TCP/IP アプリケーション開発支援キットの設計，実現とその運用

野呂昌満[††] 熊崎敦司[†] 張漢明[††]

現在, TCP/IP アプリケーションは一部の有能なプログラマによって, システムコールライブラリ等の低粒度部品を使用して手続き指向で実現されており, ソフトウェア工学的には多くの問題を孕んでいる. この問題の解決策として TCP/IP アプリケーションの開発支援キット IPak を設計, 実現する. IPak は, クラスライブラリ, 自動生成系, アプリケーションフレームワーク等の技術を応用した開発支援キットである. IPak を使用して TCP/IP アプリケーションを開発し, その有用性について考察する.

Design, Implementation and Operation of TCP/IP Application Development Kit

Masami Noro ,[††] Atsushi Kumazaki [†] and Han-Myung Chang[††]

The development of TCP/IP applications is troublesome in practice since reusable components are usually system-call-level service routines. This paper describes a development-kit, called IPak, which we have developed for TCP/IP applications. IPak is a development support kit which has a class library, a code generator, and an application framework. We demonstrate the usefulness of IPak through the implementation of several TCP/IP applications.

1. はじめに

ネットワークコンピューティングの普及とともに TCP/IP アプリケーションは OS の核, またはミドルウェアとして重要なものになってきている. アプリケーションプロトコルは頻繁に提案, 改版が繰り返されることから, その実現における生産性の向上は必要不可欠だと考えられる.

現在の TCP/IP アプリケーション開発の問題点として,

- 一部の有能なプログラマだけによって開発が行われていること,
- システムコールライブラリと呼ばれる低レベル部品を組み合わせて手続き指向で実現されていること,

などが挙げられる. これらの問題は,

- 歴史的にネットワークハッカが C 言語を用いて開発を行ってきたこと,
- アプリケーションプロトコルがプロセスとして定義されてきたこと,

などの要因が複雑に絡み合った結果として生じたものである.

我々は, こられの問題の解決を目指し, TCP/IP アプリケーションのソフトウェアアーキテクチャ IParch-R[2] を構築した[8]. このソフトウェアアーキテクチャ上で, アプリケーションフレームワーク, 自動生成系, クラスライブラリ等の技術を応用して TCP/IP アプリケーションの開発支援キット IPak を設計, 実現した.

これまでに TCP/IP アプリケーションの開発支援を目的として, クラスライブラリ Socket++[3] やアプリケーションフレームワーク Conduit+[6] が実現されている. Socket++はオブジェクト指向インタフェースを提供するものではあるが, 低レベルな部品であることにかわりはない. Conduit+には, その背景となるソフトウェアアーキテクチャが存在しない. IPak は, 構造, 開発プロセスを示すソフトウェアアーキテクチャを背景に持つ

CORBA[1] など, 遠隔オブジェクトへのメッセージ通信を実現する技術も実現されている. これは我々の考えである遠隔オブジェクトアクセスの一番自然な実現方法である. しかし, これらでは, 実行効率が考慮されていない. さらに, 通信は独自のプロトコルで実現されている. IPak は, 既存の TCP/IP アプリケーションとの互換性と実行効率を考慮した ORB を提供

† 南山大学経営学部情報管理学科
Department of Information Systems & Quantitative Sciences, Nanzan University
†† 南山大学数理情報学部情報通信学科
Department of Information & Telecommunication Engineering, Nanzan University

する.

2. IParch-R^2:TCP/IP アプリケーションのソフトウエアアーキテクチャ

ソフトウェアアーキテクチャ[5] は, システムの構造を示すだけではなく, 実現の方法を含む実現のプロセスをも示唆すると我々は考える. 4+1 の側面[9] をさらに標準化し, 抽象側面, 具象側面, プロセス側面の 3 側面を持つものとして定義する.

2.1 抽象側面

抽象側面は従来から一般に提案されているソフトウェアアーキテクチャを指し, 構成要素とその関係を記述するものである.

クライアント, サーバは異なるコンピュータ上で並行動作する状態遷移機械であるとの前提の下に IParch-R^2 を構築した. IParch-R^2 の抽象側面を図 1 に示す.

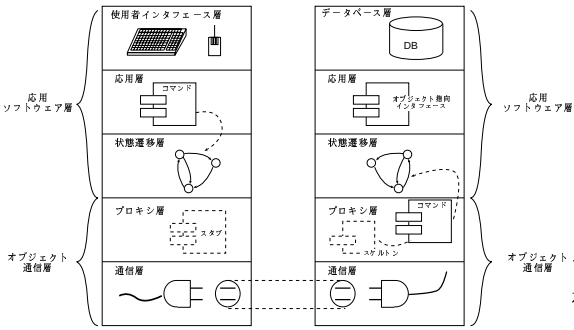

図 1 抽象側面の概要
Fig. 1 Outline of Abstract View

IParch-R^2 ではオブジェクト通信層と応用ソフトウェア層に TCP/IP アプリケーションを分割する. 生産性の向上を目指し, 再利用可能, または自動生成可能な構成要素を分離する. 状態遷移機械の記述から, 遠隔オブジェクトアクセスに関する記述を分離する. 遠隔オブジェクトアクセスの記述は一般に定型な記述でサーバのインタフェースの記述から自動生成が可能である.

オブジェクト通信層をさらにプロキシ層と通信層に分割する. どのアプリケーションプロトコルにも共通な下位 3 層のプロトコルを通信層に分離して実現する. プロキシ層ではアプリケーションに依存する通信を実現する. 状態遷移層に対しては遠隔オブジェクトの代理を提供する.

応用ソフトウェア層をさらに 3 層に分割する. これらの層は状態遷移機械を実現する層なので, 状態遷移機械への変更を考慮して層の分割を行う. 状態遷移機械の処理は状態の遷移と状態遷移時の処理の実行である. これらを独立に変更することを考え, 状態の遷移を実現する状態遷移層と応用の論理 (状態遷移時の処理) を実現する応用層に分割する. クライアントでは, 最上位層に使用者インタフェース層を, サーバでは, データベース層を配置する.

2.2 具象側面

具象側面は, 抽象側面にしたがってシステムを開発するさいに用いる開発技術, また, それを用いてできあがる半製品を示す. IParch-R^2 の具象側面では, TCP/IP アプリケーション開発に 3 技術 (クラスライブラリ, 自動生成系, アプリケーションフレームワーク) を用いることを示している.

通信層はクラスライブラリとして実現する. 通信層のプロトコルが大幅に変更されることは考えづらいのでクラスライブラリを提供することにした. コードを自動生成したり, アプリケーションフレームワークのカスタマイズを行うよりも開発者の労力は小さくなる.

プロキシ層には,

- コマンドとその引数で仕様が厳密に定義できる,
- 仕様から機械的な変換でコードが生成できる,
- 仕様と生成されるコードの隔たりが大きい,

などの理由から自動生成系の技術を用いる.

上位 3 層にはアプリケーションフレームワークを設計, 実現する. これらの層はアプリケーションへの依存度が高く, クラスライブラリ化は不可能である.

2.3 プロセス側面

プロセス側面は, 具象側面が示す開発技術を用いてシステムを開発するさいのプロセスを示す. つまり, システムを実現するさいに行う作業とその半順序関係を示す.

IParch-R^2 のプロセス側面が示す作業とその半順序関係を図 2 に示す. 作業は大きく状態遷移機械の開発と自動生成の作業に分割できる. 図中の四角はサブプロセスを表す.

3. TCP/IP アプリケーション開発支援キット IPak の設計と実現

IPak は IParch-R^2 に基づきクラスライブラリ, 自動生成系, アプリケーションフレームワークの開発支援技術を応用したものである. 我々は C++でこれらを実現した.

3.1 クラスライブラリの設計と実現

通信層を実現するクラスライブラリは TCP/IP 階層モデル下位 3 層のプロトコルを実現するものである. これらのプロトコルの実現はシステムコールライ

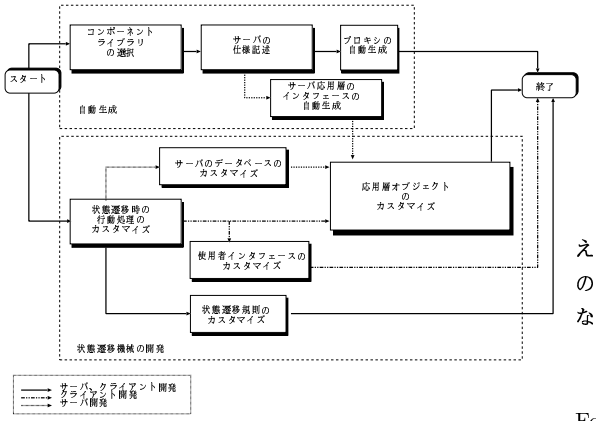

図 2 IParch-R^2 を用いたソフトウェア開発プロセス
Fig. 2 Software Process in IParch-R^2

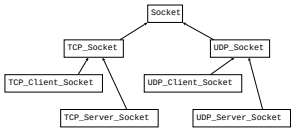

図 3 クラスライブラリの階層
Fig. 3 Hierarchy for Class Library

ブラリを使って行われるのが現状である. しかし, これらは粒度が低すぎ, 開発者は使用方法の理解に労力を費すことになる. この問題の解決策として, オブジェクト指向ソケットを提供する. クラスライブラリの粒度とオブジェクト指向の観点から考えると, ソケットをオブジェクトととらえるのが自然である. プロトコル実現の処理はオブジェクトにカプセル化し, 開発者にはパケットの読み書きのインタフェースだけを提供する.

プロトコル実現の処理は,

- プロトコル (TCP/UDP) と
- クライアント, サーバとの

2 つの次元で分類できる. TCP の処理, UDP の処理, クライアントの処理, サーバの処理をライブラリ化し, それらの組合せでプロトコルを実現できるのが理想である. しかし, 実際はクライアント, サーバの処理がプロトコルに特化し, この方法は採れない. この議論から自然に導き出されるクラス階層を図 3 に示す. `Socket` をスーパークラスとし, TCP 通信を実現する `TCP_Socket`, UDP 通信を実現する `UDP_Socket` をサブクラスとして提供する. 開発者はプロトコル (TCP/UDP) の違いを意識することなく, 通信プログラムを作成できる. サーバ, クライアント毎の異なる処理は `TCP_Socket`, `UDP_Socket` のサブクラスにカプセル化する.

開発者は, 実現したい種類にあったクラスをインスタンス化することでプロトコルを実現できる. また, 多くのオブジェクト指向プログラミング言語で多相型を提供している. TCP 通信プログラムを UDP 通信プログラムに変更する場合には, `TCP_Socket` のインスタンスの代わりに `UDP_Socket` のインスタンスを使えばよい. このライブラリ化に伴い, システムコールの使用法を覚えたり, システムコールを書き直したり, などの作業は必要なくなる.

3.2 自動生成系の設計と実現

TCP/IP アプリケーションの実現では, RFC(Request For Comments) で提唱されたアプリケーションプロトコルが事実上の標準となっている. IParch-R^2 では, 遠隔オブジェクトへのメッセージ通信を実現するプロキシ層のオブジェクトが RFC に示されたアプリケーションプロトコルを理解することになっている. プロキシ層のオブジェクトはサーバのオブジェクト指向インタフェースから自動生成可能である. 一方, サーバの応用層はオブジェクト指向インタフェースを提供するものであり, アプリケーション毎に異なるが, これもサーバのオブジェクト指向インタフェースから自動生成可能である.

我々はアプリケーションプロトコルの調査を行い, 仕様の記述には以下の定義が必要であることを確認した.

(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
(4) コマンド実行後, 接続を切るか否か
(5) 使用するポート番号 (待ち受け)

既存の技術との互換性を重視するという観点から, 仕様の記述にはクラスのインタフェース記述 (C++のクラスの公開部の記述) を用いる. 新たな仕様記述言語を定義したりはしない. コマンド毎の付加条件は局所変数を使用して記述する. 以下は `POP3` のコマンド `RETR` を表した例である. 詳しくは文献[7] を参照されたい.

```
class POP3 {
 Res *RETR(int){
  char Eod[] = "\r\n.\r\n";
  char *replace[2] = {"\n..","\n."};
  int connect = 1;
 }
};
```

自動生成系は非同期通信メソッドを生成する (上の例の場合, スタブには要求送信用メソッド `retr()` と

結果受信用メソッド retr_r() が生成される). これは実現時のプロセス割付けに関連する. クライアントは使用者からの入力とサーバからの結果を同時に待つ. クライアントが並行プロセスであれば問題はないが, 単一プロセスではサービス要求から結果取得の間, 他のイベントを受け取ることができない. ここでは, 既存アプリケーションとの互換性, 実現環境との関連から単一プロセスでの実現を考える. 単一プロセスでイベントの同時待ち受けを行うためには非同期メッセージ通信が必要になる.

3.3 アプリケーションフレームワーク IPaf-R^2 の設計と実現

アプリケーションフレームワークの設計, 実現では以下の点を考慮する.

- IParch-R^2 の抽象側面が示す構造に従う.
- オブジェクト指向の考えを素直に用いる.
- 既存の TCP/IP アプリケーションとの互換性を保証する.
- デザインパターンを積極的に採用する.

3.3.1 プロセス割付け

アプリケーションフレームワークを実現するさいには, どの構成要素を並行処理実体にするのかを考慮しなければならない.

クライアントは使用者からの入力とサーバからの結果をイベントとして受け取る状態遷移機械である. したがって, それぞれのイベントを同時待ち受けする必要がある.

IPaf-R^2 では, 単一の重量プロセスで同時待ち受けを実現する方法を提供する. 複数の重量プロセスでクライアントを実現する方法もあるが, 既存のクライアントに複数の重量プロセスで同時待ち受けを実現するものはなく, 互換性に問題がおきる. 軽量プロセスでの実現は開発・実行の環境を限定することになる.

単一の重量プロセスで同時待ち受けを実現するには, 入力を監視するオブジェクトが必要である. 使用者からの入力は使用者インタフェース層のオブジェクトが, サーバからの結果は通信層のオブジェクトが受け取る. 使用者入力, サーバ結果を同時に待ち受けするには, これらのオブジェクトを監視する必要がある.

サーバは複数のクライアントからの要求に対してサービスを提供する必要がある. 既存の TCP/IP アプリケーションは, 1 クライアントに対して 1 重量プロセスを生成し, サービスを提供する. IPaf-R^2 では, 既存 TCP/IP アプリケーションとの互換性を重視し, 複数の重量プロセスで複数のクライアントへの対応を実現する.

IPaf-R^2 では, 一般のサーバプロセスと同様にクライアントからのサービス要求を受け付けた後, 自分自身の複製プロセス (子プロセス) を生成する. それ以降のサービス提供は子プロセスが行う.

3.3.2 状態遷移機械のアプリケーションフレームワーク

IParch-R^2 の抽象側面では, 状態遷移機械の実現から状態遷移時の処理の生成を分離することが示されている. これはコマンドパターン[4] により実現する. 処理をコマンドパターンオブジェクトにカプセル化することにより, 生成と実行を分離できる.

状態遷移機械の実現には, ステートパターン[4] を用いる. ステートパターンを用いれば, 状態遷移機械と状態遷移規則を別々に実現できる. 状態遷移機械に影響を与えることなく状態遷移規則を変更することができ, 変更に強い状態遷移機械を実現することができる.

3.3.3 クライアントのアプリケーションフレームワーク

クライアントのアプリケーションフレームワークは, 状態遷移機械のアプリケーションフレームワークに使用者インタフェース部分を追加したものである. クライアントのアプリケーションフレームワークの構成を UML のクラス図を用いて示す (図 4 参照). 太線のクラスはホットスポットである.

使用者インタフェース層

使用者インタフェースを実現するオブジェクトはアプリケーション毎の違いが大きいので, あらゆる使用者インタフェースを作成可能なアプリケーションフレームワークを実現することは事実上不可能である. IPaf-R^2 では, もっとも一般的な標準入出力を実現するためのクラス (StdIO) を提供する.

使用者入力は, クラス Com にカプセル化し, 応用層に渡す. 入力情報を Com にカプセル化することで, 使用者インタフェースからの情報を統一的に扱うことができる.

ComCreator は, 入力情報を Com にカプセル化する役割を持つ. 使用者入力の解析はアプリケーション毎に異なるので ComCreator はアプリケーションフレームワークのホットスポットになる.

応　用　層

応用層では, 使用者インタフェースから受け取った Com をもとに, 行動処理として Action を生成する. 内部ではこれらの対応表を持つ必要があるので, それを ComResTable に保持する.

ComResTable は, 実際には Com と対応する Action を生成するための処理アルゴリズム Strategy を保持

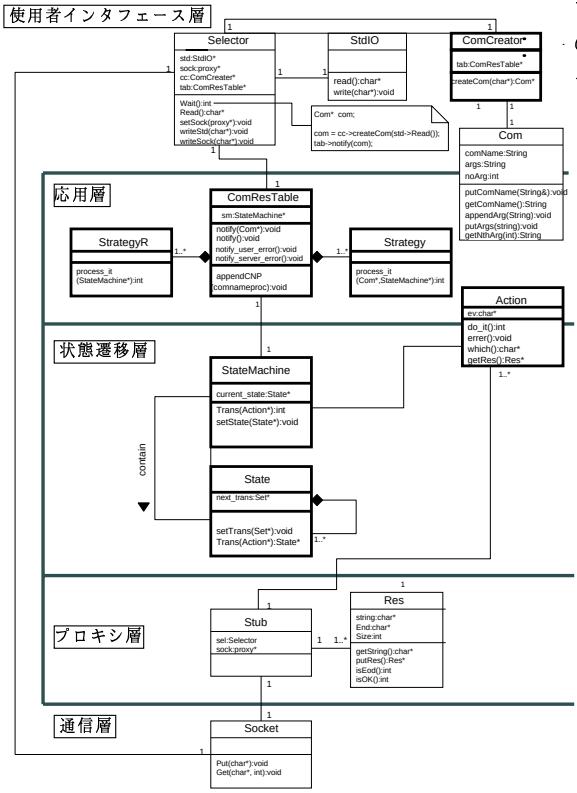

図 4　クライアントのアプリケーションフレームワーク
Fig. 4　Outline of Application Framework for Client

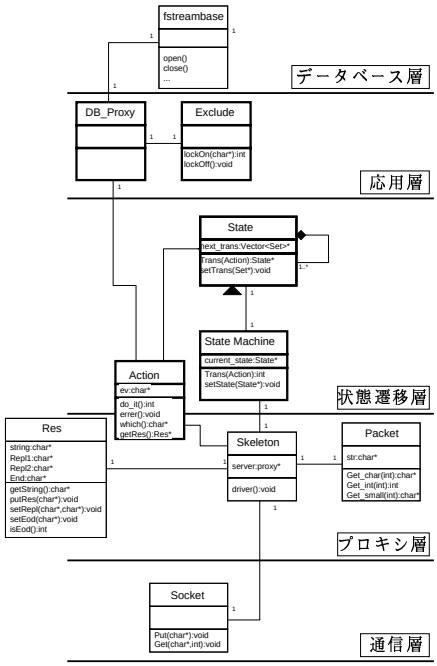

図 5　サーバのアプリケーションフレームワーク
Fig. 5　Outline of Application Framework for Server

する．Strategy はストラテジパターンを実現するクラスである．一部のプログラミング言語では，クラスを保持し，そのインスタンスを生成することも可能だが，この方法は開発環境を限定することになる．IPaf-R^2 では，一般のオブジェクト指向プログラミング言語で実現可能なストラテジパターンを使用する方法を採用する．

状態遷移層

状態遷移層には，状態遷移機械を配置する．ステートパターンを使用しているので，状態遷移機械 (StateMachine) と状態 (State) に対応するクラスを用意する．

その他 (使用者インタフェース層と通信層)

イベント監視用のクラス Selector を使用者インタフェース層と通信層のオブジェクトにアクセスできるように配置する．上で述べたように，クライアントには同時待ち受けを実現するためのクラスが必要である．

3.3.4　サーバのアプリケーションフレームワーク

サーバのアプリケーションフレームワークは，3.3.2 節で述べた状態遷移機械のアプリケーションフレームワークにデータベースを追加したものである．サーバのアプリケーションフレームワークの構成を UML のクラス図を用いて示す (図 5 参照)．

データベース層

データベース層はサーバのデータベースを実現する層である．IPaf-R^2 では，ファイルの入出力を実現するクラス (fstreambase) を提供する．一般に UNIX 上でサーバを実現する場合，データベースはファイルとして実現される．

応　用　層

応用層には，データベースアクセスのためのオブジェクト指向インタフェースを提供する DB_Proxy を配置する．このクラスはアプリケーションに依存しているので，そのインタフェースは自動生成する (スタブやスケルトン等のプロキシ層のオブジェクトと同様に，サーバのオブジェクト指向インタフェースから自動生成できる)．生成されたクラスをホットスポットとして，その実現は開発者が行う．

ほとんどの場合，サーバは並行プロセスとなるので，排他制御実現のためのインタフェースを持つ Exclude を提供する．

状態遷移層

状態遷移層には，クライアントと同様に状態遷移機

械を配置する.

4. IPak による TCP/IP アプリケーションの開発

IPak を使用していくつかの TCP/IP アプリケーションを実現した. これらと手続き指向で実現した TCP/IP アプリケーションを比較し, IPak の有用性について考察する.

まず, BBP(BulletinBoard Protocol) という我々が考案したアプリケーションプロトコルを実現した. つぎに HTTP, POP3, FTP を実現した. これらの実現を通して, IPak が新しいアプリケーションプロトコルに対応可能であること, および, 既存のアプリケーションプロトコルとの互換性を保証することが確認できた.

4.1 BBP, HTTP, POP3 の実現

クライアントの実現

2.3 節の開発プロセスにしたがって応用層, 状態遷移層のカスタマイズを行う. 応用層ではアプリケーションプロトコルのコマンド毎のカスタマイズを行う (“状態遷移時の行動処理のカスタマイズ” と “応用層オブジェクトのカスタマイズ”, 図 2 参照). 状態遷移層では状態遷移規則に関するカスタマイズを行う (“状態遷移規則のカスタマイズ”). 使用者インタフェース層のカスタマイズは, アプリケーションプロトコル毎というよりはむしろ対象とする使用者毎に異なる. 今回は, あらかじめ提供されている標準入出力用のクラスを用いて文字使用者インタフェースを実現する (“使用者インタフェースのカスタマイズ”).

各コマンドに対して,

- `Action(R)` と `Strategy(R)` の対を定義し,
- `Strategy(R)` を `ComResTable` に登録する.

状態遷移規則の各状態に対して,

- `State` のインスタンスを生成し,
- 各 `State` にイベントと遷移先の `State` を設定する.

以上の作業で BBP クライアントが実現できる.

サーバの実現

今回は UNIX 上で開発を行ったのでデータベースはファイルで実現する. したがって, データベース層ではあらかじめ提供されているファイル入出力用のクラスを使用する (“サーバのデータベースのカスタマイズ”).

クライアントと同様にコマンド毎のカスタマイズと状態遷移規則に関するカスタマイズを行う. サーバの場合, 状態遷移規則に関するカスタマイズは子プロセスの状態遷移規則に関して行う. サーバの場合, 親プロセスが接続待ちを行い, 実際のサービス処理は子プロセスが行う. IPak では, 接続要求を受信後, プロキシ層で自分自身の複製プロセスを生成し, サービス処理を行う. 状態遷移機械が起動するのは子プロセスが生成されてからとなる.

新しいアプリケーションプロトコルに対しても, 既存のアプリケーションプロトコルに対しても, 同様のカスタマイズを行うことで TCP/IP アプリケーションを実現できる.

4.2 FTP の実現

FTP は 2 本の TCP 接続 (データ接続, 制御接続) を必要とする複雑なアプリケーションプロトコルである. 接続の本数にかかわる問題はオブジェクト通信層に関係する問題であり, 応用ソフトウェア層に影響を与えてはいけないと考えられる. 応用ソフトウェア層に影響を与えなければ, IPaf-R^2 はそのまま使用することができる.

オブジェクト通信層のオブジェクトの自動生成のさい, 2 本の TCP 接続を実現する方法として以下の 2 つが考えられる.

- プロキシ層のオブジェクトを拡張し, 2 本の TCP 接続を制御できるようにする.

  スタブは異なる TCP 接続を使用して通信を行うために 2 種類のメソッドを持つ. スケルトンは 2 つのポートへの通信を監視する. 実現には自動生成系への入力を拡張しなければならない. 各メソッドで使用するポートを明示する必要がある.
- プロキシ層に 2 つのスタブを配置する.

  スタブを 2 つ生成する. 一方のスタブは制御接続を使用して通信を行い, もう一方のスタブはデータ接続を使用して通信を行う. スケルトンは両者からの通信を監視する.

ここでは前者の方法を選択する. 本来, スタブはサーバの代理である. しかし, 後者の場合, スタブ, スケルトンを通信経路の代理と考えることになる.

以上の実現方法はプロキシ層以外の層に影響を与えないので, IPaf-R^2, クラスライブラリはそのまま使用することができる. アプリケーションフレームワークのカスタマイズは, BBP, HTTP, POP3 実現時と同様である

5. 議　　論

IPak を使用して BBP, HTTP, POP3, FTP を実現した. 実現した TCP/IP アプリケーションに基づき, ソフトウェアの生産性について考察を行う.

比較の前提

ソフトウェアの生産性をはかる尺度としてコードの行数を用い考察を行う. IPak を使用して TCP/IP アプリケーションを作成した場合と C 言語を使って手続き指向で作成した場合について,

– 開発者が実際に記述するコードの行数,
– 再利用可能なコードの割合,

を比較する. IPak を使用した開発ではオブジェクト指向プログラミング言語 C++を用いる.

C の記述と C++の記述の行数を絶対数として比較することはできないので, それぞれをファンクションポイント (以下, FP とする) への換算を介在して比較する. FP では, C++の記述 1 に対して C の記述 2.41 が対応することが確認されている[2]. この値はシステムプログラムの開発における値であり, プロジェクトに依存するものである. TCP/IP アプリケーションの開発においてこの値が正確なものであるかの検討は必要である. しかし, この値は異なるグループ, 多くのプロジェクトを標本とし, 算出された平均値なので, 今回はこれを受け入れることにする.

TCP/IP アプリケーションのコードの行数

IPak を使用して TCP/IP アプリケーションを開発した場合, そのコードは以下の 5 つに分類できる.

– IPaf-R^2 のホットスポットのコード
– IPaf-R^2 のフローズンスポットのコード
– サーバの仕様 (自動生成系の入力)
– 自動生成系が生成したコード
– クラスライブラリのコード

この中で, 開発者が実際に記述するコードは, IPaf-R^2 のホットスポットのコードとサーバの仕様である. IPak を使用して BBP, HTTP, POP3, FTP を実現したときのコードの行数を表 1 に示す. 表中の C はクライアント, S はサーバを表す. 表中の FS は IPaf-R^2 のフローズンスポット, CL はクラスライブラリを指す.

手続き指向で TCP/IP アプリケーションを作成したときのコードは以下の 2 つに分類できる.

– TCP/IP アプリケーションの定型コード
– それ以外のコード

定型コードとは, システムコール呼び出し, およびそれに付随する環境設定, 例外処理など TCP/IP アプリケーション作成時の決まった記述のことである. 手続き指向で実現した BBP, POP3, FTP アプリケーションの行数を表 2 に示す.

開発者の記述量に関する結果の考察

開発者の記述量を比較したものを表 3 に示す. IPak を使用して BBP アプリケーションを実現した場合, クライアント実現にかかる記述が 260 行, サーバ実現にかかる記述が 168 行である. 手続き指向で実現する場合には, 再利用できる部品はシステムコールライブラリだけである. しかし, ここでは上述の定型コードは TCP/IP アプリケーションに共通の記述なので再利用できるものとして考える. 実際に手続き指向で BBP アプリケーションを実現する場合に開発者が記述する行数はクライアントが 673 行, サーバが 427 行である.

表 2 手続き指向で実現した TCP/IP アプリケーションのコードの行数
Table 2 The Number of Lines of Code(Procedure-Oriented)

		全体の行数	定型コードの行数
BBP	C	708	35 (5.0%)
	S	453	26 (5.7%)
POP3	C	1290	86 (6.7%)
	S	1608	98 (6.1%)
FTP	C	6265	312 (5.0%)
	S	3320	280 (8.4%)

表 3 コードの行数の比較
Table 3 Comparison of The Number of Lines of Code

		IPak	FP(IPak)	手続き指向
BBP	C	260	626.6	673
	S	168	404.9	427
POP3	C	241	580.8	1204
	S	379	913.4	1510
FTP	C	1238	2983.6	5953
	S	587	1414.7	3040

上の前提に基づいてこれらの値を比較する. クライアント実現に必要なコードの行数を FP の比率を用いて比較すると, 比率は 626:673 になる. サーバの場合, 比率は 404:427 になる. わずかに IPak を使用した場合の方が少ない労力で BBP アプリケーションを実現できる.

POP3, FTP アプリケーションに対しても同様の比較を行う. 表 3 から, POP3 クライアントのコードの比率は 580:1204, POP3 サーバのコードの比率は 913:1510 である. 同じく, FTP クライアントのコードの比率は 2983:5953, FTP サーバのコードの比率は 1414:3040 である. 明らかに IPak を使用した方が少ない記述で, POP3, FTP アプリケーションを実現できる.

以上の開発経験から IPak の有用性を実証したとはいえないが, 生産性向上に有効であることは示唆できたと考える.

再利用率に関する結果の考察

再利用率の比較では, IPak を使用した場合が高い数

表 1　IPak を使用して実現した TCP/IP アプリケーションのコードの行数
Table 1　The Number of Lines of Code(IPak)

		全体の行数	記述した行数 (仕様の行数)	生成した行数	FS	CL	再利用可能な行数 (割合)
BBP	C	1036	260(7)	84	388	311	699(73.4%)
	S	762	168(7)	72	218	311	529(76.7%)
HTTP	C	885	155(5)	36	388	311	699(82.3%)
	S	666	84(5)	58	218	311	529(87.0%)
POP3	C	1071	241(11)	142	388	311	699(75.2%)
	S	1009	379(11)	112	218	311	529(59.0%)
FTP	C	2496	1238(38)	597	388	311	699(36.8%)
	S	1401	587(38)	323	218	311	529(49.1%)

値を示しているのに対して, 手続き指向実現の場合は低い数値を示している (表 4 参照). どの開発においても IPak を使った方が再利用率が高い.

表 4　再利用率の比較
Table 4　Comparison of Reusability

		IPak	手続き指向
BBP	C	73.4	5.0
	S	76.7	5.7
POP3	C	75.2	6.7
	S	59.0	6.1
FTP	C	36.8	5.0
	S	49.1	8.4

ソフトウェアを開発, 改版する場合に, 多くのコードを再利用できれば, 結果として記述量が減り, 生産性の向上が期待できる. この結果とコード行数の比較の結果をあわせて, IPak は TCP/IP アプリケーション開発の生産性向上に有効であると考えられる.

6. おわりに

本研究では TCP/IP アプリケーションのための開発支援キット IPak を設計, 実現した. IPak を使用して複数の TCP/IP アプリケーションプロトコルを実現し, これらの例に限っては従来の開発と比較して生産性が向上することを確認した.

本研究の今後の課題は以下のとおりである.

- IPak を使用してすべての TCP/IP アプリケーションを実現する.

  実際に TCP/IP アプリケーションを実現することで, IPak が生産性の向上に有効であることを実証する.
- IPak をソフトウェア開発環境として整備する.

  IParch-R^2 のプロセス側面に基づき開発支援キットの構成要素 (クラスライブラリ, アプリケーションフレームワーク, 自動生成系) を統合し, ソフトウェア開発環境とする.

参考文献

1) R. Ben-Natan, *CORBA: A Guide to Common Object Request Broker Architecture*, McGraw-Hill, 1995.
2) R.Biuk-Aghai, "Project Planning 2: Resource Estimation," *http://www.sftw.umac.mo/~fstrpba/courses/sftw497/lecnotes/lecnotes-8.pdf* .
3) H. Bocking, "Socket++ - A Uniform Application Programming Interface for Communication Services," *IEEE Communication Magazine*, Dec. 1996.
4) E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1994.
5) D. Garlan and D. E. Perry Eds., *Special Issues on Software Architecture : IEEE Transaction on Software Enginnering, Vol. 21, No. 4*, 1995.
6) H. Huni, R.Johnson, and R. Engel, "A Framework for Network Protocol Software," *Proceeding of OOPSLA '95*, pp. 358–369, 1995.
7) 熊崎 敦司, 蜂巣 吉成, 青木 俊介, 野呂 昌満, "既存ネットワークソフトウェアとの互換性を考慮したソフトウェア構成法", 第 7 回ソフトウェア工学の基礎ワークショップ 論文集, 2000
8) 熊崎 敦司, 野呂 昌満, 蜂巣 吉成, 張 漢明, "TCP/IP アプリケーションのソフトウェアアーキテクチャ", コンピュータソフトウェア ソフトウェア工学の基礎特集号 (投稿中)
9) P. B. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, Vol. 12, No. 6, pp.42-50, Nov. 1995.