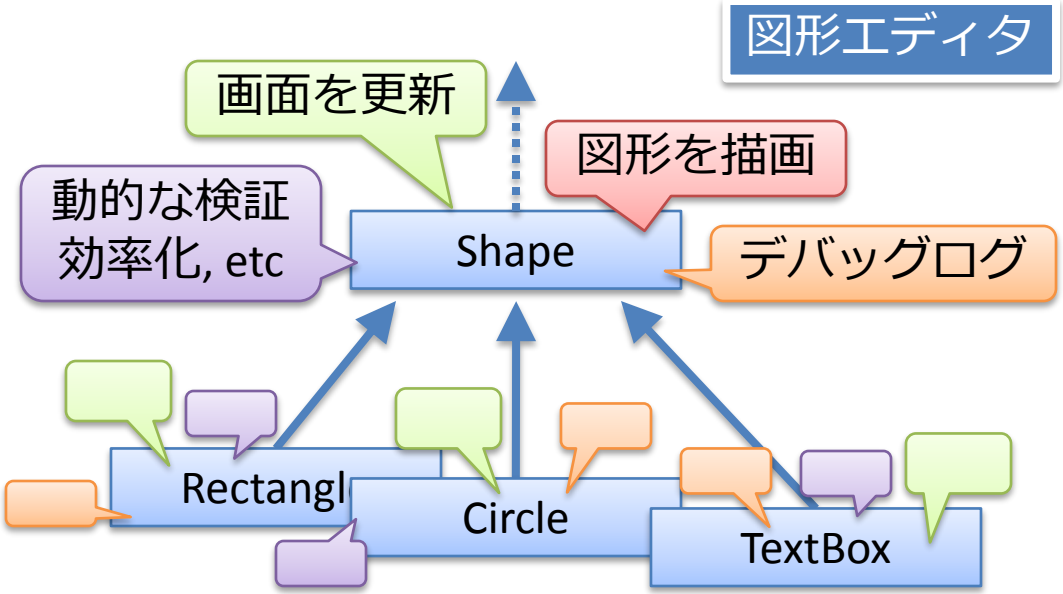
アスペクトの相互作用を解消するアスペクトの提案

武山 文信　千葉滋
東京工業大学大学院
情報理工学研究科　数理・計算科学専攻

アスペクト指向プログラミング（AOP）

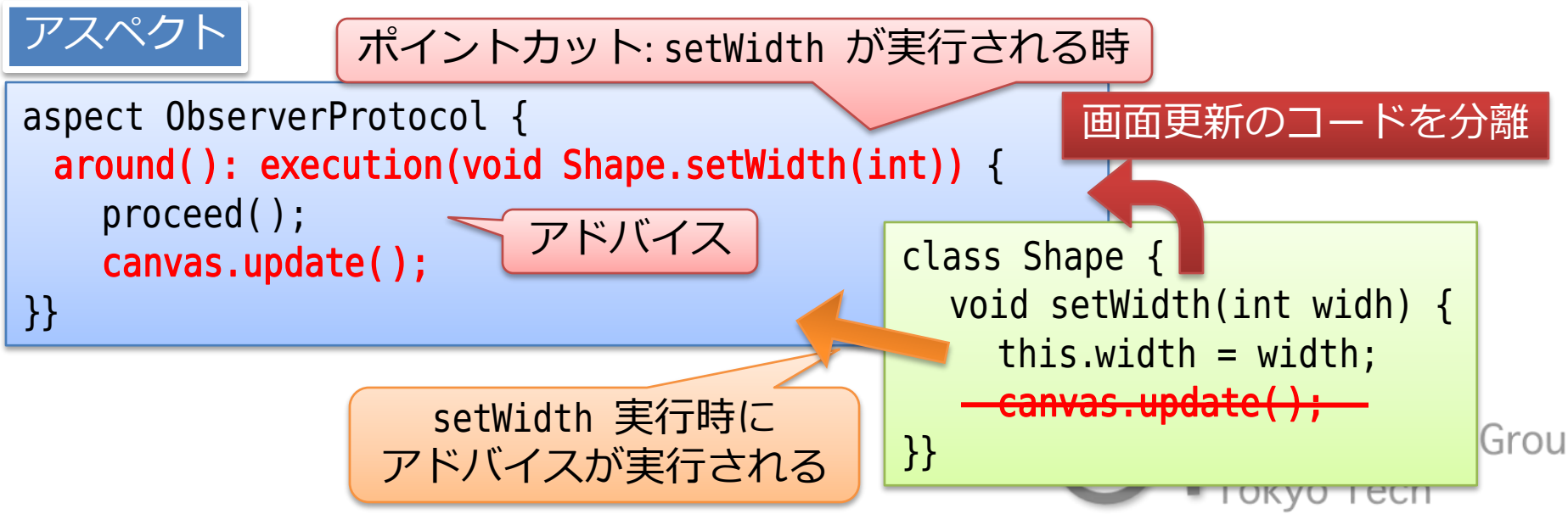
- 横断的関心事をアスペクトとしてモジュール化
  - オブジェクト指向（OOP）では上手く分離できない
  - クラス階層に、コードが散在するような機能
- AspectJ
  - Java を拡張した AOP 処理系




```
class Rectangle {
  class Shape {
    void setWidth(int w) {
      canvas.update();
    }
    void setHeight(int h) {
      canvas.update();
    }
        :
      canvas.update();
        :
  }
}
```

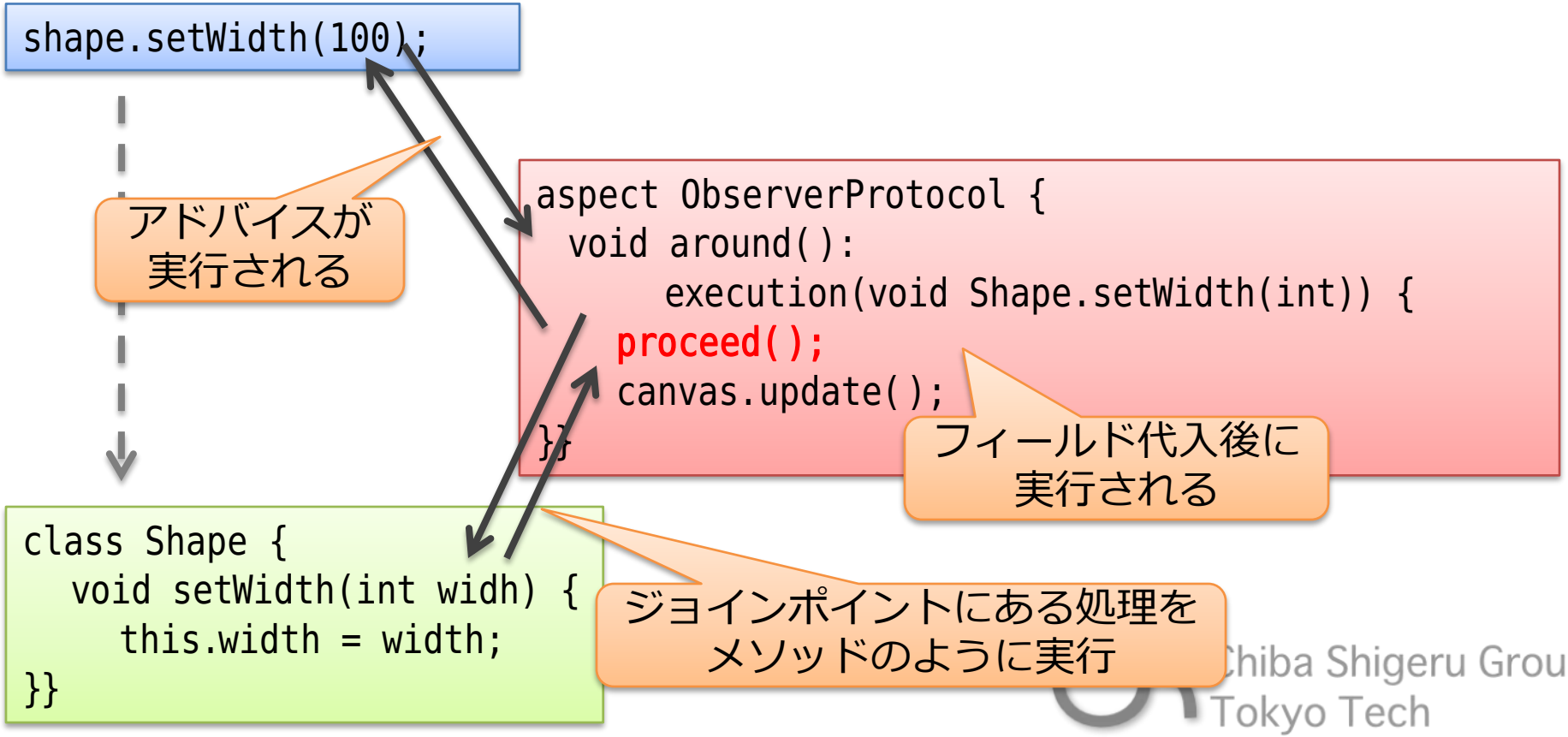
AspectJ

- 横断的関心事をアドバイスとして実装
  - アスペクトをクラスとして見たとき、メソッドに相当
- アドバイスを実行するジョインポイントを
  ポイントカットで指定
  - ジョインポイント: コード上の位置 + 動的なコンテキスト



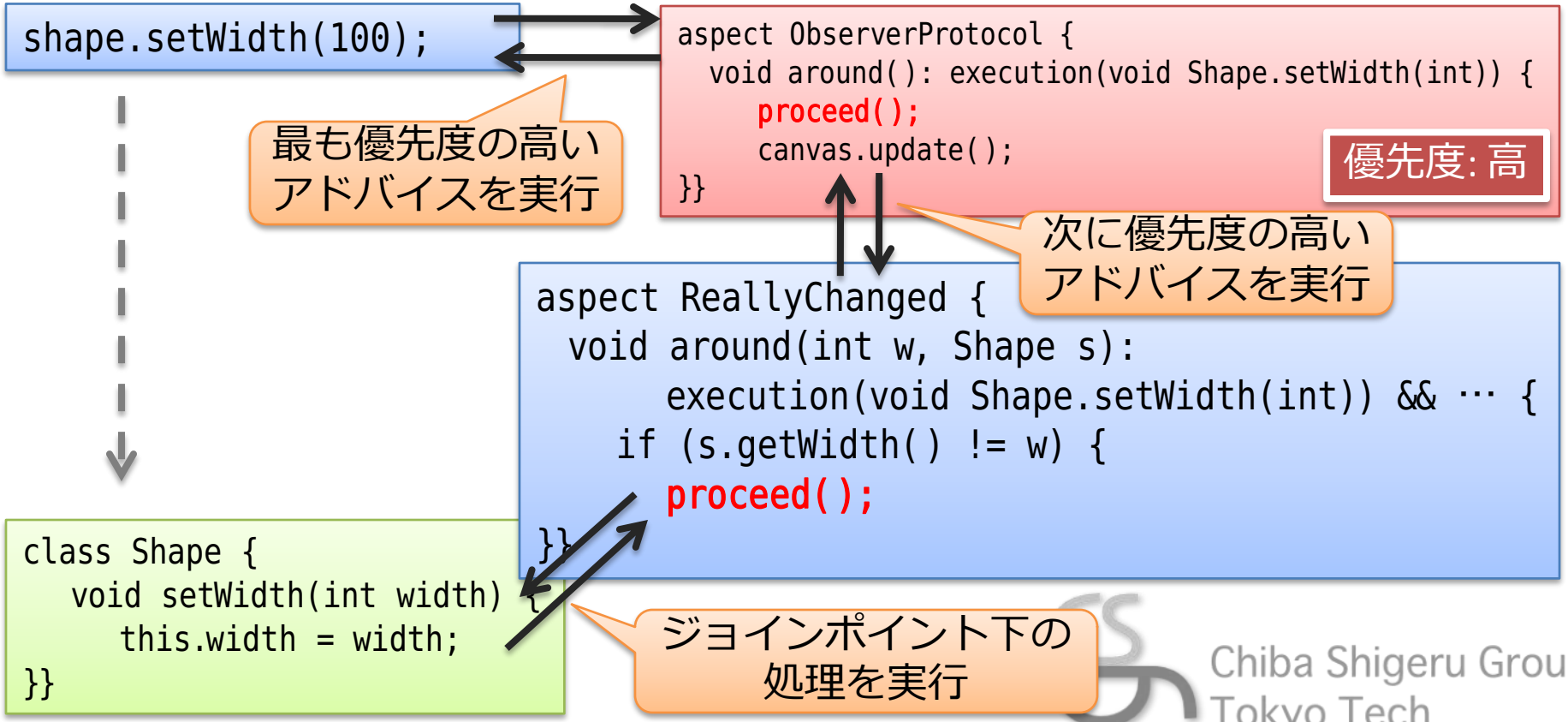
around アドバイス

- ジョインポイントにある処理を上書き(横取り)
  - アドバイス中で proceed を呼ぶと上書きした元の処理を実行
  - before アドバイスなどは特殊な around として考えられる



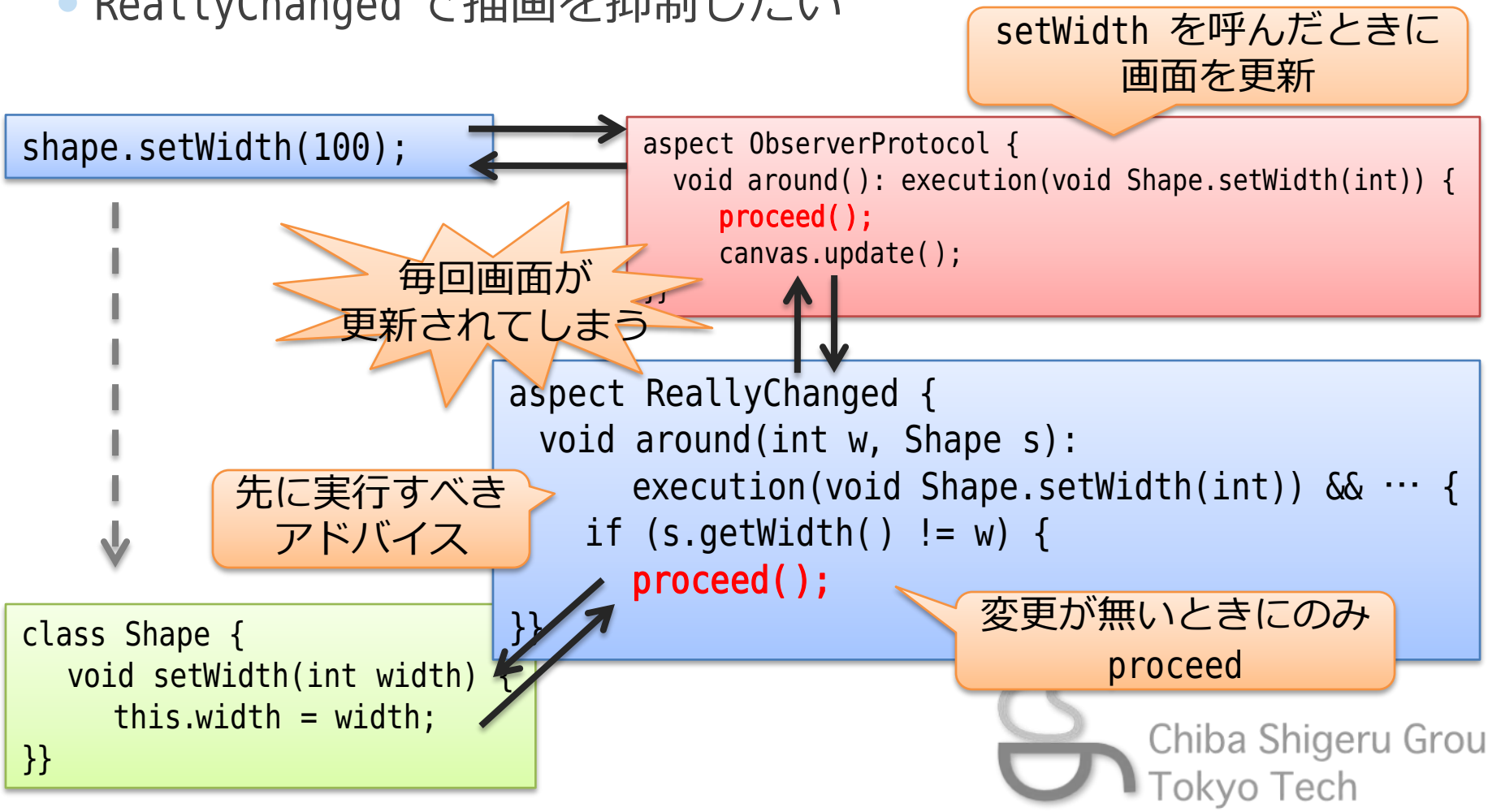
アスペクトの相互作用 1/2

- 複数のアドバイスが同一のジョインポイントに織り込まれると起きる
  - 優先度の高いアドバイスから OOP の super のように実行される



アスペクトの相互作用 2/2

- 相互作用が原因でしばしば意図した動作をしない
  - ReallyChanged で描画を抑制したい



優先度指定による解消方法の限界

- どのような順序で実行しても解消しないことがある
- 明示的な優先度の指定方法
  - declare precedence
    - アスペクト単位で静的に優先度を指定

```
aspect Precedence {
  declare precedence: DoubleCoordinate, ReallyChanged, ObserverProtocol;
}
```

優先度

  - Context-Aware Composition Rules [Antoine Marot, et al. '08]
    - アドバイス単位で動的な優先度を指定

Chiba Shigeru Group
Tokyo Tech

相互作用による不具合の例 1/4

- 図形エディタの Shape クラス
  - 幅を設定する setWidth メソッド
  - 倍率を指定して大きさを変える scale メソッド

```
public class Shape {
   int width, height;
   public void setWidth(int width) {
      this.width = width;
   }
   public int getWidth( ) {
      return width;
   }
   public void scale(double scale) {
      width = (int) Math.round(width * scale);
      height = (int) Math.round(height * scale);
   } //略
}
```

浮動小数点数を整数に丸める

相互作用による不具合の例 2/4

- DoubleCoordinate アスペクト
  - 大きさを浮動小数点で管理、scale メソッドの精度を向上

```
public aspect DoubleCoordinate {
  double Shape.dblWidth, Shape.dblHeight;
  void around(int width, Shape shape):
    execution(void Shape.setWidth(int)) && args(width) && target(shape) {
    shape.dblWidth = width;
  }
  int around(Shape shape):
    execution(int Shape.getWidth()) && target(shape) {
    return (int)Math.round(shape.dblWidth);
  }
  void around(double scale, Shape shape):
    execution(void Shape.scale(double)) && args(scale) && target(shape) {
    shape.dblWidth *= scale;
    shape.dblHeight *= scale;
}}
```

インタータイプ宣言で Shape クラスにフィールドを追加

Shape の setWidth でdblWitdhに代入

Shape の getWidth は dblWidth の値をintにして返す

Scaleメソッドはdblxxxの値を用いるように変更

相互作用による不具合の例 3/4

- ReallyChanged アスペクト
  - 実際にフィールドの値が変わらないとき setter を実行しない
  - 画面を再描画する ObserverProtocol の実行を抑制

```
public aspect ReallyChanged {
  void around(int width, Shape shape):
     execution(void Shape.setWidth(int)) && args(width) && target(shape) {
    if (shape.getWidth( ) != width) {
      proceed(width, shape);
  }}
  void around(double scale):
     execution(void Shape.scale(double)) && args(scale) {
    if (scale != 1.0)
      proceed(scale);
  }}
  //略
}
```

setterで値が変わるか？

setter や scale が呼ばれたとき

```
public aspect ObserverProtocol {
  void around( ): execution(void Shape.setWidth(int)) ||
             /* 略 */ ||
             execution(void Shape.scale(double)) {
    proceed( ); canvas.update( );
}}
```

画面の描画など

相互作用による不具合の例 3/4

- 優先順位: DoubleCoordinate → ReallyChanged → ObserverProtocol
  - ReallyChanged と ObserverProtocol が実行されない
    - DoubleCoordinate アスペクトが proceed しない

```
void around(double width, Shape shape):
   execution(void Shape.setWidth(int)) &&
   args(width) && target(shape) {

   shape.dblWidth = width;
}
```

proceed しない

Chiba Shigeru Group
Tokyo Tech

相互作用による不具合の例 4/4

- 優先順位: ReallyChanged → DoubleCoordinate → ObserverProtocol
  - dblWidth が 3.2 のときに setWidth(3) を呼ぶ → dblWidth が 3.0 にならない
  - アスペクトが意味的に依存

```
void around(int width, Shape shape):
    execution(void Shape.setWidth(int)) &&
    args(width) && target(shape) {
  if (shape.getWidth() != width) {
    proceed(width, shape);
}}
```

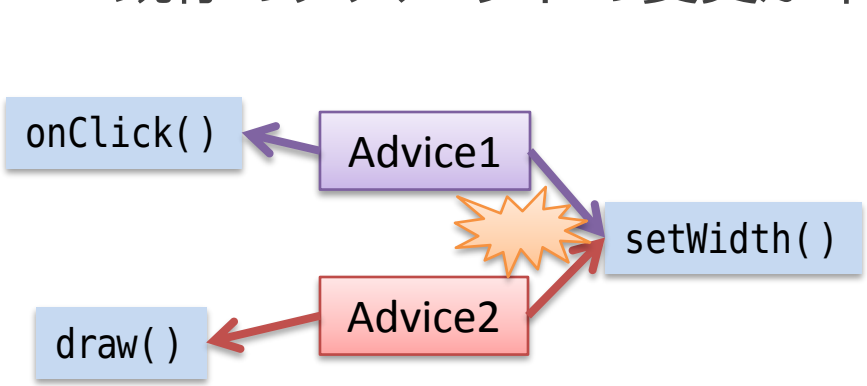
この条件式が成立しないので proceed しない

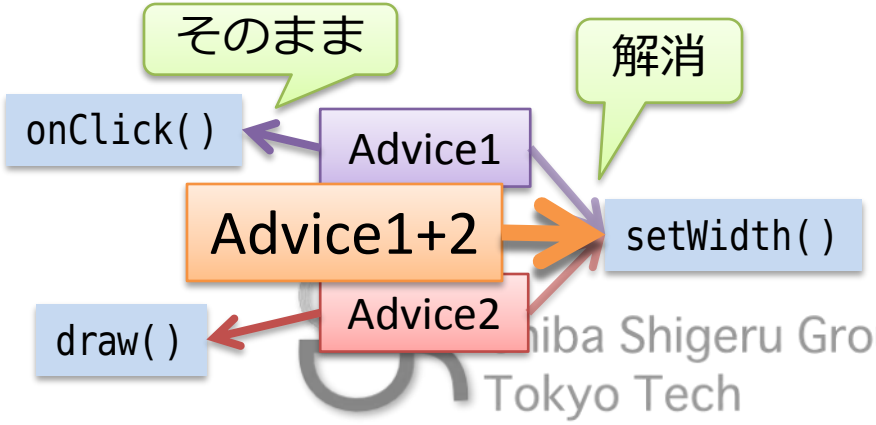
```
public aspect DoubleCoordinate {
  double Shape.dblWidth;
  int around(Shape shape):
    execution(int Shape.getWidth()) && target(shape) {
    return (int)Math.round(shape.dblWidth);
}}
```

上書きされた getWidth()

提案: アスペクトを使った相互作用の解消

- 解消のために、新しいアスペクトを定義
  - ポイントカット:
    相互作用が起きているジョインポイントだけを選択
  - アドバイス:
    相互作用を起こしている全アスペクトを融合した動作
  - 優先度:
    相互作用を起こしている全アスペクトより高い
    - それらのアスペクトの代わりに実行される
  - 既存のアスペクトの変更が不要





言語拡張の提案

- 提案方法を支援するために AspectJ を拡張
- pointcutOf ポイントカット
  - 引数のアドバイスのポイントカットを表す
  - 相互作用が起きているジョインポイントの選択を容易に
- 優先順位付き proceed
  - 残りのアドバイスの優先度を変更して proceed 呼び出し
    - 優先度を変えるだけで相互作用を解消できることも多い
  - 解消のためのアスペクトのコードが冗長になるのを防ぐ
    - 相互作用を起こしているアドバイスを再利用し、融合した動作を実装可能

Chiba Shigeru Group
Tokyo Tech

pointcutOf ポイントカット

- 引数のアドバイスのポイントカットを表す
  - pointcutOf(A) && pointcutOf(B)
    - アドバイス A と B が相互作用するジョインポイントを選択
- アドバイスの識別
  - 文法を拡張し、アドバイスに名前を付ける

```
public aspect ReallyChange {
  void around setWidth(int width, Shape shape):
      execution(void Shape.setWidth(int)) &&
      args(width) && target(shap
    if (shape.getWidth() != wid
      proceed(width, shape);
  }}
  //略
}
```

```
public aspect ObserverProtocol {
  void around onChanged():
      execution(void Shape.setWidth(int)) ||
      /* 略 */ ||
      execution(void Shape.scale(double)) {
    proceed(); canvas.update();
}}
```

優先順位付き proceed

- 既存のアスペクトを再利用
  - 優先順位を変えて proceed を呼び出す
    - 優先順位で解消できるものを再利用
  - [A, B].proceed()
  - pointcutOf で指定されたアドバイスのみ [] 内に記述可能
    - [] 内に含まれないアドバイスは実行されない
- around resolving アドバイス
  - 優先順位付き proceed を使用できる
  - 最も高い優先度が与えられる

```
void around resolve() resolving: pointcutOf(A) && pointcutOf(B)
{
  [A].proceed();
}
```

アドバイスBは
実行されない

本言語拡張を用いた解消例 1/2

- 解消のために Resolve アドバイスを定義
  - 相互作用が起きている3つのアスペクトが織り込まれるジョインポイントを選択

相互作用が発生しているジョインポイントを指定

```
public aspect Resolve {
  void around setWidth(int w, Shape s) resolving:
      pointcutOf(DoubleCoordinate.setWidth(w, s)) &&
      pointcutOf(ReallyChanged.setWidth(int, Shape)) &&
      pointcutOf(ObserverProtocol.onChanged( )) {
    if (s.dblWidth != (double)w) {
      [ObserverProocol.onChanged,
       DoubleCoordinate.setWidth].proceed(w, s);
    }
  }
  //略
}
```

本言語拡張を用いた解消例 2/2

- ReallyChanged アスペクトの処理を再定義
  - ObserverProtocol, DoubleCoordinate アスペクトを再利用

```
public aspect Resolve {
  void around setWidth(int w, Shape s) resolving:
      pointcutOf(DoubleCoordinate.setWidth(w, s)) &&
      pointcutOf(ReallyChanged.setWidth(int, Shape)) &&
      pointcutOf(ObserverProtocol.onChanged()) {
    if (s.dblWidth != (double)w) {
      [ObserverProocol.onChanged,
       DoubleCoordinate.setWidth].proceed(w, s);
    }
  }
  //略
}
```

新しい ReallyChanged アスペクトの処理

既存のアスペクトの再利用

dblWidth が 3.2 のとき、shape.setWidth(3) を呼ぶと dblWidth が 3.0 になる → 解消できた

関連研究

- Context-Aware Composition Rules [Antoine Marot, et al. ’08]
  - ジョインポイントに対してアドバイスのルールを設定
    - ジョインポイントの選択にポイントカットを使用する
    - 優先度、無効化
  - アドバイス単位で動的な優先度を指定
    - `declare precedence`: アスペクト単位で静的な優先度
  - 優先度の変更では解消できない場合がある

- JAsCo [Davy Suvee, et al]
  - ジョインポイントで織り込まれているアドバイスをリストとして操作できる
  - 本言語の使用例として示した相互作用を解消できるが、抽象度が低い

Chiba Shigeru Group
Tokyo Tech

現在の状況

- 実装
  - Aspect Bench Compiler の拡張として実装中
- 組合せ爆発の可能性
  - around resolving アドバイスの干渉
- アドバイスの指定が壊れやすい
  - pointcutOf が名前付きポイントカットに含まれる場合
    - 引数で与えられているアドバイス名が見えない

```
pointcut namedpc: pointcutOf(advice1) && pointcutOf(advice2);
```

```
void around resolve resolving: namedpc {
    [advice2, advice1].proceed();
}
```

まとめ

- アスペクトを用いた相互作用による不具合の解消方法を提案した
  - 解消のためのアスペクトで、既存のアスペクトをまとめて上書きする
  - 従来の優先順位による方法では解消できない不具合も解消できる
- この方法を支援する AspectJ 言語拡張を提案した
  - `pointcutOf` ポイントカット
    - アドバイスのポイントカットを表す
  - 優先順位付き `proceed` と `around resolving` アドバイス
    - 既存のアドバイスを再利用する

Chiba Shigeru Group
Tokyo Tech