Path Expression Pointcuts

Problem Description

Overview of «Path Expression Pointcuts» («PEP»)

Problematic AspectJ Solution:

public aspect PersistedListAspect {
  ...}

pointcut persistentList(PersistedList list);

execution(new(o)) @ target(list);

after(setting)(PersistedList list; persistLists(List) {
  orderable.list() }

public aspect changePersonAspect {
  ...}

pointcut changePerson(Object o);

set("o") @ target(o);

after (setting)(Object o; changePerson(o);

PersistedList list = getlist(o);

list.notifyChanges(o);

1. PersistedList

2. Relevant non-local context

3. Local join point context in AspectJ

4. Local join point context in AspectJ

5. Relevant non-local context information

6. Relevant non-local context information

7. Store Person that is part of a PersistedList

Technical Issues

- Field-based associations considered
- Pointcut designator syntax:
  - Path(PathExpressionPattern)
- PathExpressionPattern
  - Abstract Syntax:
    - PathExpressionPattern = ObjectPattern -> FieldPattern
  - Non-local context
- Problematic AspectJ Solution:
  - Public PersistencePattern
    - ObjectPattern = [Type]/IdPattern
    - FieldPattern := IdPattern | MemberPattern := IdPattern | * | "*" | "~" | "@" | "\" | "|" | "!" | "==" | "|="
- Resultset: A sequence of matched paths
- Context Exposure
- All objects belong to the matched paths are exposed to the pointcut context. This has a small impact since we hold the whole object graph.
- Path Parameters Binding
  - PEP is responsible for binding all of its parameters to the corresponding objects from the join point.
  - E.g.,
    - pointcut(Person p, String lname,
      Company c):
      - path(Company c → p → lname)
      - This pointcut select all paths from the object graph where the source object is "c" and destination is "lname", and goes through the object "p". The PEP will bind the parameters "lname", "p", and "c" to the right objects in the join point context.
- Ordering Issues
  - Specification of ordering via order by clause

Features

- Abstractions for objects, object relationships and actual types
- Specify sources and/or targets of the intended paths and inner objects
- Permit specify associations by name
- Object graphs are local to join points
- Multiple paths ordering mechanisms
- Operators on paths for composition, reduction, etc.

Hypothesis

- Hypothesis 1: Aspect-Oriented systems promises better quantified object persistence concern
- Hypothesis 2: Current AOP systems suffer from losing relevant object information

Introduction

- The persistence problem: quantify on persistent parts of an application code
- This quantification spreads over the application code
- AOP aims to separation of concerns
- Pointcut languages specify join points where persistence manipulations take place
- Persistence aspects need the objects participate in the join points

Motivation

- Aspect-oriented systems provide join point context for:
  - Join point selection
  - Join point adaptation
- The context exposed is "local" to the join point
- These systems fails when aspects need information "non-local" to the join point
  - E.g., uni-directional associations between objects often tend to be "non-local" to the join point
- Consequence: Need for object graphs accessible via pointcuts and advices

Other applications of PEP: Observer Aspect

- Notify Observers of the Person who is a Student on any change to its Address

Work Ahead

- Specification of path interface
- Considering signaturs for path specifications
- Implementing persistence aspects using PEP
- Integrating persistence aspects with other related aspects