ソフトウェア工学

(6) 様々なパラダイム

国立情報学研究所　　石川　冬樹

f-ishikawa@nii.ac.jp

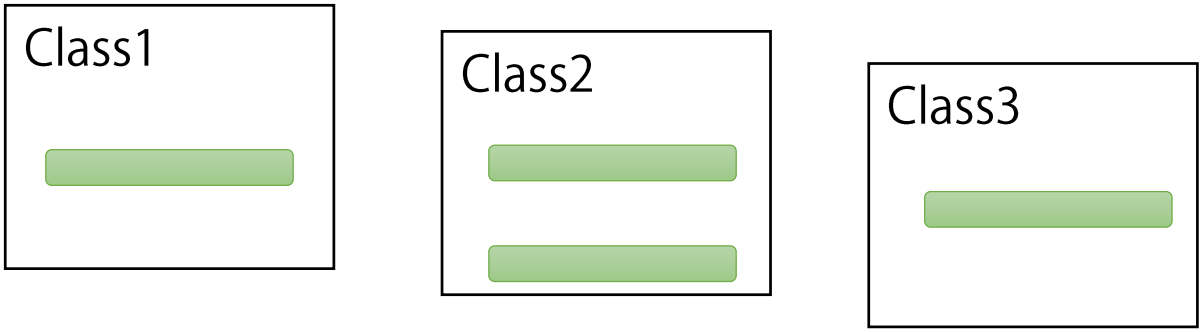
http://research.nii.ac.jp/f-ishikawa/

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
- 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る関心事がある
  - 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），・・・
  - プログラミングフレームワークとしてAspectJが有名

※ なお，「設定ファイルに応じて，処理を各所に自動で挟み込み」というミドルウェア・フレームワークの挙動についても，横断的関心事を扱っていると言える

AspectJでの例（１）

- ポイントカット：共通の振る舞いをとる対象
  - 例： 座標を変えるメソッド

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

- アドバイス：ポイントカットに共通する振る舞い
  - 例： 実行前後にログ

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

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

AspectJでの例（２）

- クラスにまたがる要素の宣言
  - 1クラスに挙動を加える例

```
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);
        }
    }
}
```

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

目次

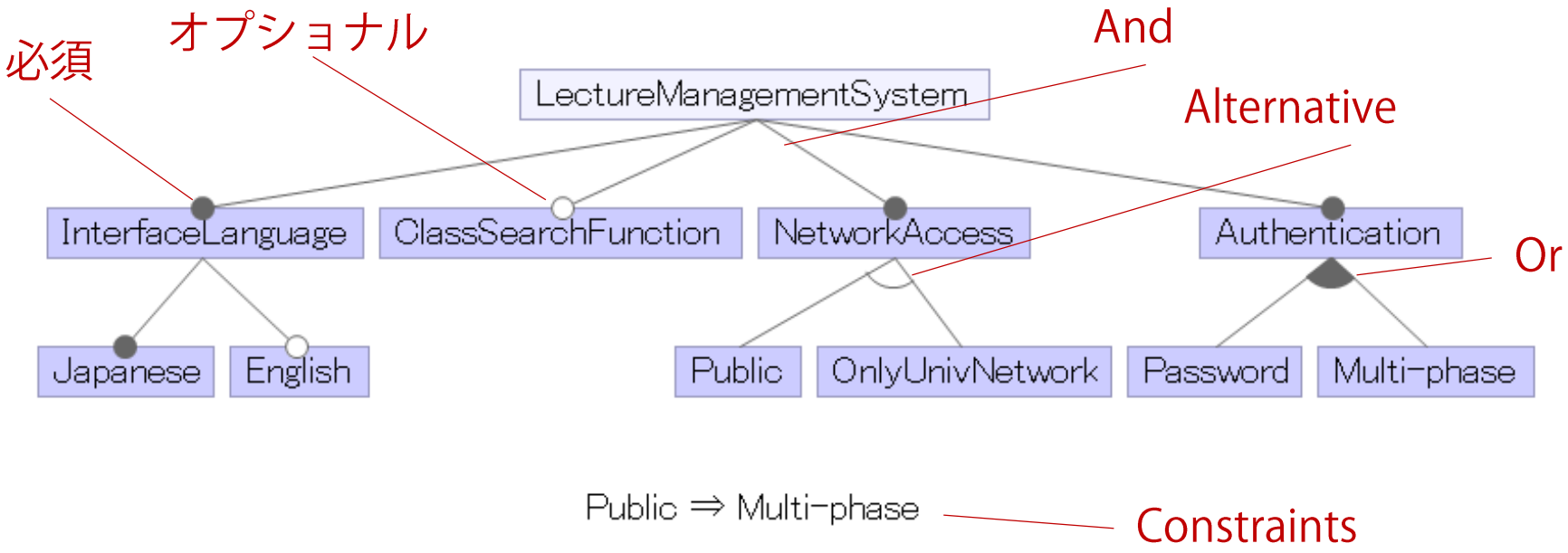
- アスペクト指向
- ソフトウェアプロダクトライン
- モデル駆動開発

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）
  - 類似するプロダクトの系列をまとめて体系的に開発する考え方
  - 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
  - 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，基本的に「選ぶ」だけのような形で効率的に各プロダクトを構築

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：プロダクトの定義に応じてモデルや実装を生成
  - アスペクト指向でプロダクト固有のオプション機能などを織り込み
  - C/C++コードのマクロ（ifdef）でプロダクトに応じた処理を含めてコンパイル処理を分岐
  - ・・・

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

- モデル駆動アーキテクチャ開発（Model-Driven Architecture: MDA）
  - モデル変換を通して開発を行う
  - 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

MDAにおけるモデル構造

- Computation Independent Model (CIM)
  - ビジネスおよびドメインに関するモデル
- Platform-Independent Model (PIM)
  - 実装プラットフォーム非依存のシステムモデル
- Platform-Specific Model (PSM)
  - 実装プラットフォーム依存の設計モデル

MDAのための道具

- UML および 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

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., 鷲崎ら訳，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
  - 特に再利用性と変化への対応を意識し，様々なパラダイムが提案され続けている