アスペクト指向ソフトウェアにおけるアプリケーション設計支援
—モデル変換による自動設計—

M2011MM079 山下優理
指導教員：野呂昌満

1 はじめに

ソフトウェアには機能，非機能に関わる多くの関心事が横断して存在している．既存のソフトウェア開発方法論では，横断的関心事の系統的な取扱いが不十分で，異なる抽象度のモデル間でそれを実現する構造や意味を一貫して表現することが難しい．これにより，要求から実装への追跡性の確保が困難となる．横断的関心事を分離してモジュール化する技術としてアスペクト指向，モデル間の追跡性の確保をする手段としてモデル駆動開発がある．横断的関心事を考慮して追跡性を確保するためには，アスペクト指向技術とモデル駆動開発の融合が有用である．

本研究の目的は，モデル変換によるアスペクト指向ソフトウェアの設計支援である．コンポーネントとコネクタで表されたアスペクト指向アーキテクチャとオブジェクト指向によってアスペクトの実現方法を示した設計の関係を整理し，カタログとしてまとめる．アーキテクチャから設計へのモデル変換のために，それらの記述ルールのモデルと変換ルールを定義する．これにより，アスペクト指向アーキテクチャから設計への自動化の基礎が確立できる．

アーキテクチャと設計の関係は，Architecture Design Map[2] で示されているように，機能・非機能要求を考慮して整理することができる．アーキテクチャ，設計，機能・非機能要求にはそれぞれ，Clements らが提案するアーキテクチャ文書化スタイルの Component-and-Connector(以下，C&C) Styles，GoF のデザインパターン，ISO9126 を用いる．本研究では，アスペクトの概念を適用するにあたり，C&C Styles，デザインパターン，品質特性の関係をアーキテクチャスタイルが本来持つ特性によって現れる構造と付加的な特性によって現れる構造で再整理する．また，アーキテクチャスタイルの組合せや特性の付加による，クラス間のメッセージ送信の方法を考察する．これにより，アスペクトごとに分離された構造を表すことができる．

アーキテクチャから設計へのモデル変換は，Mattsson らの研究 [1] によって示されているモデル変換によってアプリケーション設計を行う手法を応用する．この手法では，モデル変換によってアプリケーションをアーキテクチャスタイルやデザインパターンに沿って設計するための記述ルールモデルと変換ルールを提案している．本研究では，アスペクトとアスペクト間記述 (以下，IAD) を考慮した記述ルールモデルと変換ルールを新たに定義する．アーキテクチャルールモデルはアスペクトとアスペクトの相互作用について表す．設計のルールモデルには，UML のメタモデルを適用する．変換ルールは作成したカタログを元に定義し，変換メソッドとしてアーキテクチャルールモデル内に OCL を用いて記述する．変換ルールをモデル内に記述することで，自動生成ツールによって，そのまま実行可能となる．

事例検証として，On the Job Learning で開発に取り組んでいる ATM 監視システムを用いて，アプリケーションプラットフォームの設計を行った．結果として，定義した記述ルールモデルと変換ルールの有用性を確認できた．

2 背景技術

2.1 Architecture Design Map

Architecture Design Map[2] は，アーキテクチャ，設計，非機能特性の関係を提供するマップであり，図 1 に示す構成となっている．2 つのマップから構成されており，QualityMap はデザインパターンと非機能特性の関係を示し，Detailed Design Map はアーキテクチャスタイルと非機能特性の関係を示し，アーキテクチャスタイルの実際の設計パターンを Module Style として提供している．

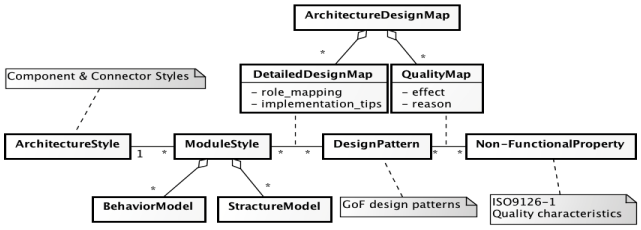

図 1 Architecture Design Map の構成

このマップには，アスペクトの概念は含まれておらず，複数の非機能要求の選択によって設計パターンが決まる．本研究では，アスペクトを適用するにあたって，アスペクトとなる要求ごとに設計パターンを整理し，アスペクト間の相互作用を表すためにパターンの組合せについて整理する必要がある．

2.2 Modeling Architecture Design Rules in UML

Mattsson らは，アーキテクチャを設計するルールをモデル化する方法を提案している [1]．アーキテクチャの設計ルールは，アーキテクトと開発者の理解のしやすさと詳細設計への自動実行のしやすさから，UML を用いて表現することが良いと述べられている．ルールモデルはアーキテクチャのメタモデルで表現されており，操作の制約については OCL を用いて表現している．

このアーキテクチャを設計するルールを示したモデルには，アーキテクチャにアスペクトの概念は含まれていない．本研究では，アスペクト指向アーキテクチャを設

計するので，アーキテクチャの記述ルールモデルを新たに定義する必要がある．その新たに定義したモデルに応じて，設計への変換ルールも定義する．

2.3 Component-and-Connector Styles

C&C Styles は，Clements らが提案するアーキテクチャ文書化スタイルで定義されているアーキテクチャスタイルであり，システムの実行時の振舞いを表している．コンポーネントとコネクタの関係で定義され，全部で 9 つのスタイルがある．このアーキテクチャスタイルを用いることで，実行システムの品質特性について論理的に捉えることが可能となる．これにより，本研究では品質特性を考慮したアスペクト指向アーキテクチャとして用いる．C&C Styles の要素であるコンポーネントはコネクタを介して他のコンポーネントと相互作用するためのポートを持ち，コネクタはどのコンポーネントと相互作用可能かを示すロールを持つ．接続可能なポートとロールの関連付けはアタッチメントで表現する．

2.4 GoF デザインパターン

GoF のデザインパターンは，オブジェクト設計において，よく現れる設計パターンをまとめ，評価したものである．パターンカタログでは，設計問題，その解決策，適用可能性，適用結果，実装方法，適用例を整理し，示している．整理されている設計問題は，システムの保守やコードの再利用など，ソフトウェアに求められる機能・非機能要求であり，要求を考慮した設計を行うのに適している．

2.5 ISO9126

ISO9126 はソフトウェア品質の評価に関する国際基準である．ISO9126 はソフトウェアの持つ様々な特徴を品質の観点から整理したものであり，機能性，保守性などの 6 項目の「品質特性」と品質特性をより細かく分類した 27 項目の「品質副特性」とから構成され ている．ISO9126 は完成したソフトウェアの品質の評価だけではなく，開発前のソフトウェアの機能・非機能要求を定義することも目的としており，機能・非機能要求の整理に適切である．

3 カタログ作成

アスペクト指向アーキテクチャとアスペクトを考慮した設計との関係を整理するにあたり，以下の 3 点について考察した．

- アーキテクチャスタイルの組合せ
- 要求間の相対関係
- アーキテクチャ・設計・要求の関係

C&C Styles は，各スタイルのコンポーネントとコネクタにどのような品質特性が関係するかが示されている．アスペクトの集合でアーキテクチャを表すアスペクト指向アーキテクチャでは，各スタイルのコンポーネントとコネクタをどう組合わせるかを考察する必要がある．アーキテクチャから複数ある設計パターンを選択する際に，その選択指針が必要となる．本研究では，要求を考慮した設計を目的としているので，要求間のトレードオフによって選択基準を与えるために，要求間の相対関係を考察する必要がある．

3.1 アーキテクチャスタイルの組合せ

アスペクト指向アーキテクチャを表すのに複数のアーキテクチャスタイルを組合わせて表現する．

表 1 組合せ可能な C&C Styles

main \ sub	P&F	CS	P2P	SOA	Pub-Sub	SD
Pipe-and-Filter	＼					○
Client-Server		＼			○	○
Perr-to-Peer			＼			
Service-oriented Architecture	○		○	＼	○	○
Publish-Subscribe					／	○
Shared-Data	○		○		○	＼

表 1 は各 C&C Styles 同士が組合わせ可能かどうかをまとめたものである．表の行のスタイルがメインアーキテクチャで，列がサブアーキテクチャとなる．例えば，Pipe-and-Filter をメインとし，Shared-Data をサブとする組合せは，Pipe-and-Filter で処理を行うデータを Shared-Data の Repository で保持するという論理を表し，Pipe 部分を Shared-Data スタイルを用いて記述することを示す．

3.2 要求間の相対関係

複数の品質特性を組合わせると，トレードオフは避けられない．表 2 に品質副特性間の相対関係の一部を示す．「○」は対応する行の特性を上げると列の特性にプラスの影響を与え，「×」はマイナスの影響を与えることを示す．「-」はほとんど影響がないことを示す．この表は対象ではない．例えば，解析性を良くすると試験性は向上するが，試験性を良くしても解析性が向上するとは限らない．

この表を用いて，アーキテクチャから設計する際に，設計のバリエーションの中から選択を行うための指針とする．

表 2 品質副特性間の関係（一部）

		analyzability	changeability	staability	testability	adaptability	installability	co-existence	replacability
maintainability	analyzability	＼	○	×	○	-	-	-	-
	changeability	○	＼	○	○	○	○	○	○
	stability	×	○	＼	○	○	○	○	○
	testability	-	○	-	／	-	-	-	-
portability	adaptability	-	-	-	○	＼	／	○	○
	installability	-	-	-	×	-	＼	-	-
	co-existence	-	-	-	-	-	-	＼	-
	replacability	-	-	-	○	-	○	○	＼

3.3 アーキテクチャ・設計・要求の関係

アーキテクチャ，設計，要求の関係は，C&C Styles を機能・非機能特性を考慮して設計する場合に，どのデザインパターンを適用可能かという視点で整理する．ある C&C Styles について，そのスタイルの適用理由であるスタイル本来の特性によって現れる構造に加えて，部分的に付加できる特性によって現れる構造を整理することで設計にバリエーションを持たせる．図 2 に設計カタログの一部を示す．設計の静的構造はクラス図，動的構造はシーケンス図を用いて表す．例として，Pipe-and-Filter Style の

設計について述べる．Pipe-and-Filter の本来の持つデータ処理の追加・変更が容易という特性（変更性・安定性）をもつ構造を設計するには，Chain of Responsibility パターンと Strategy パターンを用いる．これには，データ処理部分に例外処理，データ暗号化，圧縮などを加えると障害許容性，セキュリティ，資源効率性を持たせることができる．部分的に付加できる特性のパターンとしては，データストリーム時に，適切なタイミングでデータを送るという Pipe 部分に時間効率性を持たせるパターンがあり，Observer パターンを適用することができる．

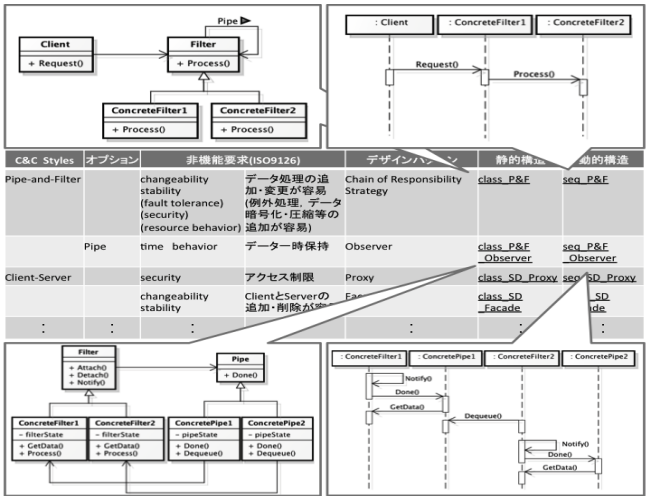

図 2 設計カタログ（一部）

4 ルールモデル

モデル変換のために，アーキテクチャと設計のそれぞれの記述ルールのモデルが必要である．設計はクラス図とシーケンス図を用いて表すこととし，設計の記述ルールモデルに UML メタモデルを用いる．アーキテクチャの記述ルールモデルは C&C Styles のメタモデルを定義して用いる．定義したルールモデルにメソッドとしてモデル変換ルールを記述する．これにより，設計の自動生成ツールを作成する際に，そのまま実行可能なモデルとなる．

4.1 変換ルール

アーキテクチャから設計への変換ルールは，作成したカタログのアーキテクチャと設計の関係から定義する．アーキテクチャのコンポーネント，ポート，ロール，コネクタが設計のどのクラス，メソッドになるかを定義する．例として，パイプ部分にオプションとして Observer パターンを付加した Pipe-and-Filter について示す．

Pipe-and-Filter(Observer) の変換ルール

アーキテクチャ	設計
Filter	→ Filter，ConcreteFilter
output	→ Notify()，GetData()
input	→ Process()
Pipe	→ Pipe，ConcretePipe
datain	→ Done()
dataout	→ Dequeue()

4.2 アーキテクチャルールモデル

アーキテクチャの記述ルールモデルは，C&C スタイルのメタモデルに変換ルールメソッドとして記述したものである．変換メソッドについては OCL を用いて事後条件の制約を記述することで変換ルールをモデルで可視化する．図 3 にアーキテクチャルールモデルの一部を示す．変換メソッドについては一例を部分的に示す．

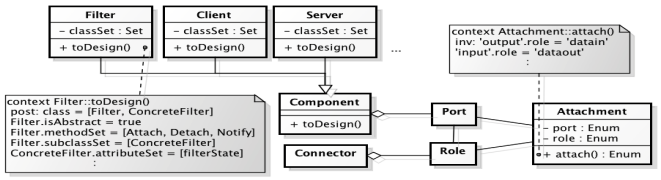

図 3 アーキテクチャルールモデル（一部）

これにより，それぞれアスペクトとなりうるコンポーネント，コネクタ部分とそれらの関係を表す IAD となるアタッチメントを分離して表現することができた．組合せ可能な C&C Styles についてはアタッチメントの中で，重複可能なタイプを定義することで実現する．

5 事例検証

5.1 事例の概要

ATM 監視システムは，様々な銀行の ATM 端末の挙動をネットワークを介して監視し，監視センタのオペレータ端末への通知及び障害対応を行うシステムである．本システムは，アスペクト指向を適用したアーキテクチャ設計を行っており，変更性，セキュリティ，時間効率など多くの非機能要求が求められる．検証には，アプリケーションプラットフォームのメッセージング部分を用いる．

5.2 メッセージング部の設計

メッセージング部分のアーキテクチャは，図 4 に示すように，障害対応及び障害処置を行う各 Web アプリケーション・Web サービス（ServiceConsumer・WebService）間を MessageServiceProvider（以下，MSP）と Registry を介してメッセージ通知するという論理になっている．適用した C&C Styles は，Service-oriented Architecture,

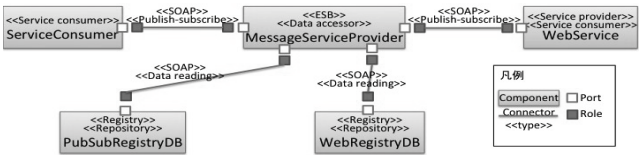

図 4 メッセージング部アーキテクチャ（一部）

Publish-Subscribe，Shared-Data である．アーキテクチャのコンポーネント，ポート，コネクタ，ロールにはそれぞれ適用するスタイルの要素のタイプを記述する．

このアーキテクチャを作成した設計カタログとモデル変換ルールを用いて，図 5 に示すように設計できた．複数の設計バリエーションの中からの選択基準として，各 Web アプリケーション，Web サービス，MSP に独立性

を持たせること，データアクセスオブジェクトの生成とビジネクロジックの分離，データベースの変更しやすさという要求から，重視する品質特性は変更性と安定性とした．これより，要求間の相対関係表を用いて，特に重要でない資源効率性の向上を考慮せずに設計パターンを選択した．

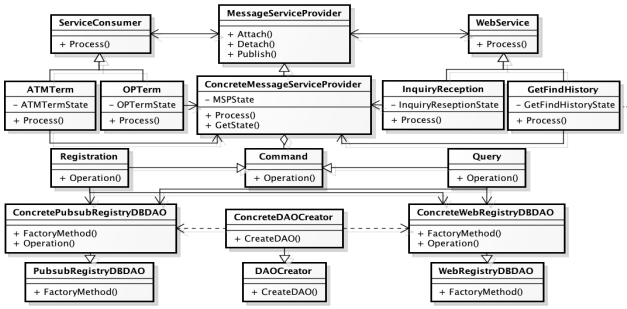

図 5 メッセージング部設計

6 考察

本研究のアプローチ，作成したカタログとルールモデルの妥当性，有用性と今後の課題について考察する．

6.1 品質特性・アーキテクチャ・設計の関係の整理

本研究では，C&C Styles をデザインパターンの集合で表現した．事例検証により，この方法でアーキテクチャから設計へ，非機能要求を一貫して表現することができることが示せた．今回一貫して表現できたのは，変更性や安定性などの保守性に関するものである．これは今回設計に適用したデザインパターンが主に再利用性や柔軟性を重視した設計パターンを提供しているからである．ただし，ソフトウェアにはこの他に時間効率や資源効率などの性能に関する要求も重要視される場合もある．今後，このような保守性以外の非機能要求を実現する方法を他の設計パターンなどから考察し，設計カタログを洗練していく必要がある．

6.2 アーキテクチャ記述

本研究では，アスペクト指向アーキテクチャに C&C Styles を適用してアーキテクチャを記述したが，この表現方法には独自の記法を用いている．今後，自動生成ツールを用いてアーキテクチャを変換することを考え，UML プロファイルなどを用いて一般的な表記方法を定義する必要がある．

6.3 アスペクト間通信

本研究では，アスペクト指向アーキテクチャをオブジェクト指向設計によって実現した．これにより，静的構造の要素についてはアスペクトによって分離できたが，異なるアスペクト間のメッセージ通知については直接アクセスしており，アスペクト間に依存がある．今後，アスペクト間の通信依存をなくすためのパターンを用いるか，AspectJ などのアスペクト指向プログラミング技術を用いることなどを検討する必要がある．

6.4 ルールモデル定義

本研究では，アーキテクチャの記述ルールモデルをクラス図を用いて，C&C Styles のメタモデルで表現した．これにより，アスペクトと IAD を設計者に理解しやすく表現できた．一方で，モデル上のコンポーネントとコネクタは 1 つのクラスとして表現している．今後，設計カタログを洗練し，多くのコンポーネント，コネクタ，両者の関係が定義されることを考慮すると，データベースのような共有データとして，まとめて管理する必要がある．

6.5 変換メソッド

自動生成ツールを作成することを考慮して，記述ルールモデルをそのまま実行可能なようにモデル変換ルールをメソッドとして記述した．モデル変換ルールは，OCL を用いてメソッド実行の事後条件の制約を記述することで表現することができた．しかし，この記述では静的構造の変換ルールしか表現できておらず，設計カタログで示した動的振舞いについては記述ルールモデル上で表現できていない．今後，モデル変換後の振舞いをモデル上に記述して，可視化する方法を検討する必要がある．また，現在変換メソッドは各コンポーネントとコネクタごとに記述されているが，ルールモデルでのデータベースなどの適用に伴って，変換メソッドの一般化も検討する必要がある．

7 おわりに

本研究では，モデル変換によるアスペクト指向ソフトウェアのアプリケーション設計支援を目的に，アスペクト指向アーキテクチャとそれを実現するオブジェクト指向設計の関係を整理したカタログの作成と，モデル変換のための記述ルールモデルと変換ルールの定義を行った．その結果，一部の品質特性についてアーキテクチャから設計まで一貫して表すことが可能となり，その自動化の基礎を作成できた．今後の課題としては，設計カタログの洗練によって今回サポートできなかった要求についてのパターンを考察することと，モデル変換によって変換後の動的振舞いの生成を行う方法の考察である．また，仕様から設計までモデル変換によって設計できるよう，仕様からアーキテクチャへの自動設計手法と統合する必要がある．

参考文献

[1] A. Mattsson, B. Fitzgerald, B. Lundell and B. Lings, “An Approach for Modeling Architectural Design Rules in UML and its Application to Embedded Software,” *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 2, article 10, 2012.

[2] A. Sawada, M. Noro, H. Chang, Y. Hachisu and A Yoshida, “A Design Map for Recording Precise Architecture Decisions,” *Proceedings of the 18th Asia-Pacific Software Engineering Conference*, pp. 298-305, 2011.