ネットワークソフトウェアの構成法に関する研究 ～ IPaf-R^2 の実現 ～

南山大学経営学部情報管理学科
97B520 服部 賢人 97B552 前吉 智之
97B555 水野 正樹 97B558 森 宰補
指導教員 野呂 昌満

1 はじめに

変更に強く，構造の整理されたネットワークソフトウェアを開発するさいに，よいアーキテクチャを提案することは重要である．アーキテクチャを提案しアーキテクチャにもとづいてフレームワークを作成することで，開発者の労力を軽減し生産性を向上することができる． これまでの研究で参照アーキテクチャ IParch-R[3] が提案され，応用フレームワーク IPaf-R が設計・実現された．研究は以下の手順で進められている．

1. 参照アーキテクチャの提案
2. 応用フレームワークの設計・実現
3. 応用フレームワークを用い，ネットワークソフトウェアを試作
4. 作成した応用フレームワークを評価

研究は上記の手順を繰り返しており， IPaf-R の評価のあと改版した参照アーキテクチャ IParch-R^2 が提案され，応用フレームワーク IPaf-R^2 の設計がなされた．

以前に提案されたアーキテクチャである IParch-R の概略を以下に示す (図 1参照)．

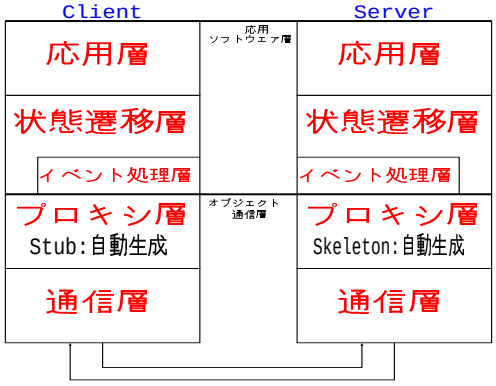

図 1: IParch-R

IParch-R は構造の整理を目的に提案されたネットワークソフトウェアのアーキテクチャである． 応用層ではクライアント側は使用者インターフェース，サーバ側はデータベースアクセスを実現する．状態遷移層は状態の遷移に関する記述を応用の論理から分離して実現する．プロキシ層はデータ通信規約を遠隔オブジェクトへのメッセージ通信として実現する． 通信層は OSI 参照モデルの 4 層までを実現する．

IParch-R には
- 入力の多重化が実現されていない．
- 排他制御が実現されていない．

という問題がある．

本研究の目的は以上の問題を解決し，IPaf-R^2 を実現することである． 設計をもとに IPaf-R^2 をオブジェクト指向言語 C ++を用いて実現した． クライアント側は使用者からの入力とサーバからの結果を監視する Selector クラスを作成することで入力の多重化を実現した．サーバ側は応用層に排他制御を実現するクラスを作成することで排他制御を実現した．

作成した IPaf-R^2 を用いてネットワークソフトウェアを試作し，その有用性を確認した．

2 IParch-R^2, IPaf-R^2

本研究のアーキテクチャ IParch-R^2, フレームワーク IPaf-R^2 の設計を以下に示す．

2.1 設計指針

アーキテクチャ IParch-R^2 の設計指針を示す．

- オブジェクト指向アーキテクチャにする．
- データ通信規約と応用の論理を分離する．
- データ通信規約では，オブジェクト指向インタフェースの記述と通信規約の記述を分離する．
- 応用の論理では，応用の論理と状態遷移機械を分離する．
- クライアント側では使用者からの入力に対応する処理と，サーバからの結果に対応する処理を分離する．

我々はこれらの指針に最も適合するモデルとして，階層モデルを選択した．

IParch-R, IPaf-R の問題点

IParch-R, IPaf-R の問題点を以下に示す．

- イベント処理層について詳細な議論がされていない．
- 入力の多重化が実現されていない．
- IParch-R はクライアントだけの研究であるので，サーバの互換性が保証されていない．
- 排他制御の構築に関して考慮されていない．

クライアントの問題とサーバの問題に分け，それぞれについて以下に考察する．

2.2 IParch-R^2 の概略

IParch-R^2 の概略を以下に示す．

クライアント: イベント処理層

クライアントは使用者からの入力と，サーバからの結果の両方に対する処理を行なわなくてはならない．IParch-R では使用者からの入力に対する処理を応用層，サーバからの結果に対する処理をイベント処理層で行なっていた．しかしイベント処理層を配置すると，サーバからの結果を処理するだけの層を作ることになりアーキテクチャが複雑なものになる．

我々はサーバからの結果に対応する処理を応用層で行なう．応用層でサーバからの結果に対応する処理と，使用者からの入力に対する処理の両方を行なうことでイベント処理層を削除できる．イベント処理層を削除することで構造の整理されたアーキテクチャになる．

設計指針と以上の考えにもとづいて提案されたクライアント側のアーキテクチャを以下に示す．

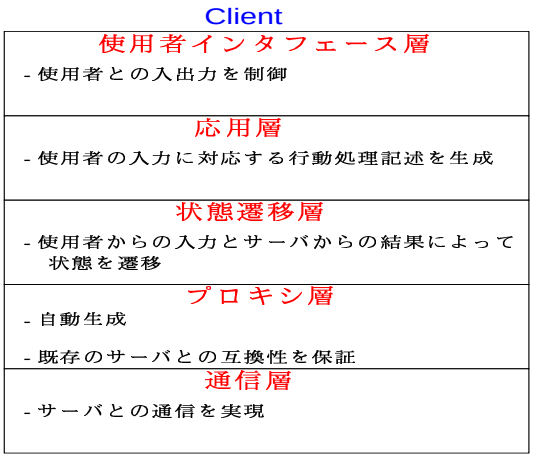

図 2: クライアントのアーキテクチャ

使用者インタフェース層，応用層，状態遷移層は OSI 基本参照モデルの応用層に相当する． プロキシ層はプレゼンテーション層，セション層に相当する．通信層はトランスポート層，ネットワーク層，データリンク層，物理層に相当する．

サーバ: 互換性の保証

IParch-R はクライアントだけの研究であるので，サーバの互換性が保証されていない．我々はサーバの互換性を保証し既存のネットワークソフトウェアと通信することを可能にする．IParch-R^2 では通信プロトコルに関するコードを実現するプロキシ層で互換性の保証を行なう．

設計指針と以上の考えにもとづいて提案されたサーバ側のアーキテクチャを以下に示す．

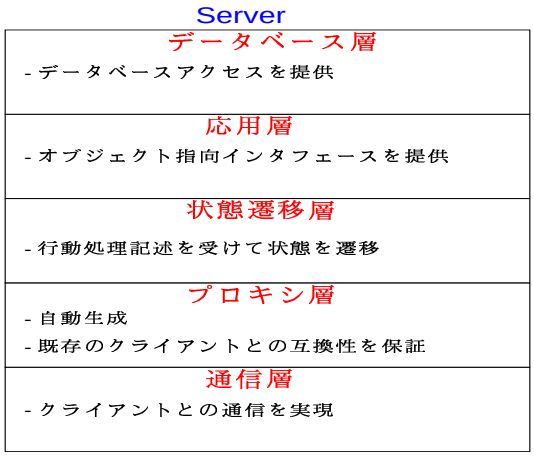

図 3: サーバのアーキテクチャ

データベース層，応用層，状態遷移層は OSI 基本参照モデルの応用層に相当する． プロキシ層，通信層はクライアントと同様である．

2.3 IPaf-R^2

アーキテクチャにもとづいてフレームワークを設計する．以下にフレームワークの設計指針を示す．

- アーキテクチャの構成を素直に反映する．
- 通信層はオブジェクト指向ソケットを提供する．
- プロキシ層はサーバの応用層の記述から自動生成し，既存プロトコルとの互換性を確保する．
- 設計パターン [4] を使用する．
- オブジェクト指向の概念を素直に用いる．メソッド再定義機能を利用する．

クライアント

IPaf-R では入力の多重化が実現されていないので，サーバからのメッセージを受信している間，使用者からの入力を受け付けることができない．例えば，サーバからデータを受信している間に中止命令を送りたい場合，使用者からの入力を受け付けることができない．このような状況を考慮すると入力の多重化を実現することが重要である．入力の多重化を実現する方法を挙げる．

1. プロセスを複数にする．

   使用者からの入力と， サーバからの入力を別々のプロセスで受け付けることによって入力の多重化を実現する．

2. ファイルディスクリプタを監視する．

   使用者からの入力と，サーバからの入力のファイルディスクリプタを監視する．

入力があった場合それぞれの処理を行なうことによって入力の多重化を実現する.

我々は以下の理由により, 2 の方法を選択する.

- UNIX 上でネットワークソフトウェアを開発する場合, クライアントソフトウェアを 1 ヘビーウエイトプロセスで実現するのは最も一般的であると考える.
- 1 の方法を用いる場合, プロセス間通信を行なう必要があり処理が複雑になる. また, ホストへの負担は増加することになる.

2 の方法を用いれば, プロセスを複数にせず入力の多重化を実現できる. 使用者からの入力と, サーバからの入力のファイルディスクリプタを監視する Selector クラスを配置することでアーキテクチャを素直に反映したフレームワークを提供できる.

アーキテクチャと設計指針にもとづいて設計したクライアントのフレームワークをクラス図を用いて説明する.

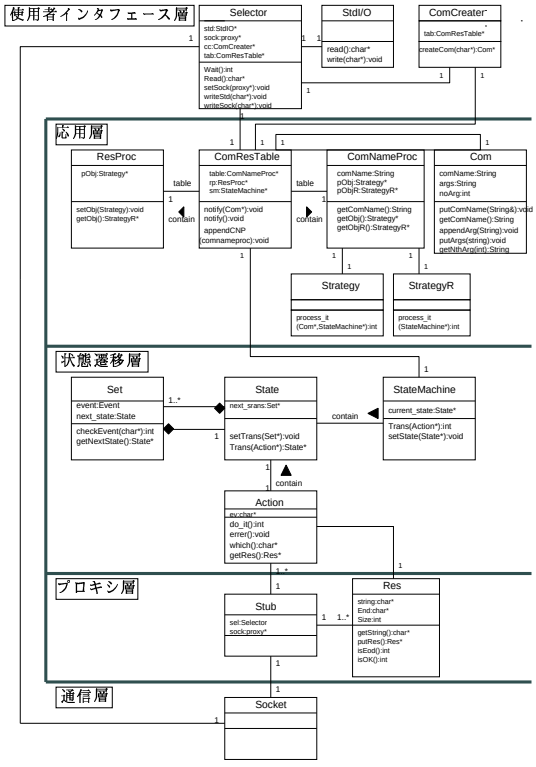

図 4: クライアントのクラス図

- 使用者インタフェース& 通信層
  - StdI/O クラスは使用者インタフェースを実現する.
  - Selector クラスは使用者からの入力, サーバからの結果を監視する.
  - Com クラスはコマンド名と引数を保持する.
  - ComCreater クラスは使用者からの入力を受けとり, Com オブジェクトを生成する.
  - Socket クラスはデータの送受信を行なう.
- 応用層
  - ComResTable クラスはコマンド名とサーバからの結果から, 対応する処理を呼び出す.
  - ComNameProc クラスはコマンドに対応する Action クラスのインスタンスを生成する.
  - ResProc クラスはサーバからの結果に対応する Action クラスのインスタンスを生成する.
  - Strategy クラスはコマンドに対応する処理を保持する.
  - StrategyR クラスはサーバからの結果に対応する処理を保持する.
- 状態遷移層
  - StateMachine クラスは現在の状態を表す State クラスのインスタンスを保持する.
  - State クラスは次にどの状態に遷移するかを知っている Set クラスを保持する.
  - Set クラスはイベントと次の状態の対を保持する.
  - Action クラスはコマンドの実行と結果に関する処理を保持する.
- プロキシ層
  - Stub クラスは各コマンドに対応するリクエスト文を作成し, サーバからの結果を読み込む.
  - Res クラスはサーバからの結果を保持する.

クライアントの可変部 (Hot Spot) を以下に示す.

- ComCreater クラス
- ComResTable クラス
- Action クラス
- Strategy クラス
- StrategyR クラス
- State クラス
- StateMachine クラス

これらのクラスを作成することでネットワークソフトウェアを実現できる.

サーバ

IParch-R は排他制御の構築に関して考慮されていない. サーバはデータベースの保護のために排他制御を実現しなくてはならない.

IParch-R^2 で排他制御を行う方法を挙げる.

1. 排他制御を行動処理記述の中に記述

   行動処理記述を実行するさいに排他制御を行ない, データベースにアクセスしている間, 他のプロセスをブロックし, アクセスが終了したらブロックを解除する.

2. 排他制御を実現するクラスを応用層に作成

応用層が排他制御をして，データベースにアクセスしている間，他のプロセスをブロックし，アクセスが終了したらブロックを解除する．

1 の場合，行動処理記述のコードと排他制御のコードが混在することになり，開発者にとって理解しにくいものになる．我々は排他制御を実現する Exclude クラスを応用層に作成する．

アーキテクチャと設計指針にもとづいて設計したサーバのフレームワークをクラス図を用いて説明する．

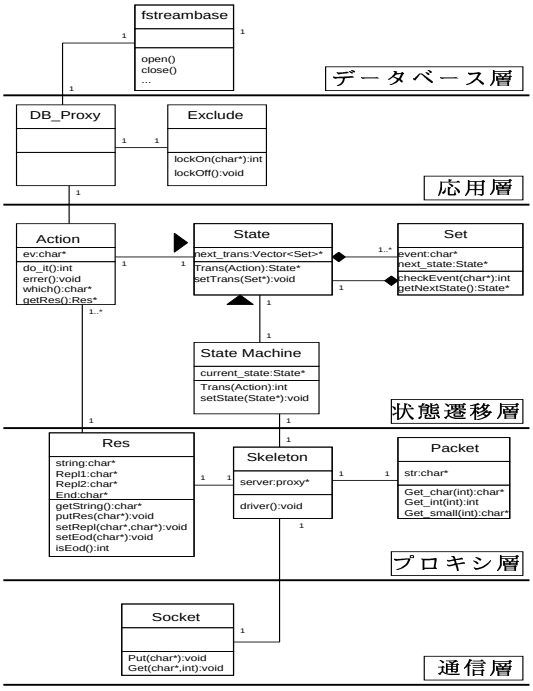

図 5: サーバのクラス図

- データベース層
  - fstreambase クラスはデータベースアクセスを提供する．
- 応用層
  - DB_Proxy クラスはオブジェクト指向インタフェースを提供する．
  - Exclude クラスはクライアントからの要求をブロックして、排他制御を行なう．データベースから結果が返ってきたら，ブロックを解除する．
- プロキシ層
  - Skeleton クラスは Action クラスのインスタンスを生成して，StateMachine クラスにインスタンスを渡す．
  - Res クラスは応用層からの結果を保持する．
  - Packet クラスはクライアントから送られてきたリクエスト文を保持する．
- 状態遷移層，通信層はクライアントと同様．

サーバの可変部 (Hot Spot) を以下に示す．

- Exclude クラス
- DB_Proxy クラス
- Action クラス
- State クラス
- StateMachine クラス

これらのクラスを作成することでネットワークソフトウェアを実現できる．

IPaf-R^2 は

- 上位 3 層はフレームワーク
- プロキシ層は自動生成
- 通信層はオブジェクト指向ソケットライブラリ

として提供する．

3 IPaf-R^2 の実現

我々は UNIX 上でオブジェクト指向言語である C++ を用いてフレームワークを作成した．フレームワークを実現するさいの考察を以下に示す．

3.1 クライアント

以下に Selector クラスの実現方法を説明する．

Selector クラスの実現

Selector クラスには使用者とサーバの入力を監視する Wait メソッドとサーバからのメッセージの受信のさいに呼ぶ Read メソッドがある．それぞれのメソッドに select システムコールを記述し，使用者とサーバからの入力のファイルディスクプリタを監視する．以下に Wait メソッドの処理の流れをシーケンス図を用いて示す (図 6参照)．

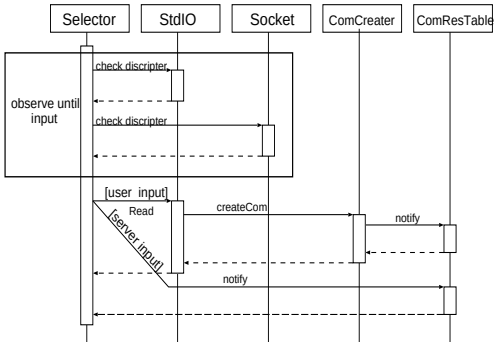

図 6: シーケンス図

Wait メソッドの select システムコールは StdIO クラス，

またはSocketクラスに入力があるまで監視を続ける．使用者からの入力を検知すると，StdIOクラスのReadメソッドを呼び，入力文字列を受けとる．入力文字列を引数としてComCreaterクラスに渡す． サーバからの入力を検知するとComResTableクラスのnotifyメソッドを呼ぶ．

ReadメソッドはStubクラスがサーバからのメッセージの受信のさいに呼ばれる． サーバからの入力を検知している間，Socketクラスからメッセージを受け取る．selectシステムコールで使用者とサーバの入力を監視しているので，メッセージを受信している間に使用者からの入力を受け付けることができる．

3.2 サーバ

我々は排他制御にロックファイルとセマフォを使う方法を比較，検討する．

- セマフォはセマフォ値を持ち， P動作とV動作によってセマフォ値を増減させることでプロセス間の同期をとり，プロセスを制御する．
- ロックファイルはあるプロセスがロックファイルの有無を確認し，ロックファイルがあればプロセスを受け付けない．ロックファイルがなければロックファイルを作成し，データにアクセスする．

排他制御を実現するExcludeクラスは可変部である．よって開発者がExcludeクラスを作成しやすいような排他制御の方法を提案しなければならない．

UNIX上ではデータベースはファイル群である． ファイルごとにプロセスを制御するロックファイルはUNIX上で排他制御を行なう方法として一般的である． セマフォを用いて排他制御を行なうことも可能である． セマフォは複雑な処理を行なうことができるが，UNIX上ではロックファイルを用いれば容易に排他制御を行なうことができる．我々は簡単な操作で排他制御を行ないたいと考えているので，ロックファイルを排他制御の方法として選択した．

Excludeクラスの実現

排他制御を実現するExcludeクラスを説明をする．

Excludeクラスはメソッド lockOn, lockOff を持つ．ロックファイルを作成したいときにlockOnメソッドを呼ぶ．mktempシステムコールを使い一時的なファイルを作成する． 次にlinkシステムコールで一時的なファイルにリンクして，データ固有の新しいファイル名をつける． このデータ固有のファイルがロックファイルとなる． ロックファイルを削除したいときはlockOffメソッドを呼ぶ．unlinkシステムコールを使い，ロックファイルを削除する．

3.3 自動生成系の拡張

ネットワークソフトウェアを作成する場合，通信規約の実現に多くの労力がかかる．IPaf-R^{2}ではプロキシ層のオブジェクトであるStub, Skeletonクラスで通信規約を実現する．これらのクラスをフレームワークとして提供することは，フレームワークの可変部(Hot Spot)が多くなりすぎる．我々はStub, Skeletonクラスを自動生成することで開発者の労力を軽減する．

IPaf-Rは自動生成する仕様に，

(1) コマンド名とその引数
(2) データの終端条件
(3) データ中の置換される文字列

が必要であるとされており，C++のクラスの記述として与えることでStubクラスの記述を自動生成した．しかしIPaf-Rはクライアントだけの研究である． IPaf-R^{2}ではSkeletonクラスを自動生成するためにプロトコルを再調査した． 結果，Stub, Skeletonクラスの記述の自動生成には上記の条件に加えて，

(4) コマンドを処理した後，サーバが接続を切断するかどうか

の条件を加える必要があることがわかった．

Resクラス，Actionクラスを変更すれば，今までの入力からStub, Skeletonクラスの記述を自動生成できるが，その場合，開発者はプロトコルを意識しながらResクラス，Actionクラスを作成しなければならない． 我々は開発者の労力を軽減するために，入力の仕様を変更し自動生成系を拡張する方式をとる．

以下に上記の(1)〜(4)にしたがった自動生成系の入力の仕様の一部をC++のクラス記述で表現する．括弧の数字は上の括弧の数字に一致する．"connect"はサーバがコマンドに対応した処理を行なった後，接続を切断する場合に0を記述する．コマンドに対応した処理を行なった後もクライアントとセッションを続ける場合は1を記述する．

```
class BBP {
 Res *READ(void)                              (1)
      { int connect = 1;
        char format[] = "READ \n";
        char Eod[] = "\r\n.\r\n";              (2)
        char *replace[2] = {"\n..","\n."};(3)
      };
 Res *LOGIN(char* user,char* pass)
      { int connect = 1;
        char format[] = "LOGIN %s %s\n";
      };
 Res *QUIT(void)
      { int connect = 0;                       (4)
```

```
        char format[] = "QUIT \n";
      };
                  :
                  :
};
```

4 ネットワークソフトウェアの試作

フレームワークの有用性を確認するために我々は既存のプロトコルにしたがったネットワークソフトウェアではなく，掲示板をモデルとする独自のプロトコル(Bulletinboard Protocol:BBP) を考案しネットワークソフトウェアを試作した.

4.1 BBP

BBP は掲示板をモデルとしたプロトコルである．以下にBBP の操作，コマンド，セッションの例，クライアントとサーバの状態遷移について説明する.

操作

クライアントはサーバに対して接続の要求を行なう．接続が確立すれば切断されるまで，クライアントとサーバはコマンドと応答をやりとりする．初めにクライアントはユーザ認証を行なう．認証が成功すると，掲示板に掲載されている文章の一覧を受けとり，文章の番号を指定し文章を受けとる.

コマンド

クライアントはコマンドにパラメータをつけサーバに送信する．サーバはコマンドを正常に受け付けると OK で始まる応答を返す．コマンドを正常に受け付けることができない場合は ERR で始まる応答をクライアントに返す.

- LOGIN コマンドはパラメータにユーザ名とパスワードを記述し，ユーザ認証を行なう.
- LIST コマンドは掲示板に掲載されている文章の一覧を受けとる.
- READ コマンドはパラメータに LIST で得た番号を記述し，文章を受けとる.
- QUIT コマンドは通信を終了する.

BBP セッションの例

クライアントを C，サーバを S として以下にセッションの例を示す.

```
S:(接続を待っている)
C:(接続する)
S:   OK service ready
C:   LOGIN 97bxxx password
S:   OK login ok
C:   LIST
S:   1 What's new?
S:   2 News
S:   .
C:   READ 1
S:   OK
S:   (サーバが 1 つ目のメッセージを送る)
S:   .
C:   QUIT
S:   OK close connection
C:   (接続を閉じる)
S:   (次の接続を待つ)
```

クライアントの状態遷移

初期状態から接続要求をサーバに行なうと接続中状態に遷移する．接続が成功すると認証待ち状態になり，失敗すると初期状態に戻る．認証待ち状態ではサーバにユーザ名とパスワードを送信すると認証中状態に遷移する．認証が成功するとコマンド待ち状態に遷移し，失敗すると認証待ち状態に戻る．コマンド待ち状態では LIST, READ を受けつけることができ，サーバに送信すると実行中状態に遷移する．サーバから結果を受信するとコマンド待ち状態に戻る．初期状態，認証待ち状態，コマンド待ち状態で QUIT コマンドを受けつけ，終了状態に遷移する．図 7に状態遷移図を示す.

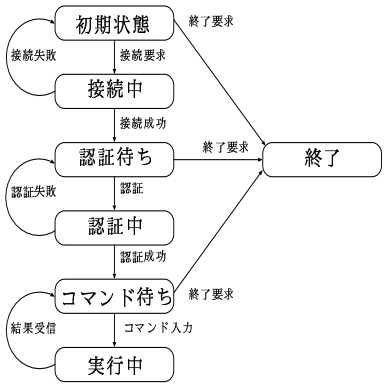

図 7: クライアントの状態遷移図

サーバの状態遷移

接続待ち状態でクライアントから接続要求を受けつけ，成功すると認証待ち状態に遷移する．失敗すると接続待ち状態に戻る．認証待ち状態ではクライアントからの認証要求を待ち，認証が成功すると要求待ち状態に遷移する．失敗すると認証待ち状態に戻る．要求待ち状態ではクライアントからの要求を受け，応答をクライアントに返す．認証待ち状態，要求待ち状態で QUIT コマンドを受けつけることができる．図 8に状態遷移図を示す.

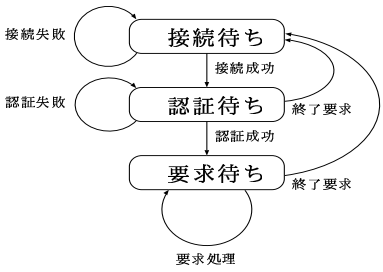

図 8: サーバの状態遷移図

4.2 BBP クライアント

BBP クライアントを試作するさいの主要な可変部を説明する.

使用者インタフェース

クライアントは使用者から入力を受け付けるさいに,
1 接続 2 ユーザ認証 3 リスト 4 閲覧 5 終了
と表示し, 例えばユーザ認証をするのであれば
2 *username password*
と入力を受け付ける. 使用者の入力は Selector クラスで StdIO クラスの Read メソッドを呼ぶことで受けとり, 入力文字列を ComCreater クラスに渡す. ComCreater クラスは先頭の番号で要求を判断し, 後に続く引数を取り出す. 要求に対応したコマンド名と取り出した引数を Com オブジェクトにセットし, 応用層に渡す.

アクション

コマンドは LOGIN, LIST, READ, QUIT の 4 つがあり, サーバへの接続要求も 1 つのアクションとして考えたのでサーバへ要求を出すさいのアクションは全部で 5 つになる. それぞれの要求に対してサーバからの返答があり, 返答に対してアクションが起こる. サーバからの返答に対してもアクションが 5 つある. Action クラスのサブクラスとして, サーバへ要求を出す時のアクションである ConnectAction, LoginAction, ListAction, ReadAction, QuitAction とサーバからの返答に対してのアクションである ConnectRAction, LoginRAction, ListRAction, ReadRAction, QuitRAction を記述した. Action クラスの do_it が実行されるとそれぞれ対応する Stub クラスのメソッドが呼ばれる. 例えば ReadAction クラスの do_it が実行されると Stub クラスの read メソッドが呼ばれ, ReadRAction クラスの do_it が実行されると Stub クラスの readR メソッドが呼ばれる. 以下に ReadAction クラス, ReadRAction クラスのコードを示す.

```
class ReadAction{
  ...
 public:
  ...
  int do_it(){
   ...
   stub->read(num);
   ...
  }
};
```

```
class ReadRAction{
  ...
 public:
  ...
  int do_it(){
   ...
   Res = stub->readR();
   ...
  }
};
```

状態

クライアントの状態は状態遷移図より初期状態, 接続中状態, 認証待ち状態, 認証中状態, コマンド待ち状態, 実行中状態の 6 つを作成しなければならない. 状態は状態の数だけ State のインスタンスを作成し, StateMachine クラスのコンストラクタにイベントと次の遷移先を記述することで状態遷移表を実現した. 以下に StateMachine クラスのコンストラクタのコードを示す.

```
BbpStateMachine():StateMachine(){

  State* auth = new BbpState();
  State* auth_ev = new BbpState();
         :

  auth->setTrans(new Set("login", auth_ev));
  auth->setTrans(new Set("quit", quit));

  auth_ev->setTrans(new Set("loginr", trnsct));
  auth_ev->setTrans(new Set("error", auth));
         :
 }
```

4.3 BBP サーバ

BBP サーバを試作するさいの主要な可変部を説明する.

アクション

クライアントからの要求に対応してアクションが生成される. クライアントからは接続要求と LOGIN, LIST, READ, QUIT コマンドを含むリクエスト文が送られてくる. サーバはそれらに対応して Action クラスのサブクラスとして ConnectAction, LoginAction, ListAction, ReadAction, QuitAction を記述した. Action クラスの do_it が実行されるとそれぞれ対応する DB_Proxy クラスのメソッドが呼ばれる. 例えば ReadAction クラスの do_it が実行されると DB_Proxy クラスの read メソッドが呼ばれる.

状態

クライアントと同様にサーバも状態の数だけインスタンスを生成することで状態遷移を実現した．状態は接続待ち状態，認証待ち状態，要求待ち状態の 3 つである．

5 IPaf-R^2 の有用性についての考察

我々は IPaf-R^2 を用いて BBP ソフトウェアを試作した．IPaf-R^2 を理解しているので短時間でソフトウェアを試作でき，正常に動作することも確認した．試作した BBP ソフトウェアをもとに IPaf-R^2 の有用性について考察する．

フレームワークが提供されていない場合，開発者は通信に関するシステムコールや UNIX 上で用意されているライブラリ関数を知らなければならない． 状態遷移機械のアルゴリズムも考えなければならない．フレームワークを用いれば開発者はシステムコールやライブラリ関数を知らなくてもネットワークソフトウェアを作成できる．よってフレームワークを提供することで開発者の労力を軽減することができる．

フレームワークの評価として可変部と不可変部の割合を比較する．不可変部の割合が大きいほど再利用率が高い．表 1に可変部と不可変部の行数を示す． 自動生成系の入力を可変部，生成されるコードを不可変部として評価する．また Socket クラスは 2803 行あり，一つのクラスとして行数が多くなり評価が分かりづらくなるので Socket クラスの行数は含めない．

表 1: 可変部，不可変部の割合 1

	全体の行数	可変部	不可変部
client	2458	1432(58.3%)	1026(41.7%)
server	1433	786(54.8%)	647(45.2%)

IPaf-R^2 には表 1で可変部として扱っているクラスも機械的なコード記述を行なうだけで作成できるクラスがある．Action クラスは呼び出すメソッド名を変えるだけでよい．StateMachine クラスはイベントと次の遷移先を記述し，状態遷移表を作成するだけでよい． Strategy クラスは生成する Action クラスのインスタンスの名前を変えるだけでよい． 表 2に Action クラス， StateMachine クラス，Strategy クラスを不可変部として扱った場合の評価を示す．

表 2: 可変部，不可変部の割合 2

	全体の行数	可変部	不可変部
client	2458	507(20.6%)	1951(79.4%)
server	1433	437(30.5%)	996(69.5%)

フレームワークは簡単なコード記述を行なうだけで多くのネットワークソフトウェアを作成できることが望ましい．IPaf-R^2 では機械的なコード記述を行なうだけで作成できるクラスが多く，表 1に比べて，表 2の不可変部の割合が大幅に高くなっている．IPaf-R^2 を用いれば開発者の労力を軽減できる．

6 おわりに

これまでに構造を整理する目的でアーキテクチャ IParch-R が提案され，フレームワーク IPaf-R が設計・実現された．新たに入力の多重化と排他制御を実現するためにアーキテクチャ IParch-R^2 とフレームワーク IPaf-R^2 が提案された．我々は提案されたフレームワーク IPaf-R^2 を実現した．

今後の課題としては

1. IPaf-R^2 は不可変部の割合が高いので再利用率は高くなるが，独自のプロトコルに従ってネットワークソフトウェアを試作したので汎用性のあるフレームワークであるかは確認できない． IPaf-R^2 を用いて既存のプロトコルに従ったネットワークソフトウェアを試作して IPaf-R^2 の汎用性を確認しなければならない．
2. 我々はコードの行数によってフレームワーク IPaf-R^2 を定量的に評価した．定性的な評価で有用性を確認するために，他者にフレームワーク IPaf-R^2 を用いてネットワークソフトウェアを試作してもらう．
3. ネットワークソフトウェアにおいてセキュリティを考慮することは重要である． アーキテクチャにおいてどの層でセキュリティを考慮するかを考察することが必要である．

謝辞

本研究を進めるにあたり，一年半御指導いただいた野呂昌満教授，有益なアドバイスをいただいた張漢明先生，大学院の熊崎敦司さん，池内仁さん，宗宮健仁さん，青木俊介さんに深く感謝致します．

参 考 文 献

[1] Masami Noro,Kunio Goto, “An Architecture and a Framework for IP Applications”, , *Proceedings of APSEC'97*, pp. 191–199, 1997.

[2] 森 忠夫： ネットワークソフトウェアの構成法に関する研究，南山大学経営学研究科修士論文，1998.

[3] 青木 俊介： 既存ネットワークソフトウェアとの互換性を考慮した構成法，南山大学経営学部情報管理学科卒業論文，1999.

[4] Erich Gamma, *et. al, Design Patterns*, Addison-Wesley, 1995.