Path Expression Pointcuts

Mohammed Al-Mansari
University of Duisburg-Essen
Institute for Computer Science and Business Information Systems
Data Management Systems and Knowledge Representation
Prof. Dr. Rainer Unland
Schützenbahn 70, D-45117 Essen, Germany
Phone: +49 (0)201 / 183 - 2467, Fax: +49 (0)201 / 183 - 4460
Email: Mohammed.Al-Mansari@icb.uni-due.de
Web: http://dawis.icb.uni-due.de/?id=40&sname=Al-Mansari

Hypothesis

- **Hypothesis 1:**
  Aspect-Orientation 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**
- **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

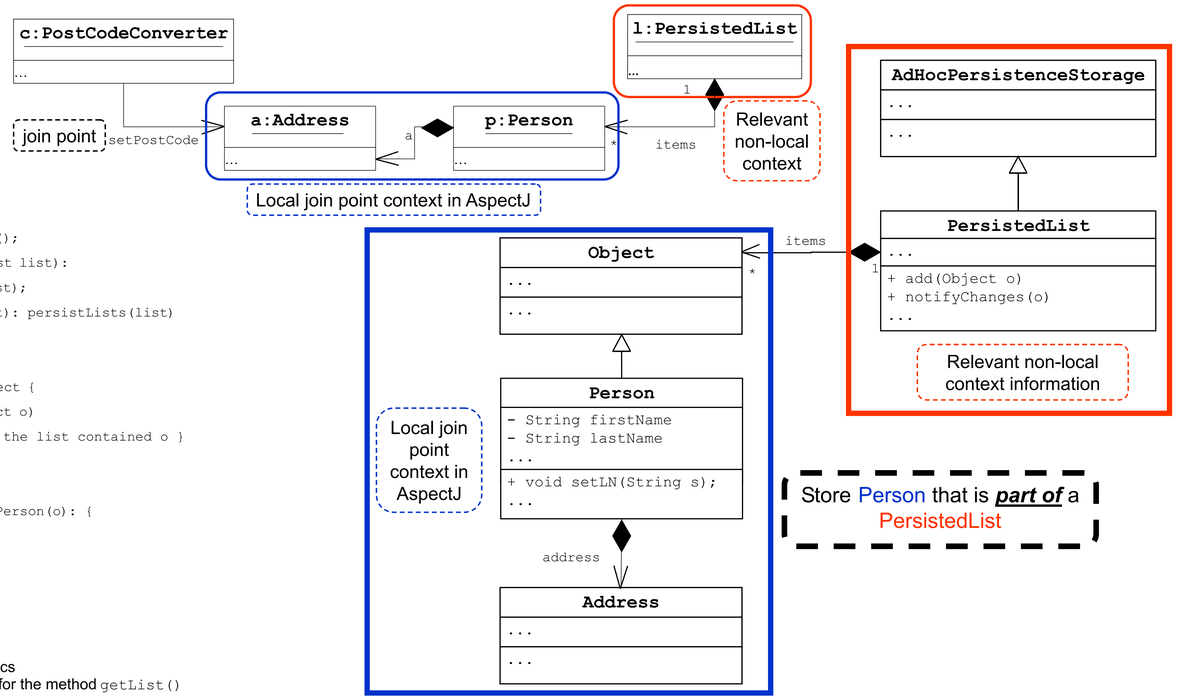
- **Aspect-oriented systems provide join point context for:**
  - **Join point selection**
  - **Join point adaptation**
- **The context exposed is "local" to the join point**
- **These systems fails when aspects need information "non-local" to the join points**
  - **E.g. uni-directional associations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

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.**

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

Problem Description



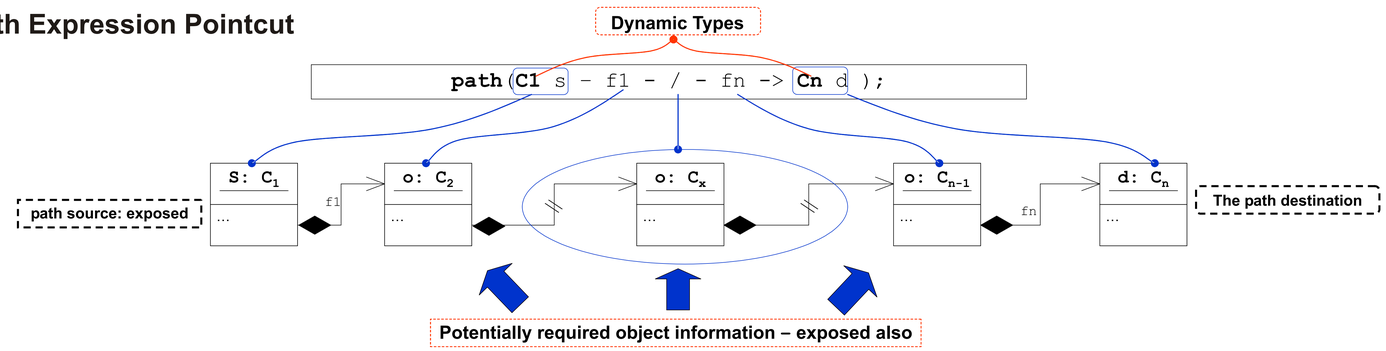
Problematic AspectJ Solution:

```
public aspect PersistListsAspect {
  ArrayList listArr = new ArrayList();
  pointcut persistLists(PersistedList list):
    execution(new(..)) && target(list);
  after returning(PersistedList list): persistLists(list)
  {listArr.add(list); }
}
public aspect ChangeNotificationAspect {
  public PersistedList getList(Object o)
  { // traverse all listArr to find the list contained o }
  pointcut changePerson(Object o):
    set(* *.o) && target(o);
  after returning(Object o): changePerson(o): {
    Persistedlist list = getList(o);
    list.notifyChanges(o);
  }
}
```

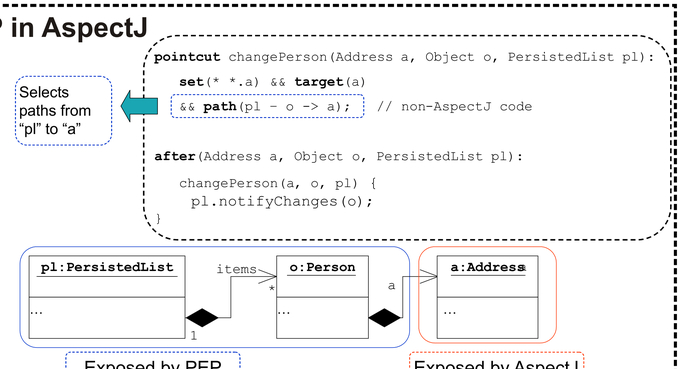
Problems with AspectJ Solution:

- Lose of the join point selection semantics
- The need for specific implementations for the method `getList()`

Path Expression Pointcut

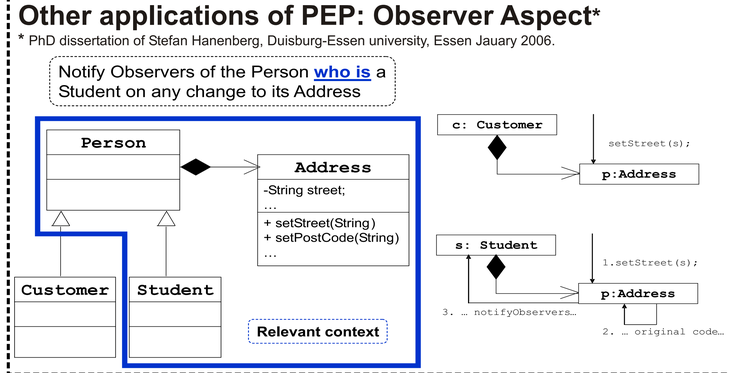


PEP in AspectJ



Other applications of PEP: Observer Aspect*

* PhD dissertation of Stefan Hanenberg, Duisburg-Essen university, Essen Jauary 2006.



Technical Issues

- **Field-based associations considered**
- **Pointcut designator syntax:**

```
Path(PathExpressionPattern);
```

- **PathExpressionPattern Abstract Syntax:**

```
PathExpressionPattern ::= ObjectPattern -
  FieldPattern -> ObjectPattern
ObjectPattern ::= [Type] IdPattern
MemberPattern ::= IdPattern | "*" | "/"
IdPattern ::= sequence of characters
```

- **Path Expression Composition:**

```
Qualifier ::= Qualifier1 && Qualifier2
        | Qualifier1 || Qualifier2
        | !  (Qualifier)
        | PathExpressionPattern
        | (Qualifier)
```

- **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 `orderBy` clause

Work Ahead

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