

Modeling Aspect-Oriented Change Realizations

Erasmus Mobility at Lancaster University

Lecture 1

Valentino Vranić

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies

Slovak University of Technology
Bratislava, Slovakia

vranic@fiit.stuba.sk

<http://fiit.stuba.sk/~vranic/>

November 23–26, 2009

Overview

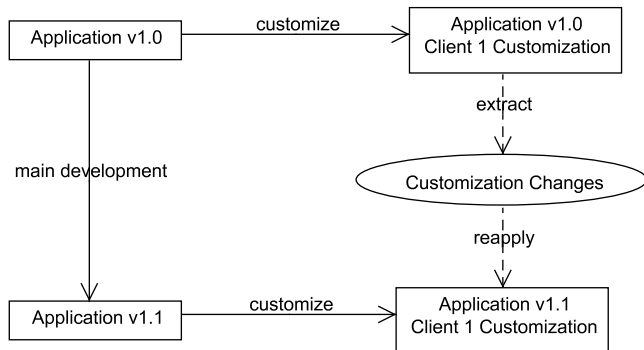
- 1 Changes as Crosscutting Concerns
- 2 Catalog of Changes
- 3 Changes at the Model Level
- 4 Generally Applicable Change Types as Paradigms
- 5 Feature Model of Changes
- 6 Transformational Analysis
- 7 Change Interaction
- 8 Related Work
- 9 Summary

Why Use Aspects in Change Realization?

- Change realization is difficult and expensive
- Changes of software applications exhibit crosscutting nature:
 - Intrinsically by being related to many different parts of the application they affect
 - By their perception as separate units that can be included or excluded from a particular application build
- Aspect-oriented programming provides suitable means to realize changes in a pluggable and reapplicable way

Motivating Example

- Customization of web applications
- A new version of the base application requires reapplication of the customization changes at the client side



Changes as Crosscutting Concerns

Change Requests as Crosscutting Requirements

- A change is initiated by a change request
 - Specified in domain notions
 - Tends to be focused, but usually consists of several requirements
- By abstracting and generalizing the essence of a change, a *change type* can be identified
- Such a change type is applicable to a range of applications of the same domain

Crosscutting Nature of Change Realizations

- A change often affects many places in the code
 - E.g., modification of selected calls of the given method
- Even if it affects a single place, we may want to keep it separate
 - To be able to revert it and reapply it
 - Especially useful in the customization of web applications
- Thus, changes can be seen as crosscutting concerns

Example Scenario

- Aspect-oriented change realization will be presented on an example scenario
- A merchant who runs his online music shop purchases a general affiliate marketing software to advertise at third party web sites (affiliates)
- Simplified affiliate marketing scheme:
 - A customer visits an affiliate's site which refers him to the merchant's site
 - When the customer buys something from the merchant, the provision is given to the affiliate who referred the sale
- Affiliate marketing software has to be adapted (customized) to the merchant's needs through a series of changes
- Assume the affiliate marketing software is written in Java
- We use AspectJ to implement changes

Why Aspect-Oriented Programming?

- Aspect-oriented programming enables to deal with change explicitly and directly at programming language level
- The logic of a change is modularized
- Changes implemented by aspects are pluggable and reapplicable to similar applications (e.g., in a product line)
- Increased changeability of components has been reported if they are implemented using
 - Aspect-oriented programming as such¹
 - Aspect-oriented programming with the frame technology²
- Enhanced reusability and evolvability of design patterns has been achieved by using generic aspect-oriented languages to implement them³

¹ J. Li, A. A. Kvale, and R. Conradi. A case study on improving changeability of COTS-based system using aspect-oriented programming. *Journal of Information Science and Engineering*, 22(2):375–390, Mar. 2006.

² N. Loughran et al. Supporting product line evolution with framed aspects. In *Workshop on Aspects, Components and Patterns for Infrastructure Software (held with AOSD 2004, International Conference on Aspect-Oriented Software Development)*, Lancaster, UK, Mar. 2004.

³ T. Rho and G. Kniessel. Independent evolution of design patterns and application logic with generic aspects—a case study. IAI-TR-2006-4, University of Bonn, Germany, Apr. 2006.

Domain Specific Changes

- Example: adding a backup SMTP server to ensure delivery of the notifications to users
 - Each time the affiliate marketing software needs to send a notification, it creates an instance of the SMTPServer class which handles the connection to the SMTP server
- A generalization:
 - An SMTP server is a kind of a resource that needs to be backed up
 - In general, it's a kind of *Introducing Resource Backup*
 - Abstract, but still expressed in a *domain specific* way—a *domain specific change type*

Domain Specific Change Implementation (1)

- The crosscutting concern identified: maintaining a backup resource that has to be activated if the original one fails
- Can be implemented in a single aspect without modifying the original code

Domain Specific Changes

```
class NewSMTPServer extends SMTPServer {  
    . . .  
}  
public aspect BackupSMTPServer {  
    public pointcut SMTPServerConstructor(URL url, String user, String password):  
        call(SMTPServer.new(..)) && args (url, user, password);  
    SMTPServer around(URL url, String user, String password):  
        SMTPServerConstructor(url, user, password) {  
            return getSMTPServerBackup(proceed(url, user, password));  
        }  
    SMTPServer getSMTPServerBackup(SMTPServer obj) {  
        if (obj.isConnected()) {  
            return obj;  
        }  
        else {  
            return new SMTPServerBackup(obj.getUrl(), obj.getUser(),  
                obj.getPassword());  
        }  
    }  
}
```

Domain Specific Change Implementation (2)

- If we abstract from SMTP servers and resources altogether, it's actually a class exchange
- *Class Exchange* change type based on the *Cuckoo's Egg* aspect-oriented design pattern ⁴

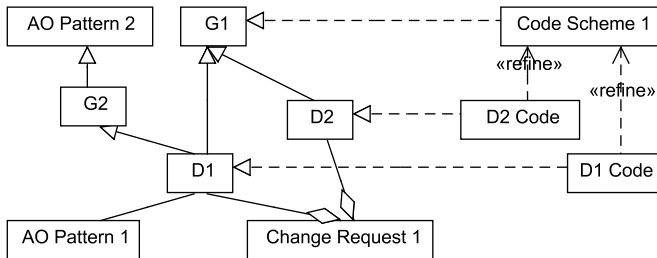
```
public class AnotherClass extends MyClass {  
    . . .  
}  
public aspect MyClassSwapper {  
    public pointcut myConstructors(): call(MyClass.new());  
    Object around(): myConstructors() {  
        return new AnotherClass();  
    }  
}
```

- *Class Exchange* is a *generally applicable* change type

⁴R. Miles. *AspectJ Cookbook*. O'Reilly, 2004.

Applying a Change Type

- How to give a hint to developer to use a particular change type?
- We have to maintain a catalog of changes
- Each domain specific change type is defined as a specialization of one or more generally applicable changes



Catalog of Changes

Applying a Change Type

- To support the process of change selection, the catalog of changes is needed
- It explicitly establishes generalization–specialization relationships between change types
- The following list sums up these relationships between change types we have identified in the web application domain (the domain specific change type is introduced first)

Catalog of Changes in Web Application Domain (1)

- Integration Changes
 - One Way Integration: Performing Action After Event
 - Two Way Integration: Performing Action After Event
- Grid Display Changes
 - Adding Column to Grid: Performing Action After Event
 - Removing Column from Grid: Method Substitution
 - Altering Column Presentation in Grid: Method Substitution

Catalog of Changes in Web Application Domain (2)

- Input Form Changes
 - Adding Fields to Form: Enumeration Modification with Additional Return Value Checking/Modification
 - Removing Fields from Form: Additional Return Value Checking/Modification
 - Introducing Additional Constraint on Fields: Additional Parameter Checking or Performing Action After Event
- Introducing User Rights Management: Border Control with Method Substitution
- User Interface Restriction: Additional Return Value Checking/Modifications
- Introducing Resource Backup: Class Exchange

Changes at the Model Level

Changes at the Model Level

- There may be a need to introduce changes at the model level
- Assume the model is in the Theme notation
- Change types can also be modeled in Theme

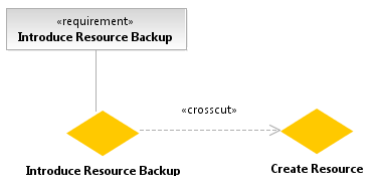
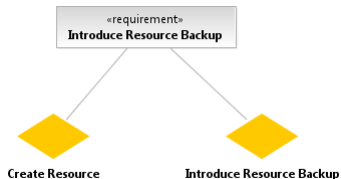
Theme/Doc Domain Specific Change Type Model

- Domain specific change types can be modeled in Theme/Doc⁵
- They are ready to be attached to the themes affected by them

⁵B. Kuliha. Realizing Changes by Aspects at the Design Level. Master thesis in preparation.▶

Theme/Doc Domain Specific Change Type Model

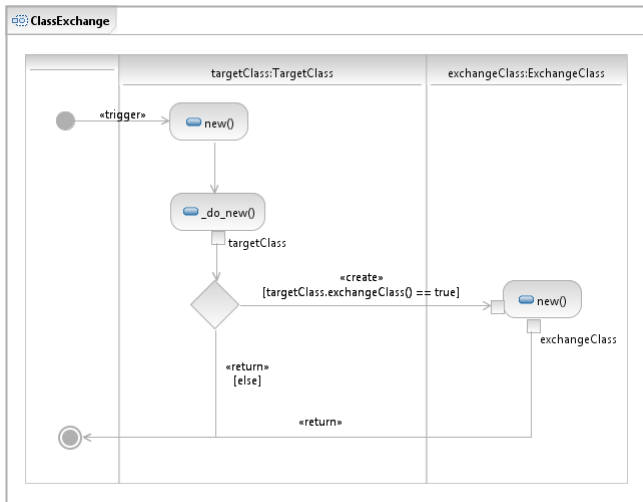
- Themes and relationships view
- Crosscutting view



Theme/UML Generally Applicable Change Type Model (1)

- Generally applicable change types can be modeled in Theme/UML
- They are to be composed with the affected themes

Theme/UML Generally Applicable Change Type Model (2)

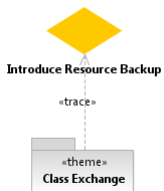


«<TargetClass.new(), TargetClass.exchangeClass(), ExchangeClass.new()> : Expression

«theme»
Class Exchange

Mapping

- We still need a catalog



What if there is no catalog?

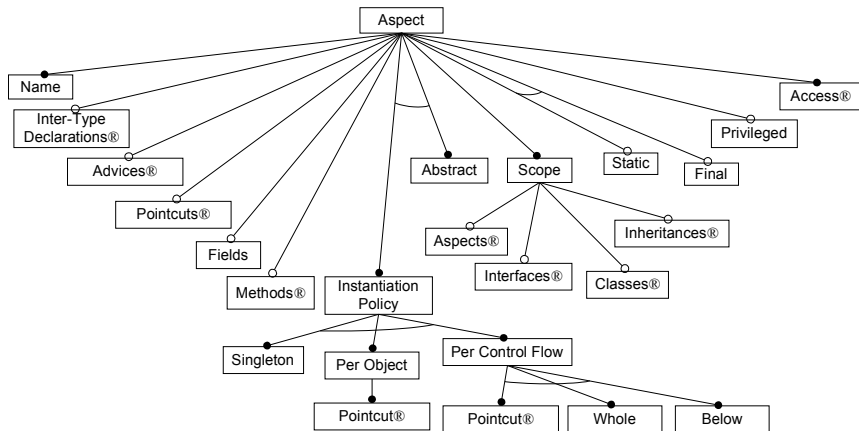
- AOP enables cleaner change realization
- The two-level framework improves this process—assuming there is a catalog of changes
- Creating the catalog of changes may be out of the scope of the momentary needs—to implement a particular change
- The expected number of generally applicable change types that would cover all significant situations is not high
- The problem is in domain specific change types and their mapping to generally applicable change types
- This resembles the problem of the selection of a paradigm suitable to implement a particular application domain concept—a subject of multi-paradigm approaches

Generally Applicable Change Types as Paradigms

Multi-Paradigm Design

- Multi-paradigm design: a process of aligning of application domain structures with the opportunities for their realization in the solution domain
- Solution domain concepts (realization mechanisms) denoted as paradigms
- Transformational analysis
- Multi-paradigm design with feature modeling (MPDFM)
 - Feature modeling is used to express both paradigms and application domain concepts
 - Transformational analysis performed as paradigm instantiation (feature model configuration) over application domain concepts

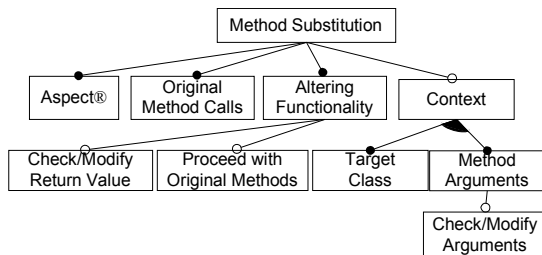
Aspect Paradigm



Constraints:

final \vee *abstract*

Method Substitution

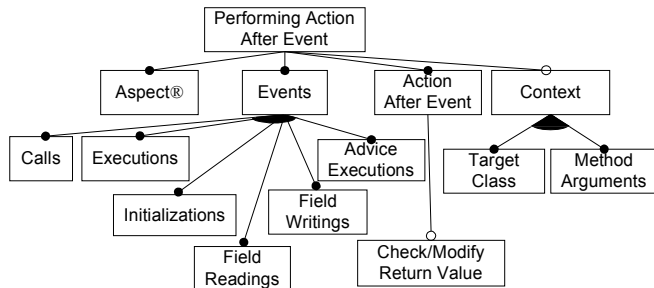


Constraints:

Aspect.Pointcut
Aspect.Advice.Around

```
public aspect MethodSubstitution {
    pointcut methodCalls(TargetClass t, int a): . . . ;
    ReturnType around(TargetClass t, int a): methodCalls(t, a) {
        if (. . .) { . . . } // the new method logic
        else proceed(t, a);
    }
}
```

Performing Action After Event



Constraints:

Aspect.Pointcut

Aspect.Advice.After

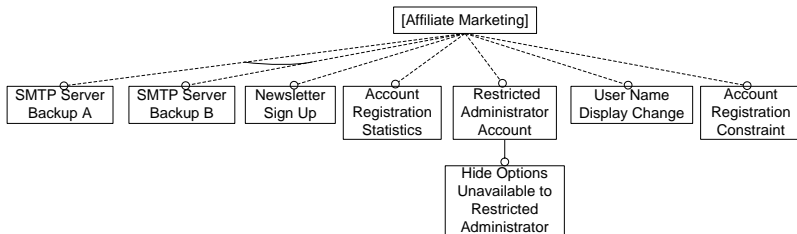
```
public aspect PerformActionAfterEvent {
    pointcut methodCalls(TargetClass t, int a) : . . . ;
    after(/* captured arguments */): methodCalls(/* captured arguments */) {
        performAction(/* captured arguments */);
    }
    private void performAction(/* arguments */) { /* action logic */ }
}
```

Feature Model of Changes

Feature Model of Changes (1)

- Changes are captured in the initial application domain feature model
- All the changes are modeled as optional features of the features they affect
- The feature model expresses constraints among changes
- But the application feature model may not be available
- We may use a partial feature model
- Initially, changes are attached directly to the application concept node

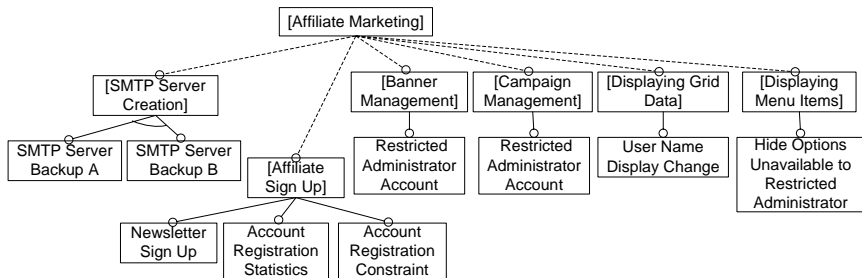
Feature Model of Changes (2)



Partial Feature Model (1)

- The rudimentary partial feature model can be developed further to uncover parent features of the change features as the features of the underlying system affected by them
- Starting at change features, we proceed bottom up identifying their parent features until related features become grouped in common subtrees

Partial Feature Model (2)



Constraints:

Hide Operations Unavailable to Restricted Administrator →
Restricted Administration Account

Transformational Analysis

MPD_{FM} Transformational Analysis (TA)

- The process of finding the correspondence and establishing the mapping between the application and solution domain concepts
- Based on paradigm instantiation (feature model configuration) over application domain concepts
- Input: two feature models—the application domain one and the solution domain one
- Output: a set of paradigm instances annotated with application domain feature model concepts and features that define the code skeleton

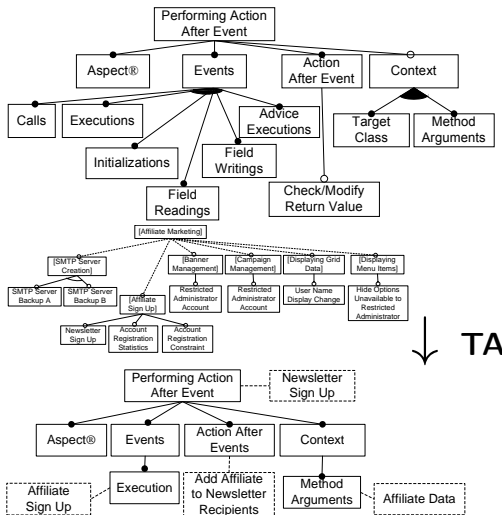
Transformational Analysis of Changes (1)

- Simplified transformational analysis can be used to determine which general change types that correspond to the domain specific changes
- Changes presented in the application domain feature model are considered to be application domain concepts
- Generally applicable change types are considered to be paradigms
- For each change C from the application domain feature model, the following steps are performed:
 - ① Select a generally applicable change type P that has not been considered for C yet
 - ② If there are no more paradigms to select, the process for C has failed.
 - ③ Try to instantiate P over C at source time. If this couldn't be performed or if P 's root doesn't match with C 's root, go to step one. Otherwise, record the paradigm instance created.

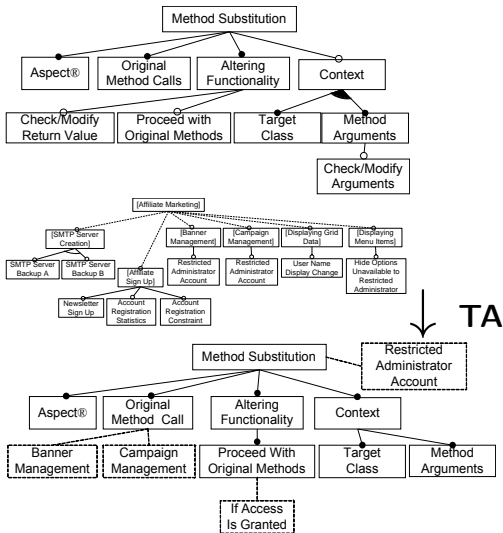
Transformational Analysis of Changes (2)

- Take the subtree in which the change resides
- Instantiate change types until there is a match for the change feature found

Example: Newsletter Sign Up TA



Example: Restricted Administrator Account TA

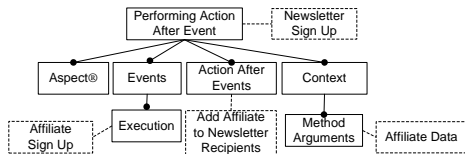


Change Interaction

Change Interaction

- Change realizations can interact:
 - They may be mutually dependent
 - Some change realizations may depend on the parts of the underlying system affected by other change realizations
- Interaction is most probable if multiple changes affect the same functionality
- Such situations could be identified in part already during the creation of a partial feature model
- Transformational analysis can reveal more details needed to identify the interaction of change realizations

Example: Change Interaction and Pointcut Type



- Account Registration Constraint change represents a potential source of interaction with Newsletter Sign Up—they target the same functionality
- Transformational analysis revealed that Newsletter Sign Up relies on method executions, not calls—it employs an **execution()** pointcut
- An interaction: if the Registration Constraint change disables new affiliate registration, Newsletter Sign Up would not be executed either
- If Newsletter Sign Up would rely on method calls, unwanted system behavior would occur


Related Work

Related Work

- Change impact has been studied using slicing in concern slice dependency graphs ⁶
- Changes modeled as application features are close to evolutionary development of a new product line ⁷
- Framed aspects can be used to keep changes separate ⁸

⁶S. Khan and A. Rashid. Analysing requirements dependencies and change impact using concern slicing. In *Proc. of Aspects, Dependencies, and Interactions Workshop (affiliated to ECOOP 2008)*, Nantes, France, July 2006.

⁷J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

⁸N. Loughran et al. Supporting product line evolution with framed aspects. In *Workshop on Aspects, Components and Patterns for Infrastructure Software (held with AOSD 2004, International Conference on Aspect-Oriented Software Development)*, Lancaster, UK, Mar. 2004. 

Summary

Summary

- The original idea: two-level AO change realization framework to facilitate easier aspect-oriented change realization
- Introducing changes at the model level: using Theme to model change types
- How to enable direct change manipulation using multi-paradigm design with feature modeling
- No need for the domain specific change types, nor catalog changes—just paradigm models of the generally applicable changes
- Revealing change interaction based on the results of transformational analysis
- We also developed paradigm models of other generally applicable change types not presented here