Aspect-Oriented Change Realization Based on Multi-Paradigm Design with Feature Modeling

Radoslav Menkyna[1] Valentino Vranić[2]

Softec, spol. s.r.o.
Kutuzovova 23, 83103 Bratislava 3, Slovakia
radoslav.menkyna@softec.sk

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

CEE-SET 2009, October 12–14, 2009, Krakow, Poland

Overview

Why Use Aspects in Change Realization?

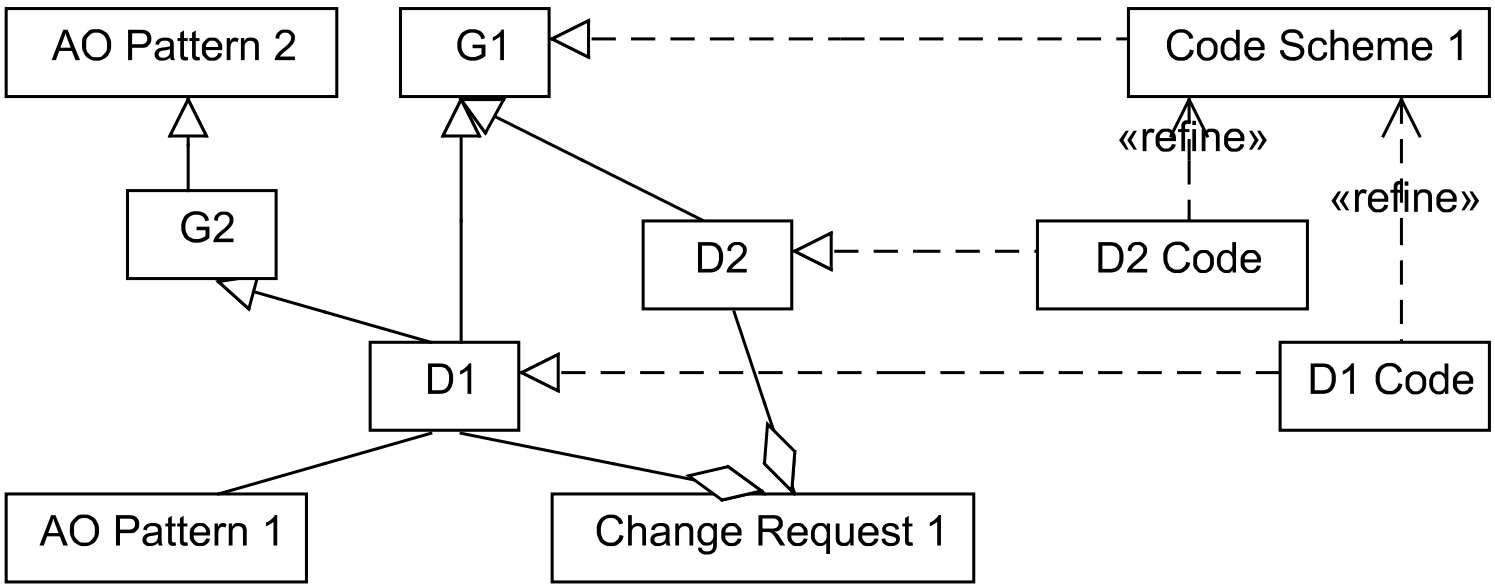
- Change realization is difficult and expensive
- Changes of software applications exhibit crosscutting nature:
  - Intrinsically by being related to many different parts of the application they affect
  - By their perception as separate units that can be included or excluded from a particular application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

Aspect-Oriented Change Realization Research

- Integrative work based on our previous research efforts
- Using aspect-oriented programming to implement changes
- Two-level aspect-oriented change realization framework
- Multi-paradigm design with feature modeling
- Modeling changes as features
- Change interaction as feature interaction

Two-Level AO Change Realization Framework

- How to get to the change realization?
- Two levels of changes:
  - Domain specific changes
  - Generally applicable changes



- Domain specific to generally applicable change mappings are catalogued

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

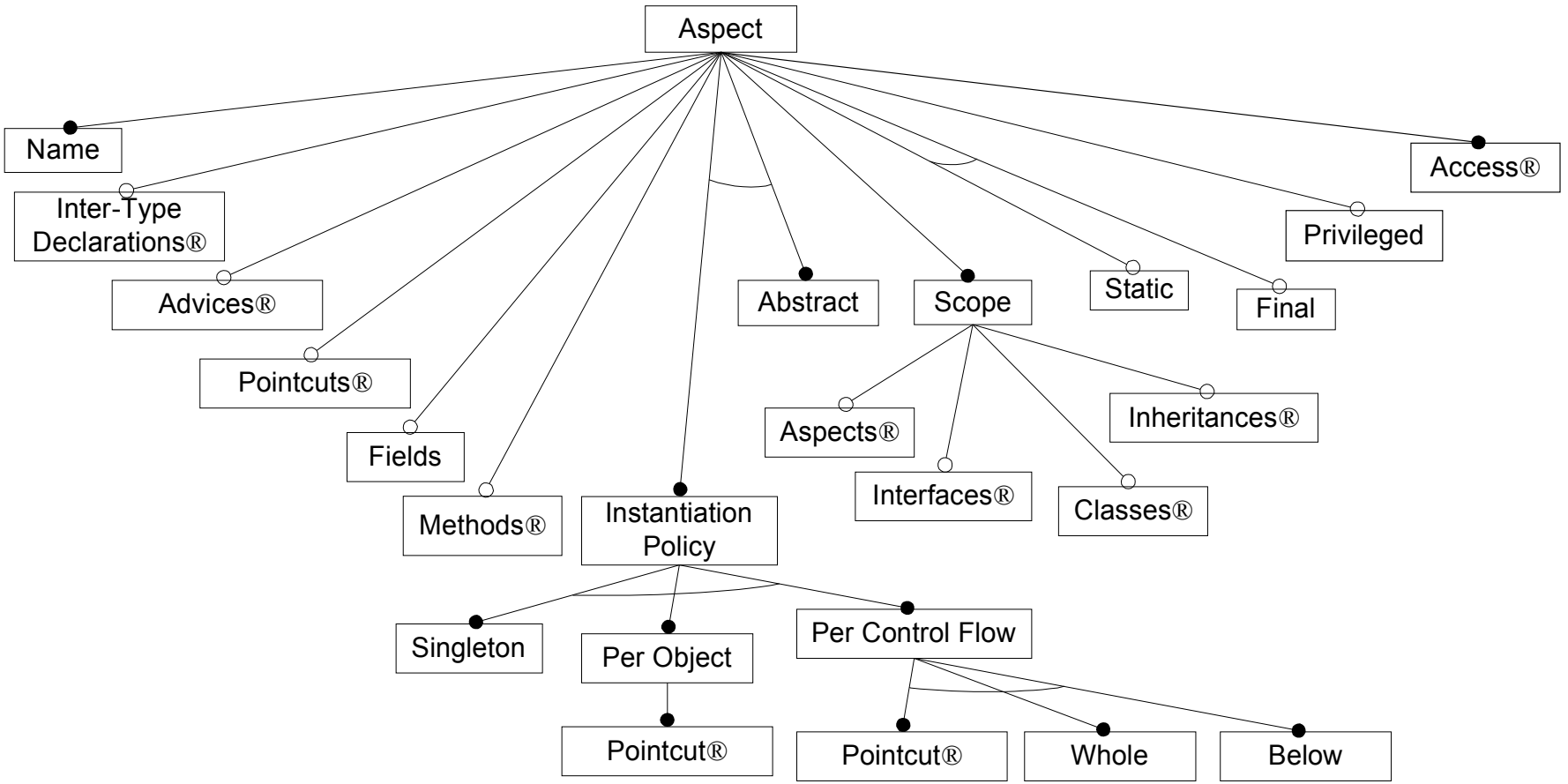
What if there is no catalog?

- 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

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 (MPD_{FM})
  - 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

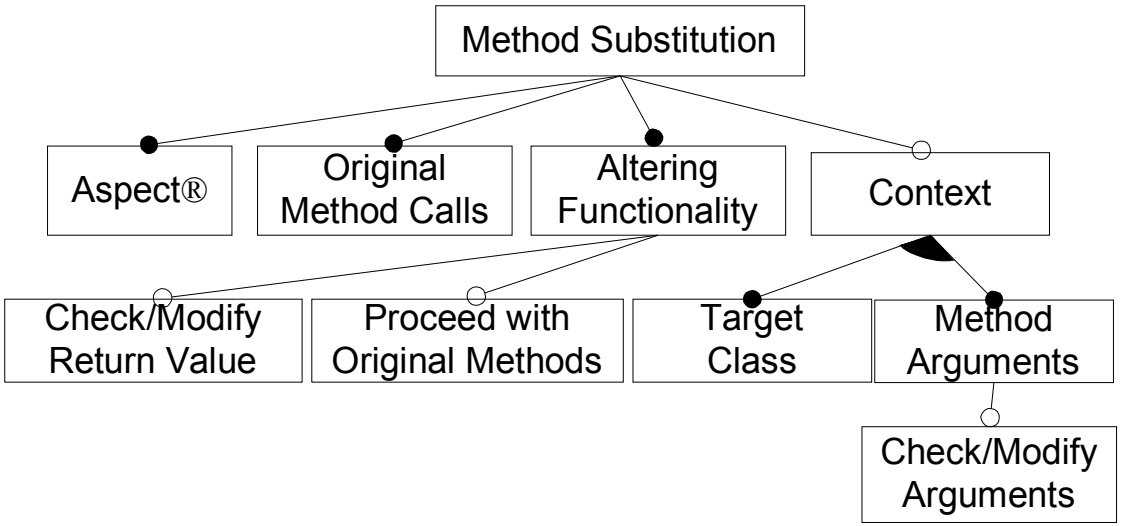
Aspect Paradigm



Constraints:

$final \veebar abstract$

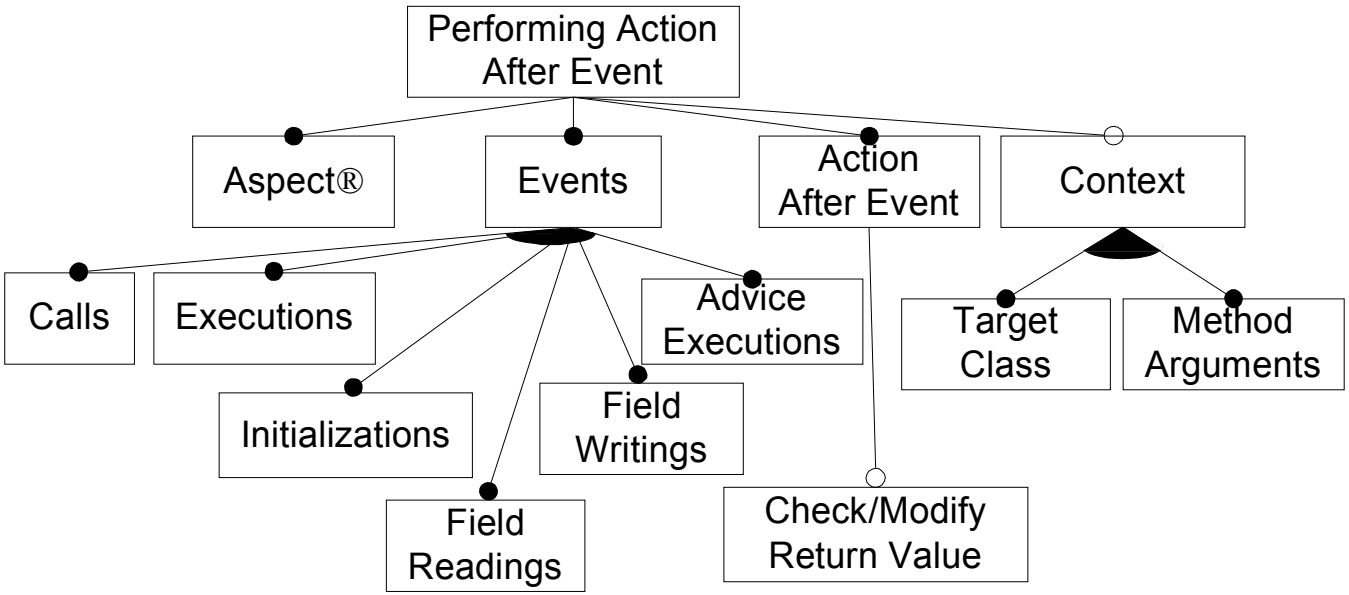
Method Substitution



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

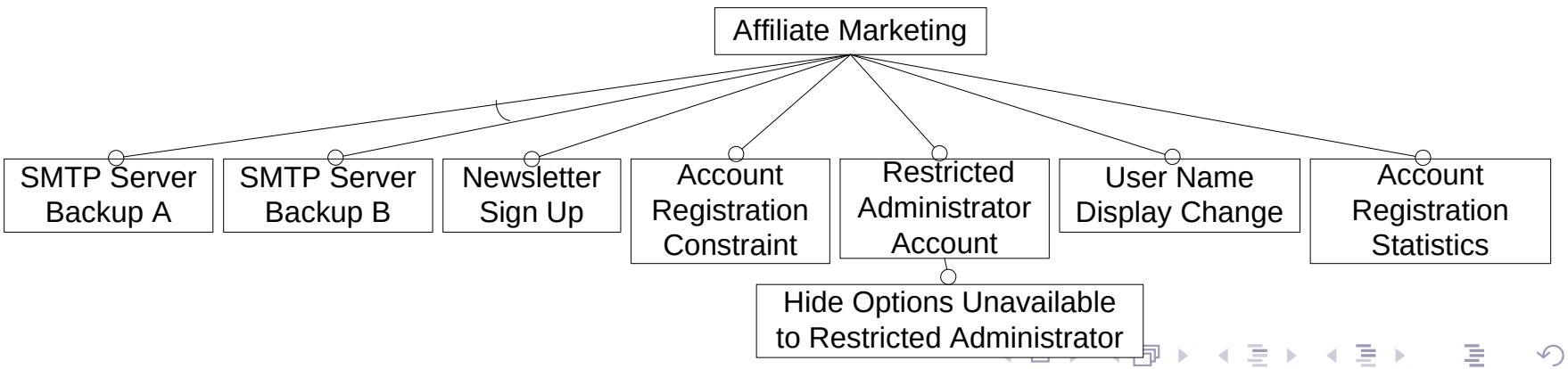


Constraints:
Aspect.Pointcut
Aspect.Advice.After

```
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

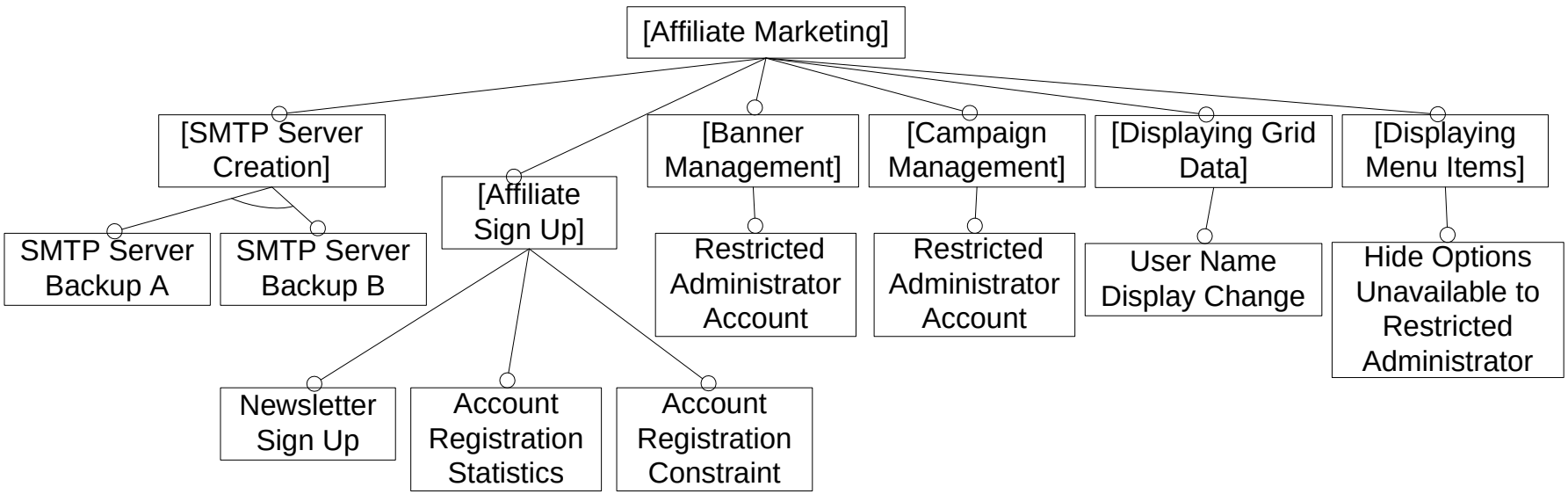
- Changes are captured in the initial application domain feature model
- All the changes are modeled as optional features of the features they affect
- The feature model expresses constraints among changes
- But the application feature model may not be available
- We may use a partial feature model
-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



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

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

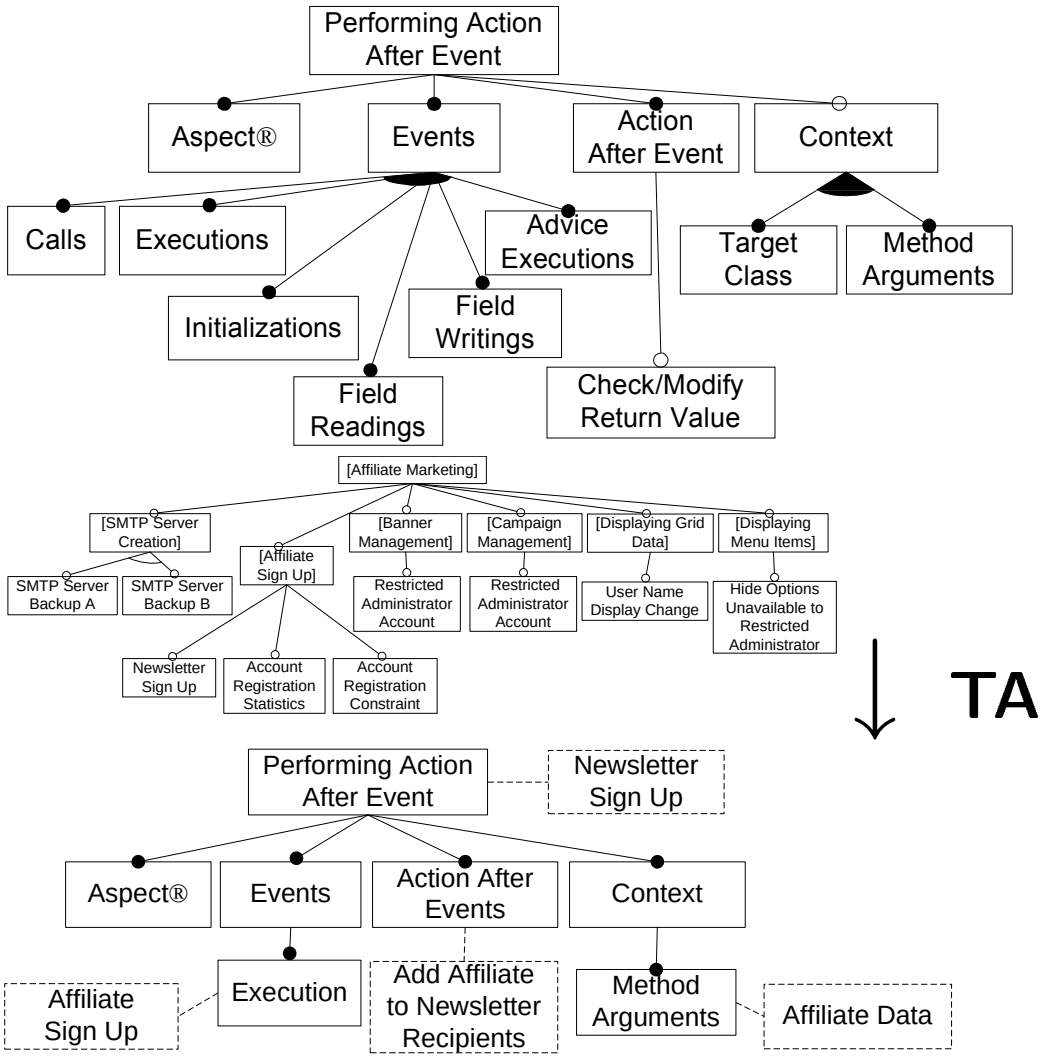
Transformational Analysis of Changes (1)

- 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.

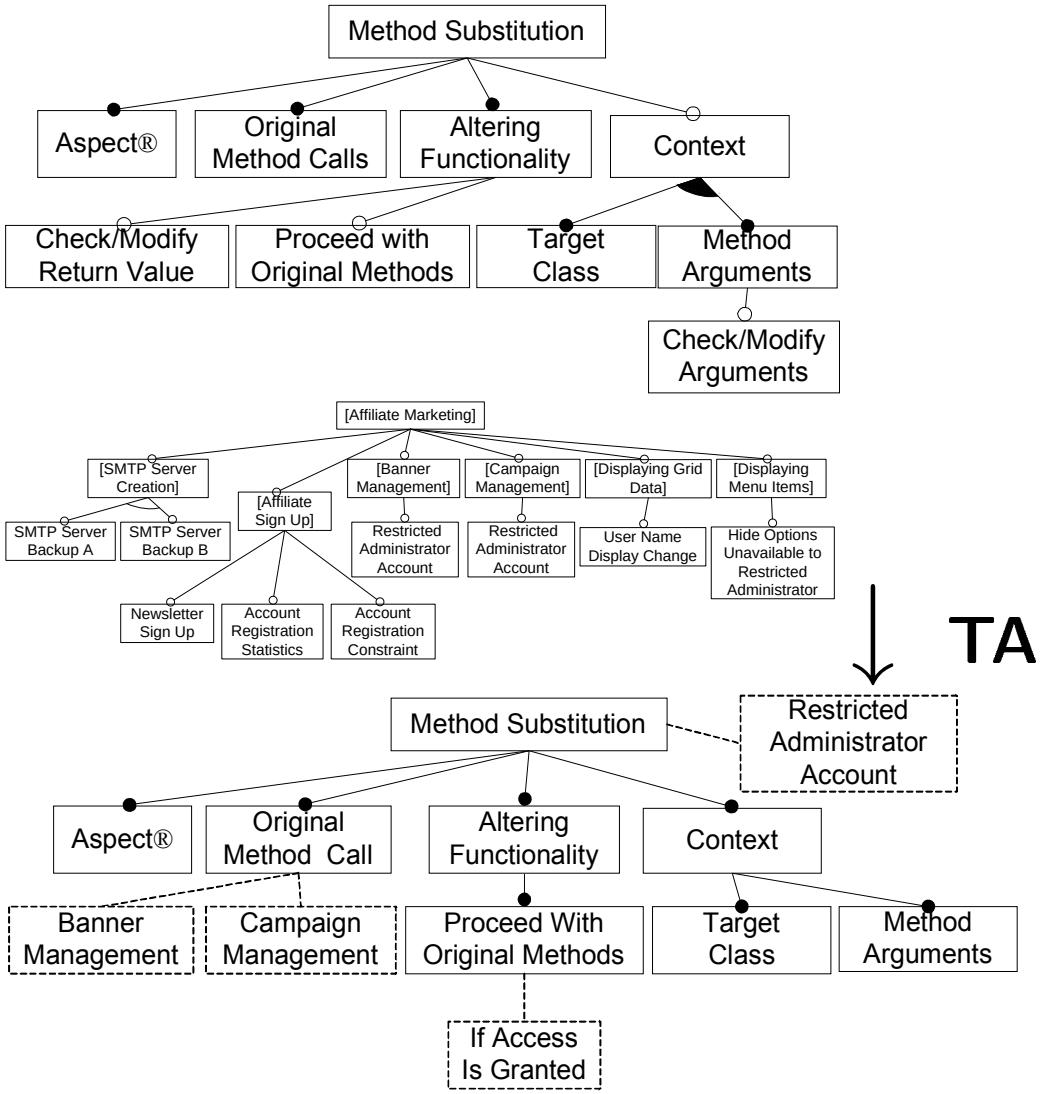
Transformational Analysis of Changes (2)

- 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



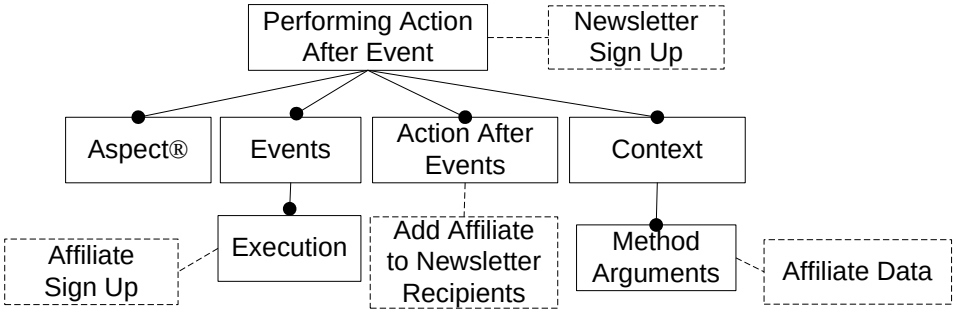
Example: Restricted Administrator Account TA



Change Interaction

- Change 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

- Change impact has been studied using slicing in concern slice dependency graphs [1]
- Changes modeled as application features are close to evolutionary development of a new product line [2]
- Framed aspects can be used to keep changes separate [3]

---

[1] S. 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.

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

[3] 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.

Summary

- The original idea: two-level AO change realization framework to facilitate easier aspect-oriented change realization
- This paper: 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
- Further work
  - Cover the changes realized by a collaboration of multiple generally applicable change types and design patterns
  - Improve change type models by expressing them in the Theme notation