プログラマのデスクトップ

佐原伸

URL: http://www.sra.co.jp/people/sahara

E-Mail: sahara@sra.co.jp

1章 はじめに

９６年９月号のSEAMAIL（Volume 8, Number 4）に、山崎さんの「プログラマの本棚」が掲載された。ソフトウェア技術者が読むべき本10冊を精選して紹介しているのだが、その題を見た瞬間、僕は「現代のプログラマなら本棚でなくデスクトップであるべきだ」と考えた。また、同じ号に掲載された玉井さんの「あるパズル」のCommon Lsipのプログラムを見て、手元にあるMacintosh Common Lisp[1]では、もっと強引な方法が成立しそうだと思った。すぐにお二人の原稿に応える原稿を書くはずだったが、３年経ってしまった。しかし、そのおかげで、関数型言語に関して少しおもしろい結果を得たので報告したい。

2章 僕のデスクトップ

僕のデスクトップはPowerMacintosh　9500/120上にある。CPUは今や「遅く[2]」、メモリーは80MB、ハードディスクは6GBほどで、今の時代では「少量[3]」である[4]。遅いのを少しでも速くするため、Speed Doublerという機能拡張ソフトを使っている。これは680*0 CPU対応のソフトウェアの命令をキャッシュして、動的にPowerPCネイティブコードを生成することにより、標準の２倍の速度でプログラムを実行する。

プログラマである以上、デスクトップ上で最初に紹介するのは「プログラミング言語」または「仕様記述言語」であるべきだろう。こういった言語は、最近数年間で劇的な変化を遂げている。そのために、僕のデスクトップからはC言語は消去された。もちろん、Basicなどは10年前に駆逐されている。

2. 1 Smalltalk

まず、今、商売[5]に使っているSmalltalkがある。本家[6]オブジェクト指向言語である。Speed　Doublerと基本的には同じ仕掛けで、キャッシュ上のSmalltalkの仮想機械のコードを、動的にPowerPCのネイティブコードへ変換することなどによって、去年までのSmalltalkより劇的に速くなった。

しかし「遅くて使いものにならない」という10年前のSmalltalkでの評価を、未だに声高に叫ぶ人がいるので困っている。僕は、下手なC言語プログラマが書くコードより、Smalltalkの方が速いコードを生成すると思っている。まあ、足し算するだけなら確かにC言語の方が速いが、C言語の関数呼び出しとSmalltalkのメソッド呼び出しとでは、ほとんど効率が同じになってきている。

また、「Smalltalkは難しい」という根拠の無い説を振り回す人もいて困っている。実際には、市販されている

[1]MCLやこのあと紹介する言語などの情報のURLは、インターネットの検索エンジンを使ってアクセスして欲しい。URLは結構頻繁に変わるので、ここでは記述しない。

[2]これは「皮肉」ではない。私のまわりの素人衆のかなりが、これより良い環境を持っている。

[3]これも「皮肉」ではない。私のまわりの素人衆でも、これより良い環境を持っている人達がいる。

[4]これは自宅での環境で、会社の環境はさらに良くない。

[5]「商売になっとらんぞ」という陰の声も聞こえる。先の見えない人達には困ったものである。

[6]元祖はSimulaであろう。

言語の中では最も簡単な言語である。

つまり、Smalltalkは「実世界で完全に実用になる」言語になったのだ。元々、開発環境やクラス・ライブラリーには定評があったので、一挙に実用言語のトップに躍りでた感がある。

2. 2 関数型言語

さて、商売は以上にして、関数型言語に話を移したい。山崎さんの「プログラマの本棚」に最初に紹介されているのが「関数プログラミング[7]」の本であるが、 関数型言語は仕様記述言語との整合性も良い。

僕のデスクトップにはベル研で作られたStandard ML of New Jerseyとオランダのナイメーヘン大学で作られたConcurrent Cleanという関数型言語がある[8]。Standard MLは強力な関数型言語であるが、コンピュータソフトウェアの９５年１月号から４回連載で大堀さんの解説が載っているし、アスキーからUllmanの本[9]が翻訳されて出ているので、ここでは主にConcurrent Cleanの方を紹介しよう。

Concurrent Cleanは、逐次・並行・分散の実アプリケーションを開発することを目的とした純粋で汎用の関数型言語で、後述するモジュール機構や高階関数やlazy evaluationを具備している。

手続型プログラミング言語（C, Basic[10], Fortran, Pascal, PL/I, COBOLなど）は、フォン・ノイマン型コンピュータあるいはそれと同値なチューリング・マシンをモデルとしている。この種の言語の利点は「アーキテクチャの単純さ」で、要するにコンパイラーを作りやすい。

しかし、手続型プログラミング言語を使ってプログラムするのが容易なわけではない。特に問題なのが「代入」で、同じコードでも変数の値によって異なる振る舞いをするという副作用を起こす。従って、手続型言語で「正確さ」を求めるのは非常に困難なことが知られている。しかも、アルゴリズムが必要以上に逐次型になるので、並行処理に向いていないた、実行効率も本質的には良くない。

実は、コンピュータ言語のモデルがフォン・ノイマン型のコンピュータである必要はない。オブジェクト指向という人間のコミュニケーションをモデルにした言語や、数学をベースにした論理型言語や関数型言語が生まれた所以である。

Concurrent Cleanは、代入を許さない「純粋」な関数型言語[11]であり、変数[12]もない。従って、プログラムを宣言的に書くことができ、関数の評価順序は結果に影響しない。このため、プログラムの正しさを証明しやすいし[13]、並行実行しやすい。メモリ管理は自動化されていて、C言語のように実行時にメモリーを破壊することがないし、そもそも、データ構造に不正にアクセスするとコンパイル時に検出される。後述するパターンとガードを使って、プログラムを短く書けるので、拡張性と保守性に優れている[14]。

[7]武市正人訳。関数プログラミング。近代科学社、１９９１年

[8]Moscow MLとCAMLというMLもあるが、まだ味見していない。いずれの処理系も無料である。

[9]Jeffrey D. Ullman著。神林靖訳。プログラミング言語ML。アスキー出版局、1996年

[10]巷でBasicと言われている言語は、実はBasicですらない。本家ダートマス大学のBasicはもっとまともな言語である。

[11]MLは「不純」な関数型言語で、奨励されてはいないが代入もできる。

[12]変数という言葉は、別の意味で使われている。

[13]とは言っても、実際の証明はそれなりに大変だし、全部証明できるわけでもない。

[14]このあたりは、最近のすべての関数型言語で共通の利点である。

問題は関数型言語ですべての処理を書けるのかという点だが、関数型言語が基盤としているλ計算は、チューリング・マシン・モデルと本質的に同じことが証明されており、問題はない[15]。

残る問題は、最近まで「効率が悪かった」ことと、もともと「関数」になじまない性質をうまくプログラムできるのかという点だけである。

「効率が悪かった」主原因は、今までのコンピュータが関数型言語を支援するアーキテクチャを持っていなかったことであり、今までのアルゴリズムが主に手続型であったことである。しかし、項書き換えをグラフの書き換えすなわちポインターの付け替えに置き換える「項グラフ書き換え[16]」や、メモリーのガーベージコレクション技術の発展により、関数型言語のコンパイラーはほとんどの場合に効率的に問題ない[17]コードを生成するようになった。

また、対話的プログラムやデータベース・OS・プロセス制御といった、基本的に手続的アルゴリズムが主だったアプリケーションにも、関数型のアルゴリズムが実現されはじめた。Concurrent Clean自体のGUIインタフェースや例題に、これらの実装例が示されている。図１に、Concurrent Cleanで実装されたテトリス実行中の画面を示す。

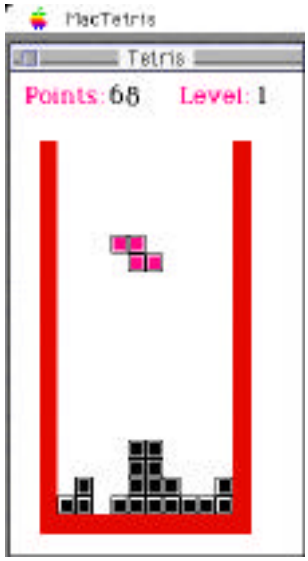

図 1　テトリス

ちゃんと遊べる。僕がConcurrent Cleanを気に入っている理由も、MacintoshのGUIインタフェースを関数型ライブラリーで実装していて、Macintoshの実用的なソフトウェアが作れるからである。

2. 3 Concurrent CleanとMCLによる「あるパズル」のプログラム

さて、口上ばかりでは何なので、実際に玉井さんの「あるパズル」のCommon Lispのプログラムを Concurrent Cleanで書き直してみよう。パズルの問題は、次の□に１から９までの数字を重複無く入れて、等式を成り立たせるようにするというものである。

□　　□　　□
＿＿＿＿＿＿＿＿

[15]厳密には、「プログラマの本棚」で紹介されている「井田哲雄著。計算モデルの基礎理論、岩波講座ソフトウェア科学１２。岩波書店、１９９１年」を参照されたい。

[16]Concurrent Clean Language Report Version 1.1, University of Nijmegen, March 1996

[17]ベンチマークでは、Macintosh II-fxのConcurrent Cleanで、Sun 3のC言語より速いが、まだPowerPCネイティブコンパイラーではないPowerMacintosh 9500版ではさらにその10倍ほど速かった。

$$\frac{}{\square\square} \quad \frac{}{\square\square} \quad \frac{}{\square\square} = 1$$

玉井さんのCommon Lispのプログラムは、9! = 362880の順列を生成するのは「無限大に等しい」と見なし、最初の解を求めるところで停止する。しかし、最近の動的言語のすさまじいまでの効率向上を実感している僕は「多分いけるだろう」という予測のもとに全部の組み合わせを計算するようちょっと手直しをした。そのMacintosh Common Lsip（MCL）[18]のプログラムがプログラム１である。

```
(defvar N 9)
(defvar N1 (1- N))
(defvar digits (make-array N))

(defun init-digits()
   (setq digits #(1 2 3 4 5 6 7 8 9)))

(defun a-by-bc ( a b c)
   (/ a(+ (* 10 b) c)))

(defun add-three-rationals ()
   (let ((sum 0))
      (do ((i 0 (+ 3 i))) ((> i 6) sum)
      (incf sum (a-by-bc(aref digits (+ i 2))
                                  (aref digits (+ i 0))
                                  (aref digits (+ i 1)))))))

(defun print-result ()
   (format t "~d/~d~d + ~d/~d~d + ~d/~d~d = 1~%"
               (aref digits 2)(aref digits 0)(aref digits 1)
               (aref digits 5)(aref digits 3)(aref digits 4)
               (aref digits 8)(aref digits 6)(aref digits 7)))

(defun swap (i j)
   (let ((x (aref digits j)))
      (setf (aref digits j)(aref digits i))
      (setf (aref digits i) x)))

(defun next-permutation()
   (let ((i (1- N1)) (j N1))
      (loop (when (or (minusp i)
                  (< (aref digits i)(aref digits (1+ i))))
                  (return))
            (decf i))
```

[18]PowerPCネイティブ・コンパイラーで、マニュアルに「CやFortranより速い」と豪語する記述がある。

```
      (when (minusp i)
      (init-digits)(return-from next-permutation))
      (loop (when (> (aref digits j)(aref digits i))
                  (return))
            (decf j))
      (swap i j)
      (incf i)(setq j N1)
      (loop (when (>= i j)(return))
            (swap i j)
            (incf i)(decf j))))

(defun find-solution()
   (init-digits)
   (dotimes (i (fact 9))
      (when (= (add-three-rationals) 1)
         (print-result))
      (next-permutation)))
```

プログラム1　パズルの答（MCL版）

実行結果は、以下の通りである。

```
? (time (find-solution))
9/12 + 5/34 + 7/68 = 1
9/12 + 7/68 + 5/34 = 1
5/34 + 9/12 + 7/68 = 1
5/34 + 7/68 + 9/12 = 1
7/68 + 9/12 + 5/34 = 1
7/68 + 5/34 + 9/12 = 1
(FIND-SOLUTION) took 51,605 milliseconds (51.605 seconds) to run.
Of that, 1,930 milliseconds (1.930 seconds) were spent in The Cooperative Multitasking Experience.
660 milliseconds (0.660 seconds) was spent in GC.
 34,840,504 bytes of memory allocated.
NIL
```

玉井さんの予想通り、項の入れ替えという自明の別解以外に正解はなかったが、全数探索をしても、玉井さんのプログラム（Sun　3/60上のKCL）の半分の実行時間である。ちなみに、最初の解を見つけるだけなら0.712秒しかかかっていないので、百倍以上速いことになる。

さて、このプログラムをConcurrent　Cleanで書いてみようとしたが、順列を生成するnext-permutation()**という関数が極めて手続的なアルゴリズムで構成されているので、関数型言語初心者の僕の週末の片手間仕事としてはちょっと歯が立たなかった。全部の順列を生成するプログラムは、「プログラマの本棚」の「関数プログラミング」という本の中にあったので、それを一部手直しして流用した。効率はともかく、というよりメモリ空間を相当食い**

そうなので実行可能かどうかも分からなかったが、問題を記述するだけでもまあ良いだろうという気持ちでプログラムを書いてみた。

まず、全部の順列を生成するプログラムはプログラム２のようになった。

```
implementation module ExtendList
import StdEnv

interleave :: a !.[a] -> [[a]]
interleave x ys = [take i ys ++ [x] ++ drop i ys \\ i <- [0..length ys]]

concat = foldr (++) []

allPermutations:: !.[a] -> [[a]]
allPermutations [] = [[]]
allPermutations [x:xs] = concat (map (interleave x) (allPermutations xs))
```

プログラム2　全部の順列を生成するプログラム（Concurrent Clean 版）

interleaveという関数は、任意の型のデータ（aで表す）と、その型のデータのリスト（[a]で表す）を引数として、[a]のリスト（[[a]]で表す）を返す関数で、例えば、
interleave 1 [2,3,4]を評価すると、[[1,2,3,4] ,[2,1,3,4][2,3,1,4][2,3,4,1]]を返す。
ここで、\\ i <- [0..length ys]という記法はZF記法と言い、0からリストysの長さまでのiを生成する。今の場合、[1,2,3,4]である。また++はリストの連結を示し、takeはリストの最初のi個の要素からなるリストを返し、dropはリストの最初のi個の要素を削除したリストを返す。
つまり、interleave1 [2,3,4] のプログラムは、最初、

take 0 [2,3,4] ++ [1] ++ drop 0 [2,3,4]]= [] ++ [1] ++ [2,3,4] = [1,2,3,4]

を計算し、次に

take 1 [2,3,4] ++ [1] ++ drop 1 [2,3,4] = [2] ++ [1] ++ [3,4] = [2,1,3,4]

を計算し...という具合に計算を進めていく。
allPermutationsはすべての順列の組み合わせをリストで返す。例えば、allPermutations [1..3]を評価すると、[[1,2,3],[[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]を返す。allPermutationsの実装は「パターン」を使って実現されている。つまり、allPermutationsの引数が空のリスト[]であれば、空のリストのリスト[[]]を返し、xという要素の後にxsというリストが続くパターン[x:xs]であれば、concat以下の定義を実行する。この表現が関数型言語の強力な武器の一つである。他にも、以下の階乗のプログラムのように「ガード」と呼ばれる条件分岐も使って、プログラムを簡潔に書くことができる。

```
factorial :: Int -> Int
factorial 0 = 1
factorial n
   | n >= 1 = n * factorial (n - 1)
   | otherwise = abort "factorial called with negative number."
```

プログラム3　階乗のプログラム[19]

allPermutations定義中のmapは第１引数の関数を第２引数のリストの各要素に適用した結果をリストで返す高階関数である。例えば、map ((*) 2) [1..4] は [2,4,6,8]を返す。関数型言語では、このように関数をデータと同じように扱えるので、かなり複雑な機能でも、簡潔に読みやすく書くことができる。関数の合成のための演算子oもあらかじめ定義されている。例えば、odd = not o evenなどということができる。このあたり、手続型言語だと逆立ちしてもできないことが多い。

concatは、別の高階関数foldrを使って、リストの要素を連結して一つのリストにする。ただし、ここではconcatを[[[1,2,3,4] ,[2,1,3,4][2,3,1,4][2,3,4,1]]]の一番外側のカギ括弧を取り去りたいためだけに使っている。

高階関数foldrはリストの各要素に右から順に引数の関数を適用する。例えば、foldr (+) 0 [1..10]は1..10の合計55を返す。

全部の解答を求めるプログラムは以下の通りである。

```
module tamai
import StdEnv, ExtendList

Start = filter isIt (allPermutations [1..9])
where
   isIt x = (sum x >= 0.99999999) && (sum x <= 1.00000001)
   where
      sum x = (addThreeRationals (x!0) (x!1) (x!2)) + (addThreeRationals (x!3) (x!4) (x!5)) +
(addThreeRationals (x!6) (x!7) (x!8))
         where
            addThreeRationals :: Int Int Int -> Real
            addThreeRationals x y z = (toReal x) / (10.0 * (toReal y) + (toReal z))
```

プログラム4　**パズルの答（Concurrent Clean版）**

Concurrent Cleanのプログラムは、左辺のグラフを右辺のグラフで書き換える規則の集まりである。Startを左辺に持つ規則は「スタート・ルール」で、手続型言語で言えば、
メイン・プログラムにあたる。

filterは、第一引数の関数を満たさない要素をリストから削除する高階関数である。ここでは、allPermutations**を使って1..9の順列をすべて生成し、その中でisItを満たすものだけを残す。**

[19]**Concurrent Cleanのマニュアルから引用。**

isItはローカルに定義した関数で、isItの内部でローカルに定義したsumという関数が1.0であればTrueを返す。sumはパズルの左辺の計算をしている。ここでx!iという記法はリストxのi番目の要素（ i = 0.. ）を表す。addThreeRationalsは、sumの中でさらにローカルに定義された関数で、パズルの左辺の一つの項を計算する。

2. 4 Concurrent Cleanの速度

さて、これでプログラムとして正しいはずだが、まともな時間で実行できるかどうか...
昼寝でもして結果を待つ予定だったが、実行は42.16秒で終わった。ガーベージ・コレクションに0.03秒、入出力に0.45秒費やしているので、正味は41.68秒[20]である。何と、CやFortranより速いと豪語するMCLより速かった[21]。

Concurrent Cleanは、全体としてMCLやSmalltalkとほとんど同じ速度で動く。つまり実用になる速度と言って良い。TAK関数[22]などは滅茶苦茶に速い。TAK 24 16 8の場合、MCLやSMalltalkでは数分かかったが、Concurrent Cleanでは０秒 :-) であった。

テトリスやライフゲームや簡単なドローソフトなども、見たところ十分実用的な速度で動いている。

```
Tak::Int Int Int -> Int
Tak x y z | x<=y  =  y
          =  Tak (Tak (dec x) y z)
                 (Tak (dec y) z x)
                 (Tak (dec z) x y)

Start::Int
Start = Tak 24 16 8
```

プログラム5　TAK関数

2. 5 Concurrent Cleanのその他の機能

Concurrent Cleanは通常の関数型言語と同じく、整数・実数・文字・文字列・リスト・タプル・レコード・配列といったデータ型が用意され、プログラマ自身が抽象データ型[23]を定義でき、型や演算子のオーバーロードが定義できる。

また、lazy evaluationを使って無限のリストを操作できる。例えば、すべての素数を求めるプログラムは、次のように書ける。

---

[20]Heap Sizeは3400KB取った。

[21]もちろん、これはすべての順列を全部計算してから結果を求めるのでなく、lazy evaluationによって必要な都度順列を計算して判定している結果である。つまり、玉井さんのプログラムの工夫はlazy evaluation機構が自動的に行っている。

[22]竹内郁男さんの関数型言語いじめのベンチマーク・プログラム。欧米に間違って伝わった計算量の少ないものと、本来のかなり計算量の多いものがある。

[23]「プログラマの本棚」で紹介されている「小林光夫訳。データ型序説。共立出版、１９９０年」を参照して欲しい。

```
module prime
import StdEnv

Start = primenums
where
   primenums :: [Int]
   primenums = map hd (iterate crossout [2..])
   where
      crossout [x:xs] = filter (not o (multiple x)) xs
      where
        multiple x y = divisible y x
        where
           divisible t n = t rem n == 0
```

プログラム6　すべての素数を求めるプログラム[24]

ここで関数crossoutは、[2,3,4,5..]という無限リスト[2..]から、先頭の整数で割り切れる整数を除外する。iterate crossout [2..]は[[3,5,7..],[5,7,11..],[7,11,13..]]というように、それぞれ２で割り切れない整数のリスト、３で割り切れない整数のリスト、５で割り切れない整数のリスト、...のリストを返す。それぞれのリストの先頭を集めてリストとして返すのがmap hd (iterate crossout [2..])、すなわち素数のリストを返すprimenumsである。

もちろんprimenumsは中断しない限り延々と結果を表示し続ける。

2. 6 Standard ML

Concurrent Cleanの唯一の欠点は、コンパイラーのソースコードが付いていないことである。関数型言語のコンパイラーを触ってみたい方は、Standard MLのソースを手に入れた方がよいだろう。Macintosh版は680*0 CPUのコード生成までMLで書いてある。C言語で実装されている部分はごくわずかである。ただ、巨大な処理系なので、マニュアルも関連論文も例題もソースコードもConcurrent Cleanよりかなり多く、ありがたい反面なかなか全体を把握できない。また、FPUを必要とする680*0 CPUのコードを生成するので、PowerMacintoshではSoftwareFPUあるいはPowerFPUといったFPUエミュレータ・ソフトウェアが必要になる。

2. 7 仕様記述言語

プログラマである以上、まず、問題を分析し、モデル化し、仕様を記述してからプログラムを作っているはずである[25]。そうであれば、仕様記述言語が必要になってくる。

僕のデスクトップには、代数仕様記述言語OBJ3がある。実用的な仕様記述言語は、RSL（RAISE Specification Language）やVDM-SL（Vienna Development Method Specification Language）など色々あるのだが、残念ながらMacintosh上にないとか、UNIX上にはあるのだが個人で買うにはやや高いといった理由で、僕のデスクトップにはOBJ3しかない。最新版のCafeOBJがIPAのWWWページから優れたマニュアルと共に入手できるので、参考にして欲しい。CafeOBJをMCLでコンパイルしたMacintosh版も同僚の澤田氏の努力でできた。

[24][24]Concurrent Cleanのマニュアルから引用。

[25]何、行き当たりバッタリだ？それじゃ、僕と同じじゃないですか :-)

OBJ3は68040 Macでは動いていたのだがちょっと実用には苦しい速度だった。しかし、PowerMacintosh用のMCL 3.9でコンパイルすると、突如10倍くらい速くなり、結構実用的な速度で動くようになってきた。

代数仕様記述言語が何であるかは、前述のIPAのマニュアルにも書いてあるが、「プログラマの本棚」の「計算モデルの基礎理論」も参考になる。ともかく、OBJなどを使ってみると、仕様の書き方あるいは心構えが今までと変わってくることは間違いない。

2. 8 コンパイラ支援ツール

プログラマであるからには、自分の言語を作りたくなるだろう。このための道具として、僕のデスクトップにはGNUのFlex（字句解析ツール），Bison（構文解析ツール）とML-Yacc（構文解析ツール），lexgen（MLによる字句解析ツール）がある。関数型言語自体も、構文解析ツールとして使える。実際に、MLにもConcurrent Cleanにも関数型言語のインタプリタの例が付いている。

このようなコンパイラ支援ツールを使えば、「プログラマの本棚」のAho達の本[26]の勉強も捗ろうというものである。

2. 9 コミュニケーション用ツール

プログラマであるからには、プログラミングの息抜きに仲間とおしゃべりするための道具が必要である。インターネット用には電子メール・ニュース・WWWクライアントなどが、パソコン通信用には専用のツールが必要になろう。

電子メールはUNIX上のMHを使っている。EudoraとかCE QuickMailその他の便利な電子メール用ツールはあるのだが、僕のようにあらゆる場所からあらゆるPCを使ってメールを読む必要がある人間には、どうもMH以外適当なものが見あたらない。他の電子メールツールは、使っているPCにメールをダウンロードしてしまうので、別のPCから昔のメールを読もうとするときに困ってしまうのだ。Macだけでも数台持ち、UNIX端末からもメールを読み、時にはDOS/Vマシンからもメールを読む人間にとって、今のところNTTに多額の電話代を払っても、MH以上のメールシステムはない :-<

ニュースはNewsWatcherに日本語パッチが当たったものを使っている。これは、ニュースのホストマシン上の設定ファイルと連動して、どこからニュースを読んでも矛盾が起きないので重宝している。

WWWクライアントはNetscape Nvaigatorを使っている。Smalltalkより重いところが笑える。C++で開発するとこうなるという見本だろう。それでも、世界最大のOSメーカーの製品より圧倒的に速い。大きいと重いというところは、僕にも当てはまる。

パソコン通信用には、日経MIXの場合、友人の諏訪君がC++で開発したCoMIXを使っている。これも、著者本人の性質を継承して大きくて重いが、日経MIXへの接続からオフライン編集からファイルのダウンロードまで一手に引き受けてくれるので快適である。

一方、NIfty-ServeはComNiftyは接続とファイルのダウンロードしかしてくれないので、補助ソフトと連動させなければならないし、最近、なぜかメールが読めなくなったので困っている。そこで、NIfty-Serveから配布されているNIFTY Managerも使うのだが、見てくれだけ綺麗で中身が貧弱なので使うたびにイライラする。

2. 10 ドキュメント作成ツール

プログラマである以上、ドキュメントは自主的に品質の良いものを書くであろう。このとき、ワープロなどは使いものにならない[27]。特にMS*というワープロなどは絶対に駄目である[28]。ワープロは高々数ページしか作成しな

[26]原田賢一訳。コンパイラ原理・技法・ツールⅠ,II。サイエンス社、１９９０年

[27]Nibai Word、MS-WORD、FlushWriter、MacWord、QuarkExpress、クラリスワークス、CrarisWorksの正

いような前提のものが多いからである。少なくともA4版で10ページ、普通は100ページ以上、役所の場合は1000ページ以上ものドキュメントを要求されるプログラマならば、まともなドキュメント作成・処理ツールを持っていなければならない。

僕のデスクトップにはTeXとFrameMakerがある。いずれも、1000ページ以上のドキュメントを書き、索引や目次を付け、途中に章や節を挿入しても大丈夫だし、後から統一的に書式を変えることができる。この原稿のように小規模なドキュメントの場合はNisus Writerを使っている。

他人が作ったドキュメントを見たり印刷するにはAcrobatｪ　Readerというプログラムを使っている。商品版はPostScriptのドキュメントも見たり部分印刷できる。 Concurren CleanやMLのドキュメントはこれで印刷している。

2. 11 CASEツール

プログラマだけでは糊口をしのげないので、コンサルタントなどもやっている。そうすると、近頃はやりのオブジェクト指向CASEも必要になる。OOA ToolとTurbo CASEを持っている。どちらも、まあ、使える。

また、Windows版のRational/ROSEの評価版をMacintosh上で動かしている[29]。BoochやRumbaughやJacobsonらの作っているUML（Unified　Method　Language[30]）というオブジェクト指向記法を使いたいからである。

ClarisDrawという図形描画ツールも、結構ダイアグラムなどを書くのに使える。が、最近は仕様記述言語を使う方が多い。いくらダイアグラムを書いても、肝心な仕様記述が曖昧では役に立たないからである。しかし、まあ、分析や設計ができてからCASEツールに入力して、コミュニケーションの道具として使うのには役立つ[31]。

2. 12 辞書あるいは自然言語処理あるいは百科事典

プログラマである以上、世界情勢に明るくなくてはならない。そうしないと、この国ではいつ役所に殺されるか分からないし、亡命先も考えておかなければならないからである[32]。　僕のデスクトップには書見台というフリーソフトがあり、これを使って、広辞苑・研究社英和辞書・和英辞書・昭和データベース・知恵蔵・仏教辞典などに瞬時にアクセスできる。内部メモリーに知識がないので、外部メモリーに頼っているわけである。また、グローリア百科事典もあるので欧米の知識には強い。コリャ英和もあるので、いい加減な翻訳なら速いし、にやりと笑える効用もある。

ATLASMATEやUSA　Atlasという地図ソフトもあるので、亡命潜伏先やプログラミングに篭る場所の選定にも苦労しない。が、米国の探偵小説の地名や道路名から、探偵の行動範囲をイメージする作業に時間を取られ過ぎることもある[33]。さらに、その町の上空をフライト・シミュレータ[34]で飛ぶと、寝不足になる。

規ユーザーであり、vi、emacs、TeX、troffや、もっとひどいワープロやテキストエディタを使ったこともある、僕の結論に異議を唱えるにはかなりの財力を必要とする :-)

[28]競合製品を打ち落としてしまうアプリケーションは初めて見た。

[29]Soft WindowsというWindowsのシミュレータでなんとか動いている。

[30]3氏の方法論を混ぜ合わせただけで、どこがUnifiedなのか僕にはちっとも理解できないが...

[31]コンサルタントとしての権威付けにも役立っているかもしれない。山崎さんと違って、下手な字で資料を書いても相手を説得できる力量がないからでもある。しかし、日本人は形が綺麗なのは好むが、中身の評価は...

[32]そうならないために市長になろうと言う人にはSimCity 2000がお勧めである。無理矢理都市計画をしても思い通りにならないことが体験できる。青島都知事に必須のツールであろう。

[33]しかし、米国に住むのは危ない。住所か電話番号を調べれば、すぐにStreetまでこのソフトで確定できてしまう。ピザ百人前を、頼んでもいない友人宅に届けるなんてのは朝飯前である。その点、日本のソフトはできが悪く、

2. 13 シミュレーション・ツール

プログラマでなくても、何かのモデルを作ってシミュレーションで確認する必要があるだろう。このような道具として、オブジェクト指向シミュレーション・ツールEX･TDとダイナミック・プログラミング用シミュレーション言語をMacintoshで実現したStellaがある。

いずれも、強力なシミュレーション用ツールで、特にEX･TDは生産計画用のライブラリが付いていて、工場のラインをどういう構成にすれば効率が良いかといったことを簡単にモデル化できる。しかし、日本の工場ではベルト・コンベアーを実際に配置してから考えるようだし、僕も、最寄りの駅前のファーストフード店の待ち行列を考察するくらいしか「問題」を持ち合わせていないので、宝の持ち腐れになっている。

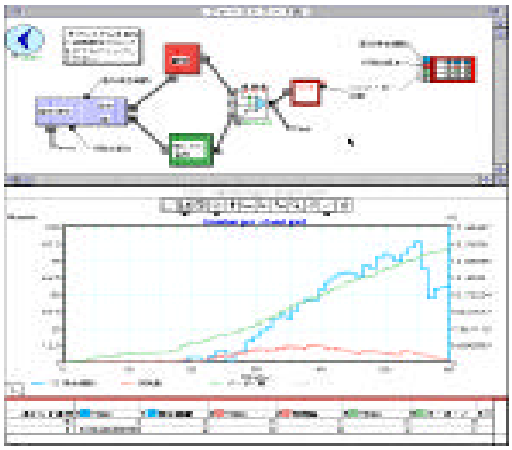

図 2　ファースフード店の待ち行列分析例

3章 とりあえずのおわりに

ここまでで僕のデスクトップの百分の一くらいは紹介できたと思うが、このまま続けると、無限リストになりそうなので、ここらあたりで止めておく。

いずれにせよ、自分のデスクトップにこの程度のソフトウェアを揃えておいて、「プログラマの本棚」の本を読み、それをデスクトップ上で動かして見るといったことをやれば、昔よりは簡単に、プロサッカーで言えばサテライトの選手くらいにはなることができるだろう。僕自身は、まだ草サッカー選手程度であることが最近判明してきたので、デスクトップのさらなる増強を図っているため、大蔵大臣に監視されている。

このようなことはできない。と書いていたら、日本でも地図ソフトを悪用した空き巣が捕まったという話が新聞に載った。SEAMIALの編集長が原稿を貯め込んでいる間に、日本のソフトも進化したようである :-)

[34]フライト・シミュレータは５機ある。Apple II版の白黒・カラー各１機とPC 98版とMac版の白黒・カラー各１機である。