7M-6

MELCOM　PSI上での ネットワーク・アプリケーションの開発手法

多々良　浩司*　長谷川　隆之*
*三菱電機（株）

1　はじめに

現在、MELCOM　PSIシリーズ（以下PSIと略す）は、研究開発ツールとして広く用いられている。PSIの特長はPrologにオブジェクト指向の概念を導入した言語であるESPを高速に実行でき、また開発に適したさまざまな機能をサポートしていることである。これらの機能およびESPの特徴を生かして効率良くネットワークアプリケーションを開発する手法について発表する。特に、特徴的な「状態遷移の記述」、「接続試験の方法」について重点的に説明する。

2　全体の構成

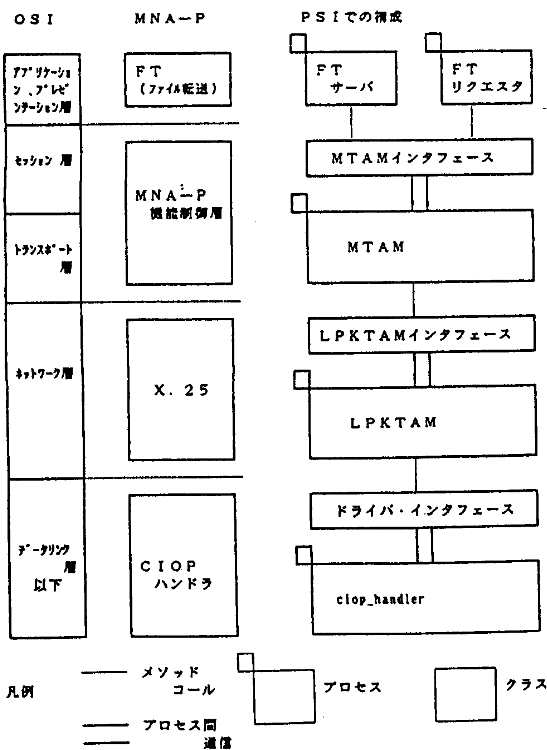

図　1　構成

今回PSI上で開発したのは、Multi_shared Network Architecture Packet（以下MNA−Pと略す）というプロトコルを用いたファイル転送システム他である。図1はその構成である。参考のためにOSIのモデルを併記した。

2．1　各レイヤ間のインタフェース

図1に示すようにMNA−Pを、レイヤごとにプロセスに分けて構成した。これは今後の拡張を容易にするためである。各プロセスは、上位に機能を提供するためのメソッドを定義したクラス（図1のMTAMインタフェース 等）と状態遷移に基づいた処理を行う部分（図1のMTAM等）に分けられる。MTAMとMTAMインタフェースはOSが提供するプロセス間通信の機能によりインタフェースがとられている。

2．2　状態遷移の記述

プロトコルを記述するためには、状態遷移表は欠かせない。PSI上で開発するにあたり、ESPの特徴を生かし、継承の機能を利用して状態遷移表をできるだけそのままのイメージでプログラムに記述するようにした。

この方法により次のようなメリットが発生する。

(1) プログラムが見やすく保守しやすい。

(2) 遷移表中のひとつのセル（ます）の処理がひとつのメソッドとして定義されるためデバッグが容易になる。

状態 / 要因	状態 1	状態 2	・・・・	状態 N
要因 1	処理 11	処理 12		処理 1N
要因 2	処理 21	処理 22		処理 2N
：：：：：：				
要因 n	処理 n1	処理 n2		処理 nN

図　2　状態遷移

A development method of Network Apprication on MELCOM PSI

Hiroshi Tatara*　Takayuki Hasegawa*
*MITSUBISHI ELECTRIC CORPORATION

```
class 状態遷移  has
nature
      with_状態遷移１、
      with_状態遷移２、
          :
      with_状態遷移N；

instance
：処理１１（）:-
：処理１２（）:-
        :
        :
        :
：処理ｎN（）:-

end.


class  with_状態遷移１ has
instance
：要因１（）:-
    ：処理１１（） ；
：要因２（）:-
    ：処理２１（） ；
        :
        :
：要因ｎ（）:-
    ：処理ｎ１（） ；

end.


class  with_状態遷移２ has
instance
：要因１（）:-
    ：処理２１（） ；
：要因２（）:-
    ：処理２２（） ；
        :
        :
：要因ｎ（）:-
    ：処理ｎ２（） ；

end.
```

図　３　　記述例

図３は図２の状態遷移をプログラムに記述した例である。各状態をひとつのクラス（ with_状態遷移ＸＸというクラス）に定義し遷移要因（要因１、要因２など）をメソッドとして定義している。またそれらのクラスを継承したクラス（状態遷移　というクラス）の中ですべての処理のためのメソッドを定義している。

２．３　試験、デバッグの方法

試験は３つの段階に分けて行なった。

すなわち、各モジュールごとに行なうモジュールテスト、プログラムとして各モジュールを結合して行なうプログラムテスト、及びシステムテストである。

```
class example_class has

  nature
      example_class_debug :
            :
            :
  instance
            :
            :
      :example(Obj, Arg1, Arg2):-      % ターゲットになるメソッド
            :
            :
end.
```

% ターゲットになるクラス

```
class example_class_debug has
            :
            :
  instance
      before:example(Obj, Arg1, Arg2):-
              :putt(Window, :example(Obj, Arg1, Arg2)),
              :putl(Window, "");
      after :example(Obj, Arg1, Arg2):-
              :putt(Window, :example(Obj, Arg1, Arg2)),
              :putl(Window, "");
```

図４

（１）　モジュールテスト

下位のモジュールよりテストを行なった。上位からメソッドを呼び出すときに与える引数はＯＳが提供するデバッガから与えられるので、とくにテスト用のドライバを作る必要はなかった。

（２）　プログラム、システムテスト

この段階になると実際に他の計算機と接続してテストをすることになる。従って、応答待ちのタイマ等が動くためデバッガによって１ステップずつ追ってゆくような方法は不可能である。（タイムアウトが発生するため）

そこで今回は、ＥＳＰの特長を生かし、前デーモン述語、後デーモン述語を利用して主要なメソッドのすべての引数をメモリ上に記録することにした。

＊前デーモン述語、後デーモン述語
　　… 通常の主処理述語（メソッド）の前処理、後処理を定義するもの。

図４はコーディング例である。クラスexample_classの:example/3をターゲットのメソッドとする。デバッグ用のクラスexample_class_debug の中にこのメソッドの前デーモン、後デーモンを定義する。example_class はデバッグ用のクラスを継承する。

メソッド:example/3が呼ばれると、まず前デーモンが実行され主処理述語が行なわれる前の引数の内容が記録される。次に主処理述語が実行され、引数の内容が変わる。その後、後デーモン述語が実行され引数の内容が記録される。

こうしてメモリ上に記録したものを必要に応じてプリンタ、ウィンドウ等に出力すればよい。これらのデータがデバッグに役にたった。

３　おわりに

以上のような手法を用いることにより、短期間で効率良く開発、試験を進めることができた。

また、できあがったプログラムソースも大変見やすく保守性がよい。

ネットワークアプリケーション以外にもここに述べた手法の応用が考えられるが、これらは今後の課題である。

本開発に際し、御協力いただきました（株）ｱｰﾃｨﾌｨｯｼｬﾙ･ｲﾝﾃﾘｼﾞｪﾝｽの皆様に感謝いたします。