Vol.2016-SE-191 No.27
2016/3/15

柔軟なプロダクトラインアーキテクチャ設計に関する一考察

野田夏子[†1]　岸 知二[†2]

概要：近年，ソフトウェアプロダクトライン開発の概念が広く知られるようになり，さらに様々な現場への適用プロセスも提案されるようになったことから，多様なドメインにおいてこの開発形態が普及しつつある．ソフトウェアプロダクトラインにおいて，全体のベースとなるプロダクトラインアーキテクチャの設計は非常に重要であるが，近年の変化の早い環境にあっては，より多くのバリエーションを含む傾向にあり，当初設計したプロダクトラインアーキテクチャの変更が余儀なくされるような事態も起こり得るようになっている．本稿では，このような状況に対処するために，現在検討中のアスペクト指向を用いたアーキテクチャの柔軟化について述べる．

A Brief Study on Design of Flexible Product Line Architecture

NATSUKO NODA[†1]　TOMOJI KISHI[†2]

1. はじめに

近年，ソフトウェアプロダクトライン開発の概念が広く知られるようになり，さらに様々な現場への適用プロセスも提案されるようになったことから，多様なドメインにおいてこの開発形態が普及しつつある．

ソフトウェアプロダクトライン(以降，プロダクトライン)とは，共通の管理された特徴を持ち，特定のマーケットやミッションのために，共通の再利用資産に基づいて作られる，ソフトウェア集約的なシステムの集合である[1]．この共通の再利用資産を如何に適切に設計し，これらを使って各プロダクトを開発できるかがプロダクトライン開発では重要になる．再利用資産が体系的に設計され適切に利用されるためには，プロダクトライン全体のベースになる，プロダクトラインアーキテクチャの役割が重要である．様々な差異を持ったプロダクトが全てひとつのプロダクトラインアーキテクチャの上に実現されなければならない．

しかしながら近年の変化の早い環境にあって，プロダクトラインはより多くの製品バリエーションを含む傾向にある．また，開発初期には十分想定できなかった差異をプロダクトラインに含めなければならない状況も起こってきている．このようなバリエーションの増大に伴って，当初設計したプロダクトラインアーキテクチャではプロダクト全体を支えられないような状況も起こってきている．

このような状況に対処するため，我々はアーキテクチャを柔軟化する方法を検討している．アーキテクチャがプロダクトライン全体を固く縛るのではなく，アーキテクチャを最初から様々な変化を柔軟に受け止めることができるように設計しておくのである．

本稿では，プロダクトラインアーキテクチャの柔軟化について考察する．以降，プロダクトラインアーキテクチャを設計する際の前提となるプロダクトラインの可変性について第 2 章で述べ，第 3 章ではプロダクトラインアーキテクチャの典型的な設計例を取り上げ, その課題を考察する．そして第 4 章で，柔軟なプロダクトラインアーキテクチャ設計の試行と考察をする．

2. プロダクトラインの可変性

プロダクトラインアーキテクチャは，様々な差異を含むプロダクトの共通の基盤となるアーキテクチャである．したがってこの設計においては，プロダクトライン中のプロダクトにおいて共通するものは何か，それぞれに異なり得るものは何かを分析しなければならない．このプロダクトライン毎に異なり得るものを可変性と呼ぶ．プロダクトラインアーキテクチャの設計において，可変性の分析は重要である．そこで，本章ではまずプロダクトラインの可変性について考える．

可変性には，機能的な特徴だけではなく，非機能的な特徴も含まれる．また，外部に直接見えるものだけなく，ソフトウェアの動作環境に関わるもの，利用するドメイン技術や実装技術など，様々な場所に可変性は現れる．

Lee らは，これらのフィーチャを分類するフィーチャカテゴリを提案しており，フィーチャモデルをこれらのカテゴリ毎に階層に分けて記述する方法を提案している[2]．表 1 はフィーチャカテゴリをまとめたものである．

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

表 1　フィーチャカテゴリ

フィーチャカテゴリ	内容
能力	サービス(システムのサービス) 操作(システムの操作) 非機能特性(外観，コスト，制約)
動作環境	SW/HW インタフェース(API, デバイスドライバ) SW/HW プラットフォーム(OS, CPU)
ドメイン技術	ドメイン特有の手法(勧告，ドメインでの理論，法規，標準)
実現技術	設計判断(アーキテクチャスタイル，プロセス協調の方法) 実装判断(アルゴリズム，プロトコル，実装方法)

3. プロダクトラインアーキテクチャの設計例

3.1 階層的レイヤによるアーキテクチャ

プロダクトラインアーキテクチャは，プロダクトラインが持つ可変性について分析した結果を踏まえ，プロダクトライン中のすべてのプロダクトがその上に実現可能となるように設計される．

具体的なプロダクトラインアーキテクチャは様々であるが，典型的なスタイルとして，階層的レイヤによるアーキテクチャが採用されることがある．これは 2 章で述べたように，プロダクトラインは様々な可変性を持つが，これらはカテゴライズして捉えることが可能で，これをシステムの構成と対応付けて管理することにより，可変性の実現が容易になるためと考えられる．また，組込みシステムの開発においてプロダクトライン開発が適用されることが多いが，組込みシステムの場合には様々なハードウェアとの関連を持つため，それらとの関係を単純化し，各コンポーネントの再利用性を向上させるため，階層的レイヤによるアーキテクチャが採用されることが多い．

図 1 は，階層的レイヤによるプロダクトラインアーキテクチャの例である．これは，自動車の室内ライト制御のプロダクトラインアーキテクチャのうち，一部を示したものである．このプロダクトラインにおいては，ドアの開閉により室内ライトの点灯消灯を行うことは必須の機能である．また，この室内ライトの制御を行うにあたって，バッテリーセーバーの機能を持つものと持たないものがある．バッテリーセーバー機能を持つものは，バッテリー残量がある値を下回っている場合には，室内ライトが点灯しても一定時間後に自動で消灯する．このバッテリーセーバーに関する可変性は，機能の可変性であると同時に，この機能を実現するためにはハードウェアとしてバッテリーに関するセンサが装備されていなければならず，動作環境としての可変性にもなっている．これらの可変性が，アプリケーション層及びハードウェア制御層に実装される．

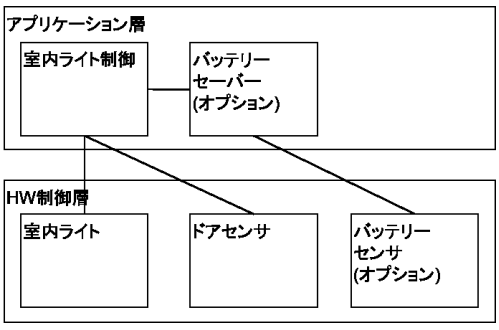

図 1　階層的レイヤによるアーキテクチャの例

3.2 階層的レイヤによるアーキテクチャの課題

階層的レイヤによるアーキテクチャは，ハードウェア，ハードウェアラッパ，アプリケーション等のように，実現技術の階層に対応した階層を定義し，各階層固有の問題が他の階層に及ぼす影響を低減できるため，再利用性を向上させることができる．各階層間の関係は単純化されるが，上位の階層は下位の階層が提供するインタフェースに依存した構造となる．

プロダクトラインにおいてその可変性を実現するために，利用するハードウェアを追加したり入れ替えたりする場合がある．例えば，図 1 の例においてもバッテリーセーバーの機能を持たせたければ，バッテリーセンサを搭載し，そのセンサの値を見ながら制御を行うモジュールをアプリケーション層に置くことで，バッテリーセーバーの機能を実現している．このように，下位層のモジュールの単純な追加・入れ替えで可変性が実現できる場合には，階層的レイヤによるアーキテクチャで問題ない．

しかし，常に下位層と上位層を 1:1 に対応させられるとは限らない．例えば，アプリケーションで用いるデータをあるハードウェアから取得していたが，ハードウェアの変更により取得できなくなったり，単純なセンサデータの利用では状況を取得することができず複数のセンサデータを合わせて値を推測しなければならなくなったりすることがある．あるいは，ひとつの赤外線センサが室内ライト照度制御，パネル表示照度制御といった複数の機能で使われたり，室内ライト制御が室内ライトやドアセンサなど複数のハードを使ったりするなど，上位層と下位層の関係は一般には多対多になる．さらに，例えばひとつの近距離レーダのセンサ値は衝突回避，駐車支援，オートクルーズなど複数の機能に使われるが．同じセンサからのデータでも加工の方法が異なるなど，その関係性は一般に複雑である．

このような関係性に絡む可変性が出現すると，層内のモジュールの単純な入れ替えでは可変性を実現できなくなり，アーキテクチャ自体の変更が必要になってしまう．

4. 柔軟なアーキテクチャ設計の試行

3.2 で述べた階層的レイヤによるアーキテクチャの問題は，階層的レイヤでは一般に上位層が下位層に依存して作られることに原因がある．本来，それぞれの層は独立した問題を扱うものであり，他に依存すべきではない．しかし，上位層が下位層を利用する関係を直接的な依存関係として実現していることが，アーキテクチャの柔軟性を損ねる結果になっていた．

我々はこの問題に対して，アスペクト指向設計を利用し，アーキテクチャに出現する依存関係を徹底的に局所化することを検討している．我々は過去に，関心事毎にアスペクトに分離し，各アスペクトは状態遷移を持ったオブジェクトの集合として定義し，各アスペクト間の関係を関連付けのルールとして定義するアスペクト指向の設計メカニズムを提案した[3][4]．関連付けのルールは，あるアスペクトで状態遷移が起こると別のアスペクトにイベントとして伝達するというルールと，あるアスペクトにおける状態遷移の条件として別のアスペクトにおけるオブジェクトの状態を参照するというルールの 2 種類がある．このようなルールを用いることにより，アスペクト自体は他のアスペクトから全く独立して振る舞いを定義することができ，他のアスペクトへの依存を排除することができる．アスペクトの集合として定義されるソフトウェア全体の振る舞いを実現する時に出現するアスペクト間の関連性は，ルールに局所化される．3.1 で述べた室内ライト制御のプロダクトラインアーキテクチャを，このメカニズムを用いて設計した例を図 2 に示す．

バッテリーセーバーやライト制御といったアプリケーションが，パワーセンサやドアセンサ，あるいは室内ライトなどと関係を持ちながらその機能を果たしているという構造は同一だが，依存関係が全く異なっている．図 1 のレイヤ構造では，アプリケーションは下位のセンサなどに依存して作られるが，図 2 のアスペクトを用いたアーキテクチャでは，そうした依存性はない．従って，仮に別のセンサを利用する際にも各アプリケーション自体の修正は必要ではなく，両者を関連付けるルールの変更によって関連付けを行うことができる．

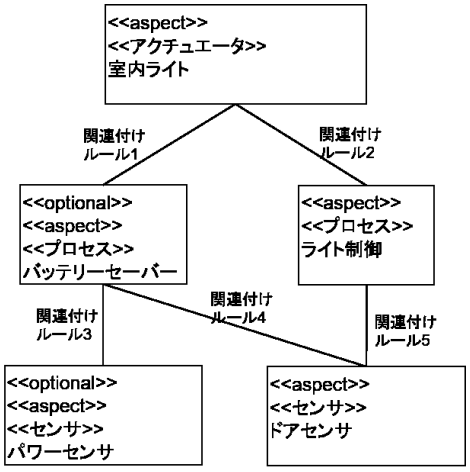

図 2　アスペクトを用いたアーキテクチャ

一般にプロダクトラインアーキテクチャにおいては，可変性に対応して，アーキテクチャ中に可変点とバリアントを定義して，製品系列への適用を図る．製品毎の可変性の違いに対応して，対応するバリアントを自由に組み合わせて製品が実現されることが望ましい．上記の例では，様々なアプリケーションやセンサの選択肢がバリアントとなる．しかしながら，レイヤ構造のようにバリアントがお互いに複雑な依存関係を持つと，そうした自由な組合せが困難となる．

一方，アスペクト指向アーキテクチャでは，それらの依存関係はルールに集約されるため，最も重要なコア資産であるアプリケーションやセンサそのものは，きわめて独立性高く実現することができる．

なお，前述したルールは，各アスペクトで扱っているデータの違いを直接的に吸収することができない．例えば照度データを扱うなら，各アスペクトでその表現や桁数をそろえておかなければならない．これは柔軟性の点からは不十分である．こうしたデータの違いによってアスペクトを関連付けて使うことができなくなる現象は，いわゆるデータに起因するアーキテクチャ不整合であり，再利用性を損なう．アスペクト間でデータの表現が異なっていても，それらを対応付けるルールをつけることでこの問題を回避できると考えており，この詳細な検討は今後の課題である．

5. おわりに

プロダクトライン開発が普及しつつあるが，変化の早い環境において当初想定した以上に多くの可変性を抱えることになり，プロダクトラインアーキテクチャの維持が困難になるといった問題が起こっている．本稿では，こうした問題に対処するために，アスペクト指向を用いてプロダクトラインアーキテクチャを柔軟化する試みについて述べた．データ不整合の解消などの課題があるが，さらなる検討を重ね，アーキテクチャの柔軟化の実現を図りたい．

参考文献

[1] Clements, P. and Northrop, L., Software Product Lines – Practices and Patterns, Addison-Wesley, 2001.
[2] Lee, K., Kang, K. C. and Lee, J., Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, 7th International Conference on Software Reuse (ICSR-7), 2002
[3] Noda, N. and Kishi, T., Aspect-Oriented Modeling for Embedded Software Design, Proc. 14th Asia-Pacific Software Engineering Conference (APSEC 2007), 2007.
[4] Noda, N. and Kishi, T., Aspect-Oriented Modeling for Variability Management, Proc. 12th International Software Product Line Conference (SPLC2008), 2008.