Feature Interactions in Aspect-Oriented Scenario Models

Gunter Mussbacher¹, Daniel Amyot¹, Thomas Weigert², and Thomas Cottenier³

¹SITE, University of Ottawa, Canada
²Missouri University of Science and Technology, Rolla, MO, USA
³Hengsoft LLC, Palatine, IL, USA

June 11, 2009
Background: Aspect-oriented Modeling

- Aspects address the problem of one concern crosscutting other concerns in a system or model.
- Aspects can encapsulate concerns even if they are crosscutting.

**Without Aspects**

Concern A

Concern B

Concern C

Tangling

Scattering

**With Aspects**

Concern A

Aspect 1

Aspectual Properties

Concern B

Aspect 2

Concern C

Aspect 3

(each aspect contains a **composition rule** illustrated by the arrows that defines where to add the aspect)

... 3 Crosscutting Concerns (Aspect 1, Aspect 2, Aspect 3)
Motivation

• Aspect-oriented Feature Development
  • Separation of concerns and minimization of crosscutting
  • One feature = one concern
  • Define features incrementally and independently
  • Resolve feature interactions (FI) incrementally and in a modular way

• Model features and resolve FI in a specification model
  • In this case: Use Case Maps (UCM) model
  • A validation model ensures that the specification model remains compliant with feature specification and FI resolutions
    • UCM validation model is a low effort pre/postcondition approach
  • Concerns should be kept separate in both models
<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Overview of Use Case Maps (UCM)</td>
</tr>
<tr>
<td>• Overview of Aspect-oriented Use Case Maps (AoUCM)</td>
</tr>
<tr>
<td>• Overview of Approach</td>
</tr>
<tr>
<td>• Example: Radio Control Software</td>
</tr>
<tr>
<td>• Features</td>
</tr>
<tr>
<td>• UCM Scenario Definitions</td>
</tr>
<tr>
<td>• Problems</td>
</tr>
<tr>
<td>• Feature Interactions</td>
</tr>
<tr>
<td>• Conclusion and Future Work</td>
</tr>
</tbody>
</table>
Use Case Maps: Overview

• User Requirements Notation (URN)
  • First and currently only standard which explicitly addresses goals-based and scenario-based models in one unified language

• Use Case Maps (UCMs) – scenario models

  ![Diagram of UCM example]

  - UCM scenarios describe one path through the UCM model (only one alternative at any choice point is taken)
  - jUCMNav’s traversal mechanism executes the UCM model given a UCM scenario description (i.e. highlights the scenario)
Use Case Maps: Notation

UCM Example: Tiny Telephone System

Start Point
Dynamic Stub
AND (fork)
Component
Originating

Responsibility
Condition
End Point

a) Basic Call map
b) OCS plug-in map
c) default plug-in map

AND (join)
Static Stub (at most one plug-in map)
Synchronizing Stub with synchronization threshold
Blocking Stub with replication indicator

waiting place
timer

AND (fork)

Use Case Maps: Path Traversal Mechanism

UCM Example: Tiny Telephone System

Selection Policy:
OCS: OCS
default: not(OCS)

Branch Conditions
[not (Busy)]
[OCSdenied]

a) Basic Call map

- Scenario Definition “Simple Basic Call”
  - Start point: req
  - OCS = false
  - Busy = false
  - End points: ring, sig

b) OCS plug-in map

c) default plug-in map
Use Case Maps: Path Traversal Mechanism (2)

UCM Example: Tiny Telephone System

Selection Policy:
- OCS: OCS default: not(OCS)

Branch Conditions

a) Basic Call map

- Scenario Definition “Busy Call + OCS”
  - Start point: req
  - OCS = true
  - OCSdenied = false
  - Busy = true
  - End point: sig

b) OCS plug-in map

c) default plug-in map
An aspect defines its structure/behavior and a pattern called pointcut expression for its composition rule stating where the aspect is to be applied in a model.

**Aspect Map**

- **Start**
- **Pointcut**
- **Behavior.before**
- **Behavior.after_success**
- **Behavior.after_fail**
- **End.Success**
- **End.Fail**

**Base Model**

- **Start**
- **End.Success**
- **End.Fail**
- **R0**
- **R1**
- **Fail**
- **Success**

**Pointcut Map**

- **Pointcut stub:**
- **Aspect marker:**

(Plugin bindings (dashed and long-dash-dot-dotted arrows) and matched pointcut expression only shown for illustration purposes – not part of concrete syntax of the AoUCM notation.)
Overview of Approach

• Step 1: Model each feature as a concern (aspect)
• Step 2: Create scenario definitions (= validation model)
  • Complete branch coverage for each feature
  • Global variables, formalization of conditions for choice points, pseudo-code for responsibilities
• Step 3: Model FI resolutions with aspects
  • Model as part of existing feature concerns
• Step 4: Create scenario definitions for FI resolutions
  • Reuse scenario definitions of individual features as much as possible
  • Requires composition rules for aspect-oriented scenario definitions
Features of Radio Device

- **10 Features**
  - Select Band, Tune, Autotune, Save, Recall, Traffic News, Power On / Off, Standby On / Off

- **Update Display**
- **Remember Settings**
  - Radio device remembers last band and frequency settings when turned off, defaults to these settings when turned on
Example of Features: Select Band

• Step 1: Model feature

User

Radio

select

band

setBand(AM)

selected

[FM]

[AM]

• Step 2: Create scenario definitions

• Global enumeration variable: Band = { AM, FM }

• Formalize conditions: [FM] … Band == FM, [AM] … Band == AM

<table>
<thead>
<tr>
<th>Select Band</th>
<th>Scenario One</th>
<th>Scenario Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Included Scenarios</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Start Points</td>
<td>select</td>
<td>select</td>
</tr>
<tr>
<td>Initialization</td>
<td>Band = FM</td>
<td>Band = AM</td>
</tr>
<tr>
<td>End Points</td>
<td>selected</td>
<td>Selected</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Band == AM</td>
<td>Band == FM</td>
</tr>
</tbody>
</table>

• Code for responsibilities changes the value of variable Band

• setBand(AM) … Band = AM; setBand(FM) … Band = FM
UCM Scenario Definitions

• Scenario definitions may be included in other scenario definitions
  • Union of start points, end points, and pre/postconditions
  • Start points ordered by scenario include order
  • Initializations: applied in scenario include order before scenario starts
    – later ones may override earlier ones

<table>
<thead>
<tr>
<th>Scenario Definition</th>
<th>Scenario One</th>
<th>Scenario Two</th>
<th>Combined Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Included Scenarios</td>
<td>n/a</td>
<td>n/a</td>
<td>One, Two</td>
</tr>
<tr>
<td>Start Points</td>
<td>start1</td>
<td>start2</td>
<td>start1, start2</td>
</tr>
<tr>
<td>Initialization</td>
<td>var1 = X, var2 = Y</td>
<td>var1 = Z</td>
<td>var1 = Z, var2 = Y</td>
</tr>
<tr>
<td>End Points</td>
<td>end1</td>
<td>end2</td>
<td>end1, end2</td>
</tr>
<tr>
<td>Postcondition</td>
<td>var2 = A</td>
<td>var1 = B</td>
<td>var1 = B, var2 = A</td>
</tr>
</tbody>
</table>

• Elements may be added to combined scenario definitions
• Next start point of scenario definition is only considered if the traversal of the previous start point is finished or stuck
Problem 3

3) Control variables for loops (counters)
   - **NEW** Traversal mechanism exposes element hit count
   - `_hitCount`
Problems 1, 2, and 4

1) Scenario stops at path element that is not an end point (e.g. OR-fork of infinite loop)
   - NEW Allow any UCM element to be end of scenario

2) <feature> Active variables
   - NEW Traversal mechanism exposes activity status
   - _activeCount

4) Same feature several times in a row
   - NEW Support array type variables
   - TuneDirection = { UP, UP, DOWN, UP, DOWN, UP… }
Problem 5

5) Redundant start points

<table>
<thead>
<tr>
<th>Feature</th>
<th>Start Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power On</td>
<td>power</td>
</tr>
<tr>
<td>Remember Settings</td>
<td>initialize</td>
</tr>
<tr>
<td>Interaction</td>
<td>power, initialize</td>
</tr>
</tbody>
</table>

- The Remember Settings map is traversed another time after the enabled end point was reached
- **NEW** declare start point initialize as redundant
Problems 6 and 7

6) Changes to scenario definition
   • **NEW** Elements may be deleted from scenario definitions

7) Interrupting a scenario
   • **NEW** Allow interruption of a scenario before/after a path element if a condition evaluates to true
Example of Feature Interaction: Abort Autotune

• Step 3: Model FI resolution
  • Autotune must be aborted when Select Band, Save, Recall, Power Off, or Standby Off are activated
  • Modeled as part of the Autotune feature concern
Example of Feature Interaction: Abort Autotune

- Step 4: Create scenario definitions for FI resolution
  - Interaction Scenario Definition:
    - **include** Autotune, Select Band
    - **interrupt** after OR-join in Autotune if _hitCount == 50
    - **delete** OR-join; **add** abort

<table>
<thead>
<tr>
<th>Scenario Definition</th>
<th>Autotune</th>
<th>Select Band</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Included Scenarios</td>
<td>n/a</td>
<td>n/a</td>
<td>Autotune, Select Band</td>
</tr>
<tr>
<td>Start Points</td>
<td>autotune</td>
<td>select</td>
<td>autotune, select</td>
</tr>
<tr>
<td>Initialization</td>
<td>Freq = 32, StrongFreq = -1</td>
<td>Band = AM</td>
<td>Freq = 32, StrongFreq = -1 Band = AM</td>
</tr>
<tr>
<td>End Points</td>
<td>OR-join</td>
<td>selected</td>
<td><strong>OR-join, abort,</strong> selected</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Freq != StrongFreq</td>
<td>Band == FM</td>
<td>Freq != StrongFreq Band = FM</td>
</tr>
</tbody>
</table>
Feature Interactions

<table>
<thead>
<tr>
<th>#</th>
<th>Features</th>
<th>Sel. B.</th>
<th>Tune</th>
<th>Autotune</th>
<th>Save</th>
<th>Recall</th>
<th>Tr. N.</th>
<th>P. On</th>
<th>P. Off</th>
<th>S. On</th>
<th>S. Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Update Display</td>
<td>ASF</td>
<td>ASF</td>
<td>ASF</td>
<td>ASF</td>
<td>ASF</td>
<td>ASF</td>
<td>ABF</td>
<td>BEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Remember Settings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **ASF**…After setting frequency; **ABF**…After beginning of feature; **BEF**…Before end of feature

• 12 Abort: one feature aborts another
• 14 Ignore: one feature is ignored because of another feature
• 2 Synergy: creates something useful and new for user
• 3 for Display, 2 for Remember Settings
• Impossible: due to radio device interface constraints
Conclusion

- Integrated aspect-oriented specification and validation scenario models at the requirements stage
- Incremental development of features
- **Composition rules** for aspect-oriented validation models
  - Include, add, delete, interrupt, redundant

- Reduced complexity of AoUCMs
- Improved scalability
- Feature concerns and FI resolution concerns are properly encapsulated
- Specification model is kept strictly separate from the validation model (thus further improving separation of concerns)
Future Work

• It may be possible to infer composition rules for the validation model from the aspect-oriented composition rules of the specification model

• It may be useful to provide more internal information of the path traversal mechanism

• Context information still needs to be addressed in scenario definitions
  • Requires the introduction of concrete component instances
  • Requires a rethinking of the global data model
  • Requires the introduction of scoping rules