Deriving Refactorings For AspectJ

Leonardo Cole and Paulo Borba
Software Productivity Group
Informatics Center - Federal University of Pernambuco - Brazil
{lcn,phmb}@cin.ufpe.br
http://www.cin.ufpe.br/spg

Problem:

- Refactorings are generally complex and global
- Aspect-oriented developers are identifying common transformations for aspect-oriented programs
- It is difficult to verify if a defined refactoring preserves behaviour

Solution:

- Basic laws of programming, defining two behaviour preserving transformations: one from left to right and another in the opposite direction
- Guarded by pre-conditions on each direction
- We can derive complex and global refactorings, including their pre-conditions, as a composition of basic laws
- It is possible to verify that complex refactoring preserves behaviour
- Laws are simple, localized, intuitive and easier to understand
- Syntactic pre-conditions simplifies tool implementation to provide automation

Drawbacks (future work):

- Laws were not proved to be sound with respect to a formal semantics
- Set of laws is incomplete; for instance, it does not cover abstract aspects and get and set pointcuts

Laws

Bi-directional behaviour preserving transformations
Simple, local, and handles one construct at a time

Examples

Merge advices before

```
eds
privileged aspect A {
  pcds
  before(pds):exp1{
    body
  }
  before(pds):exp2{
    body
  }
  ads
}
```

=

```
eds
privileged aspect A {
  pcds
  ads
  before(pds) :
    exp1 || exp2 {
    body
  }
}
```

(↔) *exp1* e *exp2* bind all parameters in pds.

Make aspect privileged

```
eds
aspect A {
  pcds
  ads
}
```

=

```
eds
privileged aspect A {
  pcds
  ads
}
```

(←) *ads* do not reference private classes, aspects, methods and fields in *eds*.

Add before-execution

```
eds
class C {
  fds
  mds
  T m(pds) {
    body';
    body
  }
}
```

```
privileged aspect A {
  pcds
  ads
}
```

=

```
eds
class C {
  fds
  mds
  T m(pds) {
    body
  }
}
```

```
privileged aspect A {
  pcds
  ads
  before(context) :
    exec(C.m) &&
    bind(context) {
    body'[cthis/this]
  }
}
```

(→) *body'* does not declare or use local variables; *body'* does not call **super**; body' only access C members through **this**
(↔) body' does not call **return**;

Summary

Law	Name
1	Add empty aspect
2	Make aspect privileged
3	Add before-execution
4	Add before-call
5	Add after-execution
6	Add after-call
7	Add after returning-execution
8	Add after returning-call
9	Add after throwing-execution
10	Add after throwing-call
11	Add around-execution
12	Add around-call
13	Merge advices
14	Remove this parameter
15	Remove target parameter
16	Remove argument parameter
17	Add catch for softened exception
18	Soften exception
19	Remove exception from throws clause
20	Remove exception handling
21	Move exception handling to aspect
22	Move field to aspect
23	Move method to aspect
24	Move implements declaration to aspect
25	Move extends declaration to aspect
26	Extract named pointcut
27	Use named pointcut
28	Move field introduction up to interface
29	Move method introduction up to interface
30	Remove method implementation

Refactorings

Behaviour preserving transformations
Usually global and complex

Composing Laws to Derive Refactorings

Law 26 ⇒ Law 27

Extract Pointcut Refactoring

```
eds
aspect A {
  pcds
  ads
  before(pds) : exp {
    body
  }
  after(pds) : exp {
    body'
  }
}
```

=

```
eds
aspect A {
  Pcds
  ads
  pointcut p(pds) : exp
  before(pds) :p(αpds) {
    body
  }
  after(pds) :p(αpds){
    body'
  }
}
```

(→) There is no pointcut named *p* in *pcds*;
(←) *eds*, *ads* and *pcds* do not reference *p*

Other Derived Refactorings

Extract method call

Law 3 ⇒ Law 13 ⇒ Law 14 ⇒ Law 16 ⇒ Extract Pointcut

Extract Interface Implementation

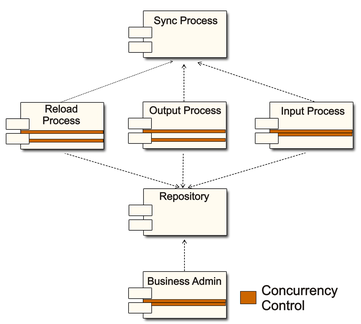
Law 23 ⇒ Law 22 ⇒ Law 28 ⇒ Law 29 ⇒ Law 30

Extract Exception Handling

Law 17 ⇒ Law 18 ⇒ Law 19 ⇒ Law 20 ⇒ Law 21

Case Study: Concurrency in a Commercial Replicated Database Application

- Intended to provide replication and synchronization of data that must be used off-line in different platforms (including mobile devices)
- Keeps information of the modifications made by each user on each platform
- Solve modification conflicts
- Propagate the resulting modifications to replicas



Modularizing concurrency control

General OO Refactorings
+
Laws: 1, 2, 3, 5, 11, 12, 15, 23
+
Derived refactorings:
Extract Pontcut
Extract Exception Handling