# 5
# Um Modelo Generativo Orientado a Aspectos

Nesse capítulo é apresentado um modelo generativo orientado a aspectos que é usado para instanciação de variabilidades OO e OA encontradas em arquiteturas de famílias de sistemas. Esse modelo generativo compõe nossa abordagem OA para desenvolvimento de frameworks. Seu objetivo central é habilitar a instanciação automática de arquiteturas orientadas a aspectos baseado em informações coletadas por um modelo de características.

## 5.1.
## Visão Geral

Nosso modelo generativo orientado a aspectos contempla o uso de técnicas OA para habilitar a implementação e instanciação de arquiteturas de famílias de sistemas. Ele pode ser visto como uma instanciação do modelo genérico proposto por Czarnecki e Eisenecker [33]. Três modelos são definidos para habilitar a instanciação automática de uma arquitetura de família de sistemas, são eles: (i) modelo de características com relações tranversais – responsável pela especificação e coleta de características a serem instanciadas na arquitetura de família de sistemas; (ii) modelo de arquitetura – define os principais componentes e elementos de implementação que compõem a arquitetura; e (iii) modelo de configuração – especifica o mapeamento entre características e componentes (e respectivos sub-elementos) provenientes dos modelos de característica e arquitetura, respectivamente.

A Figura 20 apresenta as relações existentes entre os diferentes modelos existentes na nossa abordagem. Ela apresenta os modelos sob as perspectivas da engenharia de domínio e de aplicação. Na engenharia de domínio, cada um dos modelos (característica, arquitetura e configuração) é especificado considerando uma arquitetura OA de família de sistemas particular. Durante a instanciação de um membro da família (engenharia da aplicação), uma instância do modelo de característica é definida pelo engenheiro de aplicação. Uma ferramenta usa tal

modelo para customizar e instanciar a arquitetura OA, usando os modelos de arquitetura e configuração. As seções seguintes detalham as atividades de especificação e uso do modelo generativo.

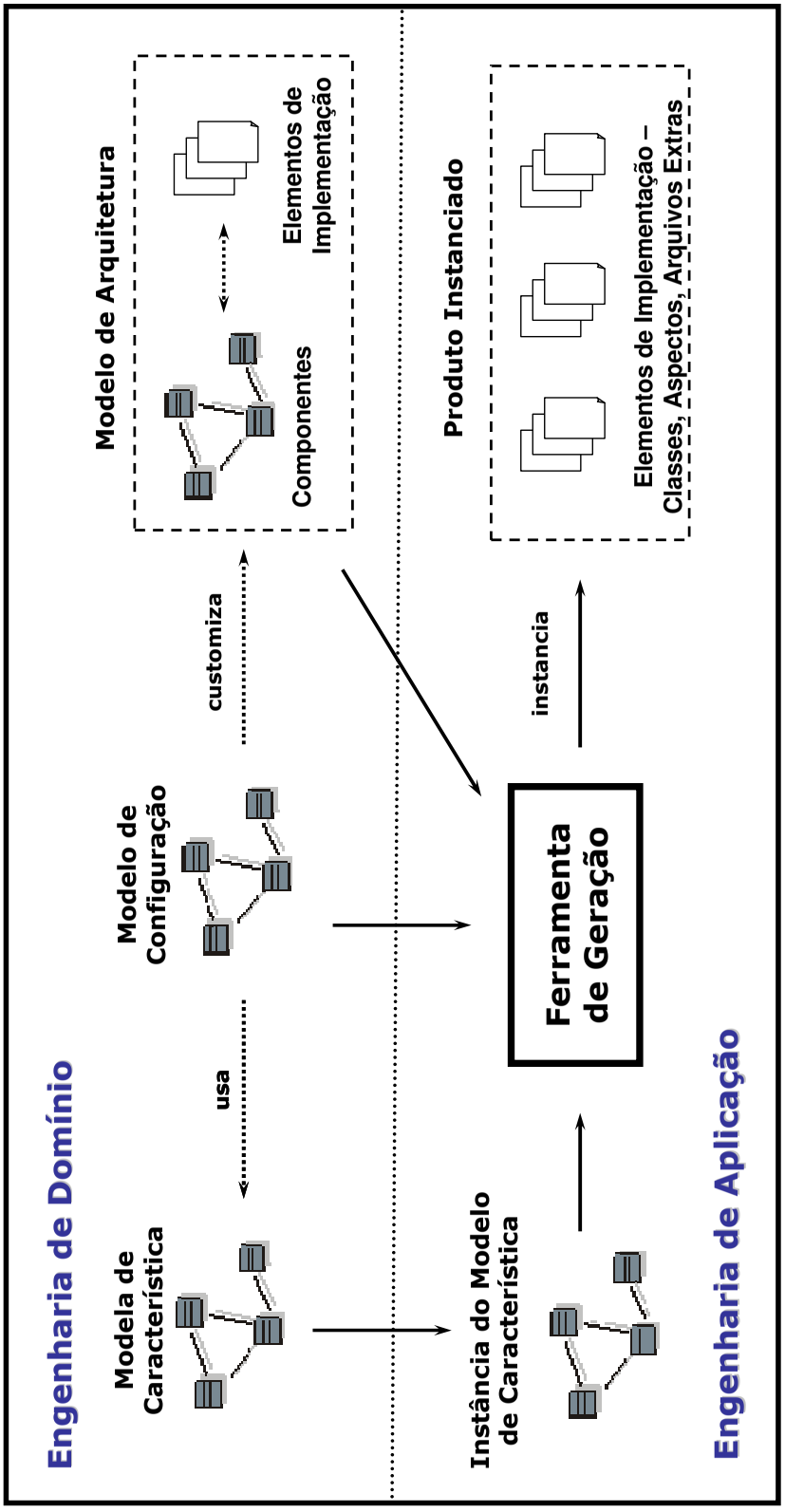


**Figura 20**. Visão Geral dos Elementos do Modelo Generativo

## 5.2. Engenharia de Domínio: Definição do Modelo Generativo OA

Nossa abordagem endereça na engenharia de domínio, a produção dos modelos de características, de arquitetura e de configuração que compõem o modelo generativo orientado a aspectos. Esses modelos devem ser produzidos (ou revisitados caso já tenham sido desenvolvidos anteriormente), após a implementação do framework (ou arquitetura de família de sistemas) usando as diretrizes apresentadas no Capítulo 4.

A arquitetura OA do JUnit (apresentada no Capítulo 4) será usada para ilustrar as atividades de definição do modelo generativo de forma a permitir a instanciação automática de suas variabilidades OO e OA. Nas seções seguintes são apresentadas as atividades de concepção dos elementos do modelo generativo, mostrando como as variabilidades do JUnit podem ser preparadas para serem instanciadas automaticamente.



### 5.2.1. Especificação do Modelo de Arquitetura

A implementação de uma arquitetura de família de sistemas resulta tipicamente em um conjunto de artefatos/elementos de implementação, tais como, classes, interfaces, templates, aspectos e arquivos extra. Uma agregação de vários desses elementos concretiza os componentes definidos anteriormente no projeto de uma arquitetura OA. Nossa abordagem OA para desenvolvimento de frameworks, por exemplo, produz um conjunto de elementos de implementação bem definidos, tais como, núcleo do framework, conjunto de classes abstratas e interfaces que representam os pontos flexíveis do framework, aspectos do núcleo, aspectos EJPs, e aspectos de variabilidade e integração.

A primeira atividade de definição do modelo generativo, é a especificação do seu modelo de arquitetura. Ele permite relacionar os elementos de implementação de uma arquitetura de família de sistemas com a especificação de seus componentes arquiteturais. O propósito principal do modelo de arquitetura é criar uma representação visual dos elementos de implementação de forma a relacioná-los com um modelo de característica que expressa as variabilidades existentes nos componentes. Ele é desenvolvido para ser usado e processado por

uma ferramenta de derivação de membros da família de sistemas. Nosso modelo de arquitetura é formado por um conjunto de componentes. Cada componente é responsável por implementar um conjunto de funcionalidades relacionadas. Eles agregam elementos de implementação, tais como, classes, interfaces, aspectos, arquivos extras e templates resultantes das atividades de projeto e implementação da arquitetura de família de sistemas. Durante a especificação do modelo de arquitetura, esses elementos devem ser agregados em componentes para facilitar o processo de instanciação da arquitetura. Os arquivos extras são aqueles necessários para a implementação do componente, tais como, arquivos de configuração, de imagens, etc. Já os templates são usados para implementar elementos da arquitetura (classes, interfaces, aspectos ou arquivos de configuração) que necessitam ser customizados durante o processo de instanciação da arquitetura. Para auxiliar na modularização de um componente, cada componente pode também ele próprio ser formado por um conjunto de sub-componentes.

Os templates são os únicos elementos que precisam ser implementados para complementar a especificação do modelo de arquitetura. Eles podem ser usados para representar: (i) especializações de classes que representam pontos flexíveis de um framework; (ii) subaspectos que definem implementações concretas para um dado interesse transversal; e (iii) alguma classe, aspecto ou arquivo de configuração que necessita sofrer customizações baseado em informação coletada pelo modelo de característica. Existem diversas tecnologias que possibilitam a implementação de templates, tais como, JET/EMF [18], Velocity[7], XSLT[8], etc.

A Figura 21(c) apresenta uma representação hierárquica do modelo de arquitetura da implementação OA do JUnit. Essa representação foi gerada usando o plugin EMF (*Eclipse Modeling Framework*) [18], a partir da interpretação de um meta-modelo simplificado para representação de modelos arquiteturais.

Como pode ser visto na Figura 21(c), dois componentes principais foram definidos para agregar os elementos de implementação presentes na arquitetura OA do JUnit: core e extensions. O componente core agrega os elementos de implementação que determinam o núcleo do framework, sendo composto por dois sub-componentes: (i) testing - que agrega as classes responsáveis pela definição

[7] The Apache Velocity Project. URL: http://velocity.apache.org/. 2007.

dos testes propriamente ditas; e (ii) runner - que agrega três diferentes sub-componentes cada um responsável pela implementação de uma diferente alternativa de interface gráfica para o JUnit. O componente testing é também responsável por agregar o aspecto EJP `TestExecutionEvents`, assim como os dois templates `TestSuiteTemplate` e `TestCaseTemplate` que são usados para criar, respectivamente, casos e suites de teste para uma dada aplicação.

O componente extensions agrega três sub-componentes (active, repeat, setup) que definem as extensões a serem aplicadas a suítes e casos de teste. Esses componentes definem aspectos abstratos de variabilidade, responsáveis por definir a funcionalidade de uma dada extensão, assim como templates que possibilitam a geração de subaspectos de cada um desses aspectos de variabilidade, sendo eles: (i) `RepeatTestTemplate` – define um subaspecto que será customizado para introduzir código de repetição em casos de teste específicos. Esse template pode ser usado para gerar automaticamente o aspecto concreto `RepeatAllTests` apresentado na Seção 4.2.1 (Figura 19); (ii) `ActiveTestTemplate` – define um subaspecto que será customizado para introduzir a funcionalidade de execução concorrente em suítes de teste específicos; e (iii) `TestDecoratorTemplate` – é usado para permitir a adição de código de configuração no início ou final de casos ou suítes de teste.

### 5.2.2. Especificação do Modelo de Características

A segunda atividade de definição do modelo generativo é a especificação do modelo de característica da família de aplicações sendo desenvolvida. Um modelo de característica permite representar as características comuns e varíaveis existentes em um dado domínio. Como o propósito do nosso modelo generativo é habilitar a instanciação automática de variabilidades existentes numa dada arquitetura, a especificação de um modelo de característica na nossa abordagem focaliza, principalmente, a representação das características varíaveis existentes em tal arquitetura.

Existem várias ferramentas disponíveis atualmente que oferecem suporte para a criação de modelos de características. Nesse trabalho foi utilizado o modelo

[8] XSL Transformations (XSLT) Specification. URL: http://www.w3.org/tr/xslt. 2007.

de característica proposto em [35], o qual permite modelar características obrigatórias, opcionais e alternativas. Além disso, esse modelo de característica também permite a representação de cardinalidades para especificação da quantidade de ocorrência de características, assim como a criação de propriedades e atributos em cada uma das características definidas. O plugin FMP (*Feature Modelling Plugin*) [6] oferece suporte para a modelagem dessa proposta de modelo de característica.

A Figura 21(a) apresenta o modelo de características com as variabilidades do framework JUnit modeladas usando o plugin FMP. Ele é composto de três características principais: Testing, Runner e Extensions. A característica Testing é modelada como obrigatória, porque ela deve existir em qualquer instanciação do JUnit. Ela é usada para especificar as suítes e casos de teste que serão criados numa dada instanciação. A característica Runner modela as três alternativas (TXT, AWT e Swing) de interface gráfica que são oferecidas pelo framework, apenas uma delas deve ser escolhida. Finalmente, a característica opcional Extensions permite definir diferentes extensões a serem aplicadas a suítes e casos de teste.

Nossa abordagem define uma extensão simples para o modelo de característica com o objetivo de permitir a customização de aspectos presentes no modelo de arquitetura. Nossa extensão define duas propriedades, denominadas `<<crosscutting>>` e `<<joinpoint>>`, as quais podem ser atribuídas a determinadas características. Uma característica `<<crosscutting>>` é usada para representar aspectos do modelo de arquitetura que podem estender o comportamento de outras características do sistema. Uma característica `<<joinpoint>>` é usada para representar pontos de junção específicos de classes (ou aspectos) do sistema, os quais são candidatos a serem estendidos por aspectos no espaço de solução. Relações transversais podem ser definidas entre esses dois tipos de características, durante o processo de instanciação automática, para permitir a customização de pontos de corte de aspectos para afetar classes/métodos específicos do sistema. Na nossa abordagem, características `<<crosscutting>>` e `<<joinpoint>>` são mapeadas, respectivamente, para os seguintes elementos de implementação: aspectos de extensão e pontos de corte existentes em EJPs.

A Figura 21(a) mostra exemplos de tais propriedades atribuídas ao modelo de característica do JUnit. As características RepeatedTest, ConcurrentTest e Configuration Test são modeladas como sendo <<crosscutting>>, porque podem ser aplicadas a outras características difusas no modelo de características. Já as características Test Suite e Test Case são modeladas como sendo <<joinpoint>>, porque são candidatas a serem estendidas por características <<crosscutting>>. Durante o processo de engenharia de aplicação, são definidas relacões tranversais, que indicam como as características Extensions são aplicadas às características Test Suite e Test Case.

### 5.2.3. Especificação do Modelo de Configuração

A atividade de especificação do modelo de configuração permite relacionar elementos dos modelos de característica e arquitetura definidos anteriormente. O modelo de configuração é usado para definir como uma configuração específica de características deve ser mapeada para uma configuração de componentes da arquitetura. Dessa forma, ele representa a especificação do conhecimento de configuração existente no desenvolvimento generativo [33]. A especificação de modelos de configuração possibilita entender, modificar e evoluir o conhecimento de configuração de forma independente do espaço de problema (modelo de característica) e do espaço de solução (modelo de arquitetura).

Nosso modelo de configuração é composto por três diferentes elementos: (i) relações de dependência entre componentes (e elementos de implementação) provenientes do modelo de arquitetura e características presentes no modelo de característica; (ii) relações tranversais válidas entre características <<crosscutting>> e <<joinpoint>>; e (iii) especificação do mapeamento entre características <<joinpoint>> e pontos de junção concretos existentes em classes do modelo de arquitetura. A Tabela 4 apresenta os elementos de nosso modelo de configuração e seu respectivo propósito para o processo de instanciação. Os parágrafos seguintes detalham cada um desses elementos.

As relações de dependência entre os componentes e sub-elementos (tais como classes, aspectos, templates e arquivos extras) e características são usadas para especificar quais componentes e sub-elementos devem ser instanciados

quando um conjunto de características é selecionado na etapa de engenharia de aplicação. Elas representam a concretização de modelos de decisão [117]. A Tabela 5 descreve regras de mapeamento entre: (i) diferente tipos de características que podem ser definidas; e (ii) elementos de implementação de nossa abordagem OA para desenvolvimento de frameworks. Essas regras de mapeamento podem ser usadas como base para derivar as relações de dependência entre elementos de implementação e características.

| Elemento do Mod. de Configuração | Propósito Principal |
|---|---|
| Relações de Dependência entre Elementos de Implementação e Características | • Escolha de Variabilidades |
| Relações Transversais Válidas entre Características `<<crosscutting>>` e `<<joinpoint>>` | • Restringir Relações Transversais no Espaço de Problema |
| Mapeamento entre Características `<<join point>>` e pontos de junção concretos de EJPs | • Customização de Pontos de Corte de Aspectos |

**Tabela 4.** Elementos do Modelo de Configuração

Além de usar as regras de mapeamento, as seguintes diretrizes podem ser usadas para especificar tais relações de dependência: (i) se um componente (ou sub-elemento) deve ser instanciado para todo membro da família de sistemas, nenhuma relação de dependência partindo do mesmo deve ser especificada; (ii) se um componente (ou sub-elemento) depende da ocorrência de uma característica específica, uma relação de dependência deve ser criada entre eles. As relações de dependência são usadas por uma ferramenta de geração de código para decidir quais módulos serão incluídos na aplicação sendo instanciada, baseado em uma instância do modelo de característica definido por engenheiros de aplicação.

As seguintes diretrizes são usadas para especificar tais relações de dependência: (i) se um componente (ou sub-elemento) deve ser instanciado para

todo membro da família de sistemas, nenhuma relação de dependência partindo do mesmo deve ser especificada; (ii) se um componente (ou sub-elemento) depende da ocorrência de uma característica específica, uma relação de dependência deve ser criada entre eles. As relações de dependência são usadas por uma ferramenta de instanciação para decidir quais módulos serão incluídos na aplicação sendo instanciada, baseado em uma instância do modelo de característica especificado por engenheiros de aplicação. No caso específico de elementos de implementação do tipo template, as relações de dependência definem se eles serão processados e incluídos no produto final gerado. Todo template depende necessariamente de uma característica, a qual oferece informação necessária para a sua instanciação.

| **Tipo de Característica** | **Elemento de Implementação** |
|---|---|
| Característica Obrigatória | • Núcleo do Framework<br>• Aspectos do Núcleo |
| Característica Alternativa Existente no Núcleo[9] | • Classes de Pontos de Extensão (*Hot-Spots*) do Núcleo do Framework |
| Característica `<<joinpoint>>` | • EJPs (aspectos) |
| Característica Opcional | • Aspectos de Integração e Variabilidade |
| Característica Alternativa `<<crosscutting>>` | • Aspectos de Integração e Variabilidade |

**Tabela 5**. Regras de Mapeamento entre Características e Elementos de Implementação

A Figura 21(b) ilustra as diferentes relações de dependência existentes no modelo de configuração do JUnit. O modelo mostra que cada um dos sub-componentes do componente runner (textui, awtui, swingui) e seus respectivos elementos de implementação (classes, interfaces, etc) serão instanciados baseado na alternativa selecionada para a característica Runner. Como podemos ver na Figura 21, todo template definido para o JUnit (seção 5.2.1) possui uma relação de dependência com elementos do modelo de característica. Os templates `TestCaseTemplate` e `TestSuiteTemplate`, por exemplo, dependem das características Test Case e Test Suite, respectivamente. Isso significa que para cada característica desse tipo que for solicitada pelo engenheiro de aplicação, uma nova classe de caso ou suíte de teste será criada, como fruto do processamento dos respectivos templates. Informação coletada pelo modelo de característica é usada durante esse processamento. O nome de cada caso ou suíte de teste definido no modelo de característica, por exemplo, será usado para nomear cada classe de teste (suíte ou caso de teste) a ser criada.

Embora as relações de dependência sejam usualmente especificadas para relacionar um determinado componente ou elemento de implementação a apenas uma característica, relações de dependência mais complexas podem também ser definidas, caso necessário, para permitir relacionar componentes ou elementos de implementação a um conjunto de características organizadas em expressões booleanas. Tais expressões booleanas podem ser usadas para indicar que um dado componente ou elemento de implementação depende: (i) da ocorrência de um conjunto de duas ou mais características; ou (ii) da ocorrência de determinadas características e não ocorrência de outras, simultaneamente.

Nosso modelo de configuração também define um conjunto de relações válidas que podem existir entre características `<<crosscutting>>` e `<<joinpoint>>`. Na prática essa informação é usada para especificar quais aspectos (representados por características `<<crosscutting>>`) podem afetar quais pontos de junção de classes/aspectos do sistema (representados por características `<<joinpoint>>`). Assim, uma ferramenta de instanciação pode usar tal informação para checar se engenheiros de aplicação estão especificando

[9] Tais características exigem a instanciação de no mínimo uma das alternativas existentes.

relacionamentos válidos entre características transversais e pontos de junção. A Tabela 6 mostra as relações transversais válidas para o estudo de caso do JUnit. Ela especifica que: (i) a característica <<crosscutting>> Repeat pode apenas estender o comportamento da característica <<joinpoint>> Test Case; (ii) a característica Concurrent Execution pode modificar apenas o comportamento da característica Test Suite; e (iii) a característica Test Setup pode estender ambas as características Test Case e Test Suite.

O terceiro e último elemento que deve ser definido no modelo de configuração é o mapeamento entre características <<joinpoint>> e pontos de junção concretos presentes em classes/aspectos do modelo de arquitetura. Essa informação é usada por uma ferramenta de instanciação para customizar pontos de corte durante a geração do código de aspectos. O mapeamento envolve a identificação de quais trechos de código de classes (ex: execução ou chamada de métodos e construtores) correspondem a uma dada característica <<joinpoint>>. Se todos os aspectos de extensão têm pontos de corte fixos, não há necessidade de especificar esse mapeamento. Na nossa abordagem, os pontos de junção concretos podem ser diretamente encontrados e extraídos dos aspectos EJP definidos para nossa arquitetura de família de sistemas. A Tabela 6 mostra o mapeamento das características <<joinpoint>> para pontos de junção concretos expostos pelos EJPs do framework JUnit.

| Modelo de Configuração | Framework JUnit |
|---|---|
| Relações Transversais Válidas | • Repeat feature <<crosscuts>> Test Case feature<br>• Concurrent Execution <<crosscuts>> Test Suite feature<br>• Test Setup <<crosscuts>> Test Suite feature<br>• Test Setup <<crosscuts>> Test Case feature |
| Mapeamento entre Características <<joinpoint>> e pontos de corte de EJPs | • Test Case feature <<maps>> TestExecutionEventsEJP. testCaseExecution(…);<br>• Test Suite feature <<maps>> TestExecutionEventsEJP.testSuiteExecution(…); |

**Tabela 6.** Elementos do Modelo de Configuração do JUnit.

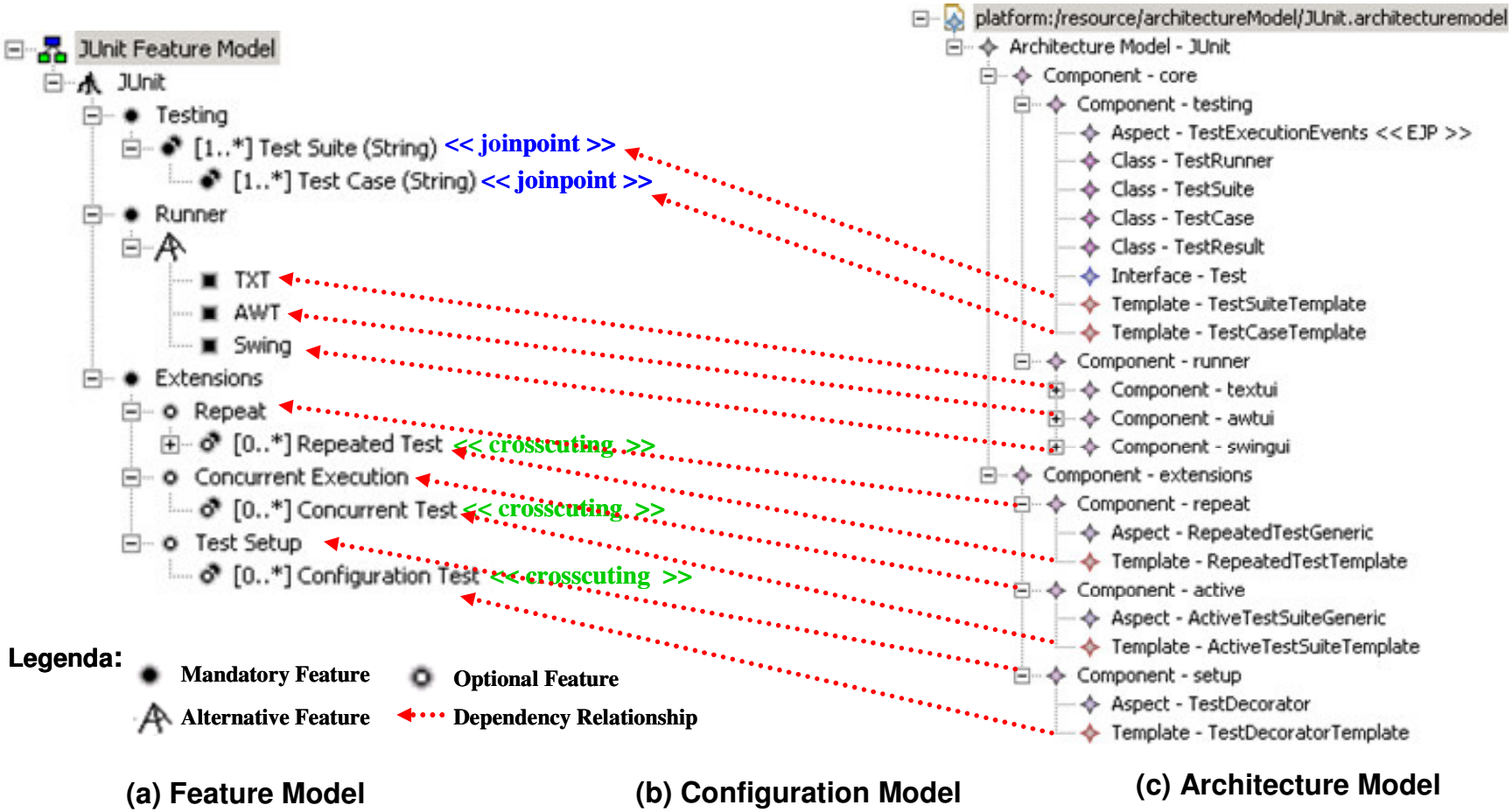


**Figura 21**. Modelo Generativo do Framework JUnit

### 5.2.4. Codificação de Templates

A última atividade de definição do modelo generativo OA é a codificação de templates. Conforme mencionado anteriormente, os templates são usados para codificar elementos de implementação que precisam ser customizados durante a instanciação de um membro da família de sistemas. Embora a estrutura geral dos templates possa ser definida durante a atividade de especificação do modelo de arquitetura, sua codificação só pode ser completamente realizada, após a especificação dos modelos de característica e de configuração. Isso ocorre porque estes últimos modelos oferecem informações relevantes para a especificação dos templates. O modelo de configuração indica as características que o template depende para ser instanciado. O modelo de característica por sua vez é usado para coletar os dados que auxiliam na customização das variabilidades presentes no template.

Existem muitas ferramentas disponíveis que implementam a tecnologia de templates [34]. Em uma implementação protótipo [79] de uma ferramenta de instanciação, a tecnologia *Java Emitter Templates* (JET) foi utilizada na codificação de nossos templates. JET é o motor de templates do plugin *Eclipse Modelling Framework* (EMF). Ele pode ser usado para implementar templates que representem qualquer tipo de elemento, tais como, código de classes e aspectos ou arquivos de configuração. A Figura 22 mostra um exemplo de implementação do template `TestSuiteTemplate` usando o JET. O template contém inicialmente seu código de configuração (linhas 1-5). A varíavel `testSuite` do tipo `Feature`[10] é usada para armazenar uma referência para uma sub-árvore de objetos que representam as informações coletadas pelo modelo de característica. No caso específico do template `TestSuiteTemplate`, como no modelo de configuração foi definido que ele depende de características Test Suite, o gerador de código irá processar esse template para cada característica desse tipo que for encontrada e passará como informação toda a sub-árvore definida por tais características. A implementação restante do template define como será o código gerado para a

[10] O tipo `Feature` é proveniente do meta-modelo do modelo de característica, sendo capturado pela ferramenta de instanciação e repassado para o template durante seu processamento.

classe de suíte de teste, oferecendo possíveis trechos de customização, tais como: (i) o nome da classe de suíte de teste (linha 9); e (ii) quais classes de caso de teste serão incluídas em tal suíte (linhas 13-18). Como pode ser observado no exemplo apresentado na Figura 22, diretivas específicas do JET são usados para processar as informações provenientes do modelo de características, tais como: (i) `<%= %>` – para acessar o valor de uma dada variável (linhas 9, 12 e 17); e (ii) `<% %>` – para definir um conjunto de comandos Java a serem executados (linhas 5 e 13-18). A Seção 5.3.3 apresenta em detalhes um algoritmo de customização do modelo de arquitetura baseado nos modelos de característica e configuração. Templates de aspectos devem também usar a informação sobre o mapeamento de ponto de junção do modelo de configuração para customizar pontos de corte[11].

```
01 <%@ jet package="translated"
02    imports="org.eclipse.emf.common.util.Elist ..."
03    class="TestSuiteTemplate" %>
04
05 <% FeatureElement testSuite = (FeatureElement) argument;%>
06 import junit.framework.Test;
07 import junit.framework.TestSuite;
08
09 public class <%=testSuite.getName()%>TestSuite {
10
11   public static Test suite(){
12     TestSuite suite = new TestSuite("<%=testSuite.getName()%>");
13     <% EList features = testSuite.getChildren();
14       for (Iterator iter=features.iterator(); iter.hasNext();){
15         FeatureElement testCase= (FeatureElement) iter.next(); %>
16         suite.addTest(
17               new TestSuite(<%=testCase.getName()%>Test.class));
18     <% } %>
19     return suite;
20   }
21 }
```

**Figura 22**. Template `TestSuiteTemplate`

[11] Essa informação de mapeamento deve também ser repassada pela ferramenta de instanciação ao template. Uma opção no exemplo acima, seria incluir um método chamado

## 5.3. Engenharia de Aplicação: Instanciação da Arquitetura OA

Durante a etapa de engenharia de aplicação, desenvolvedores solicitam a geração de uma instância da arquitetura OA. Esse processo é feito a partir da escolha das diferentes variabilidades presentes na arquitetura e que estão expressas no modelo de característica. Duas atividades principais estão presentes nessa etapa: (i) escolha das variabilidades em uma instância do modelo de característica; e (ii) escolha de relações tranversais válidas entre características. Uma ferramenta de instanciação usa a informação coletada nessas atividades e o modelo de configuração para gerar uma instância da arquitetura de família de sistemas, a partir da customização do modelo de arquitetura.

### 5.3.1. Escolha de Variabilidades no Modelo de Característica

A primeira atividade da engenharia de aplicação é a escolha de variabilidades e definição de seus respectivos parâmetros dentro do modelo de característica. Uma instância do modelo de característica é criada para possibilitar tal escolha. Essa instância representa um membro da família de sistemas que será solicitado à instanciação. Diversas ferramentas disponíveis na indústria, tais como, pure::variants[12], Gears[13] e FMP [6], suportam a criação de instâncias de modelos de características.

A Figura 23 mostra uma instância do modelo de características do JUnit. Nessa instância, o engenheiro de aplicação solicita duas suítes de teste (`TestSuiteModule1` e `TestSuiteModule2`) cada um com 2 casos de teste (`TestCaseA`, `TestCaseB`, `TestCaseC` e `TestCaseD`). A interface Swing é escolhida para a característica Runner. Finalmente, o engenheiro também define duas extensões a serem aplicadas a casos e/ou suítes de teste, são elas: (i) duas características Repeated Test com atributos de repetição (Quantity) 10 e 5; e (ii) uma característica Concurrent Test.

---

`getJoinPointFeatures()` na classe `Feature` de forma a poder acessar as características `<<join point>>` que uma dada característica `<<crosscutting>>` afeta.

[12] Pure::Variants. URL: http://www.pure-systems.com/, 2007.

[13] Big Lever. Gears. URL: http://www.biglever.com. 2007.

### 5.3.2.
### Escolha de Relações Transversais no Modelo de Característica

Uma vez escolhida as variabilidades, através da especificação de uma instância do modelo de característica, a próxima atividade envolve a escolha de relações tranversais entre características `<<crosscutting>>` e `<<join point>>`. Essas relações transversais são usadas para habilitar a customização de pontos de corte de aspectos. Elas definem, portanto, como será a composição dos aspectos com elementos do núcleo do framework. Nossa proposta é permitir a especificação de tais relações tranversais, de forma separada da escolha das variabilidades, dando mais organização, simplicidade e sistematização para cada uma das atividades. A especificação visual de tais relações tranversais pode ser feita através do uso de diagramas ou mesmo *wizards* que permitam conectar características `<<crosscutting>>` às características `<<join point>>`.

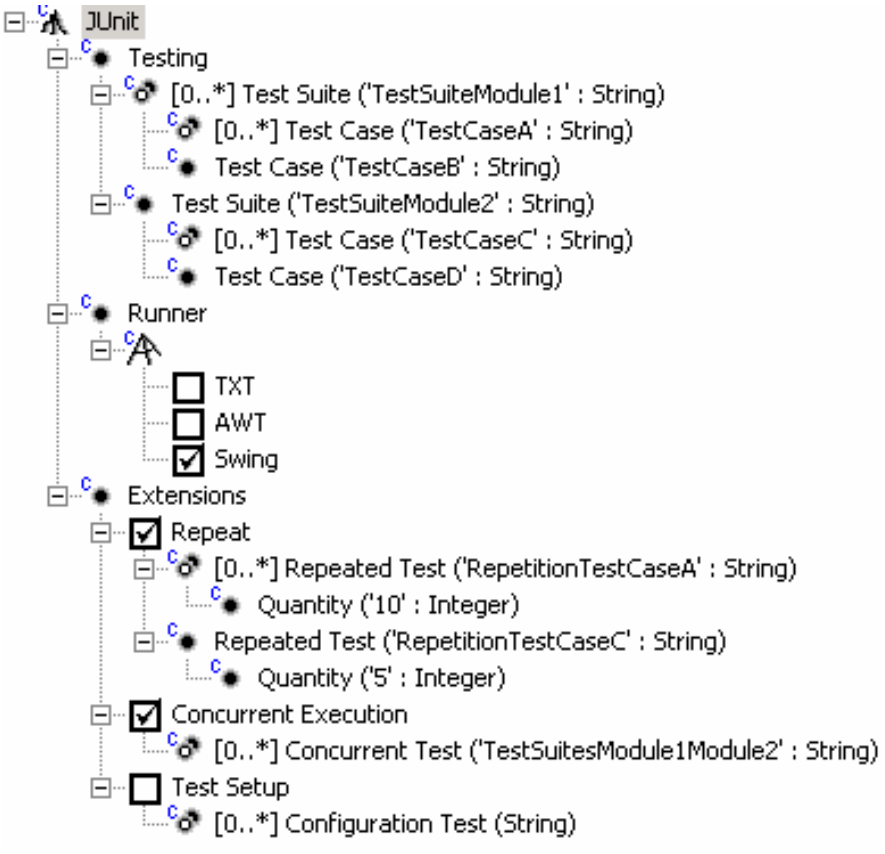


**Figura 23**. Instância do Modelo de Característica

A Figura 24 mostra a definição de relações transversais considerando a instância do modelo de característica do JUnit apresentada na seção anterior. Duas características <<crosscutting>> Repeated Test são aplicadas individualmente a duas características <<join point>> Test Case definidas. Já a característica <<crosscutting>> Concurrent Test é aplicada às duas suítes de teste solicitadas pelo engenheiro de aplicação.

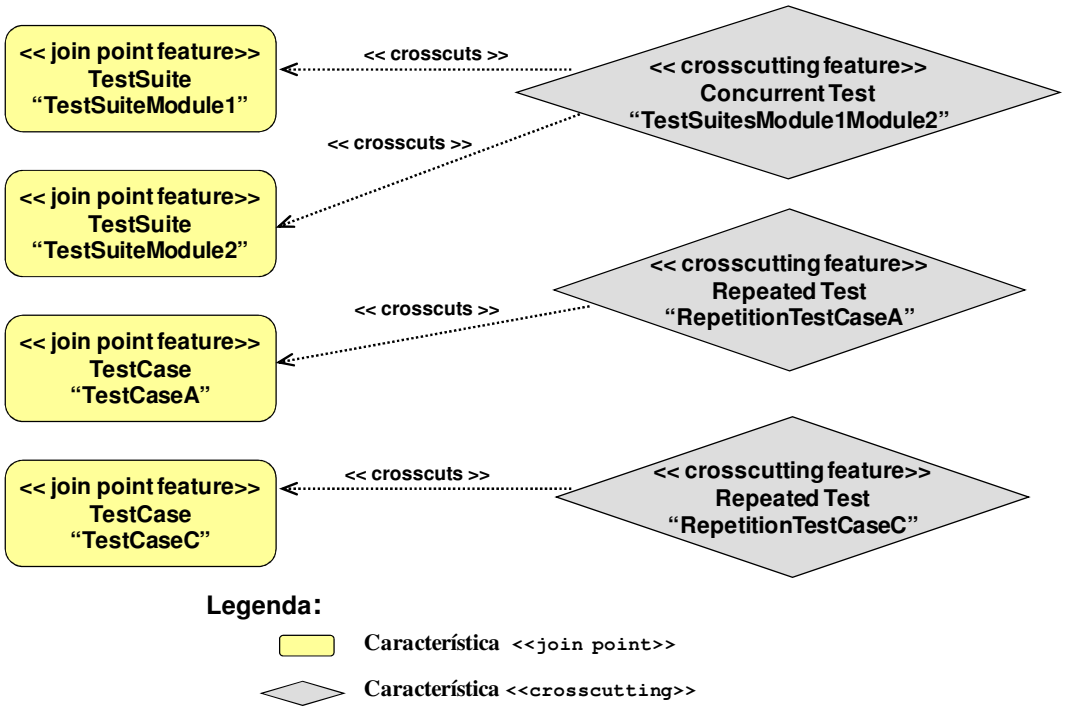


**Figura 24**. Definição de Relações Transversais

### 5.3.3. Algoritmo de Instanciação de Arquiteturas OA

A atividade final da engenharia de domínio é a solicitação de geração do código da arquitetura de um membro da família de sistemas, a partir de uma instância do modelo de característica e dos modelos de arquitetura e de configuração especificados para a família. Dessa forma, uma ferramenta de instanciação deve ser codificada para processar cada um desses modelos[14].

[14] Vale ressaltar que o processamento dos modelos de característica, arquitetura e configuração pela ferramenta de instanciação envolve o entendimento de seus respectivos meta-modelos como fonte de informação para navegar na estrutura de cada modelo criado.

Durante o processo de instanciação da arquitetura, a ferramenta deve realizar uma seqüência de 3 passos principais:

(i) *detecção de relações tranversais inválidas* – inicialmente, deve ser verificado se alguma relação transversal entre características que foi criada é inválida[15]. A detecção de relações transversais inválidas deve interromper o processo de geração, de forma a permitir que o engenheiro de aplicação resolva tal inconsistência;

(ii) *processamento do modelo de arquitetura* – no passo seguinte, o modelo de arquitetura é processado pela ferramenta da seguinte forma: para cada componente encontrado a ferramenta de instanciação verifica no modelo de configuração se ele depende de uma característica específica. Caso isso ocorra, a ferramenta apenas instancia tal componente (e processa seus respectivos sub-elementos) se existir uma ocorrência daquela característica na instância do modelo repassada para o gerador. Os componentes são instanciados através da criação de um pacote Java correspondente. Quando processando os elementos de implementação de um componente (classes, interfaces, aspectos, templates e arquivos extra) o mesmo processo é usado. Isso significa que a instanciação de cada elemento de implementação depende da ocorrência de características que eles eventualmente dependem. A instanciação de cada elemento de implementação implica em: (I) recuperá-lo de algum repositório ou do sistema de arquivos contendo todos os artefatos da família de sistemas; e (ii) carregá-lo em um projeto dentro de um ambiente integrado de desenvolvimento (IDE). Componentes e elementos de implementação que não possuem dependências de características são sempre instanciados. Elementos do tipo template devem sempre depender de alguma característica. Conseqüentemente, eles são processados para cada ocorrência daquela característica. Durante o processamento do template, a informação sobre a característica (e respectivas sub-características) da qual ele depende é usada para contemplar a customização do mesmo.

(iii) *processamento de relações transversais* – a ferramenta de instanciação usa as relações transversais entre características para determinar quais aspectos devem afetar quais classes do sistema. Templates de aspectos são definidos para representar aspectos com pontos de corte a serem customizados. Todo template

que representa um aspecto deve necessariamente depender de uma característica <<crosscutting>>. Dessa forma, quando ele é processado, o template pode obter a informação de quais características <<join point>> estão sendo afetadas pela respectiva característica <<crosscutting>> que o representa. De posse das características <<join point>> que se deve afetar, o template de aspecto usa o mapeamento entre características <<join point>> e pontos de corte concretos definidos no modelo de configuração, para realizar a geração dos pontos de corte dos aspectos.

Vamos considerar, por exemplo, a instanciação do framework JUnit, como resultado do processamento: dos modelos de característica, arquitetura e configuração da Figura 21; e da instância de modelo de característica da Figura 23. A Figura 25 apresenta as classes e aspectos gerados durante o processo de instanciação. Dessa forma, quando esses modelos são processados por uma ferramenta, o framework JUnit é instanciado da seguinte forma:

(i) criação de dois suítes de teste (TestSuiteModule1 e TestSuiteModule2) cada um referenciando duas classes de teste (TestCaseA, TestCaseB, TestCaseC e TestCaseD), através do processamento, respectivamente, dos templates TestSuiteTemplate e TestCaseTemplate;

(ii) inclusão das classes do componente swingui e não inclusão das classes dos componentes awtui e txtui;

(iii) criação de dois aspectos que implementam a característica de repetição para dois diferentes casos de teste. Esses aspectos são criados a partir do processamento do template RepeatTestTemplate. Os pontos de corte de tais aspectos são customizados baseados em informação de mapeamento de características em pontos de corte definidos no modelo de configuração;

(iv) inclusão dos aspectos ActiveTest e RepeatedTest e não inclusão do aspecto TestDecorator;

(v) criação de um aspecto que entrecorta as classes de suítes de teste (TestSuiteModule1 e TestSuiteModule2) para executá-las em *threads* separadas. O template ActiveTestTemplate é usado nesse processo..

[15] Dependendo da forma de modelagem visual das relações transversais isso pode ser garantido em tempo de especificação durante o processo de escolha das relações transversais (Seção 5.3.2)

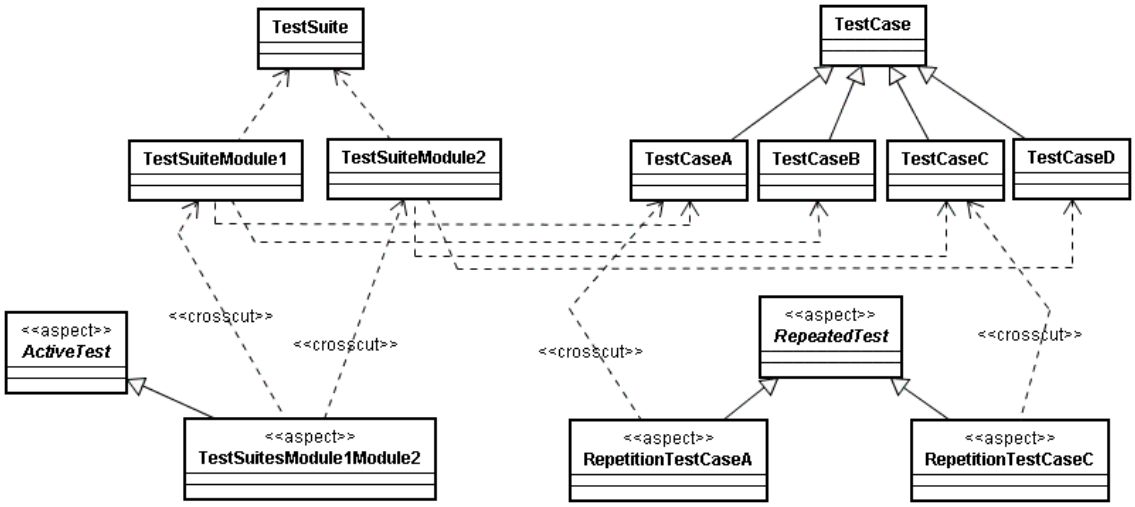


**Figura 25**. Classes e Aspectos Gerados para uma Configuração do JUnit

## 5.4. Sumário

Técnicas generativas vêm sendo cada vez mais adotadas no desenvolvimento de aplicações e famílias de sistemas. Neste capítulo foi apresentado um modelo generativo que incorpora abstrações de DSOA. O objetivo central do modelo é habilitar a instanciação de variabilidades OO e OA presentes em arquiteturas de famílias de sistemas e frameworks, usando como base modelos de característica. Uma implementação protótipo do modelo generativo foi implementada na criação de uma abordagem generativa OA para sistemas multi-agentes [77, 79]. A ferramenta foi implementada como um *plugin* da plataforma Eclipse usando como base: (i) o EMF (*Eclipse Modeling Framework*) – para criação e manipulação dos modelos de arquitetura e característica; e (ii) o JET (*Java Emitter Template*) – para criação e processamento de templates. Tal implementação não contemplava ainda a criação explícita do modelo de configuração. Uma nova ferramenta está sendo desenvolvida como parte das pesquisas de continuidade dessa tese de doutorado [26]. Novas tecnologias para desenvolvimento dirigido por modelos [114] disponíveis na plataforma Eclipse [109], tais como o OpenArchitectureWare[16], estão sendo adotadas. O capítulo seguinte apresenta três estudos de caso, os quais exemplificam o uso do modelo generativo.

[16] OpenArchitectureWare. URL: http://www.eclipse.org/gmt/oaw/. 2007.