Strategic reuse with software product lines

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Code variability mechanisms

What would you do to implement slightly different services to different products from the same family?

Conditional execution

```
canvas_midp2 = Boolean.valueOf(args[1]);
canvas_siemens = Boolean.valueOf(args[2]);
if (canvas_midp2 || canvas_siemens) {
      this.mainCanvas.paint();
      this.mainCanvas.flushGraphics();
} else {
   this.mainCanvas.repaint();
   this.mainCanvas.serviceRepaints();
}
```

Conditional compilation

```
//#if canvas_midp2 || canvas_siemens
      this.mainCanvas.paint();
      this.mainCanvas.flushGraphics();
//#else
//#   this.mainCanvas.repaint();
//#   this.mainCanvas.serviceRepaints();
//#endif
```

From Meantime Mobile Creations

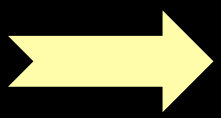
Pre-processing followed by compilation

Antenna file

```
//#if canvas_midp2 || canvas_siemens
//#   this.mainCanvas.paint();
//#   this.mainCanvas.flushGraphics();
//#else
//#   this.mainCanvas.repaint();
//#   this.mainCanvas.serviceRepaints();
//#endif
```

Tags file

```
canvas_midp2,
device_screen_128x128,
...
```



product

```
this.mainCanvas.paint();
this.mainCanvas.flushGraphics();
```

CC Core assets

Antenna file

```
//#if canvas_midp2 || canvas_siemens
//#   this.mainCanvas.paint();
//#   this.mainCanvas.flushGraphics();
//#else
//#   this.mainCanvas.repaint();
//#   this.mainCanvas.serviceRepaints();
//#endif
```

Makefiles

Tags files

`canvas_midp2,`	`canvas_nokiaui,`
`device_screen_128x128,`	`device_screen_128x128,`
`...`	`...`

Inside commands...

```
if (display.getCurrent()==this.mainCanvas)
{
//#if canvas_midp2 || canvas_siemens
//#  this.mainCanvas.paint();
//#  this.mainCanvas.flushGraphics();
//#else
//#  this.mainCanvas.repaint();
//#  this.mainCanvas.serviceRepaints();
//#endif
}
```

From Meantime Mobile Creations

and everywhere

```
//#if canvas_nokiaui
//# public class MainCanvas extends FullCanvas {
//#elif canvas_midp2 || canvas_siemens
//# public class MainCanvas extends GameCanvas {
//#else
//# public class MainCanvas extends Canvas {
//#endif
private static int keyStatesNotRepeatable;
//#if device_screen_128x128
//# public static final int SCREEN_WIDTH  = 128;
//# public static final int SCREEN_HEIGHT = 128;
//#elif device_screen_128x117
//# public static final int SCREEN_WIDTH  = 128;
...
```

Analyzing conditional compilation

- Variation point **(site)**
  - any point in the program text
- Variation
  - any program element
- Binding time
  - precompilation time, based on makefile (build time)
- Mechanism
  - decision based on tags file, specified by makefile

Parametrization via property files (PFP)

```
Properties p = new Properties();
p.load(new
    FileInputStream("device_screen.prop"));
...
    p.getProperty("SCREEN_WIDTH")
...
```

```
SCREEN_WIDTH=128
SCREEN_HEIGHT=128
```

Parametrization via property files, with reflection

```
Properties p = new Properties();
p.load(new
    FileInputStream("repositories.prop"));
    ...
    c = Class.forName(p.getProperty("ACCREP”));
    accrep = (Interface) c.newinstance();
...
```

```
ACCREP=RelationalRepository
```

PFP core assets

Java file

```
Properties p = new Properties();
p.load(new
    FileInputStream("device_screen.prop"));
...
     p.getProperty("SCREEN_WIDTH")
...
```

Makefiles

Property files

```
SCREEN_WIDTH=128
SCREEN_HEIGHT=128
```

```
SCREEN_WIDTH=128
SCREEN_HEIGHT=117
```

Properties provide data and behavior too

```
Properties p = new Properties();
p.load(new
    FileInputStream("device_screen.prop"));
...
    if (p.getProperty("SCREEN_WIDTH")==128) {
        ...
    } else {...}
...
```

Analyzing PFP

- Variation point
  - expressions in the program text, and commands too (with conditionals)
- Variation
  - constants and values, commands
- Binding time
  - execution time, based on makefile choice (build time)
- Mechanism
  - value instantiation, decision

Parametrization via parametric polymorphism (PPP)

```
public class Record<ComplaintType> {
    ...
    public void insert(ComplaintType o) {
       ...
    }
}
```

```
...
Record<HealthComplaint>
...
```

PPP core assets

Java file, generic

```
public class Record<
   ComplaintType extends CodeIdentifiable> {
   ...
   public ComplaintType search(int code){
      ...
   }
   public void insert(ComplaintType o) {
      ...
   }
}
```

Makefiles

Java files, specific

```
...
Record<HealthComplaint>
...
```

```
...
Record<SafetyComplaint>
...
```

Parametrizing data + operations

```
public class Record<
    ComplaintType extends CodeIdentifiable> {
    ...
    public ComplaintType search(int code){
       ...
    }
    public void insert(ComplaintType o) {
       ...
    }
}
```

```
interface CodeIdentifiable {
   int getCode();
}
```

Analyzing PPP

- Variation point
  - class parameters (types and methods) in the program text
- Variation
  - types (including methods)
- Binding time
  - compilation time, based on makefile choice
- Mechanism
  - type instantiation, variation and "common" code modularization

Mixing mechanisms: PPP, but instantiation with PFP and conditional

- Variation point
  - class parameters (types and methods) in the program text
- Variation
  - types (including methods)
- Binding time
  - execution time, based on makefile choice
- Mechanism
  - type and value instantiation, variation modularization, decision (conditional)

Mixing mechanisms: PPP, with conditional and input parameter

- Variation point
  - class parameters (types and methods) in the program text
- Variation
  - types (including methods)
- Binding time
  - execution time, based on user choice
- Mechanism
  - type and value instantiation, variation modularization, decision (conditional)

Same modularization mechanism, 3 decision mechanisms, 2 instantiation mechanisms:

- Makefiles
- Makefiles + property_files + conditionals
- Input + conditionals

Aspects core assets

Java files

```
class Account {
  private String number;
  public double balance;
  ...
}
```

Makefiles

Aspects

```
aspect HelloAOPWorld {
  pointcut printCall():
    call(public void *.print(String));
  after():
    printCall() && within(HelloWorld) {
    System.out.println(" AOP World!");
  }
}
```

```
aspect HelloAOPWorld {
  pointcut printCall():
    call(public void *.print(String));
  after():
    printCall() && within(HelloWorld) {
    System.out.println(" AOP World!");
  }
}
```

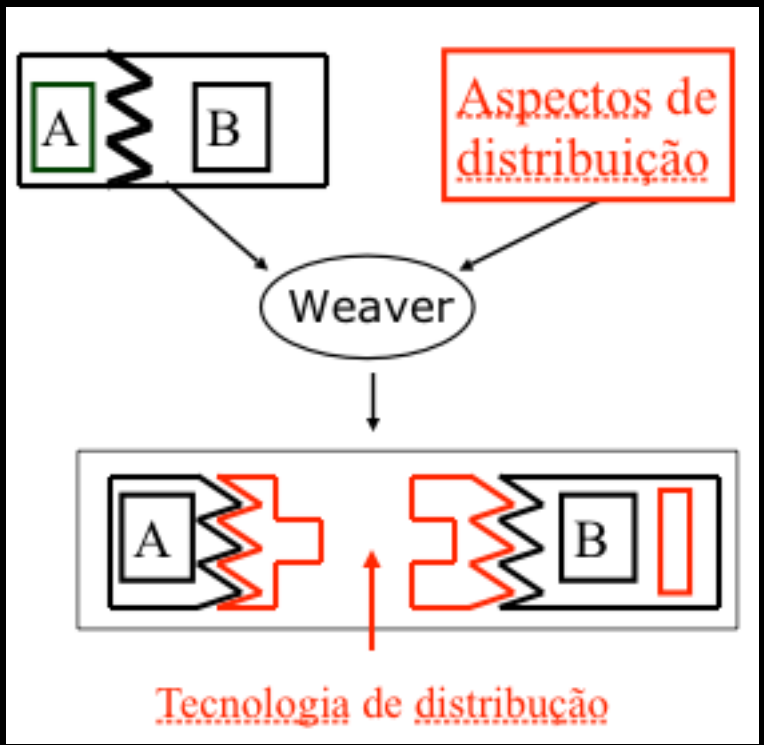
```
aspect HelloAOPWorld {
  pointcut printCall():
    call(public void *.print(String));
  after():
    printCall() && within(HelloWorld) {
    System.out.println(" AOP World!");
  }
}
```

Weaving é usado para…

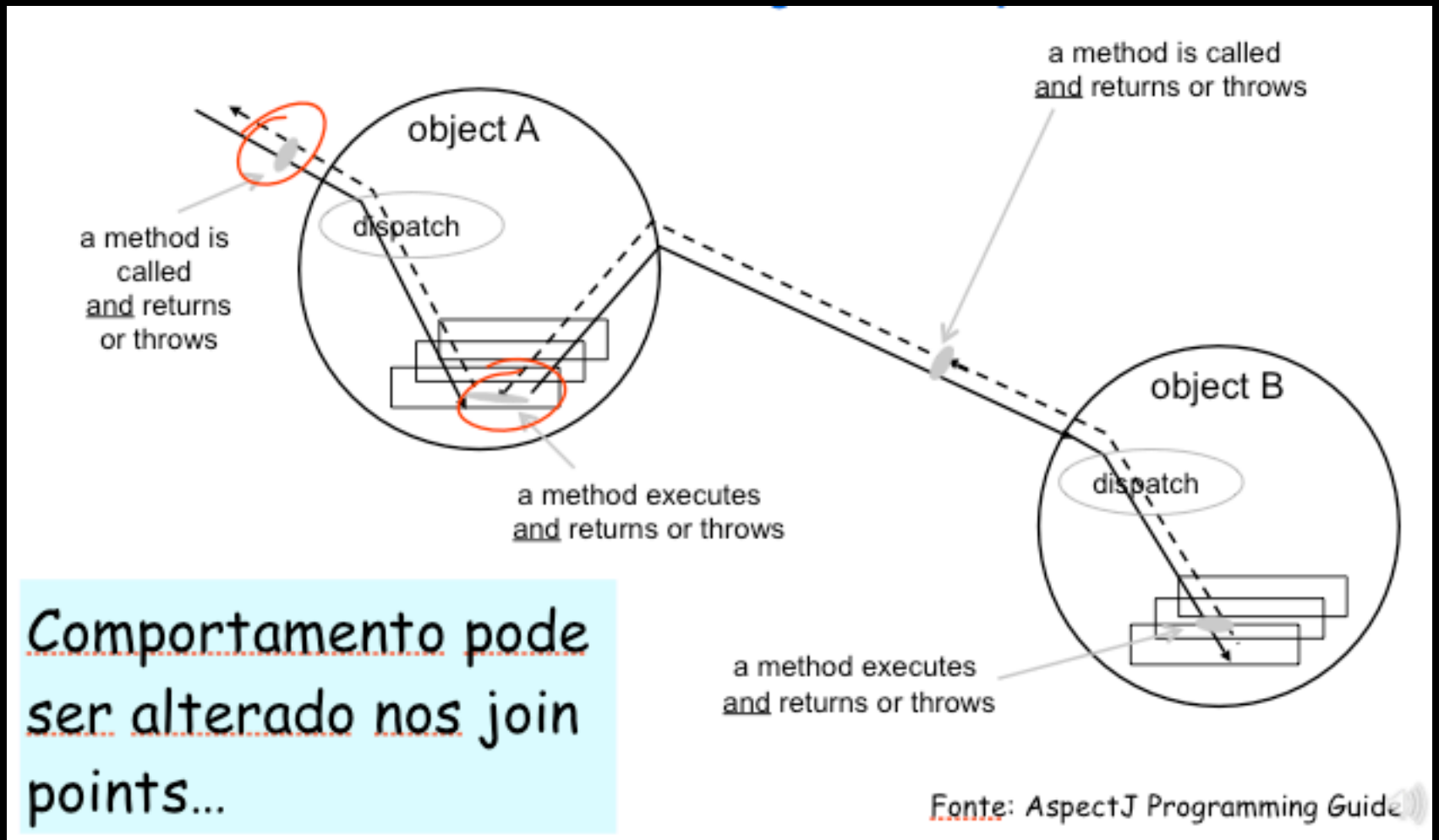
Sistema original
chamadas locais entre A e B

Processo de composição

Sistema distribuído
chamadas remotas entre A e B



Composição nos join points



Pointcuts especificam join points

- Identificam joint points de um sistema
  - chamadas e execuções de métodos (e construtores)
  - acessos a atributos
  - tratamento de exceções
  - inicialização estática e dinâmica
- Composição de joint points
  - `&&`, `||` e `!`

Identificando chamadas de métodos

```
pointcut printCall():
  call(public void *.println(String));
```

nome do pointcut

identifica chamadas de …

método println de qualquer classe

com argumento string

Advice especifica comportamento nos join points

- Define código que deve ser executado…
  - `before`
  - `after`
    - `after returning`
    - `after throwing`
  - **`ou`** `around`

  join points

Alterando o comportamento de chamadas de métodos

qualquer chamada a Write dentro de HelloWorld

após...

```
after(): printCall() && within(HelloWorld) {
  System.out.println(" AOP World");
}
```

a ação especificada será executada

Changing class hierarchy

```
declare parents: HealthWatcherFacade
     implements IFacade;

declare parents: Complaint || Person
     implements java.io.Serializable;

public static void
  HealthWatcherFacade.main(String[] args){
  try {...
    java.rmi.Naming.rebind("/HW")
  } catch  ...
}
```

Alterando a hierarquia de tipos

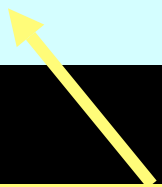
Adicionando o método main na classe fachada

Aspectos agrupam pointcuts, advices, etc.

```
aspect HelloAOPWorld {
  pointcut printCall():
    call(public void *.print(String));
  after():
    printCall() && within(HelloWorld) {
    System.out.println(" AOP World!");
  }
  ...
}
```

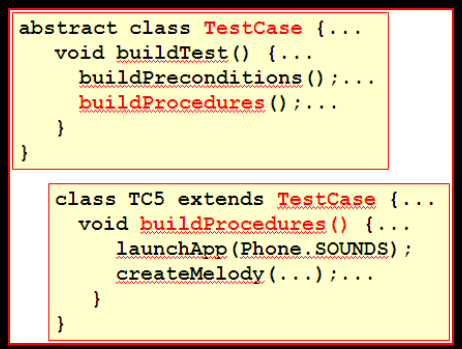
Hello AOP World!

```
public class HelloWorld {
  public static void Main(string[] args) {
    System.out.println("Hello");
  }
}
```



Chamada afetada pelo advice, caso `HelloAOPWorld` tenha sido composto com `HelloWorld`

Aspects for variations




```
privileged aspect Bluetooth {
  pointcut bluetooth(BaseTestCase test):
      execution(void TC5.buildProcedures(..))
      && this(test);
  after(BaseTestCase test):bluetooth(test){
      setBluetooth(ON);...
   }
}
```

Variations interaction, and precedence

```
privileged aspect Transflash {
  pointcut transflash(BaseTestCase test):
     execution(void TC5.buildProcedures(..))
     && this(test);
  after(BaseTestCase test):transflash(test){
     storeMultimedia(...);...
  }
}
```

```
public aspect TransflashBluetooth {
  declare precedence: Transflash,Bluetooth;
}
```

Analyzing aspects

- Variation point
  - joinpoints
- Variation
  - behavior, methods and fields, hierarchy
- Mechanism
  - variation separation, behavior modification, structure transformation

Aspects and binding time

- Compilation time, based on makefile choice
  - but can be used to implement variation 1, 2, and dynamic decision about 1 and 2
- Execution time
  - with dynamic aspect loading
  - based on user, developer, or environment choice, depending on decision mechanism

Code variability mechanisms

Strategic reuse with software product lines

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba