Guidelines for Using Aspects in Product Lines

Ján Kohut Valentino Vranić

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology,
Ilkovičova 3, 84216 Bratislava 4, Slovakia
vranic@fiit.stuba.sk

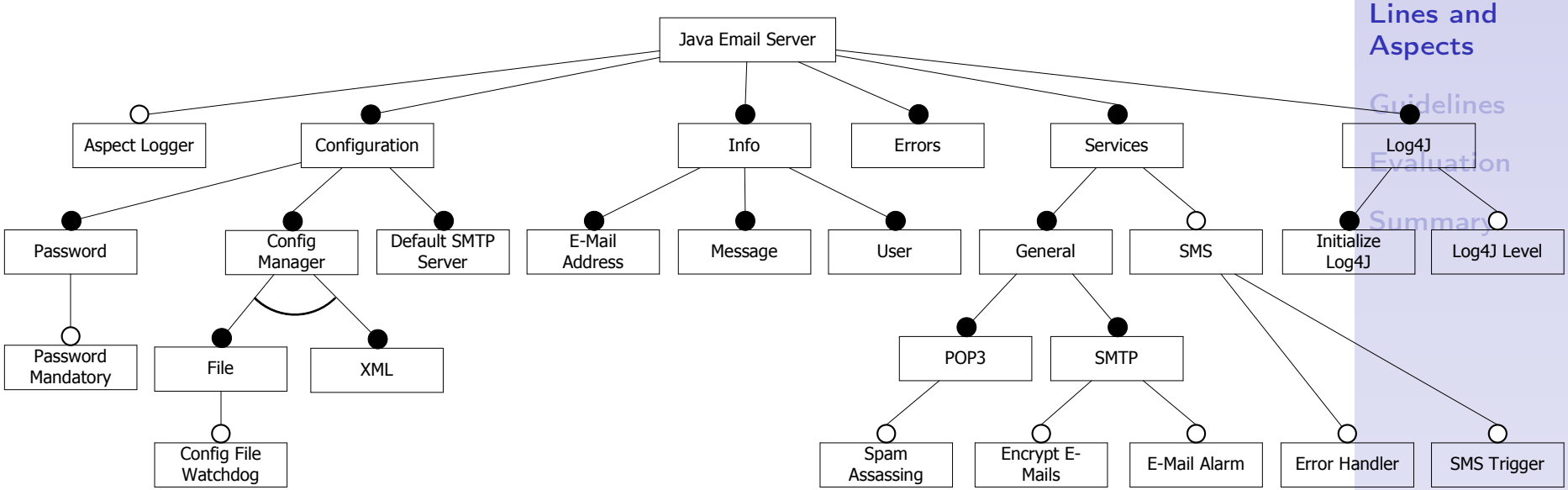
SAMI 2010, January 28–30, 2010, Herľany, Slovakia

Overview

Software Product Lines

- An organized approach to software reuse
- Successful because it's focused on a particular domain
- Common and variable feature
- Configuration: selection of variable features

Feature Model



Constraints:

E-Mail Alarm requires SMS

Configuration and Crosscutting

- Product line development is complicated and represents a large investment
- In part, this is due to crosscutting concerns
- A crosscutting problem affects configuration: crosscutting features cannot be easily incorporated nor taken out once they have been incorporated

Aspects

- Advanced software decomposition and composition approach: decomposition into multiple views developed separately and composed as needed
- Commonly denoted as aspect-orientation
- PARC AOP and AspectJ prevail: untangling crosscutting concerns
- But there are quite different yet still aspect-oriented approaches

Aspect

- The main construct in PARC aspect-oriented programming is an aspect
- Main parts of an aspect:
  - Pointcuts: specifying the join points the aspect affects
  - Advices: implementing the affecting functionality
  - Inter-type declarations: introducing new fields and methods, inheritance relationship, warnings, compile errors, softened exceptions, and annotations into types

How Can Aspect-Orientation Help

- Probably impossible to formulate strict rules of aspect-oriented programming application to product lines
- But recommendations on its use in the form of guidelines that represent expert knowledge can be given

Guideline Sources

- Experience reported in peer reviewed scientific work
- But also in informal contributions to discussion forums
- Available, but not in a compact and ready to use form
- An idea: extract the knowledge and express it as guidelines
- A lightweight, non-restrictive approach

Guideline Form

- The idea of guidelines is close to patterns
- We applied Coplien's form of pattern description:
  - Short name
  - Name
  - Context
  - Problem
  - Forces
  - Solution
  - Example
  - Discussion
- Not necessary to avoid difference in recommendations at all costs
- But the guideline should accommodate to them and eventually provide some possibilities of combining even the differing guidelines

The Guidelines (1)

- **Implementing Features of Refactored Legacy Applications**:
  The aspects are unsuitable for implementing features of refactored legacy applications.
- **Implementing Mandatory Features with no Crosscutting Concerns**:
  Do not use aspects in mandatory features if there are no crosscutting concerns.
- **Code Reduction in Homogenous Crosscutting Concerns**:
  The aspects are suitable for reduction of replicated code in homogenous crosscutting concerns.

The Guidelines (2)

- **Transforming a Mandatory Feature into Alternative Features**:
  Do not use aspects in transforming a mandatory feature into alternative features.
- **Implementing Features which Share no Code**:
  Use aspects in implementing features which share no code and which have crosscutting concerns.

An Example Guideline (1)

- Code Reduction in Homogenous Crosscutting Concerns (introduced in the paper)
- The aspects are suitable for reduction of replicated code in homogenous crosscutting concerns
- Aspects reduce replicated code in code with homogenous crosscutting concerns[1]
- Homogenous crosscutting concerns address multiple join points with a single piece of advice
- Heterogeneous crosscutting concerns, in contrast, address multiple join points each with a different piece of advice
- Homogeneity lies in a consistent application of the same or very similar policy in multiple places

[1] S. Apel, T. Leich, and G. Saake. Aspectual mixin layers: aspects and features in concert. In *Proc. of 28th International Conference on Software Engineering, ICSE 2006*, pages 122–131, Shanghai, China, 2006. ACM Press.

An Example Guideline (2)

- *Context:* the same concern repeated across different modules in a product line
- *Problem:* hard to maintain and keep consistent the code scattered across the application
- *Forces:* need to modularize the crosscutting concerns and represent each one by a single feature for ease of maintenance and configurabilty, but the conventional approach does not provide mechanisms for this
- *Solution:* use aspects to factor out homogenous crosscutting concerns and implement them in separate advices

A Code Reduction Example

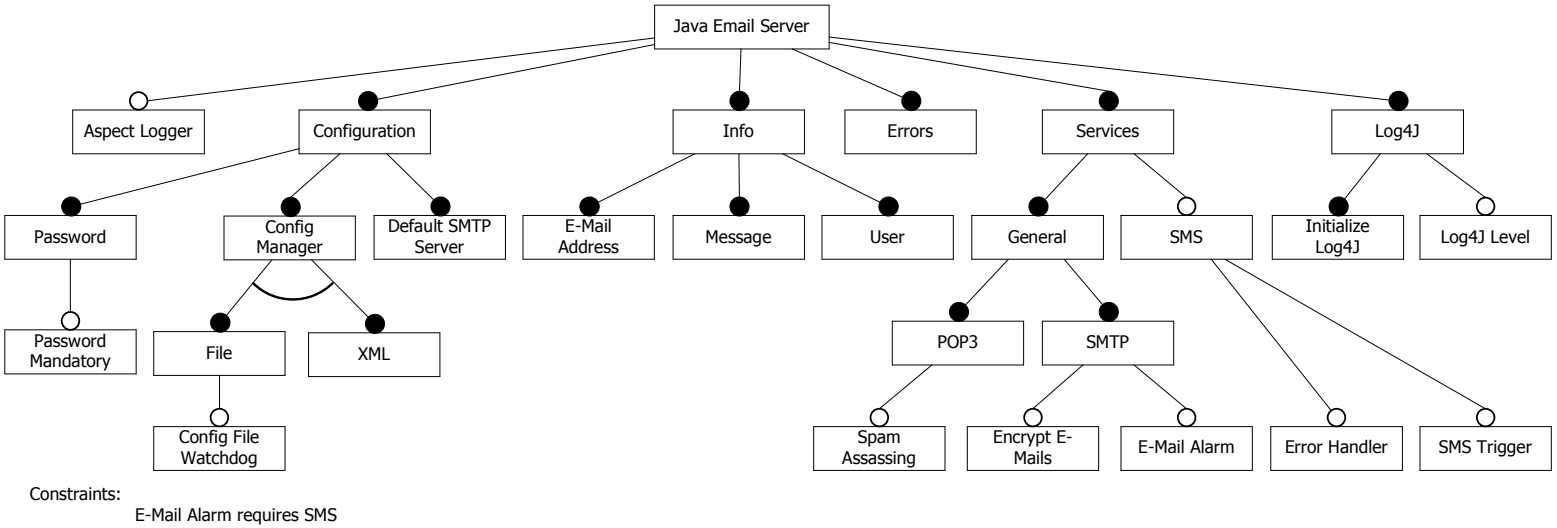
- Applicable to logging, tracing, exception handling, etc.
- An aspect affecting a set of otherwise unrelated methods using one coherent advice:

```
public aspect Homogenous {
    pointcut accessAuthorization(): call(...);
    pointcut accessDemilitarizedZone(): call(...);
    pointcut accessCommunicationInterfaces(): call(...);
    pointcut accessApplication(): call(...);

    before(): accessAuthorization() || accessDemilitariziedZone() ||
        accessCommunicationInterfaces() || accessApplication() {
        System.out.println("Accessing " + thisJoinPoint);
    }
}
```

Evaluation

- Evaluation performed on two product line implementations: an object-oriented and aspect-oriented one
- Based on the Java Email Server application[2]
- 4500 line of code, 22 classes, and several configuration files



[2] http://www.ericdaugherty.com/java/mailserver/

Metrics Applied (1)

- Scope of changes:
  - Lines of Code (LOC)
  - Number of Affected Classes (NAC)
- Quality of code:
  - Weighted Operations in Module (WOM)
  - Depth of Inheritance Tree (DIT)
  - Number of Children (NOC)
  - Crosscutting Degree of an Aspect (CDA)
  - Coupling of Method Call (CMC)
  - Coupling of Field Access (CFA)
  - Coupling between Modules (CBM)
  - Response for a Module (RFM)
  - Lack of Cohesion in Operations (LCO)

Metrics Applied (2)

- Package dependencies
  - Number of Types (NOT)
  - Abstractness (A)
  - Afferent Couplings (Ca)
  - Efferent Couplings (Ce)
  - Modified Efferent Couplings (Ce)
  - Instability (I)

Evaluation Results

- Numerical values of metrics available in the paper
- Implementing Features of Refactored Legacy Applications
  - Code with aspects shows similar level of coupling
- Implementing Mandatory Features with no Crosscutting Concerns
  - Aspect may obscure the code comprehension
  - No significant gain in LOC with aspects
- Code Reduction in Homogenous Crosscutting Concerns
  - A significant LOC gain with aspects
- Transforming a Mandatory Feature into Alternative Features
  - Both aspect-oriented and object-oriented approach add comparable number of LOC when used for adding new alternative feature
- Implementing Features which Share no Code
  - Lower method call coupling for most classes achieved with aspects

Summary

- An approach to expressing independently applicable guidelines about using aspect-oriented programming in product line development
- Evaluation performed on two product line implementations, an object-oriented and aspect-oriented one
- Results are in favor of the guidelines
- Further work
  - Identification of further guidelines
  - Evaluation of guidelines with respect to other aspect-oriented languages