ソフトウェア工学 123－6
（1999．5．14）

アスペクト指向設計に関する考察
－ 全体構想 －

野田夏子，岸 知二

NECソフトウェアデザイン研究所

<概要>
性能や信頼性等の品質特性に関する要求の達成は、ソフトウェア開発において非常に重要な課題である。従来の典型的な開発手順においては、まず機能を実現するために必要な構造が設計され、その結果に対して構造の変更や実装方法の決定などにより品質特性を作り込むことが行われてきたが、このような方法にはいくつかの問題があり、またその設計は必ずしも容易ではない。我々は、実現すべき品質特性毎のアスペクトに注目し、各アスペクトに対するソフトウェアアーキテクチャを独立に設計し、それぞれの設計結果を最終的にひとつのソフトウェアアーキテクチャに統合する、アスペクト指向設計(AOD：Aspect Oriented Design)の手法を検討している。AODはまだ構想段階であるが、本稿ではAODの枠組みや妥当性の議論を行うことを目的とし、例題を用いてAODの設計手順の一例を示すとともに、その利点や課題について考察する。

On Aspect Oriented Design
- Basic Idea -

Natsuko Noda, Tomoji Kishi

Software Design Laboratories
NEC Corporation

<abstract>
It is difficult to design software to meat its goal on quality attributes. One of the typical methods is to analyze/design software from functional aspect first, and then to modify or tune up the structure in order to meet required quality attributes. However, this type of design methods has some drawbacks and makes the design work more difficult. We are examining the design method of Aspect Oriented Design (AOD), in which we separately design software architectures from each aspect of quality attribute and weave those architectures into final architecture. Though AOD is under consideration, in this paper, we introduce an example of a design process of AOD, in order to discuss validity of the approach.

1　はじめに

ソフトウェアの開発において、性能や信頼性などの品質特性[1]に関わる要求を満たすように設計をすることは難しく、実際の開発現場においてさまざまな問題が起こっている。

品質特性に関しては、実際に動かしてみないとわからない問題も多い。開発の最終段階で実際に動かしてみて初めて性能問題に気づく等の問題も起こっている。我々は、こうした問題の解決のためには、性能に関する部分的な確認に基づいてソフトウェア全体の性能を有効に補足することが重要であると考え、性能面のソフトウェアアーキテクチャを表現するグラフを用いて、適切な実測対象部分を選定する手法を検討してきた[3][4]。

性能や信頼性等の品質特性を達成するためには、こうした設計結果に対する特性確認の手法とともに、目的とする品質特性を達成するための設計手法も必要である。

従来の典型的な手法では、まず機能を実現する構造が設計され、その設計の結果に対して品質特性の検討を行い、必要に応じて構造に変更を加えたり、実装方法を決定したりすることが多く行われてきた。しかし、機能を実現する構造の枠組みの中に品質特性を作り込もうとするため、考慮すべき制約が多く、設計は一層複雑となる。

そこで我々は、実現すべき品質特性毎のアスペクトに注目し、各アスペクトに対するソフトウェアアーキテクチャを独立に設計し、それぞれの設計結果を最終的にひとつのソフトウェアアーキテクチャに統合するアプローチを検討している。このアプローチをアスペクト指向設計(AOD：Aspect Oriented Design) と呼ぶ。プログラミングの世界では、機能の分割で捉えられないアスペクトを機能とは別にプログラミングするアスペクト指向プログラミング(AOP：Aspect Oriented Programming) [2][5]が提案されているが、AODはこのようなアスペクトへの分割の考え方を設計レベルにまで拡張したものである。

本稿では、現在我々が検討しているAODについて、例題を用いて述べる。AODは現段階では詳細まで検討されたものではないが、AOD そのものの妥当性や枠組みについて議論を行うために、その構想を提示する。2章では品質特性に関わる従来の設計方法の問題点を述べる。3章で我々の検討している AOD のねらいを述べ、4章で現在検討中の AOD の一例を示し、5章で考察を行なう。

なお、本稿では以下、ソフトウェアアーキテクチャのことを単にアーキテクチャと表記する。

2　品質特性設計の問題点

本稿では、性能や信頼性等の品質特性に関する要求を実現できるようにソフトウェアを設計することを、品質特性の設計と呼ぶ。

現時点では、こうした品質特性の設計に関して体系立った方法が整理されている状況では必ずしもない。もちろん、こうした特性が重要である分野においては、様々な方法での設計が行なわれているが、開発の初期からこうした特性の設計を行なうのではなく、後から作り込むのが通常である。すなわち、まずサービスを分析して機能を実現するための構造を設計し、その結果に対して、性能や信頼性が実現できるかどうかの検討を行い、必要に応じて構造に変更を加えたり、実装方法を決定したりする方法が多く取られている。

こうした方法には、以下のような問題点がある：

- **開発初期に特性の確認ができない**
  初期には、サービスを機能面からのみ分析するので、品質特性が達成できるかどうかの確認を行なうことができず、後から問題が発見されることが起こり得る。品質特性が重要である分野においては、開発上のリスクが大きい。
- **品質特性設計における制約が大きくなる**
  サービスの分析から決定された構造の枠組みの中で品質特性の設計を行

うため、考慮すべき制約が多くなり、設計が難しくなる。

3　アスペクト指向設計 AOD

3.1 ねらい

2章で述べたように、品質特性を後から作り込むという従来の設計方法には問題がある。

我々は、品質特性を後から作り込むのではなく、品質特性を設計の初期から機能とは独立に設計することが有効であると考える。

そこで、我々は実現すべき品質特性毎にそれに関わるソフトウェアのアスペクトに注目し、各アスペクトに対するアーキテクチャを独立に設計した後に、それぞれの設計結果をひとつのアーキテクチャに統合するアスペクト指向設計 AOD を検討している。AOD のねらいを以下に述べる：

- **問題を小さくして設計を容易にする**
  各アスペクト毎に独立に設計することから、それぞれの設計における制約が小さくなり、設計が容易になる。
- **初期段階における品質特性の確認を容易にする**
  設計の初期から品質特性に関わるアスペクトに注目したアーキテクチャの設計を行なうので、設計したアーキテクチャで品質特性の達成が可能であるか、早期から検討することができる。
- **それぞれの特性に適した構造を蓄積できる**
  機能に依存せずに各アスペクト毎の設計を行なうことから、それぞれの特性に適したアーキテクチャを機能とは独立に蓄積することができる。

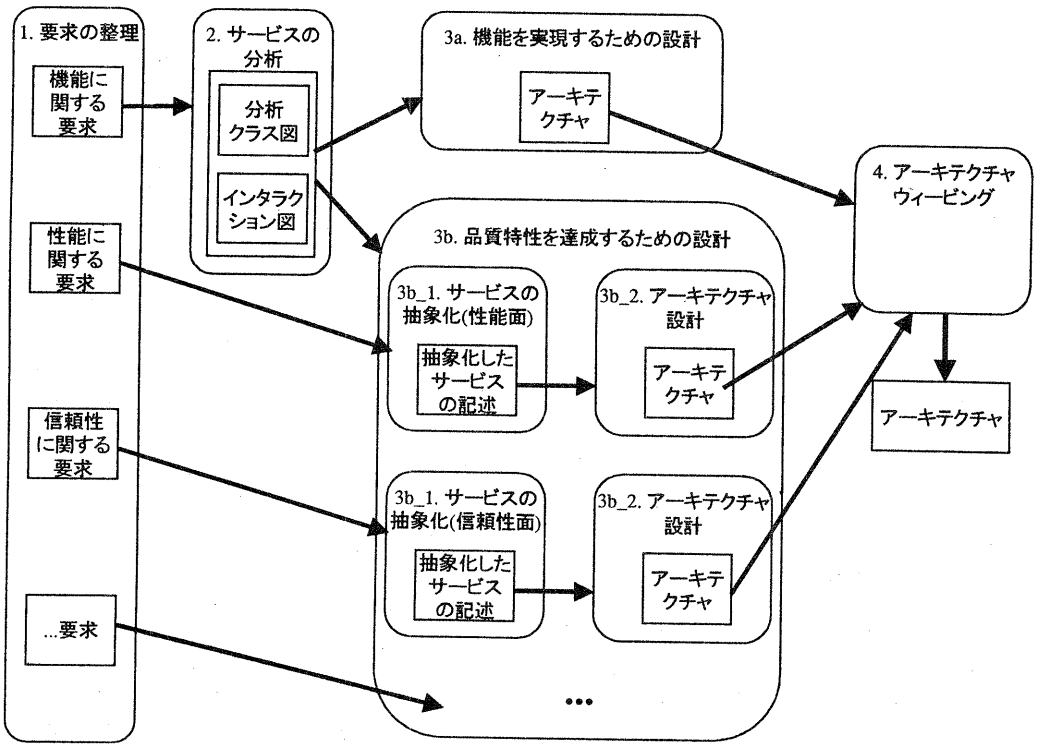

図 1　AOD の概略

3.2 AOD の概略

図 1 に AOD の設計手順を示す。それぞれの段階においては、以下を行なう。

1. **要求の整理**
   機能面だけでなく品質特性に関する要求についても整理する。サービス毎に達成すべき品質特性は異なるので、サービス毎に要求されている品質特性を明らかにする。
2. **サービスの分析**
   サービスを分析して、必要とされる情報や機能を分析レベルで明らかにする。例えば、クラス図やインタラクション図等を作成する。

3a. **機能を実現するための設計**
   分析結果に基づき、前提としている実装技術で実装するための、機能面の設計を行う。

3b. **品質特性を達成するための設計**
   以下の手順からなる：

   3b_1. **サービスの抽象化**
   品質特性の面からサービスを抽象化する。

   3b_2. **アーキテクチャ設計**
   抽象化されたサービスから、アスペクトに注目したアーキテクチャを設計する。

4. **アーキテクチャウィービング**
   機能を実現するための設計、品質特性を達成するための設計から得られたそれぞれのソフトウェアアーキテクチャを統合する。

これらの手順のうち、品質特性を達成するための設計とアーキテクチャウィービングに関しては、次章以下で例題を用いて説明する。他の手順に関しては、従来の設計と同様である。

4 AOD の一例 - 性能設計への応用

現在我々は、品質特性として特に性能を取り上げて AOD を検討している。本章では、例題を用いて性能のアスペクトに注目した AOD について説明する。

4.1 例題

例題として、情報端末を取り上げる。この情報端末は、街にあるお店と公共施設に関する情報を提供する。以下に示すサービスを提供する簡単なものとする。

- 最寄り駅が X である公共施設の名称の一覧を示す。(S1)
- X という名称の公共施設の電話番号を示す。(S2)
- 最寄り駅が X であるお店の名称の一覧を示す。(S3)
- X という名称のお店の電話番号を示す。(S4)
- お店の説明文の全文サーチを行なう。(S5)

それぞれのサービスについて要求される性能に関しては、S1、S3 は 2 秒以内、S2、S4 は 1 秒以内、S5 は 5 秒以内に終了することが求められているものとする。

サービスの分析を行って得たクラス図を図 2 に示す。

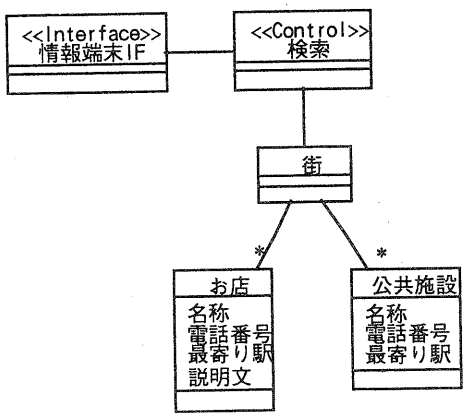

図 2 分析クラス図

4.2 性能に関わるアスペクトからの設計

4.2.1 性能面からの抽象化

この例題においては、データアクセスが性能上のひとつのポイントになる。したがって、データ量やデータ構造が重要であ

る。こうしたアスペクトに関して、それぞれのサービスを抽象化する。

まず、それぞれのサービスについて、関連するデータ量やデータ構造を明らかにしなければならない。この例題においては、以下がわかっているとする：

- 公共施設の個数は最大 100 個。
- お店の個数は最大 10000 個。
- お店の説明文については、その量に上限は定められておらず、各お店により差が非常に大きい。したがって、説明文は可変長データとして扱う。

さらに、こうした条件においてどのような実装上の制約があるかを明らかにする必要がある。この例題に対しては、以下の実装上の制約がある：

- 連続領域に格納されたデータ集合に対する検索時間は個数に比例し、100 個のデータ集合に対して最大 0.1 秒、10000 個のデータ集合に対して最大 10 秒かかる。
- 不連続領域に格納された 1000 個を越えるデータ集合に対しての検索は、10 秒以上かかる。

性能のアスペクトに注目するため、以上のようなデータ量、データ構造に関する仮定と実装上の制約をもとに、各サービスの記述からサービスの意味を落して、データ量、データ構造面からの記述に抽象化する。まず、各サービスの記述をデータ量とデータ構造の記述に単純におきかえると、以下が得られる：

- S1 → 100 個のデータ集合に対して条件に合致するデータの固定長である属性の一覧を示す。
- S2 → 100 個のデータ集合に対して条件に合致するデータの固定長である属性を示す。
- S3 → 10000 個のデータ集合に対して条件に合致するデータの固定長である属性の一覧を示す。
- S4 → 10000 個のデータ集合に対して条件に合致するデータの固定長である属性を示す。
- S5 → 10000 個のデータ集合に対して可変長である属性の集合をサーチする。

次に、実装上の制約を考慮して、各データ集合が特段の考慮をしなくても性能を達成できる許容範囲内のデータ集合であるのか、そうでないのかを判断する。さらに、そのデータ量、データ構造においては特別に扱う必要のない処理であれば、その記述も省略する。例えば、S1、S2 において属性の一覧を示すのも属性を示すのも、性能面から見れば属性を検索する性能が問題であり、この属性が固定長であるか可変長であるかは問題ではない。このような抽象化を行なうと、最終的に以下が得られる：

- 許容範囲内のデータ集合に対して属性を検索する。 → SA1 とする。
- 許容範囲外のデータ集合に対して条件に合致する属性を検索する。 → SA2 とする。
- 許容範囲外のデータ集合に対して可変長である属性の集合をサーチする。 → SA3 とする。

4.2.2 性能面からのアーキテクチャ設計

性能面に関してデータ量とデータ構造から抽象化した各サービス SA1～SA3 について、それぞれを達成できる構造を検討する。

SA1 に関しては、性能に対する特段の配慮が不要のため、機能を実現するために十分な構造であれば良い。したがって、図 3 に P1 で示す構造でサービスは実現できる。

SA2 に関しては、単純な集合で管理していては検索に関して性能目標を達成できない大量のデータ集合に対して、検索を速く行なえる構造を与える必要がある。そこで、検索の条件をインデックスとして検索を高速化する構造にする(図 3 の P2)。

SA3 に関しては、可変長である属性の集合のサーチを高速にする構造を与えなければならない。そこで、可変長である属性だけを他の固定長の属性とは別にひとつ

の連続した領域に管理することが有効であると考えられる(図3のP3)。

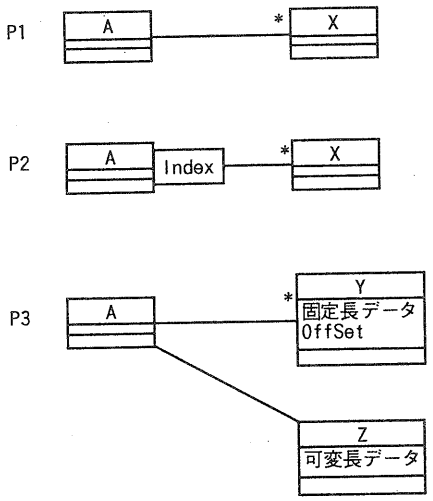

図 3 性能面からのアーキテクチャ

4.3 アーキテクチャウィービング

機能を実現するための設計の結果と品質特性を達成するための設計の結果を統合して、最終的なアーキテクチャを得る。この例題においては、機能を実現するための設計におけるクラス図は、分析結果のクラス図と変わらない。

まず、もともとのサービスS1～S5と性能面から得た構造P1～P3の対応を整理する(表1)。

表 1 サービスと性能面からのアーキテクチャの対応

	P1	P2	P3
S1	○		
S2	○		
S3		○	
S4		○	
S5			○

この対応をもとに、それぞれのサービスの機能を実現するクラスに対して性能面から得られた構造を対応づけると、以下のようになる(表2)。

表 2 機能を実現するクラスと性能を達成するためのクラスの対応

		P1	P2	P3
S1、S2	街	A		
	公共施設	X		
S3	街		A	
	お店		X	
	最寄り駅		Index	
S4	街		A	
	お店		X	
	名称		Index	
S5	街			A
	お店			Y、Z

この対応関係を整理すると、図 4に示すアーキテクチャが得られる。

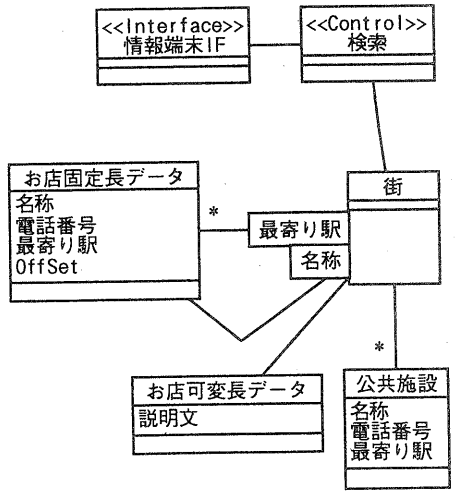

図 4 統合されたアーキテクチャ

なお、AOP ではウィービングはツールにより自動的に行なうものとしているが、AOD においては人手によるウィービングのみでも十分に効果があるものと考える。

5 考察

本稿で述べたAODは検討中のものであり詳細まで検討されたものではないが、全体像の提示により、AODのアプローチそのものの利点や課題を考察する材料となると思われる。本章では、今回の例題での検討から、AODの利点と思われたもの、今後の検討課題と考えられることを述べる。

5.1 利点

従来、典型的に行われてきた設計においては、まず機能を実現するために構造を設計し、その結果に対して品質特性を作り込む方法が取られてきた。こうした方法では、機能の実現と(複数あり得る)品質特性の達成を同時に考慮しなければならず、設計は一般に容易でない。AODにおいては、達成すべき品質特性に関わるアスペクト毎に独立に設計を行う。それぞれのアスペクト毎の設計においては、ひとつのアスペクトのみに注目しており、複数のアスペクトに対する考慮の必要がないため、設計がわかりやすくなる。

また、この手法においては、アスペクトに注目してサービスの抽象化を行うため、設計すべき問題を小さくすることができる。つまり、各サービス毎にそこで必要とされる品質特性を設計するのではなく、アスペクトからの抽象化によりサービスのグルーピングを行い、抽象化されたサービスに対してそれを実現する構造を設計すれば良い。例えば、4章で取り上げた例題においては、全体では5つのサービスを実現しなければらないが、データアクセスのアスペクトからサービスの抽象化を行うと3つのサービスに抽象化することができる。このような抽象化により設計すべき問題が小さくなる。

5.2 課題

今後検討しなければならない課題を以下に示す。

- **注目すべきアスペクトの決定方法：**
  品質特性を実現するための設計において、どのようなアスペクトに注目するのかは、品質特性に関する要求と実装において利用する技術に関する知識を基に決定される。本稿で取り上げた例題では、情報端末における大量のデータから検索を行うサービスにおいて、データアクセスが性能上のポイントになることがわかっていたために、これをアスペクトとして取り上げた。しかし、どのアスペクトに注目するのかは非常に重要なことであるので、注目すべきアスペクトを決定する方法について体系的な整理を行う必要がある。
- **アーキテクチャウィービングの方法の整理：**
  アーキテクチャウィービングについては、アスペクト毎に設計されたアーキテクチャを人手により統合することは可能であることを、例題を用いて示した。しかし、複数のアーキテクチャを統合するための一般的な方法の整理はできていない。もし、アーキテクチャウィービングの方法が繁雑なものになるなら、個々のアスペクトからの設計は容易になっても全体としての設計は難しくなり、アスペクトに分割することの利点が小さくなる。
  また、本稿では、データアクセスに関するアスペクトのみを取り上げたが、一般にはひとつのソフトウェアの設計において複数のアスペクトに注目することが必要である。複数のアスペクトに対してそれらから導かれたアーキテクチャの統合の方法も必要である。
- **アーキテクチャの評価方法の整理：**
  一般に、複数のアーキテクチャを統合して得られるアーキテクチャはひとつとは限らない。例えば、性能とサイズとのトレードオフを考慮する

等、相反するアスペクト間の優先度やトレードオフにより、最終的なアーキテクチャを選び出している。したがって、AOD により最終的に最も適当と考えられるアーキテクチャを得るためには、アーキテクチャの評価方法が必要とされる。こうした点についても、今後検討が必要である。

6 おわりに

性能や信頼性等の品質特性の設計は重要であるが、従来その設計手法について体系的に整理されていたわけではない。本稿では、品質特性設計のために現在我々が検討を進めている AOD について、例題をベースに考察を行った。

今後は具体的な問題に適用できるように、さらに手法の整理を行いたい。

参考文献

[1] Bass, L., et.al.: *Software Architecture in Practice*, Addison Wesley, (1998).

[2] Kiczales, G., et.al.: Aspect-Oriented Programming, Proceeding of ECOOP'97, (1997).

[3] 野田,他: 性能設計における協調/依存グラフの活用, 日本ソフトウェア科学会, FOSE'98, (1998)

[4] Noda, et.al., An Architectural Approach to Performance Issues - from Experiences in the Development of Network Management Systems -, First Working IFIP Conference on Software Architecture, (1999)

[5] http://www.xerox.parc.com/aop