Departamento de Sistemas Computacionais

Desenvolvimento de Software Orientado a Aspectos

Sérgio Soares
sergio@dsc.upe.br

UNIVERSIDADE DE PERNAMBUCO

ESCOLA POLITÉCNICA DE PERNAMBUCO

Programação Orientada a Objetos

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

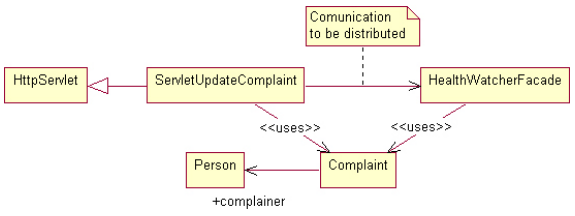
Exemplo: Sistema Disque Saúde

- 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

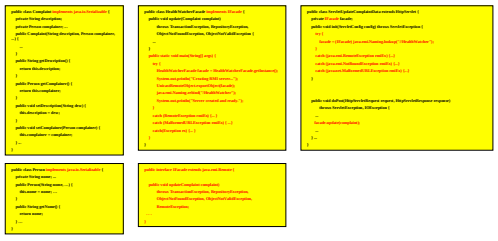
Exemplo: Sistema Disque Saúde

- 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!

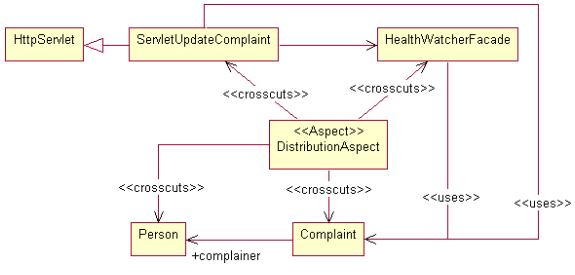
- *Tangled code* (código entrelaçado)
  - código de distribuição misturado com código de negócio e de GUI
- *Spread code* (código espalhado)
  - código de distribuição em várias classes
- Distribuição é um *crosscutting concern* (interesse transversal)
- Difícil de manter e reusar
  - mudanças no protocolo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

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

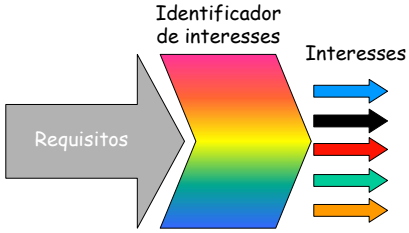


Implementação com AOP

- 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

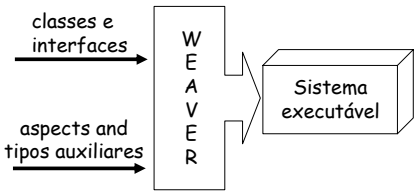
Regras de negócio

Controle de concorrência

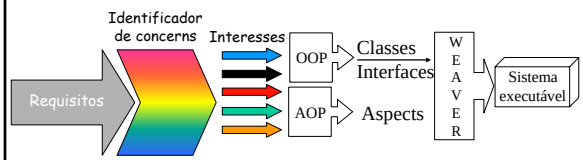
Passo 2: Implementar o sistema, separando os interesses

- 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

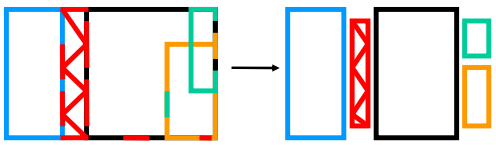


Desenvolvimento de Software Orientado a Aspectos



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

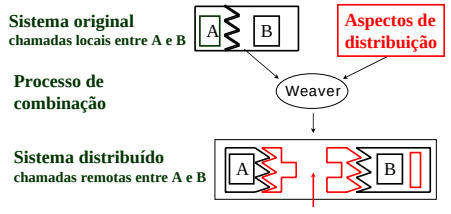
OOP vs AOP



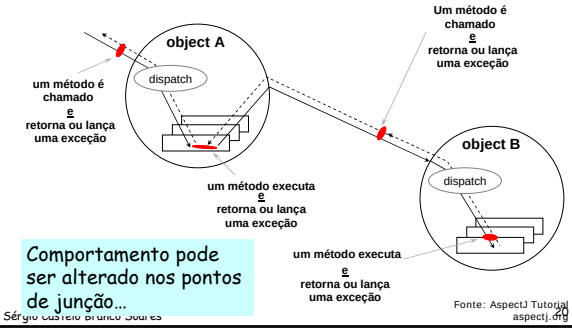
Requisitos funcionais
Controle de concorrência
Gerenciamento de dados
Interface com o usuário
Distribuição

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…

Fonte: AspectJ Tutorial aspectj.org

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

Mais construtores de AspectJ

- **target**(<nome do tipo>)
  - Identifica pontos de junção onde o objeto alvo é uma instância de <nome do tipo>
- **this**(<nome do tipo>)
  - Identifica pontos de junção cujo objeto em execução é uma instância de <nome do tipo>
- **withincode**(<assinatura de método>)
  - Identifica pontos de junção cujo código em execução pertence ao corpo do método ou construtor especificado por <assinatura de método>

Mais construtores de AspectJ

- **cflow**(<conjunto de junção>)
  - Identifica pontos de junção que estejam no fluxo de execução identificado por <conjunto de junção>
- **get**(<assinatura>) / **set**(<assinatura>)
  - Leitura/atribuição a atributos
- **declare soft**: <nome do tipo> : <conjunto de junção>;
  - A exceção <nome do tipo> será encapsulada em uma exceção não checada em tempo de compilação (runtime) em qualquer ponto de junção definido por <conjunto de junção>

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

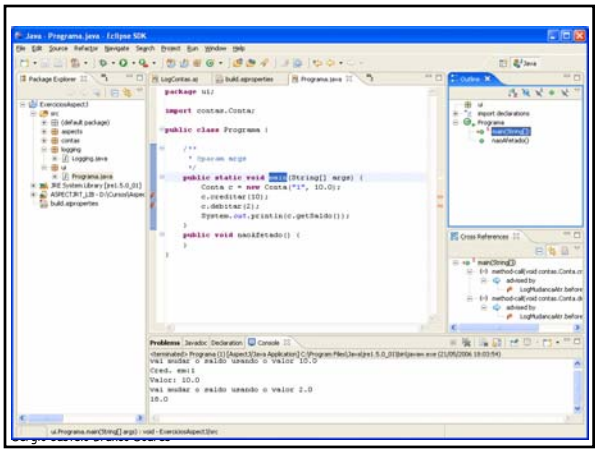
- Modularidade, reuso, e extensibilidade de software
- Inconsciência (o*bliviousness*)
- Suporte a desenvolvimento com IDEs
  - debugging
- Produtividade
- Permite implementação e testes progressivos

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

Referências

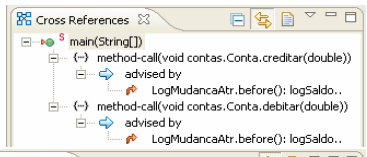
- Gregor Kiczales et. al. Aspect-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP'97
- http://groups.yahoo.com/group/asoc-br/
- http://www.aosd.net/
- http://www.eclipse.org/aspectj



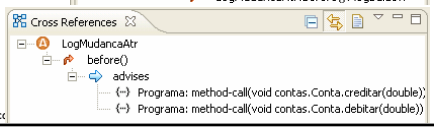
Cross References view

- Mostra relacionamentos de e para o membro selecionado
- Um clique duplo do mouse sobre uma das extremidades abre e mostra o arquivo fonte no editor

Classe

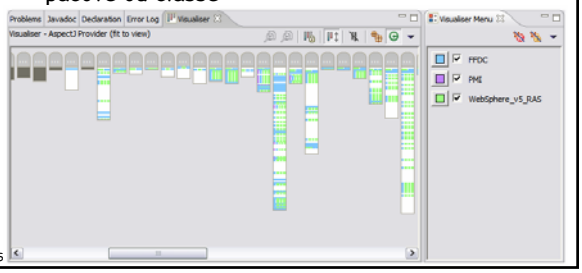


Aspecto

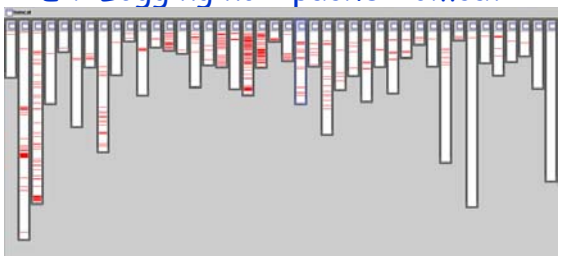


Impacto de Crosscutting

- Visualizador de aspectos: mostra o impacto de crosscutting dos aspectos de um projeto, pacote ou classe



Ex: Logging no Apache Tomcat



- Linhas vermelhas: logging
  - não estão no mesmo lugar
  - nem mesmo em poucos lugares

Software Productivity Group

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

UPE e UFPE

Sérgio Soares
sergio@dsc.upe.br