Guiding the use of AspectJ Advice: An Initial Assessment

Henrique Rebêlo and Márcio Ribeiro
Informatics Center - Federal University of Pernambuco - Brazil
{hemr, mmr3}@cin.upe.br

Characteristics:

- Initial assessment of AspectJ advice using:
  - Running time and bytecode size metrics
- Preliminary guide to implement crosscutting concerns using proper AspectJ advice

Benefits:

- Guidelines for AspectJ advice usage
- Guidelines for aspect-oriented refactoring
  - Refactorings of crosscutting concerns
  - Refactorings for AOP constructs
- A case study with preliminary results
  - before and after-returning X around
  - before and after-throwing X around

Drawbacks:

- Only one case study
- Only two scenarios
- Only one concern analyzed

Original aspect-oriented refactoring of the transaction concern in HW

```
class HWFacade {
  Complaint searchComplaint(int code) {
    Complaint c = null;
     try {
       beginTransaction();
       c = search(code);
       commitTransaction();
     } catch(TransactionException e) {
         rollbackTransaction();
     }
     return c;
  }
  ...
}
```

```
class HWFacade {
  Complaint searchComplaint(int code) {
    Complaint c = null;
    c = search(code);
    return c;
  }
  ...
}
aspect HWTransaction {
  pointcut transMeths():
    execution(* HWFacade.*(..));

  before() : transMeths() {
    beginTransaction();
  }
  after() returning: transMeths() {
    commitTransaction();
  }
  after() throwing: transMeths() {
    rollbackTransaction();
  }
}
```

Scenario 1

```
class HWFacade { ... }

aspect HWTransaction {
  pointcut transMeths():
    execution(* HWFacade.*(..));

  before() : transMeths() {
    beginTransaction();
  }
  after() returning: transMeths() {
    commitTransaction();
  }
}
```

```
class HWFacade { ... }

aspect HWTransaction {
  pointcut transMeths():
    execution(* HWFacade.*(..));

  Object around() : transMeths() {
    object result = null;
     try {
       beginTransaction();
       result = proceed();
       commitTransaction();
     } catch(Exception e) {
       rollbackTransaction();
       throw e;
    }
}
```

before and *after-returning* versus *around* advice

Scenario 2

```
class HWFacade { ... }

aspect HWTransaction {
  pointcut transMeths():
    execution(* HWFacade.*(..));

  before() : transMeths() {
    beginTransaction();
  }
  after() throwing: transMeths() {
    rollbackTransaction();
  }
}
```

```
class HWFacade { ... }

aspect HWTransaction {
  pointcut transMeths():
    execution(* HWFacade.*(..));

  Object around() : transMeths() {
    object result = null;
     try {
       beginTransaction();
       result = proceed();
       commitTransaction();
     } catch(Exception e) {
       rollbackTransaction();
       throw e;
    }
}
```

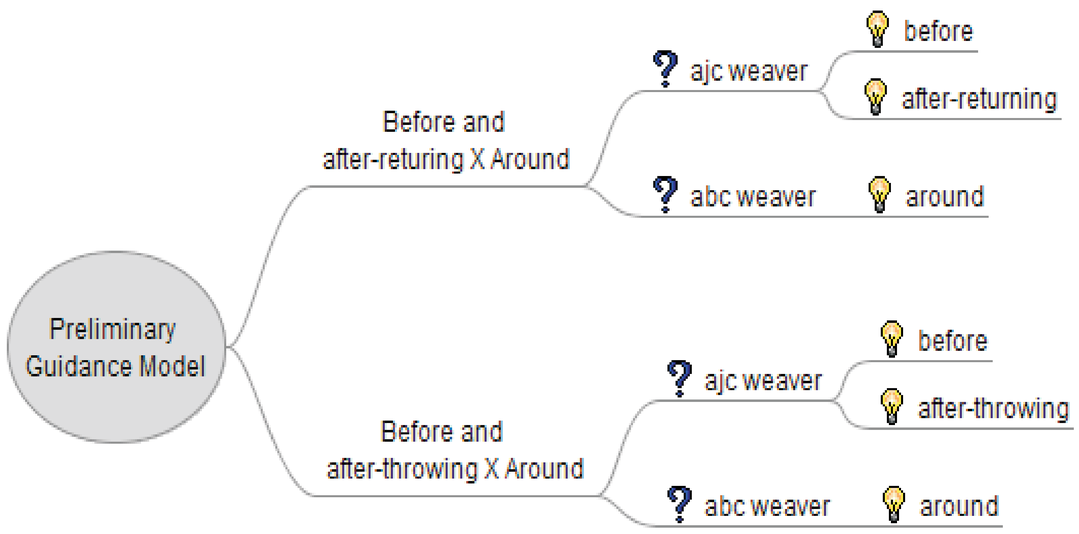
before and *after-throwing* versus *around* advice

Running time and bytecode size results

	Running time (sec)	Bytecode size (KB)
Original/ajc	**0.89521445**	**291**
Original/abc	1.26414025	264
Alternative/ajc	0.9141467	303
Alternative/abc	**0.7168156**	**273**

The results provide an evidence that the “Original/ajc” and “Alternative/abc” are the best AOP implementations

Summary



The “alternative implementation” combined with abc weaver is indicated for constrained environments