Aspect-oriented programming

Paulo Borba
Informatics Center
Federal University of Pernambuco

Advanced AspectJ concepts

Paulo Borba
Informatics Center
Federal University of Pernambuco

Banco remoto com RMI ou .NET Remoting

Distribuição

Interface

Negócio

Dados

```
public class Programa {

 [STAThread]
 public static void Main(string[] args) {
  Banco fachada;
  try {
   TcpChannel chan = new TcpChannel();
       (Banco)Activator.GetObject(typeof(Fachada.Banco),
System.Console.WriteLine("Could not locate server");
        } else {
         Programa.menu(fachada);
        }
       } catch (Exception e) {
System.Console.WriteLine("The error was: " + e.Message);
       } }

  public static void  menu(Banco fachada) {
   string numero = null;
while (opcao != 0) {
   try {
    System.Console.Out.WriteLine("Aperte <Enter> para continuar");
    Util.Util.waitEnter();
    System.Console.Out.WriteLine("\n\n\n\n\n\n\n");
System.Console.Out.WriteLine("2 - Creditar");
…}
   catch (Exception exception) {
    System.Console.Out.WriteLine(exception.Message);
   }
```

```
public class ServerInit {

 public static void Main(string[] args) {
  try {
   TcpChannel chan = new TcpChannel(8085);
   ChannelServices.RegisterChannel(chan);
   RemotingConfiguration.RegisterWellKnownServiceType
        (Type.GetType("Fachada.Banco"),
  } catch (Exception e) {
   Console.WriteLine("An error has happened:");
   Console.WriteLine(e.Message);
  }
 }
```

```
public class Banco: System.MarshalByRefObject   {

 private CadastroContas contas;

 private Banco() {
  contas = new CadastroConta
  (new RepositorioContasAccess());
 }

public void Cadastrar(Conta conta) {
contas.Cadastrar(conta);
}



 public void Transferir(string numeroDe, string n
umeroPara, double valor) {
contas.Transferir(numeroDe, numeroPara, valor);
}

}
```

```
public class CadastroContas {

 private RepositorioContas contas;

 public void Debitar(string numero, double valor) {
  Conta c = contas.Procurar(numero);
  c.Debitar(valor);
  contas.Atualizar(c);
 }

 public void Transferir(string numeroDe, string n
umeroPara, double valor) {
  Conta de = contas.Procurar(numeroDe);
  Conta para = contas.Procurar(numeroPara);
  de.Debitar(valor);
  para.Creditar(valor);
  contas.Atualizar(de);
  contas.Atualizar(para);
 }

 public double Saldo(string numero) {
  Conta c = contas.Procurar(numero);
  return c.Saldo;
 }
}
}
```

```
public class RepositorioContasArray : RepositorioContas {

 private Conta[] contas;
 private int indice;

 public RepositorioContasArray() {
  contas = new Conta[100];
  indice = 0;
 }

 public void Inserir(Conta conta) {
  contas[indice] = conta;
  indice = indice + 1;
 }

 public void Atualizar(Conta conta) {
  int i = GetIndice(conta.Numero);
  if (i == indice) {
   throw new ContaNaoEncontradaException(conta.Numero);
  } else {
   contas[i].Atualizar(conta);
  }
 }
```

```
[System.Serializable()]public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
  this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
  if (valor > saldo) {
   throw new SaldoInsuficienteException));
  }
  else {
   this.saldo = this.saldo - valor;
  }
 }
 public void Atualizar(Conta c)
 {
  this.numero = c.numero;
  this.saldo = c.saldo;
 }
}
```

```
[System.Serializable()]
 public class ContaJaCadastradaException :
 System.Exception {
  public string Numero {
   get {   return numero;
  } }
  private string numero;
  public ContaJaCadastradaException(numero;
  }
  public ContaJaCadastradaExceptionnumero =
     info.GetString("numero");
  }
  public override void GetObjectData(base.
GetObjectData(info,context);
   info.AddValue("numero", numero,typeo
f(string));
  }}
```

Distribuição na interface com o usuário

```
public static void Main(string[] args) {
   Banco fachada;
   try {...
      fachada = (Banco) Activator.GetObject(
         typeof(Fachada.Banco),
         "tcp://localhost:8085/BancoRemoto");
      ...
      Programa.menu(fachada);
   } catch...
}
```

serviço de lookup

Distribuição na fachada

```
public class Banco:
   System.MarshalByRefObject {
      private CadastroContas contas;
      private static Banco banco;
```

Manipulação de objetos por valor versus por referência

Supertipo de qualquer servidor

Distribuição na coleção de negócio

```
public class CadastroContas {
   private RepositorioContas contas;
   public void Creditar(string numero,
                        double valor) {
      Conta c = contas.Procurar(numero);
      c.Creditar(valor);
      contas.Atualizar(c);
   }...
```

Sincronização de estados entre objetos remotos e locais

Persistência na coleção de negócio

```
public class CadastroContas {
  private RepositorioContas contas;
  public void Creditar(string numero,
                       double valor) {
    Conta c = contas.Procurar(numero);
    c.Creditar(valor);
    contas.Atualizar(c);
  }...
```

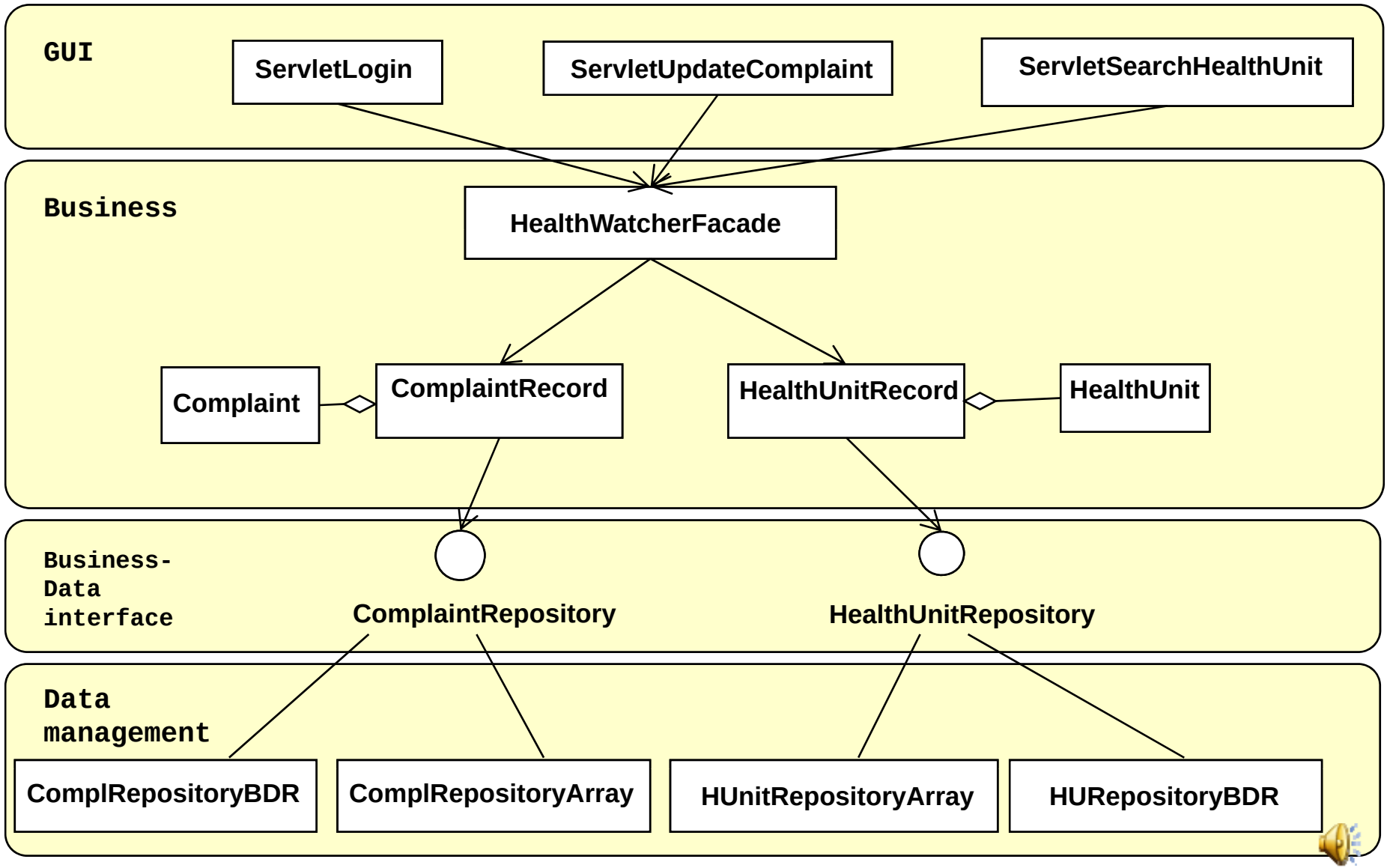
Interferência entre aspectos! deve ser identificado durante o projeto

Distribuição na classe básica e exceções

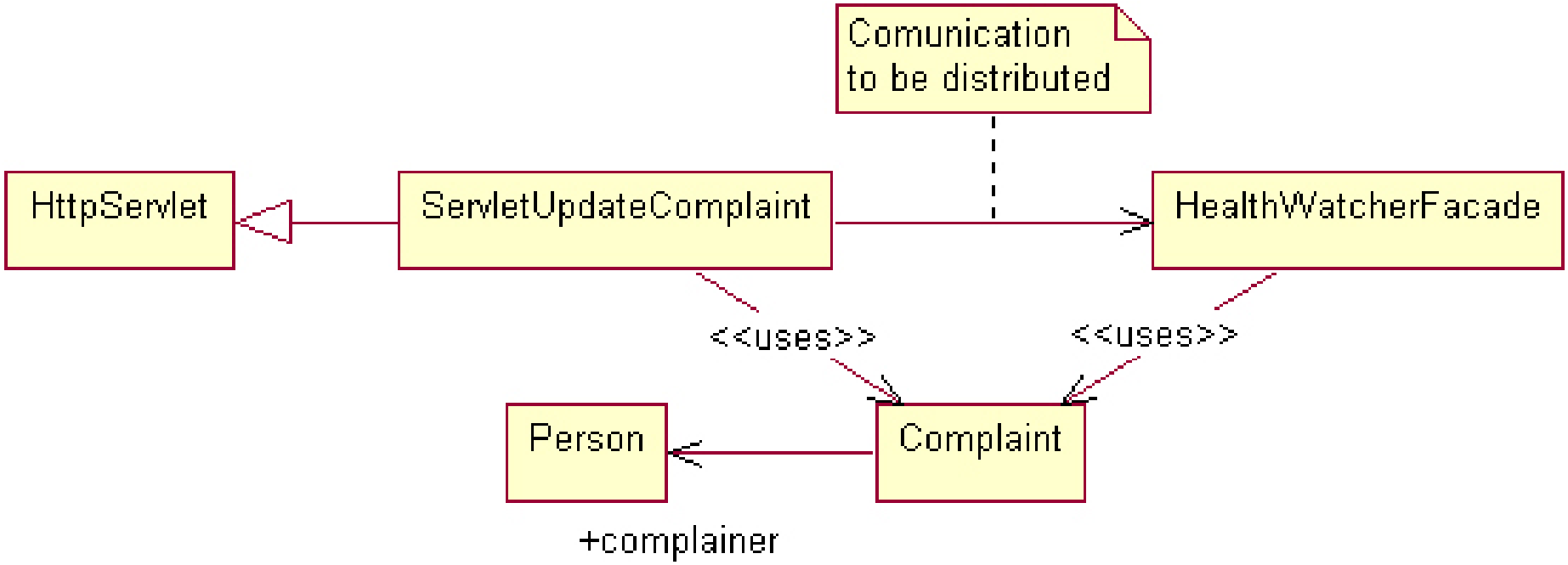
```
[System.Serializable()]
      public class Conta {
            private double saldo;...}
[System.Serializable()]
      public class ContaNaoEncontradaException
          : System.Exception  {...}
```

Possibilita a passagem de objetos por valor

Health Watcher



Local Health Watcher



Disque Saúde distribuído com RMI

```
public class Complaint implements
java.io.Serializable {
    private String description;
    private Person complainer; ...
    public Complaint(String description,
Person complainer, ...) {
        ...
    }
    public String getDescription() {
        return this.description;
    }
    public Person getComplainer() {
        return this.complainer;
    }
    public void setDescription(String
desc) {
        this.description = desc;
    }
    public void setComplainer(Person
complainer) {
        this.complainer = complainer;
    } ...
}
```

```
public class Person implements
java.io.Serializable {
    private String nome; ...
    public Person(String nome, …) {
        this.nome = nome; …
    }
    public String getNome() {
        return nome;
    } …
}
```

```
public class HealthWatcherFacade implements IFacade {
    public void update(Complaint complaint)
            throws TransactionException,
RepositoryException,
            ObjectNotFoundException,
ObjectNotValidException {
        ...
    }
    public static void main(String[] args) {
        try {
            HealthWatcherFacade facade =
HealthWatcherFacade.getInstance();
            System.out.println("Creating RMI
server...");
            UnicastRemoteObject.exportObject(facade);
            java.rmi.Naming.rebind("/HealthWatcher");
            System.out.println("Server created and
ready.");
        }
        catch (RemoteException rmiEx) {... }
        catch (MalformedURLException rmiEx) { ...}
        catch(Exception ex) {... }
    }
}
```

```
public interface IFacade extends java.rmi.Remote {

    public void updateComplaint complaint)
            throws TransactionException,
RepositoryException,
            ObjectNotFoundException,
ObjectNotValidException,
            RemoteException;
  . . .
}
```

```
public class ServletUpdateComplaintData extends HttpServlet {
    private IFacade facade;
    public void init(ServletConfig config) throws ServletException {
        try {
            facade = (IFacade) java.rmi.Naming.lookup("//HealthWatcher");
        }
        catch (java.rmi.RemoteException rmiEx) {...}
        catch (java.rmi.NotBoundException rmiEx) {...}
        catch (java.net.MalformedURLException rmiEx) {...}
    }


    public void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        ...
       facade.update(complaint);
        ...
    } ...
}
```

Código RMI é vermelho…

Disque Saúde com AOP

```
public class Complaint {
    private String description;
    private Person complainer; ...
    public Complaint(String description,
Person complainer, ...) {
        ...
    }
    public String getDescription() {
        return this.description;
    }
    public Person getComplainer() {
        return this.complainer;
    }
    public void setDescription(String desc)
{
        this.description = desc;
    }
    public void setComplainer(Person
complainer) {
        this.complainer = complainer;
    }
```

```
public class Person {
    private String nome; ...
    public Person(String nome, ...) {
        this.matricula = matricula; ...
    }
    public String getNome() {
        return nome;
    } ...
}
```

```
public class HealthWatcherFacade {
    public void update(Complaint complaint)
            throws TransactionException,
RepositoryException,
            ObjectNotFoundException,
ObjectNotValidException {
        ...
    }
}
```

```
public class ServletUpdateComplaintData extends
HttpServlet {
    private HealthWatcherFacade facade;
    public void init(ServletConfig config) throws
ServletException {
        try {
            facade = HealthWatcherFacade.getInstance();
        }
        catch (Exception ex) {...}
    }
    public void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        ...
    } ...
}
```

Sistema local

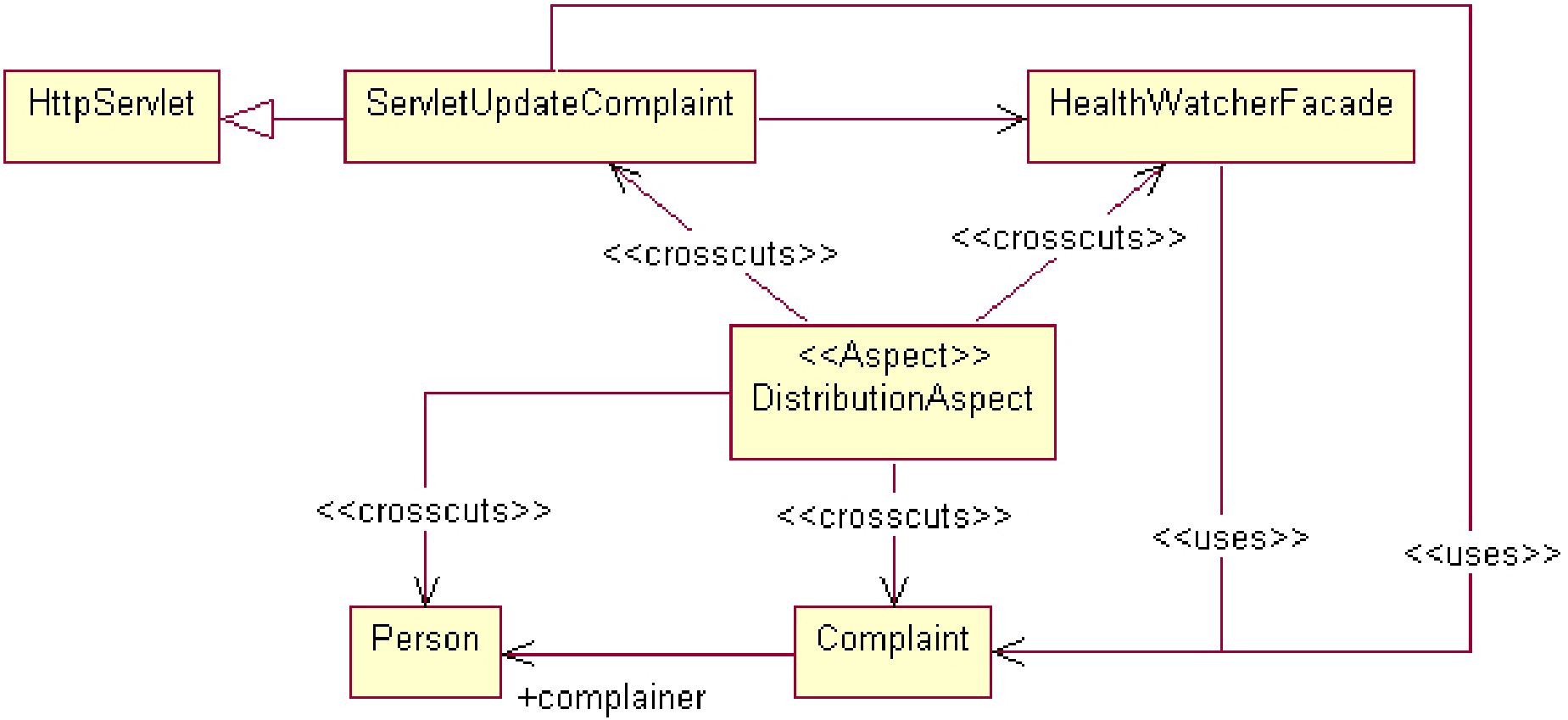
Aspectos de Distribuição para RMI

```
public interface IFacade extends java.rmi.Remote {

    public void updateComplaint complaint)
            throws TransactionException,
RepositoryException,
            ObjectNotFoundException,
ObjectNotValidException,
            RemoteException;
  . . .
}
```

```
aspect DistributionAspect {
    declare parents: HealthWatcherFacade implements IFacade;
    declare parents: Complaint || Person implements
java.io.Serializable;
    public static void HealthWatcherFacade.main(String[] args) {
        try {
            HealthWatcherFacade facade =
HealthWatcherFacade.getInstance();
            System.out.println("Creating RMI server...");
            UnicastRemoteObject.exportObject(facade);
            java.rmi.Naming.rebind("/HealthWatcher");
            System.out.println("Server created and ready.");
        } catch (RemoteException rmiEx) {...}
        catch (MalformedURLException rmiEx) {...}
        catch(Exception ex) {...}
    }
    private IFacade remoteFacade;
    pointcut facadeMethodsExecution():
        within(HttpServlet+) &&   execution(*
HealthWatcherFacade.*(..)) &&
        this(HealthWatcherFacade);
    before(): facadeMethodsExecution()  { prepareFacade();}
    private synchronized void prepareFacade() {
        if (healthWatcher == null) {
            try {  remoteFacade = (IFacade)
java.rmi.Naming.lookup("//HealthWatcher");
            } catch (java.rmi.RemoteException rmiEx) {...}
            catch (java.rmi.NotBoundException rmiEx) {...}
            catch (java.net.MalformedURLException rmiEx) {...}
    }
    void around(Complaint complaint) throws
TransactionException, RepositoryException
            ObjectNotFoundException,ObjectNotValidException:
            facadeRemoteExecutions()  && args(complaint)   &&
            call(void update(Complaint)) {
        try {    remoteFacade.update(complaint);
        } catch (RemoteException rmiEx) {...}
    }
}
```

Disque Saúde com AOP



Identificando chamadas de métodos da fachada (servidor)...

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

identifica código dentro da classe, escopo estático

inicializando fachada remota...

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

Locais ao aspecto

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) {
      ...
   }
}
```

usando argumento de um *join point*

e alterando estrutura das classes

```
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 método na fachada

Distribution aspect

```
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) ...
}
```

Aspecto de distribuição, servidor

```
public aspect DistributionAspectServer {
   declare parents:
      Banco:System.MarshalByRefObject;
}
```

Na prática, temos dois aspectos apenas por questões de implantação

Aspecto de distribuição, cliente

Acrescenta atributos a uma classe

```
aspect DistributionAspectClient {
  declare class attributes:
       (Conta || SaldoInsuficienteException)
       [System.Serializable()];
  ...
```

Aspecto de distribuição

```
void around():
  execution(...void Programa.Main(string[])) {
   Banco fachada;
   try {...
    fachada = (Banco) Activator.GetObject(
       typeof(Fachada.Banco),
       "tcp://localhost:8085/BancoRemoto");
    ...Programa.menu(fachada);
   } catch...
  }
```

O aspecto de sincronização de estado é...

- Útil tanto para persistência quanto para distribuição
- Dividido em duas partes:
  - registro de objetos modificados (sujos)
  - atualização dos objetos modificados

Registro de objetos modificados...

```
pointcut localUpdate(Conta c):
   this(CadContas) && target(c) &&
   (call(* Conta.creditar(..)) || ...;
private ArrayList dirtyObjects = ...;
after(Conta c) returning(): localUpdate(c) {
   dirtyObjects.add(c);
}
```

e atualização dos objetos modificados

Todo join point onde condição é true

```
pointcut localExecution(CadContas cad):
    if(hasDirtyObjects()) && this(cad) &&
    execution(public * *(..));
after(CadContas cad) returning():
    localExecution(cad) {
      foreach (Conta c : dirtyObjects) {
        cad.contas.atualizar(c);
      } ...
}
```

Instâncias de aspectos

- default, apenas uma instância, aspecto estático
- **`percflow,`** uma instância para cada fluxo de um joint point
- **`pertarget,`** uma instância para cada alvo de um joint point
- **`pertthis, percflowbelow`**
- **`pertypewithin,`** uma instância para cada tipo de um type pattern

Aspecto de sincronização de estado

Para evitar interferências no atributo do aspecto

```
privileged aspect StateSynchronization
  percflow(execution(* Banco.*(..))) {
    RepositorioContas CadastroContas.Contas {
      get {
        return this.contas;
      }
    }
```

publics omitidos...

Acesso às instâncias do aspecto

- **`NomeAspecto.aspectOf()`**
  - retorna a instância de um determinado aspecto, útil para acesso a membros do aspecto
  - dependendo da per clause, só disponível dentro de alguns join points
- **`NomeAspecto.aspectOf(Object)`**

Acessando mais informações dos join points

Além das informações expostas pelos pointcuts, é possível acessar mais sobre os join points usando referências especiais:

- `thisJoinPoint`
- `thisJoinPointStaticPart`
- `...`

Métodos das referências especiais

thisJoinPoint

- **getArgs()**
- **getTarget()**
- **getReturnValue()**

thisJoinPointStaticPart

- **getSignature()**

Debugging simples...

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

e um pouco mais refinado

```
public aspect LoggingAspect {
  before(): execution(public * *.*(..)) {
    System.out.println("A method from..." +
      thisJoinPoint.getTarget() + " is about" +
      “ to be executed. Its signature is " +
      thisJoinPointStaticPart.getSignature());
  }
```

Aspecto de desenvolvimento versus produção

```
after(): execution(public * *.*(..)) {
  System.out.print("Now the method" +
    ... + " has finished"); ...
  System.out.print("The return value was “
    + thisJoinPoint.getReturnValue());
}
```

Reuso e extensibilidade de aspectos via padrões e frameworks

- Tag interface
  - como na interface **Trans** do aspecto de transações
- Glue aspects
  - advices invocando serviços auxiliares
- Template pointcut
  - com aspectos abstratos

Aspecto abstrato para persistência

```
public abstract aspect APersistenceAspect {
  abstract pointcut transMethods();
  abstract PM getPM();
  before(): transMethods() {
    getPM().startTransaction();
  }  ...
}
```

Aspecto concreto para persistência

```
public aspect PersistenceAspect extends
  APersistenceAspect {
  private interface Trans {
    public void Cadastrar(Conta conta);...
  }
  declare parents: Banco implements Trans;
  pointcut transMethods():
    execution(public * Trans.*(..));
  PM getPM(){...}
}
```

Relembrando pointcut designators

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

Outros pointcut designators

- **`initialization(<constructor>)`**
  - todos os join points de inicialização com uma dada assinatura
- **`args(<Type or Id, ...>)`**
  - todos os join points com argumentos dos tipos especificados
- **`handler(TypePattern)`**
  - every exception handler for any throwable type in *TypePattern*

Mais pointcut designators

- **cflow**(<pointcut>)
  - Identifica *join points* que estejam no fluxo de execução identificado por <pointcut>
- **cflowbelow**(<pointcut>)
  - Identifica todos os *join points* no fluxo de controle do <pointcut>, exceto o inicial
- **get**(<assinatura>) / **set**(<assinatura>)
  - Leitura/atribuição a atributos

Mais intertypes

- **`declare soft`**: <nome do tipo>:<pointcut>;
  - A exceção <nome do tipo> será encapsulada em uma exceção não checada em tempo de compilação (runtime) em qualquer *join point* definido por <pointcut>
- **`declare precedence`**: <typePatternList>;
  - at any join point where multiple pieces of advice apply, the advice precedence at that join point is in typePatternList order.

Verifying design rules

```
aspect ArchitecturalRules {
  pointcut complaintRecord():
    within(ComplaintRecord);
  pointcut complaintRepositories():
    within(ComplaintRepository+);
  declare warning:
    call(* ComplaintRepository+.*(..)) &&
    !(complaintRepositories()) &&
    !(complaintRecord()):
    “Breaking design rule!”;
```

Orientação a aspectos é...

- Quantificação (quantification)
  - uma parte do programa tem efeito em várias outras
  - o aspecto afeta várias classes e outros aspectos
- Mudanças não invasivas (obliviouness)
  - uma parte A do programa tem efeito sobre uma B sem precisar alterar o código de B

Advanced AspectJ concepts

Paulo Borba
Informatics Center
Federal University of Pernambuco