

# 「ソフトウェア工学」

## (6-2) 様々な開発パラダイム

---

東大理学部情報科学科講義

石川 冬樹 (本務先：国立情報学研究所)  
f-ishikawa@nii.ac.jp / @fyufyu

<http://research.nii.ac.jp/~f-ishikawa/>

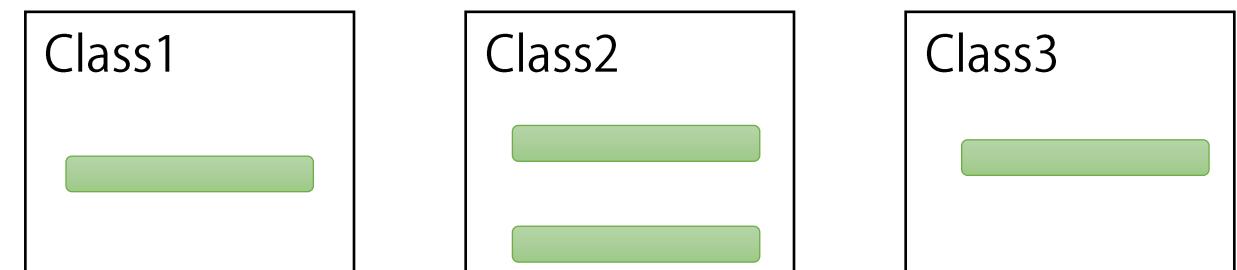
# 目次

---

- アスペクト指向
- ソフトウェアプロダクトライン
- モデル駆動開発

# 横断的関心事

- 関心事の分離 (Separation of Concerns) は、  
ソフトウェア設計の重要な側面であった (第3回)
- 横断関心事 (Cross-cutting Concerns)
  - オブジェクト指向設計・オブジェクト指向プログラミングでは、  
複数のクラスにまたがってしまう関心事が生じる
  - ロギング, セキュリティ処理 (暗号化, 署名, アクセス制御),  
画面更新, 状態保存, ...



# アスペクト指向

---

- アスペクト指向 (Aspect-Oriented XXX)
- 横断的関心事を「アスペクト」としてとらえ、  
分離して部品化するための仕組み
- アスペクト指向要求分析, アスペクト指向設計,  
アスペクト指向プログラミング (AOP) , ...
- 実装例：プログラミングフレームワーク AspectJ

# AspectJでの例 (1)

- ポイントカット：共通の振る舞いをとる対象
- アドバイス：ポイントカットに共通する振る舞い
- 例：座標を変えるメソッドすべてに対し、実行前後にログ処理

```
pointcut move():
    call(void FigureElement.setXY(int,int)) ||
    call(void Point.setX(int)) ||
    call(void Point.setY(int)) ||
    call(void Line.set*(Point));
```

[<https://www.eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html>]

```
before(): move() {
    System.out.println("about to move");
}
after() returning: move() {
    System.out.println("just successfully moved");
}
```

# AspectJでの例 (2)

## ■既存クラスPointにフィールド・振る舞いを追加

```
aspect PointObserving {  
    private Vector Point.observers = new Vector();  
  
    public static void addObserver(Point p, Screen s) {  
        p.observers.add(s);  
    }  
  
    pointcut changes(Point p):  
        target(p) && call(void Point.set*(int));  
  
    after(Point p): changes(p) {  
        for(Screen s : p.observers) {  
            s.display(p);  
        }  
    }  
}
```

PointクラスにObserverとして  
Screenオブジェクトを  
追加できるようにし、

setXXXメソッドが呼ばれた際には  
各Screenに表示を行う

[ <https://www.eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html> ]

# アスペクト指向：補足

---

## ■難しさ

- 「マージされた振る舞い」がわかりづらく、不整合が生じるかも

## ■关心事の分離は今でも重要視

- 現在「アスペクト指向」という2000年代に流行した用語を掲げて取り組むことは少ない
- 「設定ファイルに応じて、処理を各所に自動で挟み込み」というアプローチを採ることは、ロギング・デバッグなどでとても多い

# 目次

---

- アスペクト指向
- ソフトウェアプロダクトライン
- モデル駆動開発

# ソフトウェアプロダクトライン

---

## ■ ソフトウェアプロダクトライン

(Software Product Line : SPL)

- 類似する「製品系列」(Product Line)をまとめて  
体系的に開発する考え方
- 従来のProduct Line

- 経営学などでは「製品ライン」と訳すことが多い？
- ハードウェアや食品などについて、共通設備・共通材料などを  
用いて製造することで効率化（そうなるよう設計）
- 例えば、マクドナルドにおける「ハンバーガー類」「飲み物類」

# ソフトウェアプロダクトラインにおける工程

## ■ ドメインエンジニアリング

- 全プロダクトに共通する部分、個別に定まる部分を区別して、各プロダクトの開発に用いるコア資産を構築
- どれだけの変化があるかの可変性 (Variability) を分析、設計

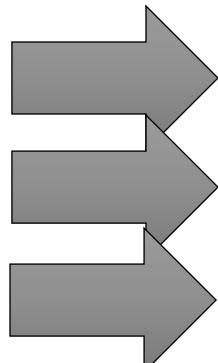
## ■ アプリケーションエンジニアリング

- コア資産に基づいて効率的に各プロダクトを構築

ドメイン：自動車の操作パネルに対する制御ソフトウェア

スマホ接続機能がある場合とない場合があり、それぞれプログラムはこう作る

...



今回のプロダクト：  
スマホ接続機能あり

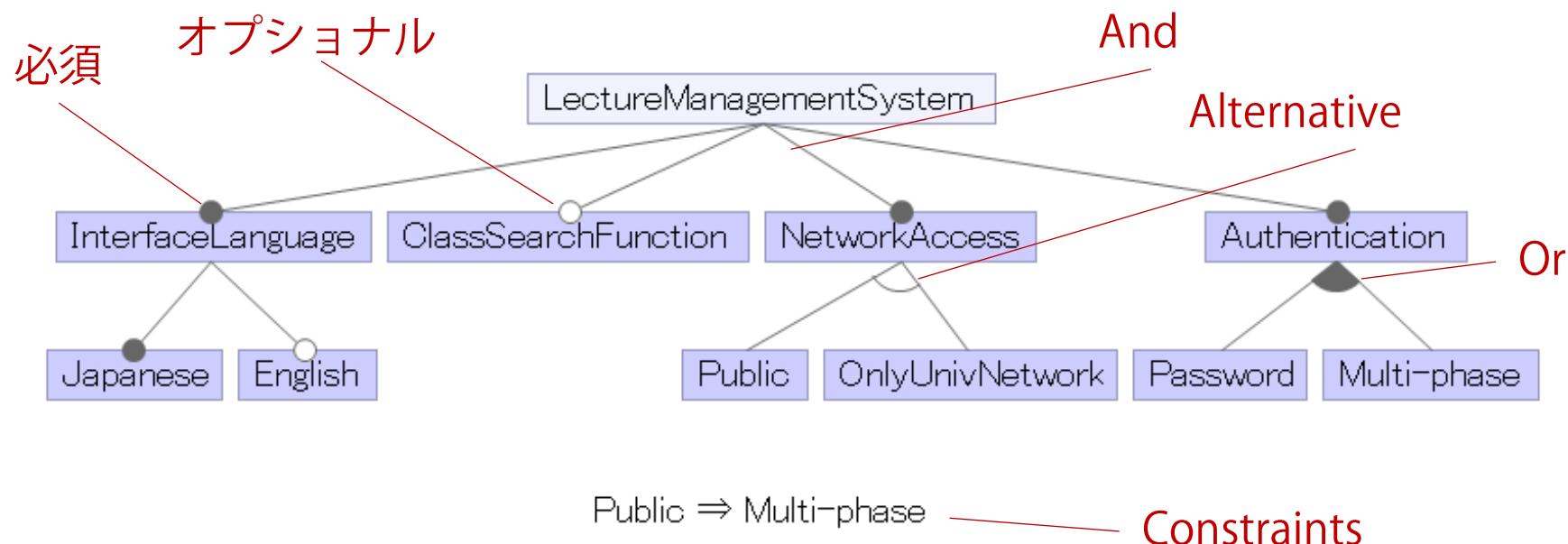
今回のプロダクト：  
スマホ接続機能なし

...

# フィーチャーモデル

## ■ フィーチャーモデル (Feature Model)

- プロダクトラインにおいて、共通部分と個別に変わる部分によって定まる各プロダクトバリアント (Variant) による可変性 (Variability) を定義するためのモデル



# プロダクトライン管理

---

- コア資産と各プロダクトが連動して変化していくような仕組み作りが必要
- 例：個々のプロダクトの定義に応じてどうやってモデルや実装を生成する？
  - アスペクト指向でプロダクト固有のオプション機能を織り込む
  - C/C++コードのマクロ（ifdef）でプロダクトに応じて処理の有無を設定してコンパイルできるようにしておく
  - ...

# 目次

---

- アスペクト指向
- ソフトウェアプロダクトライン
- モデル駆動開発

## ■ モデル駆動開発

### (Model-Driven Development: MDD)

- 要求から実装に至る工程を多段階のモデル変換処理と考え,  
特に自動でのモデル変換処理を用いて開発を行う
- ここでの「アーキテクチャ」は、開発対象システムのアーキテクチャではなく、開発活動を構成するモデルの構造化を指す
- ビジネス、実装プラットフォーム非依存のロジック、  
実装プラットフォーム依存の設計を分離する
- 特にモデル間の自動変換、コードの自動生成を指向する

## ■ モデル駆動開発

### (Model-Driven Development: MDD)

- 要求から実装に至る工程を、多段階のモデル変換処理と考える
- 「仕様モデル」から「設計モデル」、そして「実装モデル（コード）」など各モデル間の変換ルールを系統的に定める
- 理想的にはモデル変換は自動化する  
(仕様が変わると、コードを素早く再生成できる)

# 参考：モデル駆動アーキテクチャ

---

## ■ モデル駆動アーキテクチャ

(Model Driven Architecture: MDA)

### ■ モデル駆動開発において用いるモデル構造を規定

(設計のアーキテクチャではなく、プロセスのアーキテクチャ)

■ Computation Independent Model (CIM) : ビジネス・ドメイン

■ Platform-Independent Model (PIM) : システムモデル

■ Platform-Specific Model (PSM) : 設計モデル

### ■ 2001年に提案されたが、「完全なモデルを書き、モデル変換だけを通してコードを得る」理想は挫折（要素技術は有用）

# モデル駆動開発のための道具

---

- UML, 特に具体的な実行手順を定めるもの
  - fUML (foundational subset for executable UML models) : タイミングなど実行のための正確な意味論を定義
- Domain-Specific Language (DSL) の活用支援
  - 文法定義および, 文法からのパーサーやエディタの生成
  - Eclipse Modeling Framework, Xtextなど
- モデル変換の定義言語・実行ツール
  - QVT, ATLなど
- モデルからのコード生成ツール

# 例：Xtext

## 自分たちのDSL文法定義

```
MyUnivModel:  
  (lectures+=Lecture)*;  
  
Lecture:  
  'lecture' lecid=ID ('type' lectype=ID)? '{'  
  (students+=Student)*  
  '}';  
  
Student:  
  stuid=ID
```

自動生成

利用（して開発）

パーサー  
エディタ  
…

DSL用  
ツール

## 実行時：DSLを用いた一つの記述

```
lecture BS001 {  
  S4602111, T2141136  
}  
  
lecture NS005 type external {  
  S4603041. T1140855, T3421608  
}
```

エディタで編集

言語構造に沿ってプログラムで読み込み  
- myUnivModel.getLectures()  
- lecture.getLecid()

[ <https://www.eclipse.org/Xtext/> ]

# 補足：モデルベース開発

---

## ■ モデルベース開発 (Model-Based Development)

- 自動車の制御ソフトウェアなど、物理挙動を扱う機器の制御プログラム開発において用いられる用語
- 物理世界のイベント・挙動に対応した制御処理を、数式の世界で設計（モデルを作成）
- この設計モデルからプログラムコードを生成
- MATLAB/Simulinkがツールとして代表的

# 今回の参考文献

---

- ユースケースによるアスペクト指向ソフトウェア開発
  - Jacobson et al., 鶩崎ら訳, 翔泳社, 2006
- ソフトウェアプロダクトラインエンジニアリング—ソフトウェア製品系列開発の基礎と概念から技法まで
  - Pohl et al., 林ら訳, エスアイビーアクセス, 2009
- MDA モデル駆動アーキテクチャ
  - Frankel, 日本IBM訳, エスアイビーアクセス, 2003

# まとめ

---

- 様々なパラダイム
- 再利用性と変化への対応を主要な問題ととらえ、  
様々なパラダイムが提案、議論され続けている