アプリケーションプロトコルソフトウェアのアスペクト指向実現に関する研究

2000MT029 石川 智子　　2000MT046 小久保 佳将
指導教員　野呂 昌満

1　はじめに

インターネットにつながったコンピュータ数の増加に伴い，新しいアプリケーションプロトコルソフトウェアの開発や既存のソフトウェアの改版が繰り返されている．ソフトウェアの開発・改版を省力化するためには，ソフトウェアの変更に対する柔軟性の向上が必要である．アプリケーションプロトコルソフトウェアの柔軟性を向上させるための一方法として，ソフトウェアアーキテクチャの構築が考えられる．

オブジェクト指向を用いてアプリケーションプロトコルソフトウェアのソフトウェアアーキテクチャ [4] を構築してきた経験から，アスペクト指向によるソフトウェアの分割，ロールの考えによる計算場分割が必要であることを確認した．前者は機能特性と例外処理など，横断的に関連している問題の解決に必要となる．後者はプロトコル処理中の場所によって振舞を変えるオブジェクト PDU (Prot ocol Data Unit) の存在をオブジェクト指向を用いて自然にモデル化するために必要となる．これらの問題を統一的に記述する方法としてアスペクト指向ソフトウェアアーキテクチャスタイル (AOSAS)[5] を我々の研究室で提案してきた．

本研究の目的は，アプリケーションプロトコルソフトウェアにおいて，AOSAS を適用したアスペクト指向ソフトウェアアーキテクチャの有用性を考察することである．構築したソフトウェアアーキテクチャに従って作成したアプリケーションプロトコルソフトウェアの柔軟性を考察する．従来からアプリケーションプロトコルソフトウェア作成の支援を目的に x-Kernel[2] や Conduit+[3] が提案，実現されている．これらはおもに OSI の TCP/IP，UDP 層以下の通信プロトコルの実現を念頭に置いている．本研究では応用層，プレゼンテーション層，セッション層におけるアプリケーションプロトコルのソフトウェアアーキテクチャを考える．

アスペクト指向ソフトウェアアーキテクチャに従えば，横断的に関連していた処理を分離して記述することができ，プロトコル処理中の場所を移動する PDU の役割を素直にモデル化することができた．

石川は主に横断的に関連する処理に関する部分を，小久保は主に移動オブジェクトに関する部分を担当した．

2　アスペクト指向ソフトウェアアーキテクチャスタイル

AOSAS では異なった分割基準による問題を統一的に扱い解決することができる．横断的に関連する処理はアスペクト指向分割を，移動オブジェクトは計算場分割を，コンサーンを実現するオブジェクト群はオブジェクト指向分割を用いて解決する．構成要素はフィールド，ロール，オブジェクトである．分離したコンサーンによってソフトウェアを分割し，構造の規定から得られたロールの集合をフィールドとする．ロールはフィールド内でのオブジェクトの役割を規定する．構成要素間の関係を以下に示す.

- ロールとフィールドの結合関係
- ロールとオブジェクトの結合関係
- フィールドとロールの関係
- ロール間の関係

3　アプリケーションプロトコルソフトウェアのアスペクト指向ソフトウェアアーキテクチャ

オブジェクト指向階層モデルでソフトウェアアーキテクチャの構築を試みたが，横断的に関連する部品が残っていたので，厳密な階層化ができなかった．ソフトウェアアーキテクチャを観察し，横断的に関連する部品を横断的コンサーンとして分離し，ソフトウェアを分割した．これに加えて PDU は別の指針である計算場分割に従い分割した．構築したアーキテクチャを図 1 に示す．
横断的コンサーンは以下のとおりである．

- 並行処理コンサーン
- 実時間処理コンサーン
- 例外処理コンサーン
- セキュリティ処理コンサーン
- プロトコル処理コンサーン

3.1　並行処理コンサーン

並行処理は使用者とサーバからの入力処理が階層モデルの応用層と通信層に横断的に関連していた (図 2).

並行処理フィールドでは使用者からの入力とサーバからの応答の多重入力を実現する．疑似並行実行モニタで使用者とサーバからの入力を同時に監視し，入力があった場合は LocalI/O フィールドの LocalController，もしくは RemoteI/O フィールドの RemoteController が処理する．

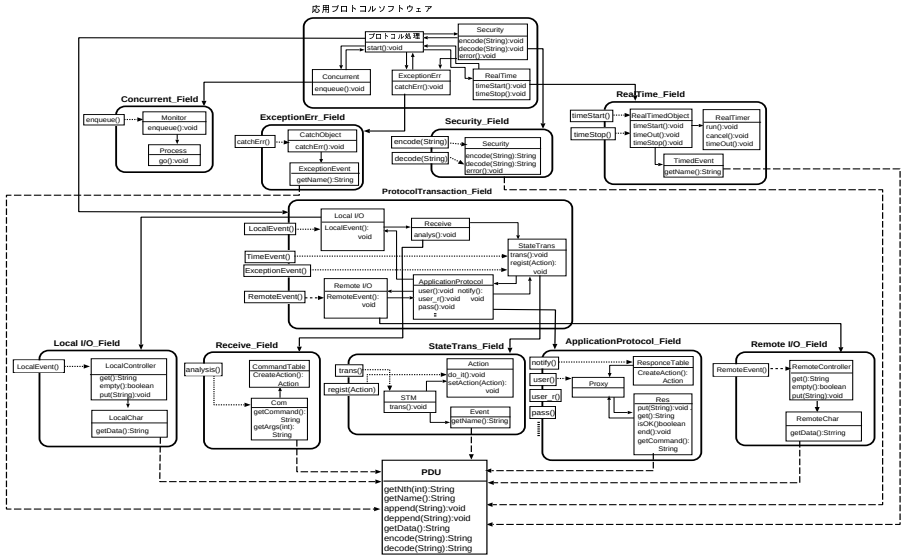

図 1　アスペクト指向ソフトウェアアーキテクチャ

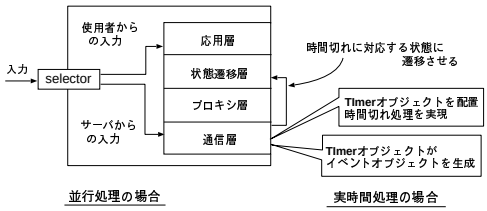

図 2　階層モデルにおける並行処理と実時間処理

3.2　実時間処理コンサーン

実時間処理は階層モデルの通信層と状態遷移層に横断的に関連している (図 2).

実時間処理フィールドでは，サーバとの通信時間を計測し通信障害やサーバの故障の検知をする．サーバに要求を送信する時間を計り，サーバから応答を受信する時間を計る．送受信中に時間切れが起きた場合は，時間切れに対応するイベントを TimedEvent に保持し，プロトコル処理フィールドのインタフェース ExceptionEvent() を通して 状態遷移機械を例外に対応する状態に遷移させる．時間切れが起きなかった場合は応答メッセージの字句解析をする．

3.3　例外処理コンサーン

例外処理は階層モデルの各階層に横断的に関連している (図 3).

例外処理フィールドでは，コマンドの例外入力やセキュリティ処理の例外に対する処理を実現する．例外メッセージを CatchObject が受け取り，例外イベントを ExceptionEvent が保持する．プロトコル処理フィールドのインタフェース ExceptionEvent() を通して状態遷移機械を例外に対応する状態に遷移させる．

3.4　セキュリティ処理コンサーン

セキュリティ処理は階層モデルの通信層の通信処理に横断的に関連している (図 3).

セキュリティ処理フィールドでは，パケットの盗聴に対してのプライバシー保護のための暗号化と復号化をする．サーバへの要求メッセージ送信の暗号化，応答メッセージの復号化を行う．

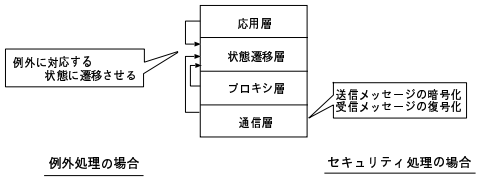

図 3　階層モデルにおけるセキュリティ処理と例外処理

3.5　プロトコル処理コンサーン

プロトコル処理フィールドでは，プロトコル処理に特化したアプリケーションロジックを実現する．プロトコル処理は使用者とサーバからの入力に対して処理をする．各入力をイベントとして状態を遷移させる状態遷移機械とみなす．使用者の入出力処理とサーバの応答処理は同様にして扱う．3.1 節〜3.4 節のようにプロトコル処理はすべてのフィールドと関連を持つ．

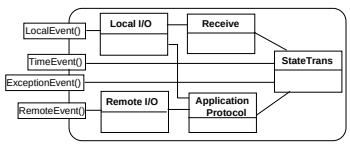

図 4　プロトコル処理フィールド

3.6 PDU

階層モデルにおける PDU は各層で役割を変えて存在していた．応用層やプロキシ層においては入力であり，状態遷移層においては遷移させるイベントとして存在した．字句解析を行う階層によって実体を持つことは，入力したものが場所によって複数存在することになるので自然なモデル化ができない．

AOSAS に従って PDU を計算場により分割する．プロトコル処理フィールドを計算場分割に従い以下のフィールドに分割した．また，実時間処理フィールド，例外処理フィールド，セキュリティ処理フィールドも計算場を用いて分割する．PDU と結合させることで，各場所においても実体は一つであり，自然にモデル化できる．

- LocalI/O_Field:使用者の入出力処理
- RemoteI/O_Field:サーバからの入出力処理
- Receive_Field:使用者からの入力の受け入れ処理
- StateTrans_Field:状態遷移機械とその振舞
- ApplicationProtocol_ Field:アプリケーション層の PDU の生成と解析

PDU は各フィールドのロールと結合することによって，以下のような役割を果たす．

- Com：使用者が入力したコマンドと引数
- Event：コマンドに対応する状態を遷移させるためのイベント
- Res：サーバからの応答メッセージ
- LocalChar：使用者から入力文字列
- RemoteChar：サーバからの入力文字列
- ExceptionEvent：例外に対応する状態を遷移させるためのイベント
- TimeEvent：時間切れに対応する状態を遷移させるためのイベント
- Security：暗号化，復号化されたメッセージ

4 アスペクト指向アプリケーションフレームワーク

アスペクト指向アプリケーションプロトコルソフトウェアアーキテクチャに従ったアスペクト指向アプリケーションフレームワークを設計し，実現する．アプリケーションフレームワークの設計方針は，アーキテクチャの構造を素直に反映することである．実現としては，一般的なアスペクト指向プログラミング言語である AspectJ[1] を用いる．

アプリケーションプロトコルソフトウェアにおいてソフトウェアの変更は主に，コマンド変更に伴う処理の変更と状態遷移の変更である．従って，コマンドに対応する処理がホットスポットとなる．並行処理，実時間処理，例外処理，セキュリティ処理はアプリケーションプロトコルにおいて共通な処理なので，フローズンスポットとなる．アスペクト指向アプリケーションフレームワークのホットスポットを図 5 に示す．

構成要素
- LocalController
- CommandTable
- PDU
- ResponceTable
- Res
- STM
- Action

構成要素間の関係
- STM と Action
- STM と Event
- Action と Proxy
- ResponceTable と Proxy
- Proxy と Res

図 5 アスペクト指向アプリケーションフレームワークのホットスポット

5 実現と考察

AOSAS をアプリケーションプロトコルソフトウェアに適用する利点は，横断的コンサーンと移動オブジェクトの問題を統一的に記述できる点である．横断的コンサーンを分離して記述することで，分離した処理を最適な構造で実現することができる．本節では，アスペクト指向ソフトウェアアーキテクチャの有用性を確認するために，作成したソフトウェアの柔軟性について考察する．

5.1 アプリケーションプロトコルソフトウェアの実現

アスペクト指向アプリケーションフレームワークを用いて，POP3(Post Office Protocol version 3)，SMTP(Simple Mail Transfer Protocol)，FTP(File Transfer Protocol) のクライアントソフトウェアを試作した．

- POP3・SMTP
  複数のコマンドがあり，1 つのコマンドに対して複数行の送受信を行う場合がある．また，コマンドがない送信がある．
- FTP
  複数のコマンドがある．FTP には制御とデータ転送の 2 つの接続を持ち，コマンドによってデータ転送を行う場合がある．

POP3，SMTP はホットスポットと，それに伴うアスペクト記述の変更だけで，ソフトウェアの作成ができた．FTP では，ホットスポットの変更に加えて，複数の接続を確立するためのロールの追加によって実現できた．

5.2 柔軟性に関する考察

作成したアプリケーションプロトコルソフトウェアの柔軟性を考察する．

コマンドの拡張

コマンドの拡張に伴う字句解析部分 (Com, Res, Event) の変更を PDU に局所化することにより，それぞれの解析部分の変更の必要がなくなる．

移動オブジェクトは場所によって振舞を変えるので，階層モデルでは，各階層の解析部分でコマンドの字句解析が必要となる．

セキュリティ処理

セキュリティ処理は SSL(Secure Socket Layer) などの通信プロトコルレベルでの暗号化・復号化と，APOP などのアプリケーションレベルでの暗号化・復号化が考えられる．これらのどちらを基準にするかはセキュリティ処理フィールドとプロトコル処理フィールドとの関係を

変更することで解決できる．通信プロトコルを基準とした暗号化・復号化では RemoteController に合流点を設定し，アプリケーションを基準とした暗号化，復号化では Proxy の各コマンドのメソッドに合流点を設定する (図 6)．

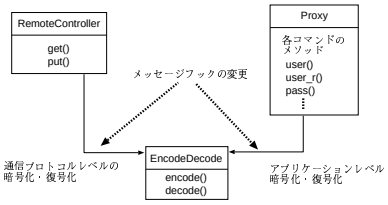

図 6　関係の変更による暗号化・復号化のレベルの変更

オブジェクト指向による階層モデルでは, 暗号化・復号化のレベルの変更は，Proxy クラスと，RemoteContoller クラスに記述の変更が必要となる (図 7)．

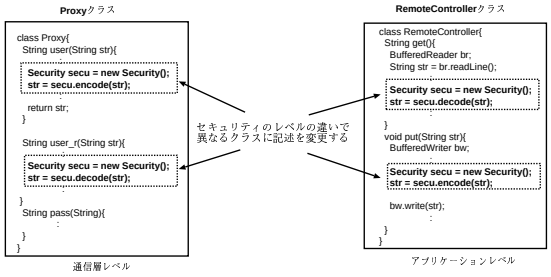

図 7　オブジェクト指向による暗号化・復号化のレベルの変更

例外処理の追加

新たに例外処理を追加することを考えた場合，追加する例外処理のための記述が必要とされる．必要となるロールに例外処理のためのメソッドを追加し，メソッドに対して合流点を設定することで例外処理の追加が可能となる (図 8)．

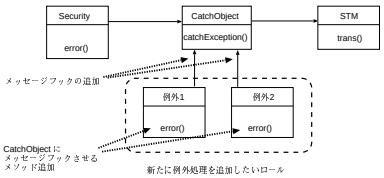

図 8　例外処理の追加

複数の接続の確立

FTP のように複数の接続を用いたアプリケーションプロトコルソフトウェアを実現するために，Application-Protocol フィールドに接続を確立するロールを追加する．そのロールは，接続が必要とされるコマンドと関係を持たせることで実現が可能となる (図 9)．
オブジェクト指向による階層モデルでは，例外の追加が各階層に必要となるので, 整理された構造とはならない．
オブジェクト指向による階層モデルでは，接続を必要とするコマンドのメソッドに接続を確立する記述が必要である．新たな接続を必要とする場合には，同じような記述をコマンドのメソッドに追加する必要がある (図 10)．

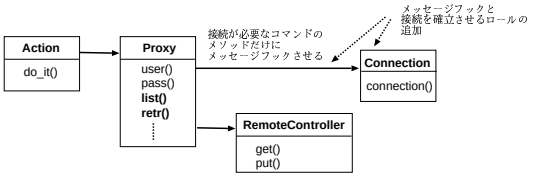

図 9　複数の接続を用いたアプリケーションプロトコルソフトウェア

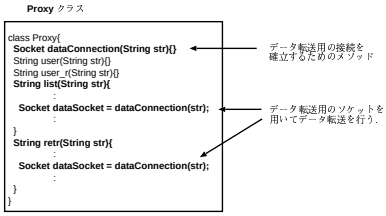

図 10　オブジェクト指向による複数接続の確立

6　おわりに

アプリケーションプロトコルソフトウェアの柔軟性を向上させるためのアスペクト指向ソフトウェアアーキテクチャを構築した．構築したアーキテクチャに基づき，アスペクト指向アプリケーションフレームワークを作成し，いくつかのアプリケーションプロトコルソフトウェアを実現した．ソフトウェアの柔軟性についての考察を行い，アスペクト指向ソフトウェアアーキテクチャの有用性が確認できた．

今後の課題はサーバのアプリケーションプロトコルソフトウェアアーキテクチャの構築と，プリケーションプロトコルソフトウェア以外のソフトウェアへのアーキテクチャスタイルの適用である．

謝辞

本研究を進めるにあたり，熱心な御指導をいただいた野呂昌満教授, 有益なアドバイスを下さった張漢明助教授, 蜂巣吉成講師，大学院生の熊崎敦司先輩，藤原泰昌先輩，森貴彦先輩，後藤修平先輩に深く感謝いたします．

参考文献

[1] AspectJ http://eclipse.org/aspectj/

[2] N. C. Hutchinson and L. L. Peterson : The x-Kernel: An Architecture for Implementing Network Protocols, IEEE Transactions on Software Engineering, Vol.17, No.1, pp.46-45 (1991)

[3] H. Hueni, R. E. Johnson, and R. Engel : A Framework for Network Protocol Software, Proceeding of OOPSLA '95, pp.358-369 (1995)

[4] A. Kumazaki, M. Noro, H. Chang and Y. Hachisu : An Application Framework for TCP/IP Applications, Proceedings of Computer Software and Applications Conference, pp,627-634 (2002)

[5] 熊崎 敦司，野呂 昌満 : アスペクト指向ソフトウェアアーキテクチャスタイルとその実現，オブジェクト指向シンポジウム 2003 特集号 (投稿中) (2003)