An Aspect for Resolving Aspect Interactions

Fuminobu Takeyama and Shigeru Chiba, Tokyo Institute of Technology, Japan

Resolution of interactions by aspect

- Create a new aspect to overwrite interacting aspects
- It is woven at join points where an interaction happens
- To avoid executing interacting aspects, a proceed is not called in the around advice
- Supports fine-grained and dynamic control of aspect (advice) precedence
- By using an if pointcut
- No need to rewrite existing aspects

Conventional methods and their problem

- Declare precedence of Aspects
  - Gives precedence between aspects statically
    - Control in advice level and dynamic precedence is needed [1, 2]
    - aspect Precedence { declare precedence: A, B, C; }
  - It is dangerous to edit existing aspects
  - Take effect with wide classes
  - Precedence is not useful to resolve some interactions
    - There is no suitable order between the aspect
      - This is like a problem of the mix-in inheritance mentioned in Traits[5].
    - See the example

What is aspect interactions?

- An interaction occurs when more than two aspects are woven in the same join point
- Executed by the advice precedence rule
- The proceed call of Advice(B) is similar to the super call of the mix-in inheritance
- It could prevent the aspects to behave well

Advice precedence rule

- The advice with the highest precedence is executed first
- If it calls proceed, the advice with next precedence is executed
- A before and after advice can be considered as an around advice

•聞declare precedence: ObserverProtocol, DoubleCoordinate, ReallyChange;
  - The ReallyChanged aspect is not executed
•聞declare precedence: ObserverProtocol, ReallyChange, DoubleCoordinate;
  - When the value of dblWidth is 3.2 and setWidth(3) is called, the value of dblWidth should become 3, but the value still be 3.2. This is because shape.setWidth() != width is false and the ReallyChanged aspect prevents to call the original setter method.

Append a name to each advice

- If only an advice specified in pointcutOf can be in the list
- Advices specified in pointcutOf but out of the list is not executed

Public class Shape {
  int intWidth;
  public Shape(int w, int h) {
    intWidth = w;
  }
  public void setSize(int w, int h) {
    intWidth = w;
  }
  public int getWidth() {
    return intWidth;
  }
  public int getHeight() {
    return intHeight;
  }
  public int getArea() {
    return intWidth * intHeight;
  }
}

DoubleCoordinate

public aspect DoubleCoordinate {
  void before(Shape s) {
    intWidth *= scale;
  }
  void after(Shape s)
}

ReallyChanged

public aspect ReallyChanged {
  void around(Shape s) {
    intWidth *= scale;
  }
}

ObserverProtocol

public aspect ObserverProtocol {
  pointcut pointcutOf(ObserverProtocol onChanged)
  - observe(ObserverProtocol onChanged)
  - observe(ObserverProtocol onChanged)
}

The setWidth advice of ReallyChanged is executed when the DoubleCoordinate, setWidth, ReallyChanged, setWidth and ObserverProtocol, onChanged are executed together. It redefines behavior of the ReallyChanged aspect, and reuses advices of ObserverProtocol and DoubleCoordinate.

Resolution of interactions by aspect

- Create a new aspect to overwrite interacting aspects
- It is woven at join points where an interaction happens
- To avoid executing interacting aspects, a proceed is not called in the around advice
- Supports fine-grained and dynamic control of aspect (advice) precedence
- By using an if pointcut
- No need to rewrite existing aspects

Conventional methods and their problem

- Declare precedence of Aspects
  - Gives precedence between aspects statically
    - Control in advice level and dynamic precedence is needed [1, 2]
    - aspect Precedence { declare precedence: A, B, C; }
  - It is dangerous to edit existing aspects
  - Take effect with wide classes
  - Precedence is not useful to resolve some interactions
    - There is no suitable order between the aspect
      - This is like a problem of the mix-in inheritance mentioned in Traits[5].
    - See the example

What is aspect interactions?

- An interaction occurs when more than two aspects are woven in the same join point
- Executed by the advice precedence rule
- The proceed call of Advice(B) is similar to the super call of the mix-in inheritance
- It could prevent the aspects to behave well

Advice precedence rule

- The advice with the highest precedence is executed first
- If it calls proceed, the advice with next precedence is executed
- A before and after advice can be considered as an around advice

•聞declare precedence: ObserverProtocol, DoubleCoordinate, ReallyChange;
  - The ReallyChanged aspect is not executed
•聞declare precedence: ObserverProtocol, ReallyChange, DoubleCoordinate;
  - When the value of dblWidth is 3.2 and setWidth(3) is called, the value of dblWidth should become 3, but the value still be 3.2. This is because shape.setWidth() != width is false and the ReallyChanged aspect prevents to call the original setter method.

Append a name to each advice

- If only an advice specified in pointcutOf can be in the list
- Advices specified in pointcutOf but out of the list is not executed

Public class Shape {
  int intWidth;
  public Shape(int w, int h) {
    intWidth = w;
  }
  public void setSize(int w, int h) {
    intWidth = w;
  }
  public int getWidth() {
    return intWidth;
  }
  public int getHeight() {
    return intHeight;
  }
  public int getArea() {
    return intWidth * intHeight;
  }
}

DoubleCoordinate

public aspect DoubleCoordinate {
  void before(Shape s) {
    intWidth *= scale;
  }
  void after(Shape s)
}

ReallyChanged

public aspect ReallyChanged {
  void around(Shape s) {
    intWidth *= scale;
  }
}

ObserverProtocol

public aspect ObserverProtocol {
  pointcut pointcutOf(ObserverProtocol onChanged)
  - observe(ObserverProtocol onChanged)
  - observe(ObserverProtocol onChanged)
}

The setWidth advice of ReallyChanged is executed when the DoubleCoordinate, setWidth, ReallyChanged, setWidth and ObserverProtocol, onChanged are executed together. It redefines behavior of the ReallyChanged aspect, and reuses advices of ObserverProtocol and DoubleCoordinate.