ICOT Technical Memorandom: TM-1253

TM-1253

PIMOS のこれまでの研究を振り返って
－PIMOS 回想録－

佐藤　正樹

March, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

PIMOS のこれまでの研究を振り返って
— PIMOS 回想録 —

1 はじめに

PIMOS は1988年に Multi-PSI 上に実装されて以来、多くの実験的応用ソフトウェアの研究開発を通じ、さまざまな拡張、改良を施してきた。今年度をもって5世代プロジェクトも後期を終え、プロトタイプシステム開発のフェーズを終えることとなる。PIMOS の研究開発も大きな区切りを迎えるにあたり、当初掲げた目標を達成できたのか、大規模並列計算機システムの OS として後生に残せる技術を確立することができたかということを振り返って考えてみることは重要であろう。

PIMOS の研究開発を進めるにあたり、目標として掲げたのは

1. 良好な並列記号処理ソフトウェア開発環境を早期に提供する。
2. 大規模並列計算機システムの OS のあるべき姿を示す。

であった。PIMOS 全体として見ると、並列ソフトウェアの実践的研究が PIMOS が提供するプログラミング環境を利用することにより飛躍的に発展していったという事実から、十分目標を達成できたということが言える。そこで、次節以降では PIMOS の構成要素、すなわち、FEP、カーネル、I/O システム、ファイルシステム、言語システム、デバッガ、シェル、ParaGraph、AYA 、リリース管理の各面から上記目標を達成できたかどうかについて各担当者に振り返ってもらった。また、数人のユーザから、ユーザの立場から見てどうだったかを振り返ってもらった。

2 FEP(天沼:JIPDEC)

2.1 良好なソフトウェア環境を提供できたか

FEP は、リモート化によるユーザ環境の整備により、さらに良好な環境になったと思う。が、エラーによるダウン時の復旧を最近になってから整備したことが、悔やまれる。

また、tcp/ip が普及してきた時点で、socket デバイスを作成しておくべきだったと、今なら思える。少々 string I/O に頼り過ぎたきらいがあった。

2.2 十分早期に提供できたか

開発を顧みてみると、基本機能は十分早かったと思うが、デバイスのいくつかに関しては、もっと早めに提供できたはずであり、すべきであったと思う。

また、UNIX 版 FEP もできればもっと早めに着手しておくべきであったと思う。

2.3 大規模並列（記号）処理 OS のための新しい技術は何か

現在の FEP の技術で、PIMOS 開発開始以後には目新しい技術はない。開発当初、まだ一般的ではなかった SCSI を採用したことが、最大の成果と思う。

3 カーネル (藤瀬:ICOT)

3.1 良好なソフトウェア開発環境を提供できたか

カーネル自身 PIMOS 照会書をほとんど書いたことがないことから判断すると「ダウンしにくかった」と言え、良好な環境を提供できたと言える。しかしその反面カーネルが楽をした分、処理系に負荷がかかってしまったのは残

念である。またメモリ不足にカーネルとしてはお手上げだっとは、設計に問題があったのかもしれない。

並列論理型 OS において非常に難しかった共有資源の問題を標準入出力で解決できたのはよかったと思う。この概念をまとめたため、後の TELNET 等の実現が非常にスムーズだった。この点からみてもカーネルは充足していると思う。

クライアントサーバモデルの実現については、結局難解であったことがネックとなったかもしれない。しかし導入したおかげで諸々のデバイスの提供が可能となったことからみて機能として充足していたと思う。しかしサーバ開発環境の整備が不足であった面をみると「提供できなかった」とも言える。

デバイスとのインタフェースについては、PIMOS 本体はほとんど FEP 独立に設計したため、FEP が絡んだユーザインタフェースがやや複雑になった。特に I/O node なる概念は最後まで理解してもらえなかったのは残念である。FEP 自身の機能をもっと省力化すべきであったと思う。FEP が SIMPOS をひきずる限り、カーネルと FEP とのインタフェースがミスマッチになってしまうのである。

3.2 十分早期に提供できたか

カーネル自身 3 回書き直しを行なったため、リリースが遅れた気がする。しかしもともとモジュール度が高く設計されていたため、他の箇所への影響度は低かったといえる。

並列論理型 OS としての共有資源の実現は、概念が非常に難しかったため難航を極めた。これについては初期の頃に解決しておきたかったと考える。

3.3 大規模並列（記号）処理 OS のための新しい技術は何か

カーネルを眺めると、並列論理型 OS は、各種管理モジュールが自律的に動作していることがわかる。つまり管理状況は、刻々と変化し、全体を把握することは不可能に近いことがわかる。つまり OS が提供する資源ひとつに至るまですべて自律的に管理する機構が必要であると思う。この観点からみると新しいとは言えないが次の技術が必須であると思う。

- エージェント指向
  エージェント間の交流のために用いた KL1 の基本機構は、「多重待ちと同期」機構であり、具体的には次の技術を必要とした。
  - 優先度付きマージ機構

  KL1 でなくともこの機構が実現できれば、エージェント同士の有機的な結合が可能となり、自律管理機構が実現できると思う。

また、OS とユーザとのメッセージ交換の道具立てとして

- プロトコル定義と保護フィルタ
  を用意した。プロトコル定義の制約により、管理側にとって厳しいメッセージ処理を排することができた。ある程度数学的にも抑えられる概念である。

まとめると大規模並列処理 OS のためには、自律分散管理機構が必要であり、逆にこれだけでほとんどすべての管理が可能であるといえる。

4 I/O システム (屋代:ICOT)

4.1 良好なソフトウェア環境を提供できたか

機能的に見れば、I/O 機能は若干貧弱であった。しかし、必要最低限の機能を提供し、その部分に関してはそれほど大きなバグもなかったという点では、「良好なソフトウェア環境」であったといえる。

足りない機能に関して、全く手を下さなかったわけでもなく、FEP 上のユーザ定義デバイス (ストリング I/O デバイス) を提供して、「足りないところは、勝手にやってね」と言う方針をとってきた。アプリケーションプログラムのほとんどが、この機能を使っていたことからも分かるように、この機能は非常に有効であった。機能拡張が容易であることは、研究システム用 OS の重要な機能であることも、このことから言えるのではないかと思う。

また、デバイスのアボート機能やアテンション機能などは、PIMOS の I/O で一番工夫してある点だと思うが、これらの機能を、ユーザが直接使っている例をほとんど見たことがない。これは、不必要な機能だったというより

も、宣伝をあまりしなかった、とか、こういう機能を使うようなプログラムはシェルやリスナのようなメタなプログラムに限定される、などの理由で、あまり使われなかったのだと思う。

4.2 十分早期に提供できたか

十分早期に提供するというのが、PIMOS の I/O 機能を開発する上での最重要課題であった。この点では、I/O 機能に関しては、十分早期に提供できたと思う。

各版の動きを追ってみると、次のようになる。

第 1 版: 低レベルな処理部分をすべて FEP に依頼して実装。KL1 で実現したのは、ファイルやウィンドウといった仮想的なデバイスの管理であり、極力、未成熟な処理系の負担を回避した。

第 1.5 版〜第 2 版: リモート機能。これも、主に FEP の改修がほとんどであり、PIMOS の KL1 部分の改修は FEP の改修に較べて、ごく少なかった。

第 3 版: FEP インタフェースの変更。FEP のインタフェースが、高機能な Multi-PSI の専用インタフェースから、PIM に対応した低レベルな SCSI インタフェースとなった。ここでは、入出力機能を BIOS という形で仮想化し、PIMOS の書き換えに柔軟に対応できた。

4.3 大規模並列（記号）処理 OS のための新しい技術は何か

PIMOS の I/O 機能については、特に、大規模並列処理 OS のための技術に関しては、メッセージ指向の I/O インタフェースであろう。たとえば、デバイスのメッセージにとり入れた attention、abort の機能は、並列処理に必要なスループットとレスポンスの相反する性能を実現している。これに、reset、resend というメッセージを付加するだけで、通信路を多重化できると言う付加価値も見逃せない。

だが、従来からあるバッファリング等の「普通の」OS の技術も、実際に使ってみた上で、やっぱり必要であることが分かった。これは、システムを作る上では当たり前の話しなのだが、上述のような並列処理のための新しい技術に関する部分は全体の数%で、後の大半は既存の OS で使われているような技術である。実際にやってみると、新しい OS 技術を生み出すためには、何でもないような既存の技術も大事にしていく必要がある。

5 ファイルシステム (佐藤:ICOT)

5.1 良好なソフトウェア環境を提供できたか

PIM は、システム全体が新規開発されたものであるので、システムダウンの発生率は商用のマシンに比べてはるかに大きいものである。そのためシステムダウンが発生してもファイルシステムの管理に必要な情報が失われないような障害対策を考えたシステムを作るというのが目的の一つであった。具体的には本来のデータ格納場所（実体）とは別にログ領域というものを用意し、更新はまずこのログ領域に対し行ない、その書き込みが終了したら、実体にデータを書き込むという方式をとった。実体書き込み中にシステムがダウンしてもログ領域に書いたデータを元にデータの復元ができるので、ユーザは安心してこのファイルシステムを使用することができる。そういう意味では良好な環境を提供できたと思う。しかし、次節でも触れるが、ディレクトリ構造を持つファイルシステムを提供できなかったことは良好なソフトウェア環境という点では不合格であった。

5.2 十分早期に提供できたか

ファイルシステムは当初二段階にユーザにリリースする予定であった。第一段階としてユーザを Kappa-P のように、大量データを扱うユーザ、すなわち、FEP のディスクでデータの I/O を行なっていたのではとても実用時間内に終らないものを扱っているユーザを対象に、多少の使いにくさはかまわないから PIM のノードに直接つながっているディスクを使用して高速に I/O を行なえるようなシステムを提供することを目的とした。第二段階として、ディレクトリ構造を持ち、ユーザフレンドリなインタフェースを持つシステムを提供することを目的とした。第一段階のものは FGCS'92 に向けてそれなりに使えるものが提供できたが、「十分早期に」という意味では半年ぐらいの遅れがあったと反省している。第二段階のものは設計が完了するレベルにしか至らず、ユーザに提供することができなかった。これはおおいに反省すべきであり、スペックの絞りこみなどの方策を早期に施すべきであった。

5.3 大規模並列（記号）処理 OS のための新しい技術は何か

このファイルシステムの研究を通して、疎結合並列計算機向けファイルシステムの要素技術として分散キャッシュの技術を得ることができた。疎結合並列計算機では、ノード間の通信を如何に削減するかが重要なポイントとなる。その解決策として分散キャッシュ技術があげられる。特に分散キャッシュ技術の中で問題となるのはデータの一貫性制御である。この研究で、一貫性制御のためのオーバヘッドを最小限に押え、ノード間の通信量を削減したキャッシング方式を確立することができた。

6 言語システム (関田:MRI)

6.1 良好なソフトウェア環境を提供できたか

これは総じて考えると「並」というところか。一応ひと通りの機能は揃え、マクロ機能もそこそこ使えるものが用意できた。また、ユーザ定義プリプロセッサ機能は他の言語システムではあまり見られないものであろう。これらは最近ハックしているユーザが出てきて頼しい。

一方、コンパイラのバグが多く、最適化もあまり頑張れなかった。クローズインデキシングには異常に開発コストがかかってしまったが、それ以外に、「最適化モード」と称して (つまり、トレースをさせることはあきらめて)、ソースレベルで書き換えてしまう (例えば inline 展開とか、データフロー解析により中断を起こしそうなゴール群をまとめてしまうとか。これらは低コストではあるが、効果はかなり大きそう) といったことがなぜ行なえなかったと悔やまれる。結局「実用」と「研究」の両立は難しい、という感想をもつ。

また、「高水準命令マシン」であるため命令 1 つづつの「重さ」がばらつき、最適化の効果の見積りは難しい (firm を読むくらいのことをすればよいのだろうが)。

6.2 十分早期に提供できたか

レスポンスは (最適化をのぞくと) そこそこよかったのではないかと思う。ただし、モジュール管理および MRB 回りのバグが発生した場合、その bug を fix することの「提供」は遅れた。モジュール管理については一般的に impure にやりたくなるようなことを pure な枠組でおこなったための困難であり、MRB は pure な機能を impure に実現する、ということからおきているといいかえることができる? とかく pure は (使う方は嬉しいが) 作る方は難しい。

6.3 大規模並列（記号）処理 OS のための新しい技術は何か

「アトムの管理」のような「大域的なデータの管理」は古くて新しい並列 (というより分散?) の課題であろう。このキャッシュがどの程度効いていたか (キャッシュがなければどのくらいひどかったか) はわからないし、キャッシュの効率はどの程度であったかなどの評価はなされていない (そもそも、write through 程度のキャッシュしかしていなかった?)。もっとも「アトム」の動的な管理が問題になる局面はあまりなかった。

7 デバッガ (中尾: 応用技術)

7.1 良好なソフトウェア環境を提供できたか

7.1.1 リスナ

- 基本的なデバッグ環境は提供
  プログラムの試験実行、トレース、デッドロック検出機能、標準入出力、マルチタスク管理、性能評価ツール (ParaGraph) とのインタフェースなど、KL1 プログラムのデバッギングに必要な基本機能は一通り提供できたと思う。

- 実用に耐え得るシステムを作るには処理系の協力が不可欠
  本格的な応用プログラムをちゃんとデバッグできる実用性のあるシステムを目指した。トレースやスパイ、デッドロック検出機能などの、メモリ消費量や実行速度がデバッグ対象のプログラムの規模に大きく左右される機能は、言語処理系が提供する荘園の機能を利用して実現した。たとえばトレーサーを実現する方法としてソフトウエアでメタインタプリタを作成する方法も検討したが、実行効率を考えると本格的なプログラムのデバッグは困難であり採用しなかった。このため処理系の開発には負担がかかったが、そのおかげでソフトウエアの

開発は比較的楽に進めることができ、実用性の高いシステムを提供することができた。また、ユーザに対しても本格的なデバッグ機能を開発するためのプリミティブを提供できたという点で好都合であった。

- 荘園のトレース機能を利用して試作したスパイ機能
  まだ処理系にスパイ機能が実現されていなかった時期に、荘園のトレース例外機能を利用してスパイ機能を試作した。すべてのゴールをトレースモードで実行して、ゴールのリダクションが報告されるたびにスパイのターゲットかどうかをソフトウエアでチェックするというものであった。何もないよりはマシと言う程度で、とても実用規模のプログラムのデバッグに使用できるものではなかった。処理系にスパイ機能が実装された時、そのパフォーマンスの違いには唖然とさせられたものである。しかしながらこのソフトウエアでのスパイ機能は意外なデバッグ機能を持っていた。デッドロック検出に利用できたのである。ゴールをトレースモードで実行する時にそのゴール情報を覚えているため、リダクションが報告されないゴールを検出することができた。当時はまだ処理系にデッドロックの検出機能がなかったこともあり、それなりに役に立った。色々と試行してみることは大切である。

- トレーサーは60点
  対話的なプログラムの実行環境としては、リスナは通常のPrologシステムと同様のユーザインタフェイスを提供したので、ほとんどのユーザはあまり違和感を感じることなく使用できているのではないかと思う。しかしながらトレーサーについては、一見、逐次型言語のステップトレーサーに似たインタフェイスでありながら、実行の順序がゴールの親子関係しか保証されず、色々な実行が交じりあって報告されるため、デバッグ対象の絞り込みという従来にはなかった操作が必要になる。このため逐次型言語でのプログラム開発に慣れている多くのユーザは、使用当初は戸惑いがあったであろう。もちろんステップトレース機能は必須の機能であり、デバッグ機能として大いに貢献していると思うが、この他にプロセス間のストリームを流れるメッセージに着目したトレース方式や、変数(引数)の具体化に着目したトレース方式も提供できれば良かったと思う。また、トレーサーにはたくさんのコマンドを用意しているが、ユーザにどの程度使用され、どの程度デバッグの役に立っているのかは疑問である。操作性という点ではもっと機能を絞って必要最小限、単純にした方が良かった。マニュアルをわかりやすく、ユーザが読む気になる量に抑えるということも大切であった。トレーサーの出来は自己採点すると60点というところである。

7.1.2 インスペクタ

- リスナとの連係で威力を発揮
  インスペクタはデータのタイプチェックや複雑な構造を持つデータの中身を調査するためのツールであり、デバッグには不可欠な機能であるが、かなり地味なシステムである。その機能や操作方法はSIMPOSやLISPマシンなどが提供しているものとほぼ同等であり、並列言語ゆえの特殊な機能は特に持っていない。そういった点では違和感なく使用されていると思う。一番良かった思うことは、リスナに組み込む形で一緒にリリースできたことである。トレーサーとの連係、変数プールを介したデータのやりとりなど使い勝手の面で威力を発揮したと思う。

7.2 十分早期に提供できたか

前述の通り、リスナが提供しているデバッグ機能の大部分は処理系が提供している機能をベースとして開発している。このためリスナのデバッグ機能としてユーザに提供するための開発にはあまり手間暇がかからなかった。処理系の新機能搭載からユーザへのリリースはスムーズに行なえたと思う。以下にデバッガのあゆみを記す。

暫定版 (89.4): 対話的なプログラムの実行環境。変数の値の出力。組込述語 unbound は提供されていない。alternatively を利用して未定義かどうかを判定。トレース機能なし。主にPIMOS開発者用。資源管理部の大改造、シェルのピンチヒッターとして役立つ。

第1版 (89.7): 正式なユーザリリース。トレース機能の搭載。スパイ機能はトレース例外を利用してソフトウエアで実現。インスペクタ搭載。従来のシェルしかなかった環境から比べるとユーザにとっては格段の進歩。

第1.5版 (89.12): 処理系のスパイ機能、デッドロック検出機能が実現。デバッグ効率が格段に向上する。やっとまともなトレーサーらしくなった。

第2版 (90.10): パッケージ機能導入。タスク内トレースが可能になった。処理系の荘園プロファイル機能実現。暫定的なプロファイルコマンドをリリース。

第 2.5 版 (91.1): ParaGraph デビュー。プロセサプロファイル機能はまだなし。ユーザインタフェイスの向上。ノードの分割運用始まる。

第 3 版 (91.11): プロセサプロファイル機能搭載。パフォーマンスメーターもプロセサプロファイル版に移行。PIM/p 版も FGCS92 で活躍する。マルチウインドウを利用したトレース方式、マルチタスク管理機能 (フォアグラウンドとバックグラウンド) 提供。デバッグ環境は試験実行、トレース、性能評価支援機能と一通り揃う。

7.3 大規模並列（記号）処理 OS のための新しい技術は何か

大規模というわけではないがトレース機能に関していうと、処理系にトレース機能を搭載する時に現在のトレース方式のほかに「変数トレース」という機能も検討した。現在のトレース方式は、トレース対象の指定はゴールに対してのみであり、ゴールの親子関係に着目したトレース方式となっている。変数トレース方式は、指定した変数が具体化される場合、ガード中でその変数の値を読むようなリダクションがトレースの対象となる。これはデータの流れに注目したトレース指定を目的としたものであり、より具体的にはユニフィケーションとリダクションの因果関係に注目したトレース指定方式となる。処理系の改修・開発に負担がかかるということで実現できなかった。非常に残念である。

8 シェル (松尾:MRI)

8.1 良好なソフトウェア環境を提供できたか

ユーザ・インタフェースに関する部分は、まだまだいろいろなサービスを提供できる部分が残されていると思う。

8.2 十分早期に提供できたか

標準入出力の部分をもっと早くシェルから分離し、独立させた方が良かったかもしれない。

8.3 大規模並列（記号）処理 OS のための新しい技術は何か

処理自体を並列分散させても、データや指示を与えたり、結果を出力する部分は、人間とのやりとりになる。この入出力部分は、操作性等を考えると、どうしても逐次的にならざるを得ない部分がある。並列に処理を行う部分と、入出力等のために逐次的に処理を行わねばならない部分との繋ぎの部分（標準入出力）を何かうまく処理する機構が必要だと思う。例えば、ストリームに priority merge を入れる（それですべてがうまく解決するとは限らないが）とか、荘園の絶対 priority をもう少しユーザから制御できるようにするとかすると良いかもしれない。あとストリームに流れるメッセージを監視する機構は是非とも欲しい。

話は変わるが、大規模なプログラムを実行する際には、exception handling 部分（荘園の機構ではなく、例外報告後の対処）を充実させる必要があろうかと思う。例えば、数万台の PE から一斉に例外報告を行われても、利用者はとても一つ一つに対処できないから、ある程度自動的に復旧が行われるように工夫する必要があろうかと思う。

9 ParaGraph(清原:ICOT)

9.1 良好なソフトウェア環境を提供できたか

ParaGraph は、荘園の中の実行状況を実行終了後に可視化してみせる機能と、実行中にプロセッサの稼働状況、ヒープの使用量などをリアルタイムに表示する機能がある。どちらも、パフォーマンスのデバッグには必要不可欠なものである。現状では、実験的なプログラムに使用する上では、かなり良好な環境といえるが、大きな問題に対しては、メモリの制限から、必ずしも十分な環境とは言えず、まだまだ改良の余地がある。

9.2 十分早期に提供できたか

実験的なプログラムがパフォーマンスを気にするようになってきたとき、とりあえず使うには十分な機能がすでに提供されていたわけで、必要最低限の機能は、十分早期に提供できていたということができる。

9.3 大規模並列（記号）処理 OS のための新しい技術は何か

パフォーマンスデバッグの観点からみると、機能的に、世間ではないものといった機能を新たに提供したわけではないが、この手のデバッグツールがなければどうしようもないということははっきりしたであろう。この後もこの手のツールは、大規模並列（記号）処理 OS には必ず使われていくものである。

10 AYA(寿崎:ICOT)

10.1 良好なソフトウエア環境を提供できたか

KL1 のユーザ用言語としての不十分さを補うために設計した言語であるが、本当の使いやすさを目指すといういみではまだ不十分である。言語の概念整理、シンタックスにも改良の余地がある。また、専用のデバッガの設計開発もこれからである。専用デバッガではプロセスとその間のメッセージを中心としたデバッグを行なえるようにすることによりプログラム開発の効率をあげまた初心者にも学びやすい環境を提供することができる。

10.2 十分早期に提供できたか

予定より大幅におくれてしまった。

10.3 大規模並列（記号）処理 OS のための新しい技術は何か

AYA については OS ではないが、コミッテドチョイス型の並列論理型言語とそれによるプロセス指向プログラミングという土台の上に、PIMOS を記述できる記述力を提供するという制約のなかで、プロセスに状態 (AYA ではシーンとよぶ) という概念を導入した。これは新しいアイデアだったと思う。しかし、現在の言語仕様はこの概念を十分に生かしきれていないのではないかという疑問がある。 デバッグにおいてもあたらしい切口を見い出したいと思う。

一般に PIMOS については、資源管理部では資源木など新しいアイデアがありおもしろかったと思うが、デバッガも含めたインタフェース部分は Prolog を踏襲したデザインが多くそれはそれで使いやすいものであるが、もっと並列性を意識したインタフェースとしての実行環境も提供できるとよかったのではないかと思う。

11 リリース管理 (西ヶ谷:ICOT)

11.1 良好なソフトウエア環境を提供できたか

障害管理を担当して内部及び外部からのクレーム処理を行なってきたが、これまでに発生した問題点はほとんど２週間以内に回答され次版の改修において反映されたと思う。また、今年度より電子メールによる照会書受付も開始した関係で照会書を 受付してから利用者への返却時間が大幅に改善されクレームの早期解決 および利用者へのサービス向上を図ることが出来たことにより良好な ソフトウェア環境を提供できたと考えている。

11.2 十分早期に提供できたか

この点については、ソフトウェアのバグ対処や環境改良をどこまで行なうかという妥協点をどこにするかということで開発当初に定めた期日を多少オーバーするのも仕方ないと思う。我々のグループでは、ＯＳ開発者よりＯＳの媒体を受けとってから外部へ発送するまでに手作業の部分が多くあったのでこの点ではリリースに時間が掛かってしまったと思う。また 3.5 版のリリースは、ネットワークを利用し外部リリースとなったが 早くからネットワークでのリリースが出来るようになっていればリリースも速やかにできたのではないかと今になるとこの点が残念である。

12 ユーザの立場から（森:ICOT、六沢: 沖電気、石田:ICOT、平田:MRI、河村:ICOT、鬼塚:ICOT、平野:ICOT）

12.1 良好なソフトウエア環境を提供できたか

- 残念ながら、もう一息と言ったところか。個々のツール（エディタ、リスナ、シェル、コンパイラ etc ）の完成度が足りなかったと思う（バグがあったりして、今一つ信用がおけない）。個々のツールのデザインに関

しては、まあまあだったと思うが、リスナ回り（インスペクタ等含む）はまだ勉強の余地がありそうだ。その他、リンカの辺りは最後まで良くわからず仕舞いだった。

- PIM というマシンを対象にした環境という意味では達成したと思う。
- FEP シミュレータによる簡易デバッグツール VM は、PIMOS 開発者にとってはかなり有効な環境を提供できたのではないか。PIMOS のオープン化を目的として開発したテルネット機能は、一応はその目的に貢献できたと考える。ただ、PSI-II または PSI-III を利用できるユーザにとっては、テルネットではウインドウデバイスが利用できないなどで十分なメリットはなかったと思う。
- 外部記憶装置に対するインタフェースが不十分（ない？）。たとえば、CMT のユーザへの見せ方（ノンラベルのファイルとするなど）と PIMOS ファイルとの関係（UNIX 風にデバイスファイルとするとか）。原因は PIM についていなかったことかもしれないが。PIMOS ファイルにソースファイル、アンロードファイルをおいてコンパイル、ロード等ができるようにしたかった。ソース、アンロードファイルが巨大だと FEP との通信ネックになる。
- Debugger、Listener などは、使い易いものであった。しかし、SIMPOS ベースであったため、ファイル転送などで、困難な部分が多かった。とくに、問題なのは、PIMOS を利用するために、SIMPOS についても有る程度の十分な知識が必要であり、初めて使う側としては、問題が多かった。また、KL1 言語処理系は、いろいろな機能をほとんど言語機能として実装しているため、膨大なマニュアルを読まないと使えないという問題があった。入出力は並列であるがゆえに難しく、デバッグも、一般の言語のように、print 文を入れて状態を知るなどということが出来なかった。
- PIMOS のユーザーといってもいろいろあると思うが、自分は (そして多くの関係者は) PIMOS 上でプログラムを開発していたわけで、「ソフトウェア環境」というのは「ソフトウェア環境を開発するための環境」になる。この視点で見ると、良好とは言えない状況である。
  - UNIX 風のルールが記述できる Make コマンドがなかった。コンパイラに !make はあるが、使い難い、どう動くのかよくわからない、マクロ定義の変更を無視する、最初のコンパイルに使えないといった欠点ばかりが目だった。UNIX 風 Make のほうが実装も簡単だったと思う。(結局、PIM/p 用 PIMOS のコンパイルは、PDSS が十分に速く動く SS10 が入ったので、PIMOS 環境から逃げ出して、make 一発でできるようにした)
  - バッチ処理がなかった (今度提供するようだが)。これが無いために、夜中に色々なテストプログラム (勿論、一部は途中でエラーが起きて終ってしまう) を流して、翌日に回収するということができなかった。エラーが起きると、いちいち「どうする？」と聞いてくる ので途中で止まっていて虚しい思いをしたことがある。エラーが起きた場合の処置が変えられるとよかった。

以上の 2 点のために、PIMOS は開発にはなるべく使わないようになっていた。結局、PIMOS の機能で役にたったのは、

- ParaGraph
  高速化にずいぶん寄与した。これで、ParaGraph 自身がもう少し速ければよかった。

ぐらいである。しかし、

- listener
  すいぶん強力なツールだった。そのため複雑で、結局、十分に使いこなせなかった。

というものもある。

12.2 十分早期に提供できたか

- PIMOS 自身は割合早期から提供され、十分だったと思う。ファイルシステムに関していえば、全然駄目だった。ただ、ハードがないというのは結構足を引っ張る要因だったと思う（ファイルに限らず、ユーザが少なくてバグを出せないという点でも）。

- PIMOS 自身は十分早い時期に存在していたと思う。ただ、PIMOS がその上で動く KL1 処理系、PIM/x ハードウェアの方は十分早かったとはいえないだろう。
- プログラミングシステムに関してはそう言える。ファイルシステムは早期に実現できなかったことが悔やまれる。AYA も、もう少し早くリリースできればよかった。
- VM の提供により、PIMOS 本体が開発しやすくなった分、ユーザに新機能が適切な時期に提供できたのではないか。特に海外からの利用者にとって有効と思われるテルネット機能は、一応の開発終了から諸環境を整えリリースする迄にやや時間がかかり過ぎた感は否めない。
- PIM の遅れで実機上で動作するものは遅かった。それ以外は○。
- かなり早い時期から、かなり仕様の固まったものが提供されたので、利用する側としては、よかった。
- Multi-PSI 用、PIM/m 用は十分に早かったのではないだろうか。自分が担当した PIM/p 用はというと、機種依存部分の切り分けやコンパイラのバグなどでずいぶん遅れている。PIMOS 3.50 もまだテストしていなくて、年度内はあと 3 日という状態である。最近になって、やっと、早期追従が可能な環境が整いつつあるので、今後は、結構早く提供できるようになると期待している。

12.3 大規模並列（記号）処理 OS のための新しい技術は何か

- ユーザがいるところに OS が行くという辺り（特に資源管理）は新しいかどうかは良くわからないが、並列 OS の考え方の基本となるのではないだろうか。
- パフォーマンスチューニングを行なうための情報 (各プロセッサの稼働状況など) を収集しユーザに提供する技術。素性のわかっているプログラムの一部についての自動負荷分散技術（完全な general purpose 自動負荷分散なんかできないので）。
- OS としては、資源の階層管理のメカニズム。KL1 言語としては、データフロー並列性による自動同期機構。負荷分散機構－プラグマにより、プログラムの意味をかえずに負荷分散の実験ができる。処理系レベルでは、クラスタ内負荷分散。

13 おわりに

以上各立場から PIMOS の研究を振り返ってみた。大筋では PIMOS は良好な並列記号処理ソフトウェア開発環境を早期に提供し、また、大規模並列（記号）処理 OS の新しい技術を確立したといえる。が、細かなところを見てみると、まだまだやり残してきたことは多い。今後このプロジェクトは KL1 の普及を目指して商用ハードウェア上に並列処理のためのソフトウェア環境を開発するというフェーズに移行していく。その際に、今までの研究開発で得られた知見を十分に生かして行きたいと思う。