Departamento de Sistemas Computacionais

Desenvolvimento de Software Orientado a Aspectos: Será que vai pegar?

Sérgio Soares
sergio@dsc.upe.br
http://sergio.dsc.upe.br

UPE UNIVERSIDADE DE PERNAMBUCO

POLI ESCOLA POLITÉCNICA DE PERNAMBUCO

Programação Orientada a Objetos

- Lida com conceitos mais intuitivos
- Permite ganhos
  - Reuso
  - Manutenção
  - Adaptação
- Padrões de projetos
  - Auxiliam a POO

OO resolve nosso problema?

- Ainda não!
- Complexidade aumenta sem parar
- Limitações com objetos
  - fatores de qualidade ainda são prejudicados

Conceito novo – CONCERN (interesse)

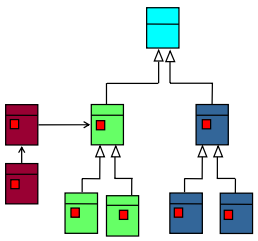
- Requisitos que devem ser implementados em um sistema
- Em qualquer sistema, vários interesses precisam ser implementados
  - sem eles implementados, seu sistema não atende aos requisitos

Tipos de interesses

- Funcionais (negócio)
  - creditar, debitar, transferir
- Não-funcionais (sistêmicos)
  - *logging*
  - tratamento de exceções
  - autenticação
  - desempenho
  - concorrência e sincronização
  - persistência...

Problema (mesmo com OO)

- Código relacionado a **certos** interesses são transversais
- Espalhados por vários módulos de implementação (classes)



Cada cor um interesse diferente

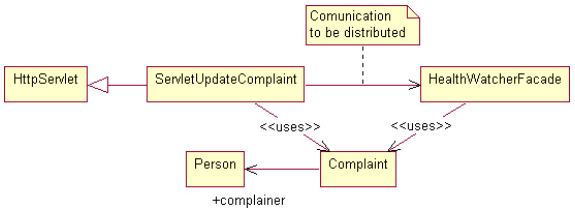
Exemplo: Sistema Disque Saúde

- Um sistema de informação
  - Registra e encaminha queixas para o sistema de saúde
  - Exibe informações sobre unidades de saúde e suas especialidades
- Requisitos não-funcionais
  - Distribuído
  - Acesso concorrente
  - Extensível
    - Armazenamento de dados
    - Tecnologia de distribuição

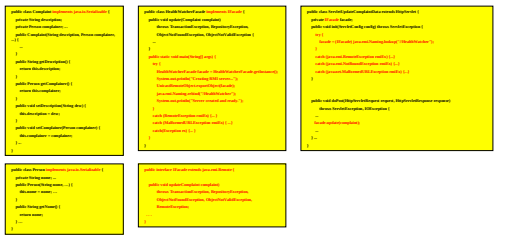
Exemplo: Sistema Disque Saúde

- Implementado em Java
  - Servlets implementam a GUI
  - Utiliza vários padrões de projetos para contemplar requisitos não-funcionais
- Como implementar a distribuição no sistema?

Disque Saúde local



Disque Saúde distribuído com RMI



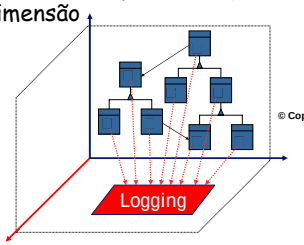
Código RMI é vermelho...

Implementação OO: problemas!

- *Tangled code* (código entrelaçado)
  - código de distribuição misturado com código de negócio e de GUI
- *Spread code* (código espalhado)
  - código de distribuição em várias classes
- Distribuição é um *crosscutting concern* (interesse transversal)
- Difícil de manter e reusar
  - mudanças no protocolo de distribuirão (RMI, CORBA, EJB) são invasivas

Causa do problema

- Implementação mapeia os requisitos em uma única dimensão

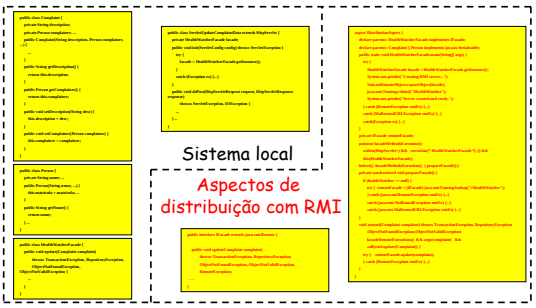


- Interesse é transversal à dimensão de implementação

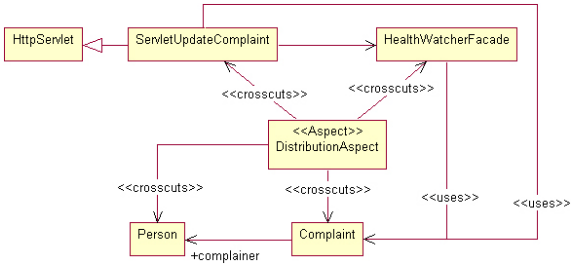
AOP — Aspect-oriented programming

- Melhora a modularidade de interesses transversais
  - distribuição, gerenciamento de dados, controle de concorrência, tratamento de exceções, logging, debugging, …
- Auxilia separação de interesses (*separation of concerns*)
  - aumenta extensibilidade e reuso

Disque Saúde distribuído com AOP (usando Java RMI)



Disque Saúde distribuído com AOP

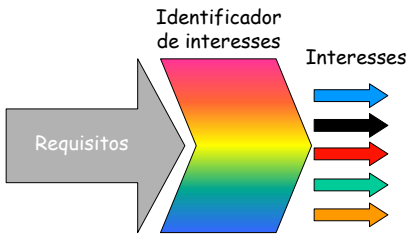


Implementação com AOP

- Aumento em modularidade, reuso e extensibidade
  - Mais unidades de código
  - Mudanças no sistema local podem causar impacto nos aspectos de distribuição
- *Separation of concerns* (separação de interesses)
  - Relação entre os aspectos e o resto do sistema nem sempre é clara
- Normalmente menos linhas de código

Em direção ao Desenvolvimento de Software Orientado a Aspectos (DSOA)

Passo 1: Identificando interesses (decomposição)



Engenharia de requisitos

Interesses do Disque Saúde

Interface com o usuário

Gerenciamento de dados

Distribuição

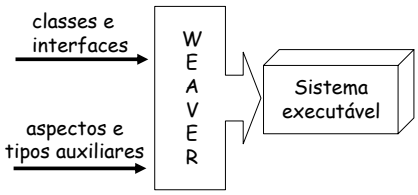
Regras de negócio

Controle de concorrência

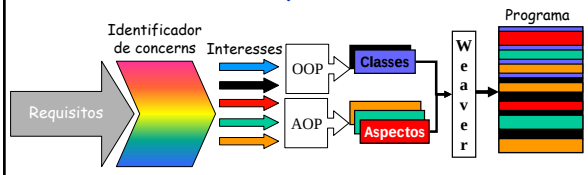
Passo 2: Implementar os interesses

- Alguns *concerns* são bem modelados como objetos (núcleo do sistema)
  - interface com o usuário
  - regras de negócio
- Outros (*crosscutting concerns*) como aspectos
  - distribuição, controle de concorrência, armazenamento de dados

Passo 3: Recompor o sistema

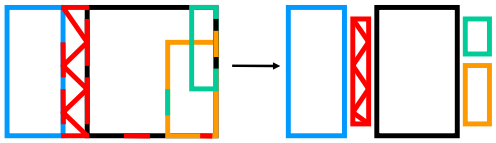


Desenvolvimento de Software Orientado a Aspectos



Requisitos funcionais
Controle de concorrência
Gerenciamento de dados
Interface com o usuário
Distribuição

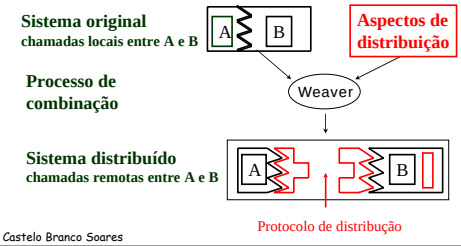
OOP vs AOP



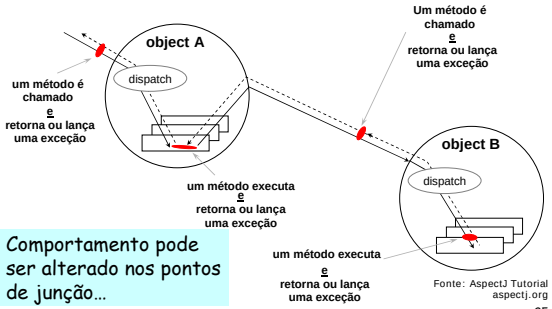
Requisitos funcionais
Controle de concorrência
Gerenciamento de dados
Interface com o usuário
Distribuição

Combinação (*weaving*) é usada para ...

- Compor o "núcleo" do sistema com os aspectos



Composição nos pontos de junção (*join points*)



Comportamento pode ser alterado nos pontos de junção…

Conjunto de pontos de junção (`pointcut`)

- Identifica pontos de junção de um sistema
  - chamadas e execuções de métodos (e construtores)
  - acessos a atributos
  - tratamento de exceções
  - inicialização estática e dinâmica
- Expõe o contexto nos *join points*
  - —argumentos de métodos, objetos alvo, atributos
- Composição de pontos de junção
  - `&&, || e !`

AspectJ: identificando chamadas de métodos da fachada (servidor)

identifica código dentro da classe ...

nome do conjunto de pontos de junção

```
pointcut facadeMethodsCall():
    within(HttpServlet+) &&
    call(* IFacade+.*(..));
```

identifica chamadas de …

qualquer método

com quaisquer argumentos

Comportamento transversal (`advice`)

- Define código adicional que deve ser executado…
  - before
  - after
    - —after returning
    - —after throwing
  - ou around

pontos de junção

AspectJ: antes (`before`) de chamar métodos da fachada

```
private IFacade remoteFacade;
before(): facadeMethodsCall() {
  getRemoteInstance();
}
synchronized void getRemoteInstance() {...
  remoteFacade =
    (IFacade) java.rmi.Naming.lookup(...);
  ...}
```


AspectJ: transformando chamadas locais em remotas

```
void around(Complaint c) throws Ex1,…:
  facadeMethodsCall() && args(c) &&
  call(void update(Complaint))
{
  try { remoteFacade.update(c);
  } catch (RemoteException rmiEx) {...}
}
```

obtendo e utilizando argumento de método em um ponto de junção

Além de mudanças (dinâmicas) com comportamento transversal…

- AspectJ suporta mudanças estáticas
  - alterar relação de subtipo
  - adicionar membros a classes

AspectJ: mudanças estáticas

```
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  ...
}
```

Adicionando o método main na classe fachada

Alterando a hierarquia de tipos

Aspecto de distribuição em AspectJ

```
public aspect DistributionAspect {
  declare parents: ...
  private IFacade remoteFacade;
  public static void
    HealthWatcherFacade.main(String[] as)...
  pointcut facadeMethodsCall(): ...
  before(): facadeMethodsCall() ...
  private synchronized void
    getRemoteInstance() ...
  void around(Complaint complaint) ...
}
```


Aspectos de desenvolvimento: *debugging* simples com AspectJ

```
public aspect DatabaseDebugging {
    pointcut queryExecution(String sql):
        call(* Statement.*(String)) &&
        args(sql);
    before(String sql): queryExecution(sql) {
        System.out.println(sql);
    }
}
```


AspectJ: pontos positivos

- Modularidade, reuso, e extensibilidade de software
- Inconsciência (o*bliviousness*)
- Suporte a desenvolvimento com IDEs
  - debugging
- Produtividade
- Permite implementação e testes progressivos

AspectJ: pontos negativos

- Novo paradigma
  - relação entre classes e aspectos deve ser minimizada
  - inconsciência (o*bliviousness*)
- Projeto da linguagem
  - tratamento de exceções
  - conflitos entre aspectos
- Requer suporte de ferramentas
- Combinação (apenas) estática

Pesquisas em DSOA

- Em todas as atividades ligadas ao desenvolvimento de software
  - requisitos, analise, projeto, implementação e testes
- Falta integrar as diversas pesquisas na área
  - teremos um "Processo de DSOA"?

AOSD na indústria

- Uso de ferramentas
  - Spring AOP
  - JBoss AOP
  - AspectWerkz
  - PostSharp (.NET)
  - Compose*
  - EOS
  - Glassbox

vide referências no final

AOSD na indústria
Rápida consulta a listas de discussão

- Instituição financeira
  - projeto médio (8 desenvolvedores)
  - logging, transaction, **violação de código fonte**
- CarnegieLearning.com
  - projeto grande (12 desenvolvedores, +1000 classes Java)
  - **violação de código fonte**

AOSD na indústria

- Uma grande empresa (Dinamarca)
  - Projeto com mais de 80 pessoas
    - —logging de desempenho, framework para GUI assíncrona e vários outros cc a implementar
  - Projeto com mais de 15 pessoas
    - —uso de Spring para implementar cc
  - Projeto com 20 pessoas
    - —usa C# e pretende introduzir AOP

AOSD na indústria

- 3 projetos no leste asiático
  - mais de 40 pessoas em cada por mais de 2 anos
  - redes domésticas, telecomunicações e desenvolvimento de produtos eletrônicos para o consumidor
  - não apenas gerenciou *crosscutting concerns*, mas separou código dependente de plataforma do código independente de plataforma
  - em dois deles, limitações de compilação obrigaram o uso de adaptações OO para simular AOP

AOSD na indústria

- Empresas conhecidas que usam AOP
  - IBM
    - —mantém AspectJ
  - Motorola
    - —WEAVR
  - Microsoft
    - —Policy Injection Application Block
  - CESAR
    - —Flip (Meantime) - SPL

Conclusão

- OO surgiu com Simula 67
  - começou a ser disseminado na indústria depois de ... 20 anos?
    - padrões de projetos
- AOP tem 10 anos
  - algum uso em grandes empresas

Será que AOP vai pegar?

Parece que vai...

cada empresa deve avaliar seus riscos
adotar AOP em projetos menores
situações menos críticas

Referências

- Spring AOP (www.springframework.org)
- JBoss AOP (labs.jboss.com/jbossaop)
- AspectWerkz (aspectwerkz.codehaus.org)
- PostSharp (www.postsharp.org)
- EOS (www.cs.iastate.edu/~eos)
- WEAVR (www.iit.edu/~concur/weavr)
  - www.jot.fm/issues/issue_2007_08/article3.pdf
- Policy Injection Application Block (msdn2.microsoft.com/en-us/library/ Bb410104.aspx)

Referências

- Gregor Kiczales et. al. Aspect-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP'97
- http://groups.yahoo.com/group/asoc-br/
- http://www.aosd.net/
- http://www.eclipse.org/aspectj

Software Productivity Group

http://spg.dsc.upe.br/

DSC/UPE e CIn/UFPE

Departamento de Sistemas Computacionais

Desenvolvimento de Software Orientado a Aspectos: Será que vai pegar?

Sérgio Soares
sergio@dsc.upe.br
http://sergio.dsc.upe.br

UNIVERSIDADE DE PERNAMBUCO

POLI ESCOLA POLITÉCNICA DE PERNAMBUCO