Instituto Nacional de Ciência e Tecnologia para Engenharia de Software

Rede de Laboratórios de Produtividade de Software

Introdução a Programação Orientada a Aspectos

Programa de Capacitação em Testes de Software

Design modular possibilita

- Desenvolvimento independente
- Evolução independente
- Entendimento independente

Programação OO

Com o uso de princípios, padrões e ferramentas, possibilita a modularização de algumas preocupações:

- GUI
- Interações entre camadas
- Regras de negócio
- Algoritmos

Limitações da POO

A decomposição predominantemente OO resulta em espalhamento e entrelaçamento de código associado a:

- Distribuição
- Persistência
- Sincronização
- Features opcionais, alternativas, ... em LPS
- ...

Persistência com um design OO

```
public class Banco {

 private CadastroContas contas;

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

public void Cadastrar(Conta conta) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Cadastrar(conta);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Transferir(numeroDe, numeroPara, valor);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

}
```

```
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 RepositorioContasAccess : RepositorioContas {

 public void Inserir(Conta conta) {
  string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
  OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
  insertCommand.ExecuteNonQuery();
 }

 public void Atualizar(Conta conta) {
  string sql = "UPDATE Conta SET SALDO = (" + conta.Sa
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
  OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
  int linhasAfetadas;
  linhasAfetadas = updateCommand.ExecuteNonQuery();
  if (linhasAfetadas == 0) {
   throw new ContaNaoEncontradaException(conta.Numero);
  }
 }

}
```

```
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;
 }
}
```

```
public class DBHandler {
 private static OleDbConnection connection;
 public static OleDbConnection Connection {
  get {
   if (connection == null) {
    string dataSource = "BancoCS.mdb";
    string strConexao =
     "Provider= Microsoft.Jet.OLEDB.4.0; " +
     "Data Source=" + dataSource;
    connection = new OleDbConnection(strConexao);
   }
   return connection;
  }
 }
 public static OleDbConnection GetOpenConnection() {
  Connection.Open();
  return Connection;
 }

 public static void StartTransaction() {
  Connection.Open();
  transaction = Connection.BeginTransaction();
 }

}
```

ines

Persistência com um design AO

```
public class Banco {

  private CadastroContas contas;

  private Banco() {
    contas = new CadastroConta
  }

  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);
  }

   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);
  }

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

```
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;
  }
 }
```

Elementos do domínio

```
public class RepositorioContasAccess : RepositorioContas {

  public void Inserir(Conta conta) {
   string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
   OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
   insertCommand.ExecuteNonQuery();
  }

  public void Atualizar(Conta conta) {
   string sql = "UPDATE Conta SET SALDO = (" + conta.Sa
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
   OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
   int linhasAfetadas;
   linhasAfetadas = updateCommand.ExecuteNonQuery();
   if (linhasAfetadas == 0) {
    throw new ContaNaoEncontradaException(conta.Numero);
   }
  }

 }
```

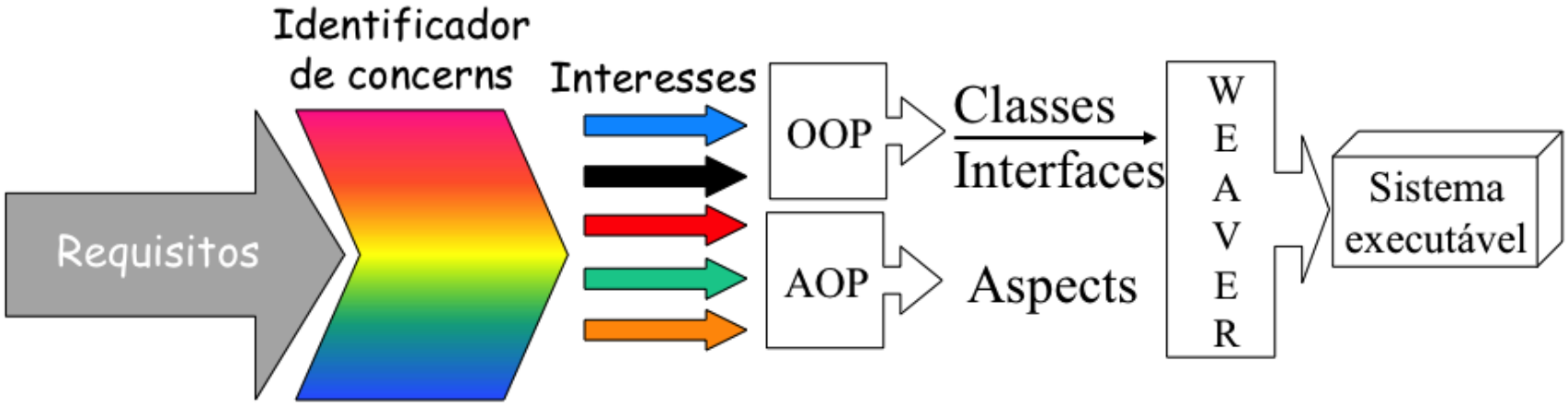
```
public class DBHandler {
 private static OleDbConnection connection;
 public static OleDbConnection Connection {
  get {
   if (connection == null) {
    string dataSource = "BancoCS.mdb";
    string strConexao =
     "Provider= Microsoft.Jet.OLEDB.4.0; " +
     "Data Source=" + dataSource;
    connection = new OleDbConnection(strConexao);
   }
   return connection;
  }
 }
 public static OleDbConnection GetOpenConnection() {
  Connection.Open();
  return Connection;
 }

 public static void StartTransaction() {
  Connection.Open();
  transaction = Connection.BeginTransaction();
 }

 }
```

```
public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
```

```
public class Conta {
  private string numero;
 ;
  }
  public void Debitar(double valor) {
```

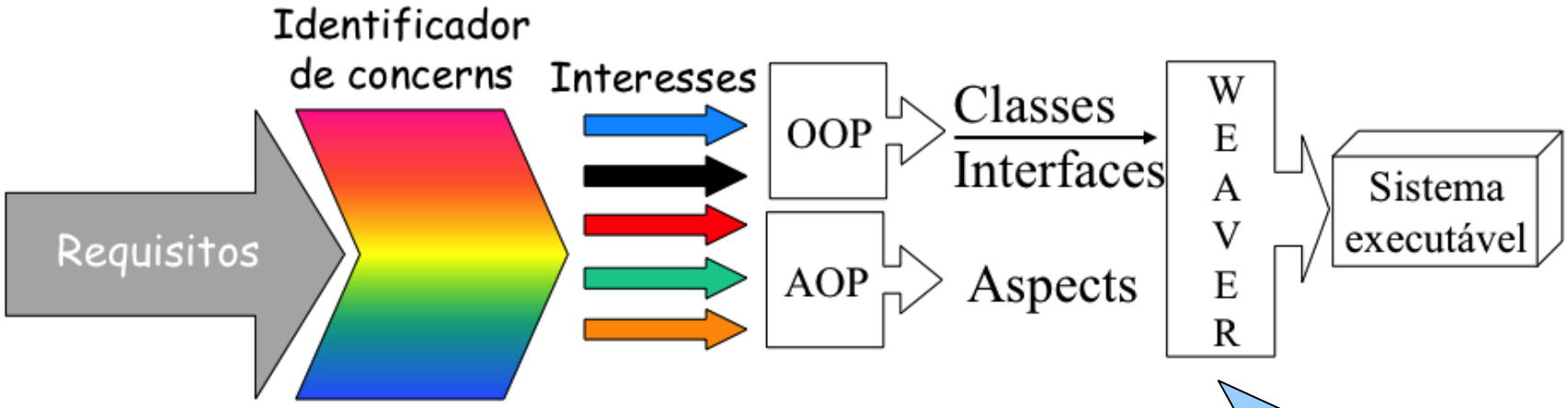
Persistência

Programação Orientada a Aspectos



Autor: Sérgio Soares (http://www.cin.ufpe.br/~scbs/aspectos/)

Programação Orientada a Aspectos



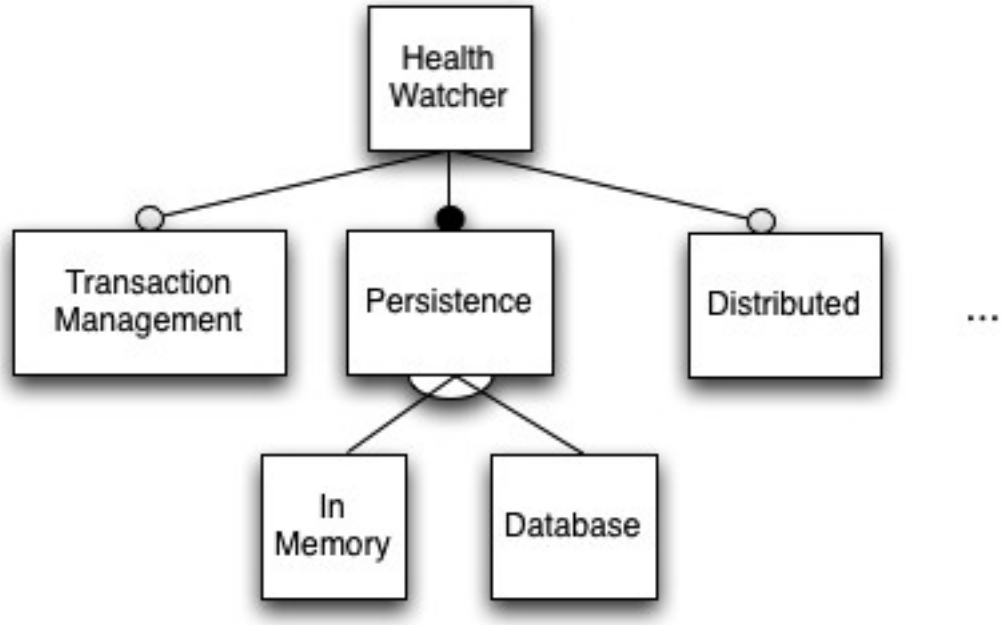
Autor: Sérgio Soares (http://www.cin.ufpe.br/~scbs/aspectos/)

Compõe os aspectos com o código OO. Tais composições ocorrem em pontos específicos.

Linguagens AOP

- AspectJ
- AspectC++
- CaesarJ
- AspectWerkz

Health Watcher SPL



Database implies Transaction Management

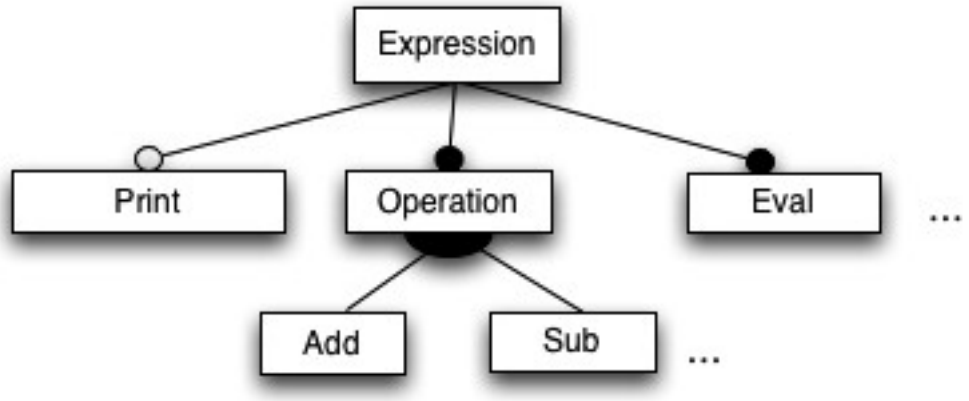
Modularizando transação com AspectJ

```
public aspect HWTransactionManagement {

    pointcut transactionalMethods(): execution(* HealthWatcherFacade.*(..));

    before(): transactionalMethods() {
        getPm().beginTransaction();
    }

    after() returning: transactionalMethods()  {
        getPm().commitTransaction();
    }

    after() throwing: transactionalMethods()  {
        getPm().rollbackTransaction();
    }

    public IPersistenceMechanism getPm() {
         return HWPersistence.aspectOf().getPm();
    }
    ...
}
```

Modularizando distribuição com AspectJ

```
public aspect HWServerDistribution {

   declare parents: HealthWatcherFacade implements IRemoteFacade;

   declare parents: healthwatcher.model..* implements Serializable;

   void around(): execution(static void HWFacade.main(String[])) {
    try {
      ...
    }
    catch (java.rmi.RemoteException rmiEx) {
      rmiFacadeExceptionHandling(rmiEx);
    }
    catch (java.net.MalformedURLException rmiEx) {
      rmiFacadeExceptionHandling(rmiEx);
     }
   }

...
}
```

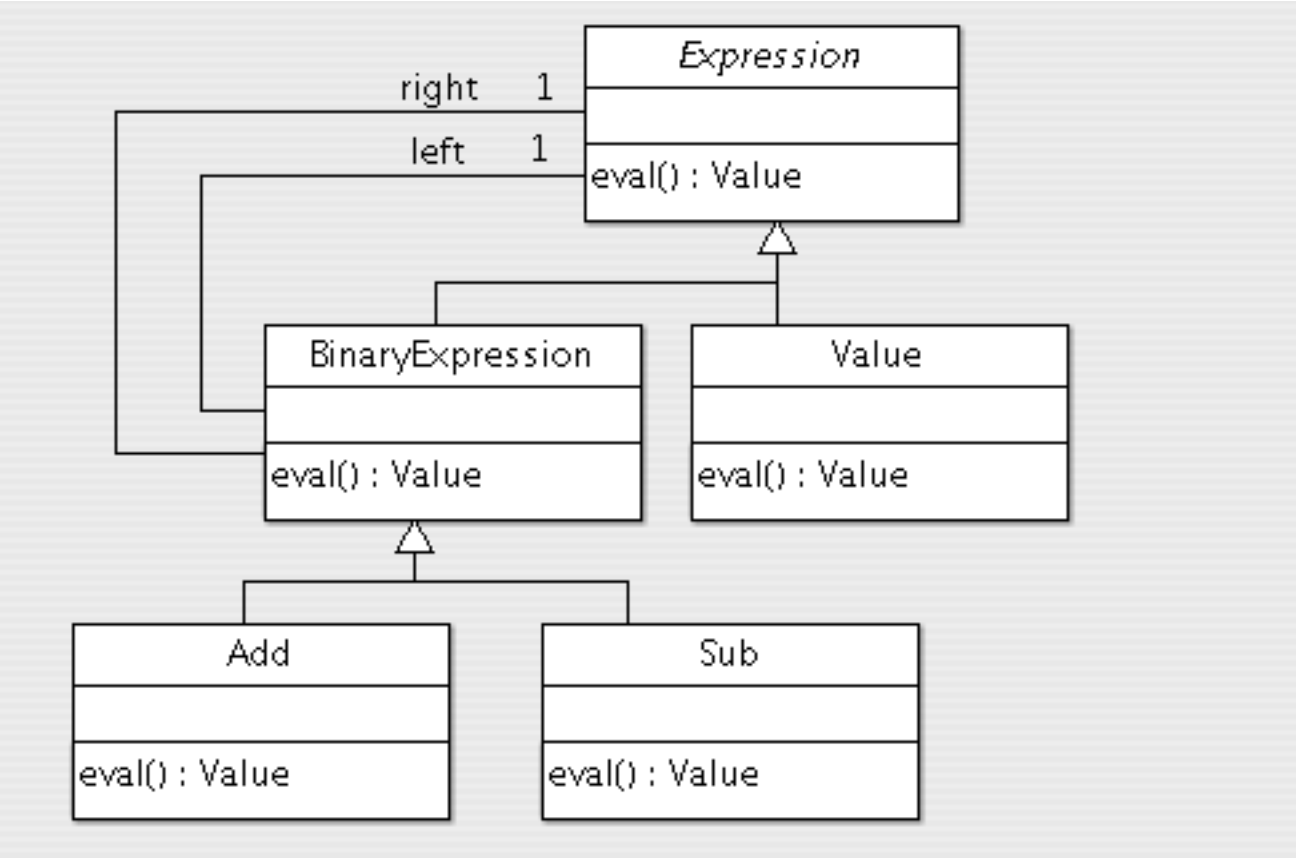
SPL para avaliar expressões



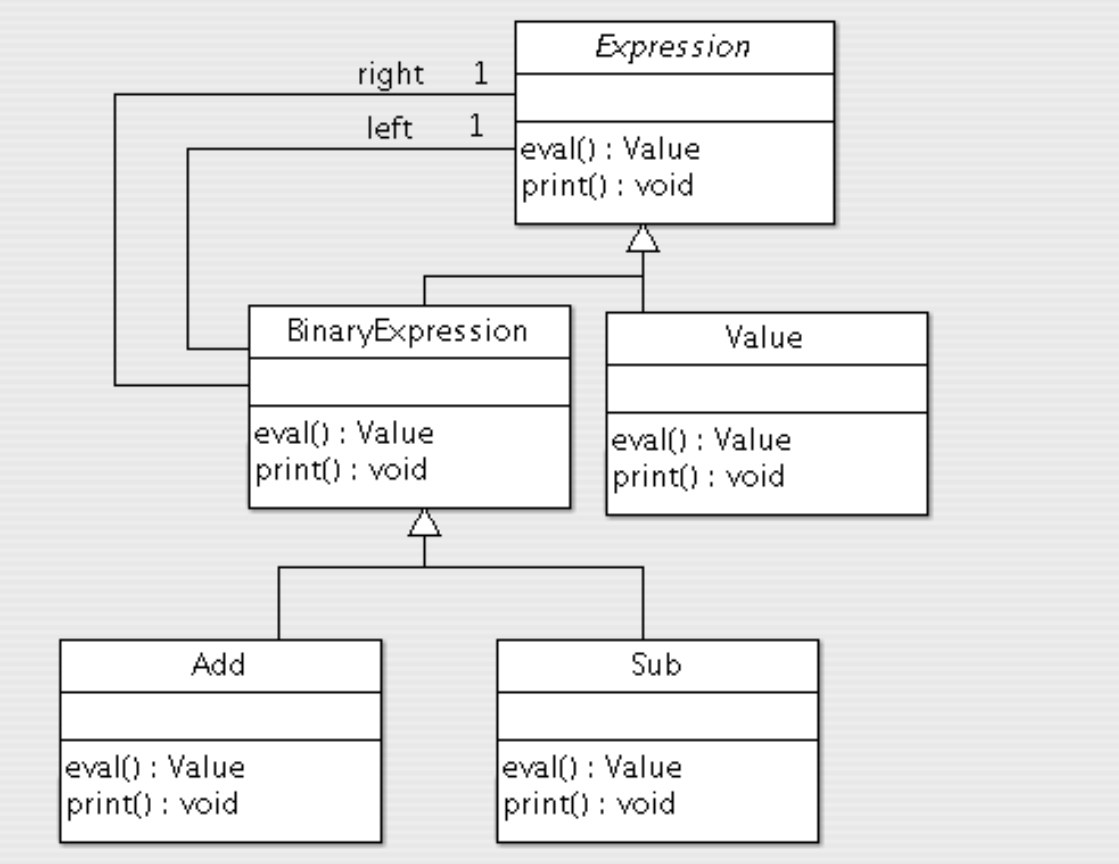
Modularizando a feature “Print” com AspectJ

```
public privileged aspect PrintAspect {

   public void Value.print(){
     System.out.print(value);
   }

   public void BinaryExp.print(){
     left.print();
     System.out.print(" "+ operator + " ");
     right.print();
   }

   ...
}
```

Antes de avaliar `PrintAspect`



Após avaliar `PrintAspect`



Sumário

- POA modulariza interesses transversais
- POA tem se mostrado interessante para modularizar features em linhas de produto de software
- Diferentes mecanismos podem ser classificados como orientados a aspectos

Referências


```
@book{laddad2003aspectj,
  title={{AspectJ in action: practical aspect-oriented programming}},
  author={Laddad, R.},
  year={2003},
  publisher={Manning}
}

@inproceedings{soares:2002,
 author = {Soares, Sergio and Laureano, Eduardo and Borba, Paulo},
 title = {Implementing distribution and persistence aspects with aspectJ},
 booktitle = {OOPSLA '02},
 year = {2002},
 pages = {174--190},
 location = {Seattle, Washington, USA}
 }
```

Referências

@article{383858,
 author = {Kiczales, Gregor and others},
 title = {Getting started with ASPECTJ},
 journal = {Commun. ACM},
 volume = {44},
 number = {10},
 year = {2001}
 }

 @proceedings{masuhara2003modeling,
  title={{A modeling framework for aspect-oriented mechanisms}},
  author={Masuhara, H. and Kiczales, G.},
  booktitle={Proc. ECOOP},
  volume={2003},
  year={2003}
}

Dúvidas

?

Obrigado!

Instituto Nacional de Ciência e Tecnologia para Engenharia de Software

Rede de Laboratórios de Produtividade de Software