Software Engineering

(9) Various Paradigms

Sokendai / National Institute of Informatics
Fuyuki Ishikawa / 石川　冬樹
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

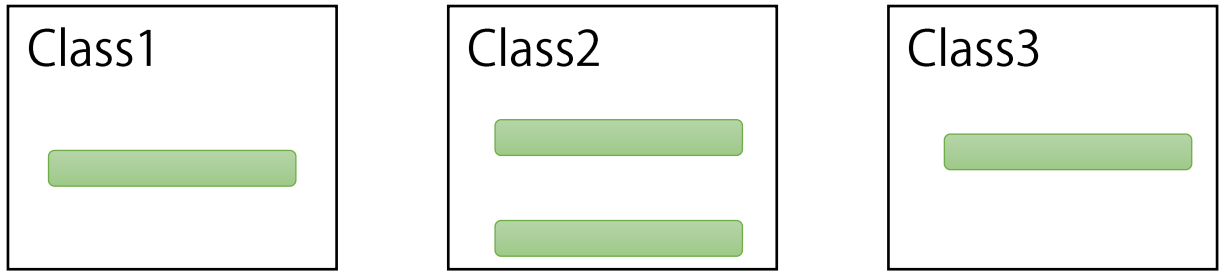
大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

目次

- Aspect-Orientation
- Software Product Line
- Model-Driven Development

Cross-Cutting Concerns

- Separation of Concerns（関心事の分離） is one of the key principles of software design
- Cross-cutting Concerns（横断的関心事）
  - Concerns that span over multiple classes in object-oriented design/programming
  - Logging, security processing (encryption, signature, access control), screen update, state preservation, …



Aspect-Orientation

- Aspect-Oriented XXX（アスペクト指向）
  - Mechanism to capture cross cutting concerns as “aspects”, separated modules
  - Aspect-oriented requirements analysis, aspect-oriented design, aspect-oriented programming (AOP), …
  - Tool example: AspectJ for Java-based AOP
  - Discussed actively around 2000

Example in AspectJ (1)

- Pointcut: targets that take the common behavior
- Advice: the common behavior in pointcuts
  - e.g., logging before/after all the methods that change the points

```
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");
}
```

Example in AspectJ (2)

- Add a field and method onto the existing class Point

```
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);
        }
    }
}
```

A new data field *observers* is added to the existing Point class

The *observers* are used to display information after any *setXXX* method is invoked

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

Aspect-Orientation: Discussion

- Difficulties in understanding the merged behavior, possibly causing inconsistency
- Separation of concerns is still considered significant
  - Maybe not the old trendy word “aspect-orientation”
  - Typical implementation: the framework (for security, for logging, for debugging) inserts common behavior according to specific configuration files

目次

- Aspect-Orientation
- **Software Product Line**
- Model-Driven Development

Software Product Line

- Software Product Line : SPL
  （ソフトウェアプロダクトライン）
  - Approach to systematically develop a similar line of products
- Traditional product line
  - Design efficient production of hardware, food, etc. by using common instruments and materials
  - e.g., “Hamburgers” in Mcdonald’s

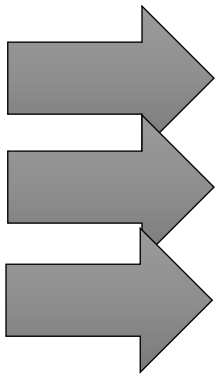
Activities in Software Product Line

- Domain engineering
  - Build core assets to be used for development of each product by identifying common/different parts
  - Analyze and design the variability（可変性）
- Application engineering
  - Efficiently build each product using the core assets

Domain: software of AV display of cars

e.g., variability: with or without smart phone connection function

→ Each case is build in this way …



Product 1:
with smart phone connection

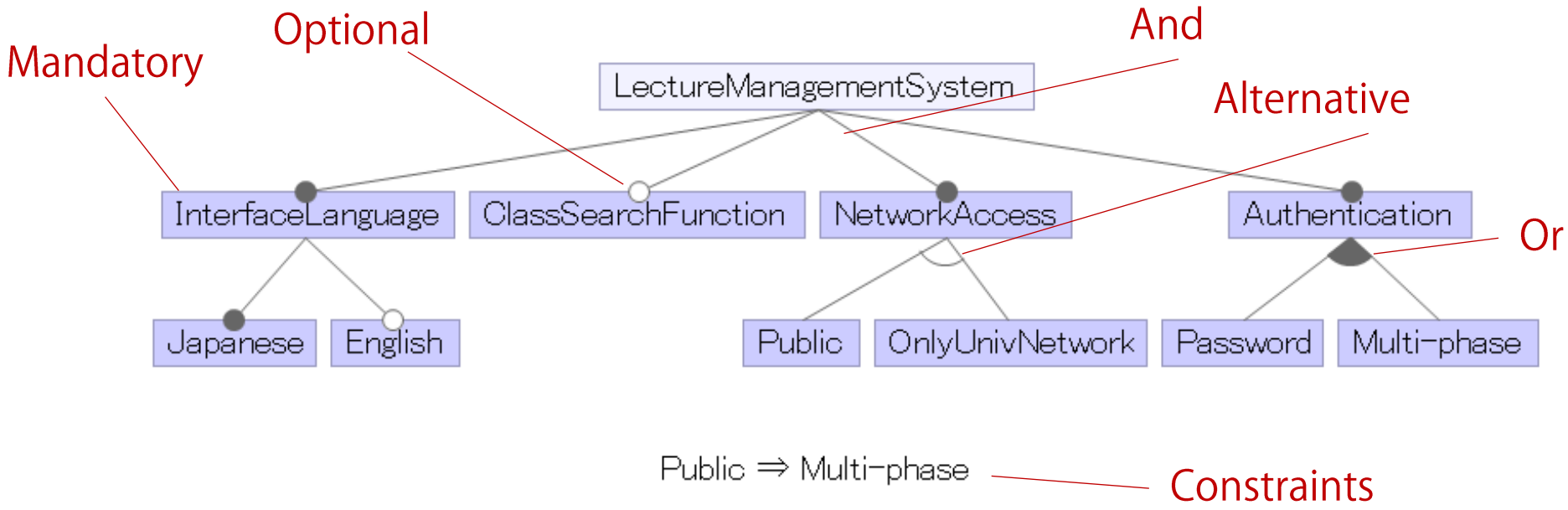
Product 2:
without smart phone connection

…

Techniques for Software Product Line

Feature Model（フィーチャーモデル）

- A model to define each product variant by specifying common and different aspects

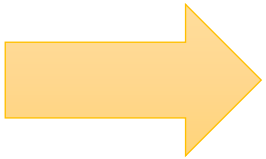


Techniques for Software Product Line

- Mechanism to generate models and code of each product
  - e.g., embed specific behavior for a certain feature by using aspect-oriented programming
  - e.g., activate/deactivate specific behaviors by C/C++ macro (ifdef)
  - …

```
@feature1
class Display{
  void show(){
    ...
    @feature2A
    show(text);
    @feature2B
    showTranslated(text);
}
```

This time, the product employs feature1 and feature 2B



```
@feature1
class Display{
  void show(){
    ...
    showTranslated(text);
}
```

目次

- Aspect-Orientation
- Software Product Line
- **Model-Driven Development**

Model-Driven Development

- Model-Driven Development: MDD（モデル駆動開発）
  - Approach to consider the development process as multi-phase model transformation activities
  - Use of systematic, ideally automated, transformation rules such as one from specification to design, from design to implementation
  - Reaction to changes by re-generation of implementation code from the updated requirements models (ideally)

Model Driven Architecture

- Model Driven Architecture: MDA
  （モデル駆動アーキテクチャ）
  - Defines the modeling structure as a process architecture (not a design architecture)
    - Computation Independent Model (CIM): business and domain
    - Platform-Independent Model (PIM): system models
    - Platform-Specific Model (PSM): design models
  - Proposed in 2001 but too ideal to think of fully automated model transformation from requirements to code

Techniques for Model-Driven Development

- UML with clear semantics for generating code
  - fUML (foundational subset for executable UML models)
- Domain-Specific Language (DSL)
  - Generating language tools from a syntax definition
  - e.g., Eclipse Modeling Framework, Xtext, …
- Languages and tools for model transformation
  - QVT, ATL, …

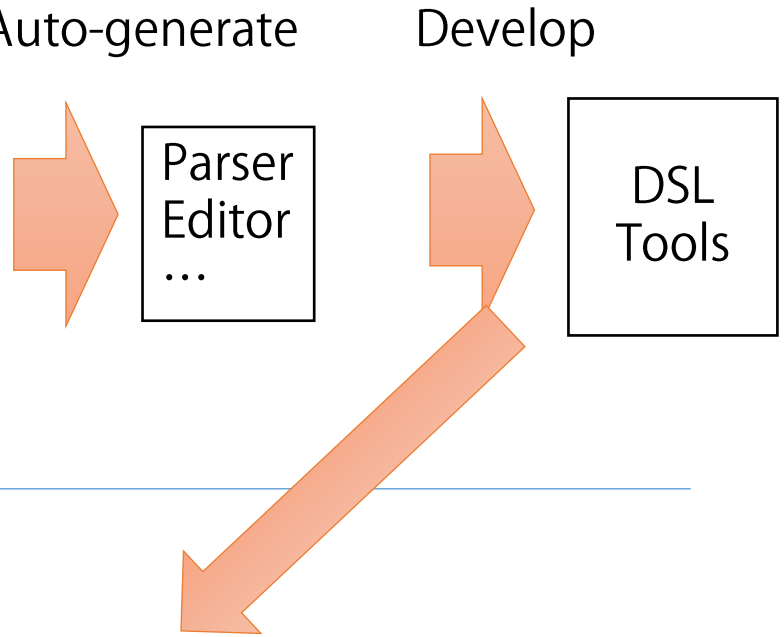
Example: Xtext

DSL syntax definition

```
MyUnivModel:
  (lectures+=Lecture)*;

Lecture:
  ‘lecture’ lecid=ID (‘type’ lectype=ID)? '{'
    (students+=Student)*
  '}';

Student:
    stuid=ID
```



Run time: DSL description instance

```
lecture BS001 {
  S4602111, T2141136
}

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

Edit with the generated editor tool

Read from code by prepared APIs
- myUnivModel.getLectures()
- lecture.getLecid()

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

Model-Based Development

- Model-Based Development（モデルベース開発）
  - Term used in control software such as in automotive systems
  - Design control behavior with mathematical formula as models, e.g., by using MATLAB/Simulink
  - Then, generate program code from the models

Summary

- Different paradigms have been still actively investigated
  - With the core focus on reusability and response to changes