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の実現

99B534 今泉富保 99B566 前田晴久
99B620 社本直子

指導教員　野呂昌満

1　はじめに

われわれの研究室では，アスペクト指向ソフトウェアアーキテクチャ [1] を提案した．ソフトウェア開発者の労力を軽減するために，ソフトウェアアーキテクチャにもとづいてアスペクト指向アプリケーションフレームワークを実現することが考えられる．

われわれは，アスペクト指向アプリケーションフレームワークを実現しようとしたときに，既存のアスペクト指向プログラミングの実現方法がアスペクトを記述するのに十分でないという問題を発見した．既存のアスペクト指向技術でソフトウェアを実現するには，アスペクト指向プログラミング言語による実現方法と，デザインパターンによる実現方法 [4] が考えられる．しかし，両者ともアスペクトがコアコンサーンに依存しているようにしか記述できない．

本研究ではこの問題を解決するために，実用性を考慮した一般的なアスペクト指向プログラミングの実現方法を提案する．われわれは一般性を考慮して，既存のプログラミング言語で実現可能という理由から，デザインパターンによる実現方法を選択する．

アスペクト指向プログラミングを実現するには，コアコンサーン側に

- 合流点の設定
- アドバイスを織り込む (AspectJ[5] でいう Weaving) 順序

アスペクト側に，

- アスペクト側のオブジェクトが受信するメッセージの順序

の 3 点を記述しなければならないと考えた．既存のアスペクト指向プログラミングの実現方法ではアスペクト側のオブジェクトが受信するメッセージの記述が考慮されていない．われわれは前者の実現に，アスペクトモデレータフレームワーク [4] とデコレータパターン [6] を用いる．後者の実現には，ファサードパターン [6] とステートパターン [6] を用いる方法を提案する．

アプリケーションプロトコルのアスペクト指向アプリケーションフレームワークを実現し，われわれの提案する方法で実現できることを確認した．

2　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

アスペクト指向ソフトウェアアーキテクチャはアスペクトの集合である．アスペクトは複数の構成要素によって実現される．アスペクトは他のアスペクトとは独立に記述し，アスペクト間の関係は別に記述する．

われわれがこれまでソフトウェアアーキテクチャを構築してきた経験 [2][3] をもとに発見したアプリケーションプロトコルソフトウェアのコンサーン [1] を図 1に示す．

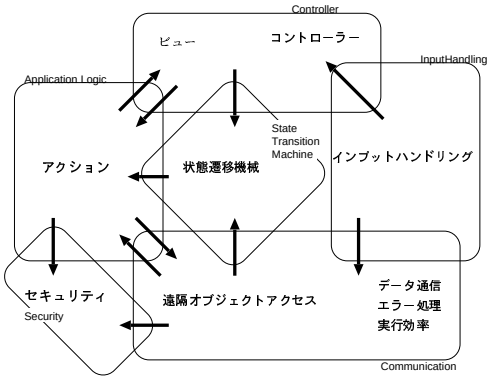

図 1:　アプリケーションプロトコルソフトウェアのコンサーンの関係

2.1　開発技術

アスペクト指向ソフトウェアアーキテクチャを実現する方法としては，次の 3 つが考えられる．

- アプリケーションフレームワーク
- 自動生成系
- コンポーネントライブラリ

われわれは，実現方法としてアプリケーションフレームワークを選択した．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を実現しようとすると，アプリケーション固有の箇所が多い．したがって，コンポーネントライブラリ化には向かない．状態遷移機械は，状態遷移図などからの自動生成も考えられるが，仕様と生成されるコードの隔たりが小さくあまり有効ではない．アプリケーションフレームワークを実現すると，アプリケーションに固有の変更箇所を局所化することができ，高いコードの再利用率が期待できる．

データ通信のコンサーンで使うプロトコルは，複数あり，アプリケーションに固有ではないので，ユーザが選択できるようにコンポーネントライブラリとして提供する．遠隔オブジェクトアクセスのコンサーンでは，仕様を厳密に記述することができ，仕様と生成されるコードの隔たりが大

きいという理由から，自動生成系の技術を用いる．

2.2 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フレームワーク

アスペクト指向ソフトウェアアーキテクチャにもとづいてアプリケーションフレームワークを設計する．アプリケーションフレームワーク設計時には，以下の点を考慮する．

- アスペクト指向ソフトウェアアーキテクチャの構造を素直に反映する．
- デザインパターンを用いて実現する．
- データ通信のコンサーンはコンポーネントライブラリとして，遠隔オブジェクトアクセスのコンサーンは自動生成系として提供する．

設計したアプリケーションフレームワークの各アスペクトを構成するクラスを図2に示す．

- InputHandling
  ユーザからの入力を監視する LocalPort とサーバからの入力を監視する ClientPort を持つ．
- Controller
  ComCreator が，ユーザからの入力を受け取り，コマンド名と引数を保持する Com オブジェクトを生成する． CommandTable が Com オブジェクトの持つコマンド名から対応する処理 (Strategy オブジェクト) を探し，その処理を実行する． Com オブジェクトから，コマンド名を保持する Event オブジェクトを生成する． StdOut は，サーバからの返答を標準出力する．
- ApplicationLogic
  Action オブジェクトは，コマンドの実行と結果に関する処理を保持する．
- StateTransitionMachine
  STM は，現在の状態を表す State のインスタンスを保持する． State は，イベントと次の状態の対を保持し，与えられたイベントに対する次の状態を返す．
- Communication
  Stub が各コマンドに対応するリクエスト文を作成し，サーバへ送信する．次にサーバから来る応答が，今の Action の返答であるとわかるために， ResponseTable に StrategyR をセットする． サーバから返答が来たら， ResCreator が受け取り， Res オブジェクトを生成する． ResponseTable が Res オブジェクトから対応する処理 (セットしてあった StrategyR オブジェクト) を実行する． Res オブジェクトから，コマンド名を保持する Event オブジェクトを生成する．

3 アスペクト指向アプリケーションフレームワークの実現

アスペクト指向ソフトウェアアーキテクチャのアスペクト間の関係を記述するためには，コアコンサーン側に，

- 合流点の設定
- アドバイスを織り込む順序

アスペクト側に，

- オブジェクトが受信するメッセージの順序

が必要であると考えた．既存の技術では，アスペクト側の順序を規定することができない．この節では，以上の3つの記述の方法を述べる．

3.1 コアコンサーン側で合流点にアドバイスを織り込む順序

コアコンサーン側で合流点にアドバイスを織り込む順序は，アスペクトモデレータフレームワークとデコレータパターンを使って記述する．アスペクトモデレータフレームワークはアスペクト指向を実現する既存の技術であり，

- 一般的な技術である
- 複数のアドバイスを織り込むことができる

ということから，この技術を用いる．

アスペクトモデレータフレームワークにデコレータパターンを加えた理由を次に述べる．アスペクト指向プログラミングでは，コアコンサーンを実現するオブジェクトとクロスカッティングコンサーンを実現するオブジェクトを独立して記述したい．しかし，アスペクトモデレータフレームワークでは，図3のクラス GetAction とクラス GetActionWithAspect が is-a 関係になっている．クラス GetActionWithAspect がクラス GetAction に依存していることになってしまう．インタフェースを設けて，クラス GetAction とクラス GetActionWithAspect をそのサブクラスとする．クラス GetActionWithAspect のメソッドがクラス GetAction のメソッドを拡張するために，デコレータパターンを使う．

デコレータパターンとは，オブジェクトのメソッドの拡張を容易にできるようにするデザインパターンの一つである．

クラス図 (図3) の各クラスについて説明する．

- ActionIF クラス
  インタフェースを提供する．
- GetAction クラス
  本来一番したいことが書いてあるクラス．このクラスのメソッド呼び出しが合流点になる．
- GetActionWithAspect クラス

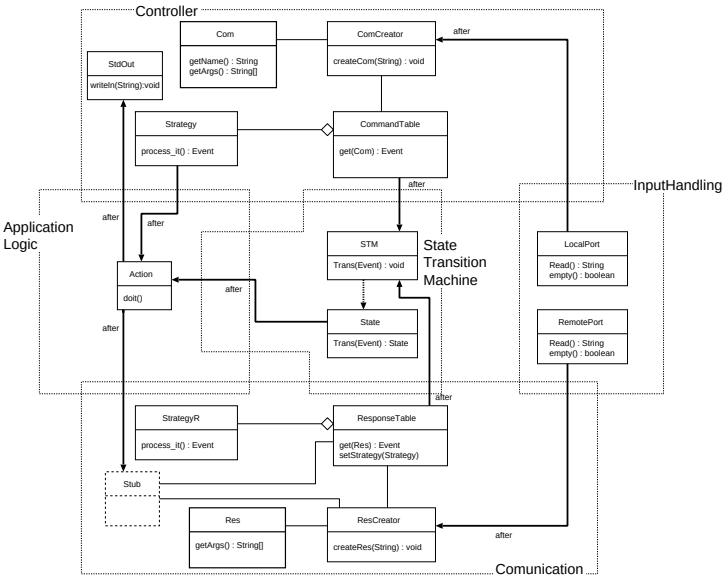

図 2: アプリケーションプロトコルソフトウェアの主要なコンポーネントのクラス図

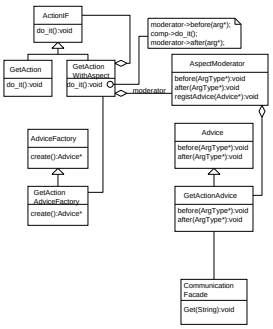

図 3: コアコンサーンのクラス図

アスペクト指向プログラミング言語でいう Weaving されたクラス．デコレータパターンを使っている．

- AdviceFactory クラス
  Advice クラスを生成する，ファクトリーメソッドパターンのオブジェクトのインターフェース．
- GetActionAdviceFactory クラス
  ファクトリーメソッドパターンのオブジェクトなので，GetActionAdvice の構造と同じ構造になる．対比する GetActionAdvice を生成する．
- Advice クラス
  フックオペレーションパターンのオブジェクトのインターフェース．
- GetActionAdvice クラス
  アドバイスを記述する．
- AspectModerator クラス
  複数のアドバイスを保持できる．登録した順にアドバイスを呼び出す．

3.2 アスペクト側のオブジェクトが受信するメッセージの順序

3.1 節で述べたアスペクトモデレータフレームワークとデコレータパターンをもちいても，アスペクト側のオブジェクトが受信するメッセージの順序は記述できない．われわれは，ファサードパターンとステートパターンを用いて，記述する方法を提案する．

ファサードパターンを用いて，アスペクト内のオブジェクトの統一インタフェースを提供する．アスペクト内の複数のオブジェクトが，他のアスペクトのアドバイスからメッセージを受信する場合，それらのメッセージを受信する順序をアスペクト内で制御することは困難である．アスペクト内のオブジェクトへのメッセージはすべてこのインタフェースを通じておこなうものとすれば，メッセージの受信できる順序を制御しやすくなる．

状態遷移機械の実現には実績があるステートパターンを利用する．アスペクト側のオブジェクトが受信するメッセージの順序の制御は，状態遷移機械での実現が可能だと考えられる．たとえば，A アスペクト，B アスペクト，C アスペクトの順でアスペクト内のオブジェクトへのメッセー

ジが送信されるのであれば，Aアスペクト，Bアスペクト，Cアスペクトへのメッセージ送信順にイベントを受け取り，遷移をおこなう状態遷移機械を実現すればよい(図4参照).

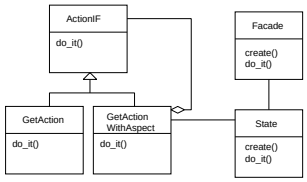

図 4: アスペクトのクラス図

- Facade クラス
  アスペクトを構成するオブジェクトへのメッセージが来たとき，このオブジェクトを必ず通る．ファサードパターンオブジェクト.
- State クラス
  ステートパターンオブジェクトのインタフェース．個々の状態に関する振る舞いを持つクラスをサブクラスで保持している.

3.3 実現

3.1節と3.2節で述べた方法と実現したいアスペクトの具体的な実現方法を示す．アプリケーションフレームワークのクラス図の一部を図5に示す．コアコンサーンにもアスペクトにもなるApplicationLogicを例に挙げた．ApplicationLogicではActionを生成された後にそのActionを実行するという順序が決められている．次にシーケンス図(図6)を使って説明する.

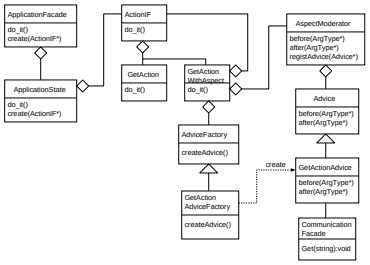

図 5: アスペクト ApplicationLogic のクラス図

1. アスペクト側のオブジェクトが受信するメッセージの順序
   (a) StateTransitionMachine のアドバイスオブジェクトから，ApplicationFacade に do_it メソッドが呼ばれる.

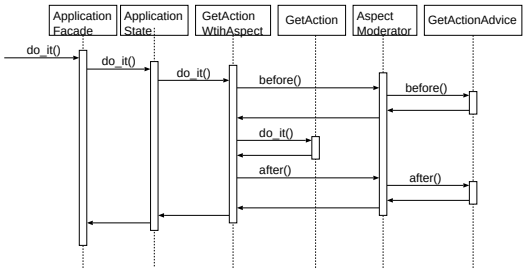

図 6: アスペクト ApplicationLogic のシーケンス図

   (b) 今の状態での ApplicationState の do_it メソッドを呼び出す.
   (c) ApplicationState が保持する ActionIF の do_it メソッドを呼び出し，実行する.
2. コンサーン側で合流点にアドバイスを織り込む順序
   GetActionWithAspect のコンストラクタで GetAction，AspectModerator，GetActionAdvice が生成されており，AspectModerator には GetActionAdvice が登録されている.
   (a) AspectModerator の before メソッドを呼び出す.
   (b) 登録された GetActionAdvice の before メソッドを呼び出す.
   (c) 本来したいことである GetAction の do_it を呼び出す.
   (d) AspectModerator の after メソッドを呼び出す.
   (e) 登録された GetActionAdvice の after メソッドを呼び出す.

以上のようにアスペクト間の関係を記述し，アプリケーションフレームワークを実現した.

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アプリケーションフレームワークの可変部(ホットスポット)を次に示す.

- ComCreator
- CommandTable のサブクラスのコンストラクタ
- Strategy
- StrategyFactory
- StrategyAdvice
- STM のサブクラスのコンストラクタ
- Action
- ActionFactory
- ActionAdvice
- StrategyR

- StrategyRFactory
- StrategyRAdvice

4 アプリケーションプロトコルソフトウェアの試作

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)，HTTP(HyperText Transfer Protocol)，SMTP(Simple Mail Transfer Protocol)，FINGER，ECHO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おこなう場合がある．また，コマンドがないアプリケーションプロトコルよりも，状態遷移が複雑になる.
- HTTP
  サーバからの返答が来たらサーバとの通信を切断する.
- FINGER・ECHO
  コマンドがない.

などの特徴のあるアプリケーションプロトコルを試作したことによって，多くのアプリケーションプロトコルソフトウェアを実現できると確信する.

5 考察

アスペクト指向プログラミングの実現方法として，デザインパターンとアスペクト指向プログラミング言語による実現方法が考えられる．アスペクト指向プログラミング言語を用いた実現方法の利点は，系統的で美しい記述ができるということである．欠点は，あらたにプログラミング言語を覚える必要があるということである．またデザインパターンを用いた実現方法の利点としては，言語依存していないことと，あらたにプログラミング言語を覚える労力を削減できるということがあげられる.

本研究では一般性，実用性の観点から言語依存していないことが望ましいと考えて，デザインパターンを使用した．コアコンサーンの合流点の設定とアドバイスを織り込む順序を，アスペクトモデレータフレームワークとデコレータパターンを用いて記述し，アスペクト側のオブジェクトが受信するメッセージの順序をあらたに，ファサードパターンとステートパターンを用いて記述した.

本節では，

- アスペクトモデレータフレームワークとデコレータパターンを使うことの妥当性
- ファサードパターンとステートパターンを使うことの妥当性

について考察する.

5.1 アスペクトモデレータフレームワークとデコレータパターンを使うことの妥当性に関する考察

コアコンサーン側でアスペクト指向プログラミングを実現するためには，

- コアコンサーンを実現するクラスを独立に記述すること
- 合流点にアドバイスを織り込む記述
- 合流点に織り込むアドバイスの順序の記述

が必要である.

合流点とアドバイスの記述を，以下の6種類のデザインパターンで記述することが可能であると考えた.

- デコレータパターン
- プロキシパターン
- アダプタパターン
- ブリッジパターン
- テンプレートメソッドパターン
- チェインオブレスポンシビリティパターン

アダプタパターンはコアコンサーンを実現するクラスの合流点と，アスペクト指向プログラミング言語でいうWeavingされたクラスの合流点のメソッド名が変わってしまう．コアコンサーンを実現するクラスとWeavingされたクラスは同等に扱いたいので，同じメソッド名を使いたい.

ブリッジパターン，テンプレートメソッドパターン，チェインオブレスポンシビリティパターンの3つは，コアコンサーンを実現するクラスにアスペクトの記述に関するコードを書かなければならない．これではコアコンサーンを独立に記述できていないことになる.

デコレータパターンとプロキシパターンでは両者ともにコアコンサーンを独立に記述することができる．プロキシパターンはコアコンサーンまたはアスペクトの動的な追加・変更ができないが，デコレータパターンではできる．したがって，合流点とアドバイスの記述にはデコレータパターンを用いる.

合流点にアドバイスを織り込む順序を記述するためには，アドバイスをカプセル化し，カプセル化したアドバイスを呼び出す順序を記述すればよいと考えた．アドバイスをカプセル化できるパターンは

- フックオペレーションパターン
- コマンドパターン
- ビジターパターン

の3種類あると考えた.

ビジターパターンはコアコンサーンを実現するクラスにアスペクト記述に関するコードを書かなければならない．コアコンサーンは再利用できるように独立に記述したいので

それは好ましくない.

コマンドパターンはアドバイスをカプセル化し，順序関係を記述することができる．しかしコアコンサーンを実現するクラスとアドバイスのクラスが，同等に扱われてしまう.

フックオペレーションパターンは，ストラテジパターンを拡張したパターンであると考える．ストラテジパターンは1つのアルゴリズムだけをカプセル化することを考えたパターンだが，フックオペレーションパターンはメソッドの前処理，後処理のアルゴリズムをカプセル化することができる.

われわれは，コマンドパターンとフックオペレーションパターンを組み合わせて実現することが，アドバイスをカプセル化する最良の方法だという結論に達した．この方法はアスペクトモデレータフレームワークでおこなわれている．したがって，合流点にアドバイスを織り込む順序の記述には既存の技術である，アスペクトモデレータフレームワークを用いる事が一般的であると考えられる.

以上の理由からコアコンサーン側のアスペクト指向実現に，アスペクトモデレータにデコレータパターンを追加して実現する．コアコンサーンを実現するクラスを独立に記述し，合流点とアドバイスをデコレータパターンを用いて記述をし，合流点にアドバイスを織り込む順序を記述にアスペクトモデレータフレームワークを用いて記述する．この実現方法を他のパターンを使った実現方法と比較することでその妥当性を確認した.

5.2 ファサードパターンとステートパターンを使うことの妥当性に関する考察

アスペクト側のオブジェクトがメッセージを受信する順序は，状態遷移機械で実現可能であると考えた．状態遷移機械で記述する方法には実績のあるステートパターンを用いることが自然であると考える.

アスペクト内の複数のオブジェクトがメッセージを受信する場合，順番を規定することが困難である．そこで以下の2通りの方法を考えた.

- ファサードパターンを用いる
- 複数の状態遷移機械を通信させる

メッセージを受信するオブジェクトがアスペクト内に複数ある場合，そのオブジェクトを状態遷移機械と考える．状態遷移機械同士を並行実行させ，通信させれば，アスペクト内の複数のオブジェクトがメッセージを受信する順序を規定することができる.

しかし，複数の状態遷移機械を通信させる方法は，実現のコスト，実行のコストがかかりすぎるという欠点がある．したがってわれわれは，アスペクト側の複数のオブジェクトがメッセージを受信する方法として，ファサードパターンを用いて実現した.

新たにアスペクト側のオブジェクトが受信するメッセージの順序を，ファサードパターンとステートパターンを用いて実現し，われわれの実現方法の妥当性を確認した.

6 おわりに

われわれは，アスペクト指向プログラミングの実現として，アドバイスを織り込む順序と，アスペクト側のオブジェクトが受信するメッセージの順序の記述が重要と考え，デザインパターンを用いて実現できた．われわれは，一般的なアスペクト指向プログラミングの実現方法が提案できたと考える.

今後の課題として，

- サーバのアプリケーションフレームワークの実現
- ア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スペクト指向プログラミングに実現方法の適用

があげられる.

7 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，蜂巣吉成先生，親身になって相談にのってくださった大学院の熊崎敦司さん，朝長武士さん，山本英貴さん，藤原泰昌さん，森貴彦さんに深く感謝致します.

参考文献

[1] Masami Noro, Atsushi Kumazaki : "On an Aspect-Oriented Software Architecture : it Implies a Process as well as a Product", *Proceedings of APSEC 2002.*

[2] 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"，オブジェクト指向最前線，2001.

[3] 服部賢人 前吉智之 水野正樹 森宰補: " 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卒業論文，2000.

[4] C. Constantinides, A. Bader, T. Elrad and M. Fayad "Designing an Aspect-Oriented Framework in an Object-Oriented Environment," *Computing Surveys 32, 41*, 2000.

[5] AspectJ - Aspect-Oriented Programming (AOP) for Java,

http://aspectj.org/

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1995.