Vol.2014-SE-186 No.17
2014/11/14

アスペクト指向を利用したアーキテクチャ設計に関する考察

野田夏子[†1]　　岸知二[†2]

ソフトウェアおよびソフトウェア開発にとって、ソフトウェアアーキテクチャは非常に重要である。その一方で、今日ソフトウェアアーキテクチャの設計はますます難しくなってきている。それは、堅牢なソフトウェアアーキテクチャが、しばしばサイズの増大と変化への迅速な対応に対して衝突するからである。我々は、サイズと変化に対応できる柔軟なソフトウェアアーキテクチャの実現のために、アプリケーションとの支配関係の逆転を実現するソフトウェアアーキテクチャについて検討を始めている。本稿では、現在検討中のアスペクト指向を利用した柔らかいソフトウェアアーキテクチャの設計について述べる。

About Design of Software Architecture using Aspect-Orientation

NATSUKO NODA[†1]　TOMOJI KISHI[†2]

Software architecture is very important for software itself and development of software. However, designing of software architecture is getting more and more difficult, because of scalability of software and agility of software development. We are considering software architecture design using aspect-oriented technologies, which enables inversion of dominance relationship between software architecture and application core functionalities. In this paper, we describe our basic idea of this software architecture design.

1. はじめに

ソフトウェアアーキテクチャとは、ソフトウェアおよびその開発を支配するソフトウェア構造や構造化の原則を意味する 4)。これは、ソフトウェア開発の方針や方向付けを行うために必要なソフトウェアのベースとなる構造を示すものであり、ソフトウェアおよびソフトウェア開発にとって重要な意味を持つ。なぜなら、ソフトウェアの構造は品質特性を決定づける大きな要因のひとつであり、ソフトウェアアーキテクチャが適切でないと求められる品質特性を達成することが難しくなるからである。また、ソフトウェアの巨大化・複雑化がますます進む中で、ソフトウェアの再利用は避けられなくなっており、その際にはソフトウェアアーキテクチャの明示化と共有を適切に行うことが求められているのである。したがって、現代のソフトウェア開発においては、ソフトウェアアーキテクチャを適切に設計することが非常に重要である。

その一方で、ソフトウェアアーキテクチャの設計はますます難しくなってきている。それは、堅牢なソフトウェアアーキテクチャが、しばしばサイズの増大と変化への迅速な対応に対して衝突するからである。今日、クラウドの台頭に見て取れるように、急速に増大するサイズへの対応がソフトウェア開発において課題となっている。しかし、サイズへの対応は、しばしばソフトウェアアーキテクチャの変更を要求する。また、アジャイル開発が注目を集めているが、これはソフトウェア開発において変化は避けられないものであり、その変化への迅速な対応がソフトウェア開発上の大きな課題であることがひとつの理由であろう。アジャイル開発は、ともするとアーキテクチャなどの設計行為の軽視と誤解されることもあるが、アジャイル開発であるからといってソフトウェア構造を無視して良いわけではない。しかし変化への対応の中で、どのようにソフトウェアアーキテクチャを捉え、どのように設計していくのかは大きな問題となっている。

ソフトウェアアーキテクチャは、従来ソフトウェアのベースとなる「堅牢な」構造であると考えられ、いかにして堅牢なアーキテクチャを設計するかが課題とされてきた。しかし、近年のソフトウェアを取り巻く環境を考えた時、むしろサイズと変化に対応できる柔軟なソフトウェアアーキテクチャが必要であると考えられる。我々は、この柔軟なソフトウェアアーキテクチャを実現するために、アプリケーションとの支配関係の逆転を実現するソフトウェアアーキテクチャについて検討を始めている。この支配関係の逆転には、アスペクト指向技術を利用する。

本稿では、現在検討中のアスペクト指向を利用したソフトウェアアーキテクチャ設計について述べる。なお、本稿では、以降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呼ぶことにする。

2. アーキテクチャ設計の課題

2.1 スケーラビリティの確保

近年、ソフトウェア開発でしばしば問題になるのが、スケーラビリティの問題である。つまり、様々な意味でソフトウェアのサイズは大きくなっており、また開発や進化の過程でもさらにサイズが増大するということが起きている。

†1 芝浦工業大学
Shibaura Institute of Technology
†2 早稲田大学
Waseda University

例えば、ソフトウェアプロダクトライン開発においては、近年スケーラビリティに関する問題が多く指摘されている 2)3)10)。

スケーラビリティには、いくつかの方向がある。

- 機能の増大：ソフトウェア開発においては、しばしばソフトウェアが持つべき機能が増える。単一のソフトウェアは、その進化の過程で機能が増えることがある。また、プロダクトライン開発を行っている場合に、プロダクトライン自体の進化が起こり、プロダクトラインアーキテクチャ上で実現すべき機能が増大することも起こっている。また、機能の増大には、ソフトウェアが実現する機能の種類自体は増えないが、その機能を実現するためのインスタンスが増大するという場合もある。
- データの増大：ソフトウェアが扱うデータ量が増大することもしばしば起こり得る。機能の増加に伴いデータ量が増大する場合もあるし、機能とは独立に、ある機能が扱うデータ量自体が増大する場合もある。

これらのサイズの増加は、アーキテクチャに影響を与える。一般に、単機能を実現する場合と多様な機能を実現する場合では、適したソフトウェア構造は異なる。また、機能のインスタンスが増えれば、それらインスタンスをどう配置するのが最適かは異なってくる。また、少量のデータを扱う場合と、大量のデータを扱うのでは、その処理のパターンが変わり得るので、最適なアーキテクチャは異なる。

このようにスケーラビリティの問題は、アーキテクチャの設計を難しくしている。

2.2 アジリティの実現

今日、ソフトウェアを取り巻く様々な環境の変化のスピードは非常に速い。また、ユーザの要求は様々に変化するものである。こうした中で、以下に迅速に変化に対応できるかがソフトウェア開発において大きな課題となっている。

このような問題意識を踏まえ、今日ではアジャイル開発が注目を集め、様々な実践も報告されている。アジャイル開発では、「包括的なドキュメントよりも動くソフトウェアを」と言われるため 1)、コーディング作業がその開発の中心のように捉えられ、アーキテクチャ設計は重要でないかのような誤解もあるが、アジャイル開発であることは必ずしもアーキテクチャの軽視を意味しない。開発スタイルがどうであろうと、アーキテクチャが重要であることは変わりない。むしろ、迅速な開発のためには再利用は必須であり、再利用のためにはアーキテクチャの明示化と共有が必須である。

しかし、様々な変化への対応はアーキテクチャの再構成を必要とし、それが迅速な開発を妨げるという問題も報告されている。例えば、Weitzel らは、実開発における Scrum の実践において、アーキテクチャの再構成への対応から徐々に開発のペースが落ちることを報告している 11)。また、この問題に対して、Weitzel らは、ユーザストーリー5)に即したストーリーアーキテクチャを導入し、アジリティを実現する手法を提案している 11)。

このように、従来的な堅牢なアーキテクチャでは変化への迅速な対応が難しくなってきている。

3. アーキテクチャの支配関係の逆転

前章で述べたように、サイズと変化への対応の必要から、アーキテクチャの設計は非常に難しくなっている。

従来のアーキテクチャは、「固い」ものであり、アプリケーション構造を支配するものと考えられてきた。つまり、本質的なソフトウェア構造であるアーキテクチャを設計し、そのアーキテクチャの上にアプリケーションを設計するのである。例えば、アーキテクチャとして 3 層アーキテクチャを採用すると決め、そのアーキテクチャ上にアプリケーションの機能を配置することにより、アプリケーションを設計する等が行われる。このように、アーキテクチャがアプリケーション構造を支配するのである。

しかし、このような堅牢なアーキテクチャでは、前章で述べたようなサイズと変化に対応することは難しい。アーキテクチャの上に、それに支配されるアプリケーションを設計するのでは、アプリケーションのサイズの増大や迅速な変化をアーキテクチャ側は受け止めきれない。サイズや変化に対応できる、より「柔らかい」アーキテクチャが求められる。

ソフトウェアが果たすコアな機能の構造は、本来はアーキテクチャとは独立したものである。しかし、従来のアーキテクチャ設計では、アーキテクチャを設計し、その上にソフトウェアの機能を実現する構造を設計するため、アーキテクチャは固いものとなっている。アプリケーションのコア機能がアーキテクチャに依存する、あるいはアプリケーションのコア機能がアーキテクチャに支配される関係にある。我々は、アプリケーションのコア機能とアーキテクチャの間に存在するこの依存性を排除し、支配関係を逆転させることが、柔らかいアーキテクチャを設計する上で重要であると考える。

4. アスペクト指向を利用したアーキテクチャの設計

柔軟なソフトウェア構造を実現するための技術として、アスペクト指向技術の利用が考えられる。アスペクト指向は、単一視点からなる分割(クラス分割、モジュール分割等)では分割された単位をまたがって現れる横断的な関心事をアスペクトとして分離することにより、単一の視点からの構造に対して柔軟性を与えることができる。こうした特徴を生かして、例えばプロダクトラインにおけるバライアビリティの実現方法としてアスペクト指向技術を活用するなどが行われている。

我々は、支配関係を逆転させた柔らかいアーキテクチャの設計に、アスペクト指向を活用することを検討している。これまでに、デザインパターンをアスペクト指向を用いて実装する手法を提案した 6)7)。この手法では、パターンに依存してアプリケーションの構造を設計するのではなく、アプリケーションのコア機能はパターンとは独立に設計し、コア機能中のクラスを、デザインパターンが規定する役割関係を示すクラス構造中のクラスと対応付けることにより、デザインパターンが提供する機能をアプリケーションで実現する。図 1 は、本手法による Observer パターンの実現を、簡略化して表したものである。この例では、アプリケーションのコア機能である Counter(増減する数をカウントするクラス)、Button(カウントした数を表示するボタン)、Display(表示一般を行うディスプレイ)がアプリケーションのコア機能として Observer パターンとは独立に設計されており、Observer パターンの役割関係を示す Subject、Observer がアプリケーションに独立なパターンとして設計されている。そして、アプリケーションのコア機能とパターンをアスペクト指向を利用することにより対応付けている。このような構造にすることにより、ソフトウェア構造の枠組みを示すデザインパターンとアプリケーションのコア機能の依存関係を逆転させ、コア機能を変更せずに、利用するパターンを切り替えることを実現した。

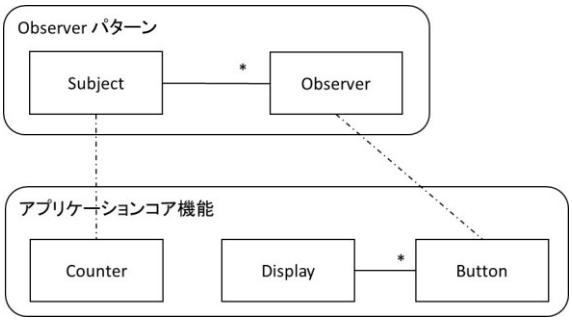

図 1 アスペクト指向によるデザインパターンの実現

6)7)で提案した手法では、アスペクト指向プログラミング言語を利用したプログラミングレベルでの設計の分離であるが、その後我々は関心事毎にアスペクトとして分離した構造をアスペクト間を関連付けるルールにより対応づけて全体の設計を行う、アスペクト指向設計手法を提案した 8)9)。この手法を用いることにより、実装言語によらず設計段階でデザインパターンが示す構造とアプリケーション構造を分離できることを確かめている。

デザインパターンは、実装に近いレベルでの構造を与えるものであり、これまでに提案した手法はそうしたレベルでの構造にアスペクト指向技術を利用して支配関係の逆転を実現するものである。しかし、8)9)で提案したアスペクト間をルールにより対応付ける手法を応用すれば、より大きなアーキテクチャ構造に対しても、アーキテクチャとアプリケーションのコア機能の支配関係の逆転が可能になると予想される。

また、2 章で述べたように、アーキテクチャ設計を難しくする要因としてサイズと変化の問題があり、さらにサイズに関していくつかの方向性があるが、これらのうちのどの問題に対して、アスペクト指向による支配関係の逆転の有効性が高いのかは、今後手法の洗練とともに精査が必要な課題であると考える。

5. おわりに

従来、ソフトウェアアーキテクチャはソフトウェアのベースとなる「堅牢な」構造であると捉えられてきたが、近年のソフトウェアを取り巻く環境の変化の中で、サイズと変化に対応できる柔軟なソフトウェアアーキテクチャが必要とされるようになってきている。本稿では、この柔軟なソフトウェアアーキテクチャを実現するための、アスペクト指向技術を利用した、アプリケーションとの支配関係の逆転を実現するソフトウェアアーキテクチャについて述べた。本稿は、今後のソフトウェアアーキテクチャ設計についてのビジョンを示すものであり、アーキテクチャ設計手法の詳細についてはまだ検討途上である。柔らかいアーキテクチャの実現のために、今後より詳細に手法を検討していきたい。

参考文献

1) アジャイル開発宣言, http://agilemanifesto.org/iso/ja/manifesto.html
2) Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE'09) held at the 13th International Software Product Line Conference (SPLC'09), 2009
3) The Second Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE'10) held at the 14th International Software Product Line Conference (SPLC'10), 2010
4) 岸知二, 野田夏子, 深澤良彰: ソフトウェアアーキテクチャ. ソフトウェアテクノロジーシリーズ, 共立出版 (2005).
5) Cohn, M.: User Stories, Epics and Themes, Succeeding with Agile -Mike Cohn's Blog, 2011. [Online]. Available: http://www.mountaingoatsoftware.com/blog/stories-epics-andthemes.
6) Noda, N., and Kishi, T.; Implementing Design Patterns Using Advanced Separation of Concerns, OOPSLA 2001 Workshop on AsoC in OOS. (2001).
7) Noda, N., and Kishi, T.; Design pattern concerns for software evolution, Proceedings of the 4th international workshop on principles of software evolution. ACM (2001).
8) Noda, N., and Kishi, T.; Aspect-Oriented Modeling for Embedded Software Design. In Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific. IEEE (2007)
9) Noda, N., and Kishi, T.; Aspect-oriented modeling for variability management, In *Software Product Line Conference, 2008. SPLC'08. 12th International*. IEEE (2008).
10) Simone, S. A., S. Almeida, E.S., and McGregor, J. D.: Scalability of Ecosystem Architectures, 2014 IEEE/IFIP Conference on Software Architecture (WICSA) (2014)
11) Weitzel, B., Rost, D., and Scheffe, M.: Sustaining Agility through Architecture, 2014 IEEE/IFIP Conference on Software Architecture (WICSA) (2014)