論の主題であるソフトウェア開発環境での動作について説明する為、当社のソフトウェア開発支援環境W-1 [4]を例にとり、その関係を示す。



（注）ＰＤは基本設計支援モジュールであり、ＤＤは詳細設計支援モジュールである。

図２．開発支援環境に於けるＰＤＬ"infer"

3.2 言語の特徴

（１）表明機構

inferは表明機構を有する。ここでのアプローチは、Eiffel[5]で実現されている方法に近い。

検証可能性については、関数型言語等で実現されている様な形式的方法によって、正当性を完全に実現できるが、ここではそのやり方はとらない。一つには副作用を持つ言語に最終的に変換する場合、写像が困難であるからである。実現されたものは、厳密性と云う点では不完全であるが、そうしないよりも明らかに安全性が増す。

inferでは、そのために、pre（事前条件）、post（事後条件）、invariant（不変条件）等の表明用の予約語を持つ。事前条件及び事後条件についてはモジュール定義と比較され、整合性の検査が行われる。

不変条件については、インタプリタ実行時に動的に検査される利用方法と、対象プログラム言語に変換される際に組み込み、チェック機構とする利用方法がある。

但し、オプションとして対象プログラム生成時に、注釈としてのみ表示することもできる。

（２）複数のプログラム言語への変換

対象言語については、標準的に用いられている手続き型言語を対象としている。オブジェクト指向言語については、基本的なパラダイムの相違（継承機能、メッセージパッシング機能を有する）が大きいため、今回は対象から外している。

なお、ＰＤＬを、それぞれの対象言語の和集合とはしていない。完全なメタ言語にしていないのは、それだけで理解不可能な大きな仕様になってしまうと云うのが理由の一つである。

現在、Fortran66/77/90、c、Adaに対応している。

（３）学習容易性

ＰＤＬで論理構造を考え、最後に対象言語へ実装するのであるから、設計者は常に二つの言語で考えることになる。伝統的な記法は基本的に踏襲しつつ、簡略化を図ることで、容易に習得可能な言語である。

（４）対象言語との相互変換

相互変換とは、ＰＤＬからソースコードへの変換と、ソースコードからＰＤＬへの変換（リバース）を指す。注意しなくてはいけないのは、リバースを単純に行うと、ＰＤＬに変換されたコードが非常に見づらくなることである。それに対する対処を幾つか行っている。

例えば、コメント文は２種類あり、一つは"大きな"ブロックを表す。自然語への変換時には、大きなパラグラフを構成する。もう一つは、特に注意を呼び起こすような行単位の"小さな"コメントである。

また、リバース機能により、ドキュメントを持たない過去のプログラム資産の保守に利用することも可能となる。

3.3 記述例

良く知られている問題、"与えられた非負整数(n)の整数根を求める"のinferプログラムを、例として以下に示す。

```
pre   { n >= 0 }                                    // 事前条件
@ld                                                 // ローカル変数宣言
    int x := 0 ; int y:= n ; int q := 1 ; int k := 0
@dl
while q <= n
do
    q := 4*q ; k := k + 1
od
while q != 1
invariant { 0 <= y == n - x*x/q < 2*x + q
          and mod(x,q) == 0
              and k >= 0 and q == exp(4,k) }
do
    q := q/4; k := k - 1
    if y < x + q
        x := x/2
    else
        y := y - (x + q); x := x/2 + q
    fi
od
post { x*x <= n < (x+1)*(x+1) and y == n - x*x }    // 事後条件
```

4. 最後に

例えば、高級言語と呼ばれている言語は、コンパイラを介することによってアセンブラと比較して一段抽象度の高い言語になり、生産性と品質の向上に大きく寄与した。それと同様に、ソフトウェア開発支援環境の枠組みの中で、言語を考えることによって、プログラマの対象とする言語をより抽象度の高いレベルに上げることが可能となる。

今回提案している言語は、これからの開発環境の枠組みの中で、生産性及び品質向上に寄与する実践的な言語である。

今後は、不変条件の自動導出や限量子のinferコードへの自動展開等について研究を進めていきたい。

【参考文献】
[1] "IEEE Recommended Practice for Ada as a Program Design Language", ANSI/IEEE Std 990-1987
[2] K.Lew et.al "Software Complexity and Its Impact on Software Reliability",IEEE TSE Vol14,No.11,1988
[3] E.W.Dijkstra "A Discipline of Programming. Prentice-Hall,1976
[4] 伊藤他、「ソフトウェア開発支援システムW-１の開発」、第４４回情報処理学会全国大会論文集３Ｊ-７
[5] B.Meyer "EIFFEL: the language" Prentice-Hall,1992