Seletores de Pontos de Junção

Cristiano Breuel

Tópicos

- Introdução à Programação Orientada a Aspectos
- Motivação
- Trabalhos relacionados
- Proposta deste trabalho
- Comparação de abordagens

Introdução

- Linguagens de programação são uma forma de aumentar o nível de abstração da programação
- A programação orientada a aspectos é uma nova forma de abstração
- Este trabalho procura aumentar o nível de abstração deste novo mecanismo

Programação Orientada a Aspectos

- Objetivo: modularizar requisitos transversais
- É complementar à orientação a objetos
- Cria uma nova unidade de modularização: o **Aspecto**
- Um aspecto tem duas partes principais:
  - **Advices**: o comportamento a implementar
  - **Pointcuts**: onde aplicar esse comportamento

Pontos de Junção e Pointcuts

- Pontos de Junção (*joinpoints*): pontos na execução do programa em que podem-se adicionar *advices*
  - *Joinpoint shadow*: ponto no código-fonte ou representação intermediária que, em tempo de execução, corresponde a um ou mais *joinpoints*
- *Pointcuts*: conjuntos de *joinpoints* selecionados
  - Expressão de *pointcut*: uma expressão que define um *pointcut*

Qualidade de um *pointcut*

- Definida pelos seguintes critérios:
  - Resistência a mudanças
  - Clareza de intenção

Estilos de definição de pointcuts

- Enumeração
  - O *pointcut* enumera os *joinpoints* para seleciona-los

```
pointcut DBActivity() :
 call(void Person.updateName(String)) ||
 call(void Person.updateAge(int)) ||
 call(void Company.insertPerson(Person)) ||
 call(void Company.deletePerson(Person));
```

```
pointcut DBActivity() :
 call(void *.update*(..)) ||
 call(void *.insert*(..)) ||
 call(void *.delete*(..));
```

Estilos de definição de pointcuts

- Anotações
  - Os *joinpoints* são marcados com indicadores (anotações) que o *pointcut* seleciona

```
public class Person {
  @DBActivity
  public void updateName(String newName) {
    DBManager.openConnection();
    ...
  }
}
aspect DBConnectionDisposal {
  pointcut DBActivity() :
    call(@DBActivity * *(..));
  after() : DBActivity() {
    DBManager.closeConnection();
  }
}
```

Estilos de definição de pointcuts

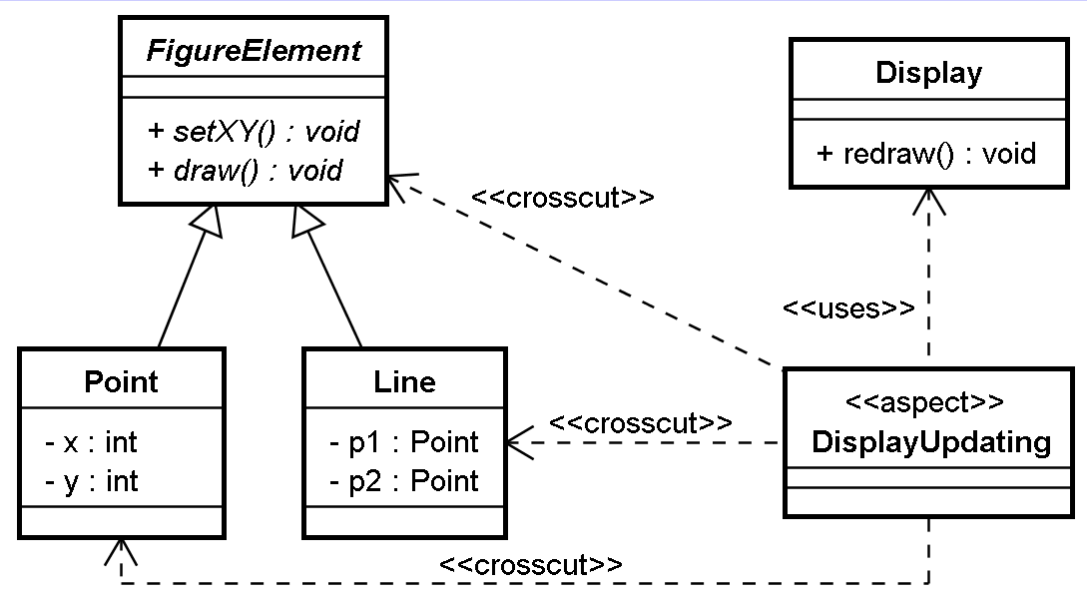
- Declarações semânticas
  - O *pointcut* seleciona os *joinpoints* de acordo com características de sua semântica

```
public class Person {
  public void updateName(String newName) {
    DBManager.openConnection();
    ...
  }
}
aspect DBConnectionDisposal {
  pointcut DBActivity(): execution(void Server.initiateProcess())
    && containsCall(* DBManager.openConnection());
  after DBActivity()(): {
    DBManager.closeConnection();
  }
}
```

Motivação

- Mecanismos de *pointcuts* atuais permitem resolver a maioria dos problemas
- Entretanto, a qualidade de muitos deles é baixa

Exemplo: Editor de figuras




```
aspect DisplayUpdating {
 pointcut move() :
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point)) ||
  call(void Point.setX(int)) ||
  call(void Point.setY(int));
 after() returning :
   move() && target(fe) {
  fe.redraw();
 }
}
```

```
aspect DisplayUpdating {
 pointcut move() :
  call(void FigureElement+.set*(..));
 after() returning :
   move() && target(fe) {
  fe.redraw();
 }
}
```

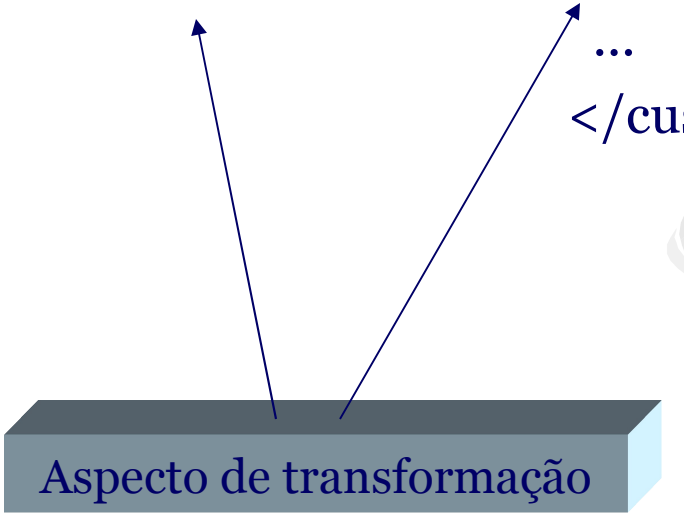
Exemplo: Aspectos para Web Services (1/3)

- Objetivo: aplicar aspectos a elementos XML específicos dentro de mensagens WebServices (SOAP)
- Estes elementos podem estar presentes em vários tipos de documentos, fazendo parte de diversos processos

Exemplo: Aspectos para Web Services (2/3)

```
<?xml version="1.0"?>
<company>
 ...
 <address>...</address>
 ...
</company>
```

```
<?xml version="1.0"?>
<customer>
 ...
 <address>...</address>
 ...
</customer>
```



Exemplo: Aspectos para Web Services (3/3)

```
request each(Address->AddressExt address):
 body(//Address, address)
```

Trabalhos relacionados

- Programação Funcional
- Programação Lógica
- Programação Imperativa

XQuery/BAT

- Transforma o programa em um modelo XML do bytecode
- Usa a linguagem XQuery para selecionar nós dentro deste modelo
- Vantagem: a XQuery é uma linguagem funcional completa, permitindo maior expressividade

Andrew (1/2)

- Linguagem de *pointcuts* baseada em PROLOG
- Modelo de *joinpoints* básico, mas com uma linguagem que permite combinações complexas

Andrew (2/2)

- Unificação de predicados
- Processamento sobre propriedades
- Regras parametrizadas reutilizáveis
- Recursão

Gamma (1/2)

- PROLOG usada como linguagem de *pointcuts*
- Modelo de *pointcuts* baseado em um traçado da execução do programa, com eventos numerados (*timestamps*)
- Permite criar *pointcuts* baseados seqüencialidade de eventos (inclusive no futuro)

Gamma (2/2)

- Exemplo

```
% Evento em T2 está no fluxo de controle da chamada em T1
cflow(T1, T2) :-
  calls(T1,_,_,_,_),
  endcall(T3,T1,_),
  isbefore(T1,T2),
  isbefore(T2,T3).
```

Alpha

- Semelhante a Gamma, porém:
  - Não permite referenciar eventos no futuro
  - Modelo de *joinpoints* possui 4 fontes de informação
    - representação da árvore abstrata de sintaxe
    - representação do armazenamento de objetos (*heap*)
    - representação do tipo estático de cada expressão do programa
    - representação do traçado (*trace*) da execução do programa

Josh

- Semelhante a AspectJ
- Baseada no arcabouço Javassist
- Linguagem de definição de expressões com baixa expressividade
- Permite a definição de novos "designadores de *pointcuts*" em Java
- Modelo de *joinpoints* estático, definido pelo arcabouço Javassist

Josh – Exemplo

```
static boolean paramTypeI(CtMethod m, String[] args,
                          JoshContext jc) {
  CtClass parType = m.getMethod().getParameterTypes()[0];
  CtClass argType = jc.getType(args[0]);
  if (parType.subtypeOf(argType)) {
    return true;
  }
  if (argType.subtypeOf(parType)) {
    jc.setIf("$1 instanceof " + argType.getName()) ;
    return true;
  } else {
    return false;
  }
}
```

Seletores de pontos de junção

Conceito

- Um seletor de pontos de junção é uma função que, dado um ponto de junção e um conjunto de parâmetros, determina se o ponto de junção deve fazer parte de um *pointcut*.
- Pode atuar em dois tempos
  - Tempo de *weaving*
  - Tempo de execução

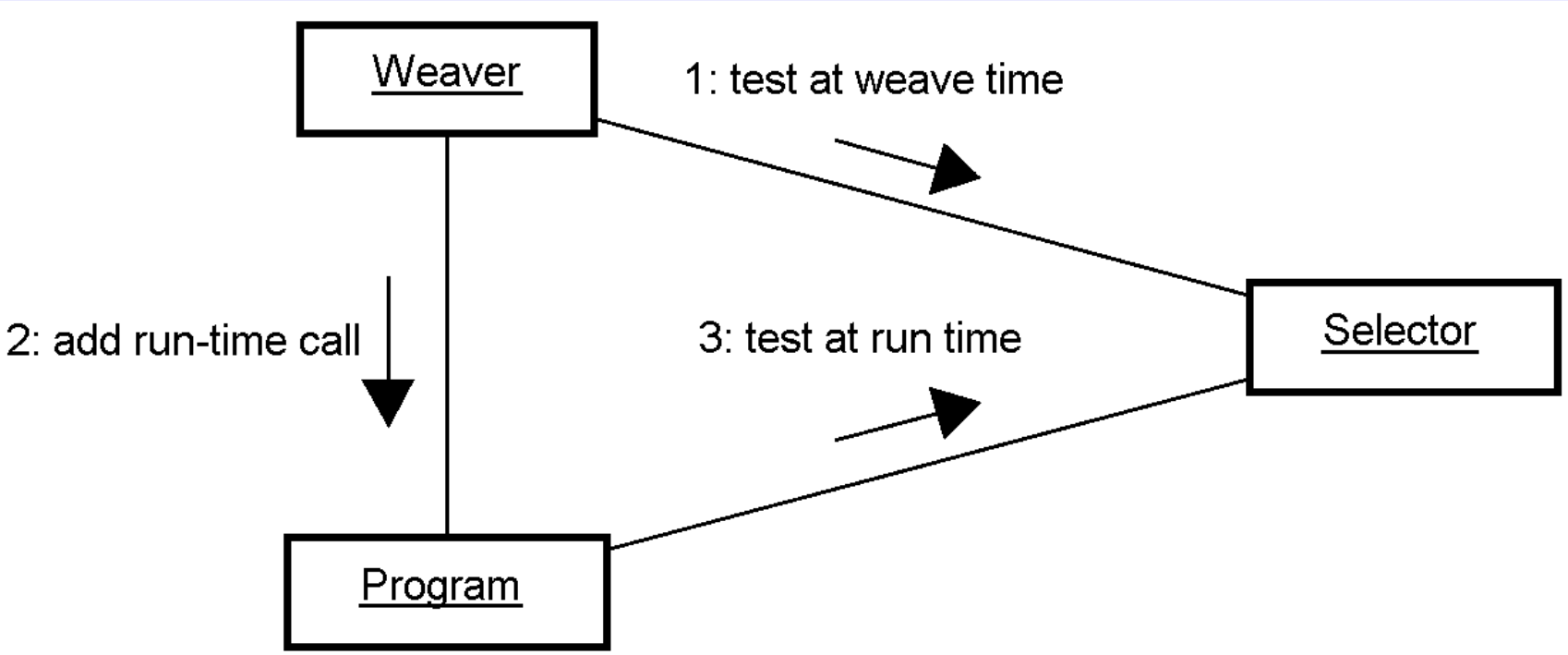
Exemplo

- `call`
  - é um seletor.
- `call(void *->setSize(..))`
  - é um *pointcut*.

Implementação

- Extensão ao arcabouço JBoss AOP
- Consiste em:
  - Extensão à gramática da linguagem de definição de *pointcuts*
  - Adição do mecanismo para declaração de seletores, através de XML ou anotações
  - Modificação do mecanismo de *weaving* do arcabouço, para adicionar chamadas a seletores em tempo de execução

Arquitetura



Declaração de um seletor

- Um seletor é uma classe que implementa a interface `org.jboss.aop.selector.Selector`
- Esta interface contém métodos para implementação de seletores para os diversos tipos de pontos de junção
- Há métodos de seleção em tempo de *weaving* e de execução.

Interface Selector

```
public interface Selector {
   /* Weaving-time selector methods */
   SelectionValue matchesExecution(Advisor advisor, CtMethod m,
                List<SelectorParam> selectorParams);
   SelectionValue matchesGet(Advisor advisor, CtField f,
                List<SelectorParam> selectorParams);
...

   /* Run-time selector methods */
   boolean matchesExecution(Advisor advisor, Method m,
                Object target, Object[] args,
                List<SelectorParam> selectorParams);
   boolean matchesGet(Advisor advisor, Field f,
                List<SelectorParam> selectorParams);
...
}
```

Declaração de um seletor

- Para que o seletor seja reconhecido, há duas formas de declará-lo
  - XML
  - Anotação
- Segue o mesmo padrão de outros elementos do JBoss AOP

Declaração de um seletor

```
<selector name="ParameterTypeIs"
   class="org.jboss.test.aop.ParameterTypeIs"/>
```

```
import org.jboss.aop.selector.SelectorDef;
import org.jboss.aop.selector.Selector;
@SelectorDef
public class ParameterTypeIs implements Selector {
...
```

Uso de um seletor

- Um seletor é usado como parte de expressões *pointcut*
- Pode receber parâmetros (no protótipo atual, apenas Strings)

```
@Bind(pointcut = "ParameterTypeIs(\"0\",\"java.lang.Integer\")")
public Object advice(Invocation invocation) throws Throwable {
  ...
```

```
<bind
   pointcut="ParameterTypeIs(&quot;0&quot;,&quot;java.lang.Integer&quot;)">
    <interceptor class="com.mc.Metrics"/>
</bind >
```

Exemplos

- Seletor de tipo de parâmetro
  - Prova de conceito simples
- Editor de figuras
  - Uma versão do exemplo clássico
- Chamadas por reflexão
  - Simples, porém com aplicação geral
- Seletor específico para arcabouço
  - Demonstra a capacidade de encapsulamento e riqueza semântica
- Doxpects
  - Outro exemplo de seletor com alto conteúdo semântico

Seletor de Tipo de Parâmetro

- Seleciona execuções de métodos com um parâmetro de um tipo específico em uma posição específica
- Leva em conta o tipo da referência em tempo de execução, não apenas o tipo formal
- Exemplo simples para demonstrar o uso de seletores

Seletor de Tipo de Parâmetro (parte estática)

```
@SelectorDef(name="parameterTypeIs")
public class ParameterTypeSelector extends SelectorHelper {

    public SelectionValue matchesExecution(Advisor advisor, CtMethod m,
                    List<SelectorParam> params) throws NotFoundException {

          int paramIndex = Integer.parseInt(params.get(0).getValue());
          String paramTypeName = params.get(1).getValue();

          CtClass parType = m.getParameterTypes()[paramIndex];

          CtClass argType = ClassPool.getDefault().get(paramTypeName);

          if (parType.subtypeOf(argType)) {
                    return TRUE;
          } else if (argType.subtypeOf(parType)) {
                    return CHECK_AT_RUNTIME;
          } else {
                    return FALSE;
          }

    }
...
```

Seletor de Tipo de Parâmetro (parte dinâmica)

```
...
    public boolean matchesExecution(Advisor advisor, Method m, Object target,
            Object[] args, List<SelectorParam> params) {

        int paramIndex = Integer.parseInt(params.get(0).getValue());
        String paramTypeName = params.get(1).getValue();

        Class parType;
        Class argType;

        argType = Class.forName(paramTypeName);

        Object arg = args[paramIndex];
        if (arg == null) {
            return true; // We convention that null always matches
        } else {
            parType = arg.getClass();
        }

        return (argType.isAssignableFrom(parType));
    }

}
```

Editor de figuras

- Busca o conjunto de métodos que atualiza, direta ou indiretamente, um campo pertencente ao conjunto de campos lidos, direta ou indiretamente, por um dado método

Editor de Figuras

```
@SelectorDef
public class UpdatesStateReadBy extends SelectorHelper {

    public SelectionValue matchesExecution(Advisor advisor, CtMethod m,
                        List<SelectorParam> params) throws NotFoundException {
        // Gets selector parameters
        String readerTypeName = params.get(0).getValue();
        String readerMethodName = params.get(1).getValue();
        // Obtains the reader method
        CtClass readerType = ClassPool.getDefault().get(readerTypeName);
        CtMethod readerMethod = readerType.getDeclaredMethod(readerMethodName);
        // Gets the sets of read and updated fields
        Set<CtField> readFields = getFieldsReadByMethod(readerMethod);
        Set<CtField> updatedFields = getFieldsUpdatedByMethod(m);
        // Compares sets
        boolean result = readFields.removeAll(updatedFields);
            return (result ? TRUE : FALSE);
    }

    private Set<CtField> getFieldsUpdatedByMethod(CtMethod m) {
        ...
    }

    private Set<CtField> getFieldsReadByMethod(CtMethod m) {
        ...
    }
}
```

Chamadas por reflexão

- Seleciona chamadas indiretas a métodos através da API de reflexão
- Dá acesso ao lado do chamador
- Funciona analogamente a um predicado "call"

Chamadas por reflexão (parte estática)

```
@SelectorDef
public class ReflectiveCall extends SelectorHelper {
   public SelectionValue matchesCall(Advisor callingAdvisor,
                MethodCall methodCall, List<SelectorParam> params)
   {
         CtMethod invoke = ClassPool.getDefault()
              .get("java.lang.reflect.Method")
              .getDeclaredMethod("invoke");
        if (methodCall.getMethod().equals(invoke)) {
                return CHECK_AT_RUNTIME;
        } else {
                return FALSE;
        }
   }
...
```

Chamadas por reflexão (parte dinâmica)

```
...
  public boolean matchesCall(Advisor advisor,
                AccessibleObject within, Class calledClass,
                Method calledMethod, Object target,
                Object[] args, List<SelectorParam> selectorParams)
  {
    String methodClassName = selectorParams.get(0).getValue();
    String methodName = selectorParams.get(1).getValue();
    Class argType = null;
    Method method = null;
    argType = Class.forName(methodClassName);
    Method[] methods = argType.getDeclaredMethods();
    for (Method m : methods) {
      if (m.getName().equals(methodName)) {
        method = m; break;
      }
    }
    Method targetMethod = (Method) args[0];
    return targetMethod.equals(method);
   }
}
```

Seletores específicos para Arcabouços

- Seletor para métodos que são “setters” de propriedades persistentes
- Utiliza o arcabouço *Hibernate*

Seletor de propriedade persistente

```
@SelectorDef(name = "hibernatePersistentSetter")
public class HibernatePersistentPropertySelector extends SelectorHelper {
    public SelectionValue matchesExecution(Advisor advisor, CtMethod m,
                    List<SelectorParam> params) {
        SelectionValue result = FALSE;
        // Gets the method's class
        Class declaringClass = m.getDeclaringClass().toClass();
        // Gets a Hibernate Session Factory
        SessionFactory sessionFactory = new Configuration().configure()
                                        .buildSessionFactory();
        // Gets the class's persistent properties an iterare over them
        ClassMetadata cmd = sessionFactory.getClassMetadata(declaringClass);
        String[] persistentProperties = cmd.getPropertyNames();
        for (String prop : persistentProperties) {
          // Gets the JavaBeans method used to set the property
          PropertyDescriptor pd = new PropertyDescriptor(prop, declaringClass);
          Method writeMethod = pd.getWriteMethod();
          // If the methods are the same, we found a match
          if (writeMethod.equals(m.getName())) {
            result = TRUE;
            break;
          }
        }
        return result;
    }
}
```

Doxpects

- Reproduz parte da funcionalidade da linguagem específica “doxpects”
- Seleciona elementos específicos dentro de mensagens XML
- Utiliza um arcabouço de *webservices*

Webservice Request (parte estática)

```
@SelectorDef(name = "request")
public class WsRequestSelector extends SelectorHelper {
   public SelectionValue matchesCall(Advisor callingAdvisor,
                MethodCall methodCall, List<SelectorParam> params)
                throws NotFoundException {
         // Calls a method to check for the apropriate static joinpoint
         if (isWsRequestMethod(methodCall)) {
                return CHECK_AT_RUNTIME;
         } else {
                return FALSE;
         }
   }
   private boolean isWsRequestMethod(MethodCall methodCall) {
         ...
   }
   private Document getWsDocument(Object target, Object[] args) {
         ...
   }
...
```

Webservice Request (parte dinâmica)

```
...
    public boolean matchesCall(Advisor advisor, AccessibleObject within,
                               Class calledClass, Method calledMethod, Object target,
                               Object[] args, List<SelectorParam> selectorParams) {
        // Gets selector parameters
        String xpathExpression = selectorParams.get(0).getValue();
        // Gets the XML Document
        Document docroot = getWsDocument(target, args);
        // Matches the document to the desired elements, given by the XPath in
        // the selector parameters
        XPath xpath = XPathFactory.newInstance().newXPath();
        NodeSet resultNodes = (NodeSet) xpath.evaluate(xpathExpression,
                                                       docroot, XPathConstants.NODESET);
        if (resultNodes != null && resultNodes.getLength() > 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

Comparação de abordagens

- Paradigma
- Pointcuts dinâmicos
- Informação temporal
- Resistência a mudanças
- Clareza de intenção
- Viabilidade prática

Comparação de abordagens

Linguagem / Abordagem	Paradigma	*Pointcuts* dinâmicos	Informação temporal	Resistência a mudanças	Clareza de intenção	Viabilidade prática
XQuery / BAT	Funcional	não	não	média	alta	média
Andrew	Lógico	sim	não	alta	alta	média
Gamma	Lógico	sim	sim	alta	alta	baixa
Alpha	Lógico	sim	sim	alta	alta	média
Josh	Imperativo	não	não	média	alta	alta
Seletores	Imperativo	sim	não	alta	alta	alta