CrossMDA: Arcabouço para integração de interesses transversais no desenvolvimento orientado a modelos

[1]Marcelo Pitanga Alves, [2]Paulo F. Pires, [2]Flávia C. Delicato, [1]Maria Luiza M. Campos

[1]Departamento de Ciência da Computação (DCC/IM) - Núcleo de Computação Eletrônica (NCE) - Universidade Federal do Rio de Janeiro (UFRJ) - Bloco C – Cidade Universitária – Ilha do Fundão – Rio de Janeiro, RJ – Brasil.

[2]DIMAp: Departamento de Informática e Matemática Aplicada - Universidade Federal do Rio Grande do Norte - Natal, RN - Brasil

mpitanga@gmail.com,{paulo.pires,flavia.delicato}@dimap.ufrn.br, mluiza@ufrj.br

Abstract. CrossMDA is a framework that encompasses a transformation process to integrate crosscutting concerns in model-oriented systems. Such integration is accomplished by combining the capacities of separation of concerns currently existent in MDA and AOP approaches. CrossMDA uses the concepts of horizontal separation of concerns from AOP to create independent business and aspect models, integrating those models through MDA transformations (vertical separation of concerns). CrossMDA comprises a development process, a set of services and support tools.

Resumo. CrossMDA é um arcabouço que incorpora um processo de transformação para integração de interesses transversais em sistemas orientados a modelo. Essa integração é feita combinando as capacidades de separação de interesses existentes nas abordagens MDA e Programação Orientada a Aspectos (POA). CrossMDA usa o conceito de separação horizontal de interesses da POA pra criar modelos de negócio e aspectos independentes, integrando-os através de transformações MDA (separação vertical de interesses). CrossMDA provê um processo de desenvolvimento e um conjunto de serviços e ferramental de apoio para dar suporte ao processo.

1. Introdução

A crescente complexidade dos sistemas de software atuais, aliada ao constante advento de novas tecnologias e a demanda sempre maior dos usuários finais por qualidade nos sistemas fornecidos, fazem com que as aplicações tenham que incorporar e lidar com um conjunto cada vez maior de requisitos de *software*. Dentre esses requisitos, os requisitos computacionais, como, por exemplo, a necessidade de concorrência, distribuição, persistência e recuperação de falhas afetam um grande número de componentes de um sistema, ou seja, cruzam as fronteiras de tais componentes, sendo responsáveis pelo espalhamento (*scattered*) e entrelaçamento (*tangled*) de funcionalidades, e conseqüentemente do código que as implementam. O espalhamento e o entrelaçamento, por sua vez, dificultam o entendimento, a manutenção e a evolução do código [Tekinerdogan et al. 2004]. Requisitos que não podem ser encapsulados em um único componente e tipicamente ficam espalhados por diversas partes do sistema,

tais como por exemplo, requisitos de monitoramento e sincronismo de código, são conhecidos como interesses transversais.

[Kiczales et al. 1997] apresentaram a abordagem de Programação Orientada a Aspectos (POA) que complementa a Programação Orientada a Objetos (POO) ao oferecer um conjunto de técnicas que permite o apropriado encapsulamento de interesses transversais através de uma nova abstração chamada aspecto, além de fornecer um mecanismo de composição (*weaving*) e reuso do código do aspecto.

A Arquitetura Orientada a Modelos (Model Driven Architecture - MDA) [OMG 2006a] é uma iniciativa do OMG para o desenvolvimento orientado a modelos que propõe três diferentes níveis de abstrações para a modelagem: Modelo Independente de Computação (Computational Independent Model-CIM), Modelo Independente de Plataforma (Platform Independent Model-PIM) e Modelo Dependente de Plataforma (Platform Specific Model-PSM). Os modelos são mapeados de uma abstração para outra através de processos de transformações sucessivas, durante as quais são incluídos novos elementos no modelo, reduzindo sua abstração até o nível de dependência da plataforma computacional aonde o sistema será implementado. A proposta MDA naturalmente provê uma forma de separação vertical de interesses, já que cada modelo contempla elementos específicos daquele nível de abstração, por exemplo, requisitos computacionais são incluídos apenas no modelo PSM. Porém, a separação de interesses segundo a dimensão horizontal não é tratada pela abordagem MDA, ou seja, não há mecanismos para identificar e encapsular interesses transversais dentro de cada modelo.

A separação horizontal de interesses está sendo atualmente abordada na área de Modelagem Orientada a Aspectos (MOA) [AOM 2006] onde trabalhos concentram-se em técnicas para identificação, análise, gerenciamento e representação de interesses transversais na fase de modelagem, usando extensões da UML [Suzuki e Yamamotto 1999, Stein 2002, Stein et al. 2002, Aldawud et al. 2003, Baniassad e Clarke 2004, Chavez 2004]. Porém, a falta de ferramentas adequadas para a modelagem e gerência do relacionamento de elementos do negócio com um determinado interesse transversal (processo de *weaving*) tem sido uma barreira na adoção desses modelos em ambientes MDA. Essa lacuna já vem sendo alvo de investigações que combinam os conceitos de POA com MDA propondo a integração de interesses transversais em modelos através do mecanismo de transformação MDA [Wampler 2003, Chaves e Zancanella 2004, Reina e Torres 2005, Solberg et al. 2005, Simmonds et al. 2005, Graziadei 2005]. Porém, ainda restam questões em aberto quanto à combinação das abordagens POA e MDA principalmente com relação à gerência do processo de combinação (*weaving*) e ao reuso de artefatos de transformação derivados desse processo.

A motivação do presente trabalho é propor uma solução que integre as abordagens POA e MDA e trate essas questões. Com esse intuito, propomos um arcabouço, denominado CrossMDA, o qual contempla um processo de transformação e provê um conjunto de serviços e ferramentas de apoio que implementam esse processo. O CrossMDA permite: i) elevar o nível de abstração na modelagem orientada a aspectos através do uso de modelos PIM de interesses transversais independentes do modelo de negócio; ii) reusar artefatos de interesses transversais no nível de modelos PIM; iii) automatizar o mapeamento do relacionamento de interesses transversais com elementos do modelo de negócio através do processo de transformação da MDA; iv) facilitar o reuso de artefatos de transformação MDA relacionados a interesses transversais; e v) favorecer o reuso de modelos PIM de negócios.

CrossMDA permite o tratamento de aspectos no nível de modelagem e fornece mecanismos que possibilitam a separação de interesses na dimensão horizontal, entre modelos de um mesmo nível, bem como a separação na dimensão vertical, entre modelos de diferentes níveis. A dimensão horizontal permite a modelagem de interesses transversais independentemente dos elementos de negócio. Para tanto, a CrossMDA sugere um processo que utiliza modelos de aspectos no nível de PIM. Tal modelo de aspectos é uma representação abstrata de um determinado interesse transversal, que permite esconder os detalhes de implementação do aspecto para o projetista de negócio, elevando assim o nível da modelagem no PIM.

Quanto à dimensão vertical, ela é endereçada através da implementação de um processo de transformação para gerar modelos das instâncias dos aspectos. Como os modelos de aspectos e de negócio são independentes, é oferecido ao projetista um processo para guiar e documentar os relacionamentos entre o aspecto e o elemento de negócio, a serem utilizados na composição do novo modelo. Através do uso de uma linguagem formal de transformação baseada no padrão MOF-QVT (*Query, View, Transformation*) [OMG 2006c], CrossMDA oferece, ao final de seu processo de composição de modelos (*model weaving*), a geração automática de um programa de transformação, o qual corresponde à implementação da especificação formal do processo de composição de modelos. Esse processo permite a edição, pelo projetista, de composições de modelos já existentes sem perda dos mapeamentos entre os aspectos e elementos de negócio.

Este artigo está estruturado em 4 seções. Na seção 2, é apresentado o arcabouço CrossMDA, seu funcionamento e sua implementação através de um exemplo ilustrativo. Na seção 3 são apresentados os trabalhos relacionados. Finalmente, na seção 4, são apresentadas a conclusão e considerações finais do trabalho.

2. Processo CrossMDA

O processo CrossMDA, apresentado no Diagrama de Processos da Figura 1, é composto de atividades, as quais são por sua vez organizadas em 3 fases: Fase 1 - seleção de fontes, Fase 2 - mapeamento e Fase 3 - composição do modelo.

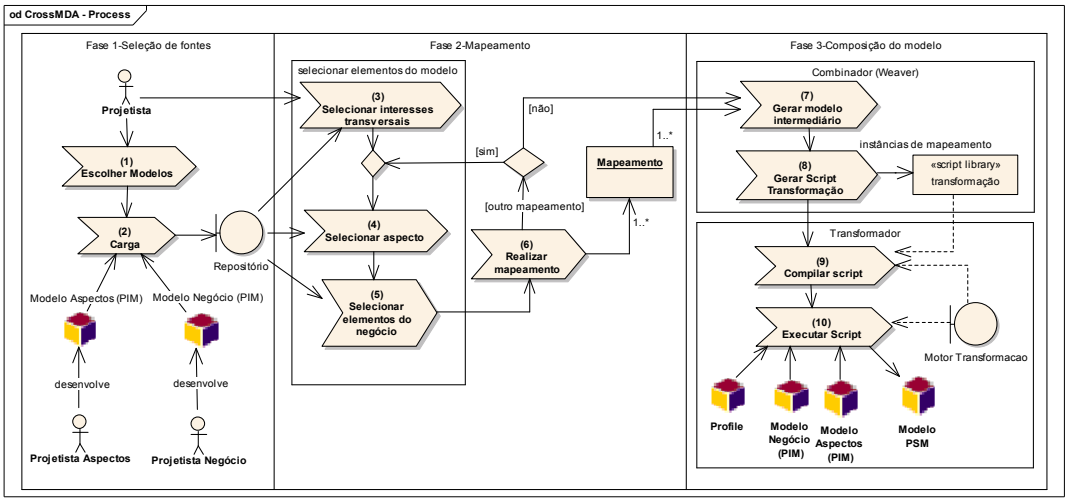

Figura 1. Processo CrossMDA.

A fase 1 engloba as atividades (1) e (2) (ver Figura 1). A atividade (1) consiste em realizar a escolha dos modelos PIM fontes a serem utilizados durante o processo de transformação e a atividade (2) é responsável pela carga e persistência dos modelos no repositório de metadados. Os modelos fontes são de dois tipos: modelo de aspectos e modelo de negócio. O modelo de aspectos consiste na representação abstrata, isto é, independente de plataforma, de interesses transversais, modelados como classes com o estereótipo <<*aspect*>> [Stein 2002] e organizados em pacotes. No CrossMDA, um pacote de aspectos é uma entidade que agrupa aspectos relacionados, ou seja, que dizem respeito a uma mesma categoria de requisito. Por exemplo, um pacote pode conter vários aspectos relacionados a autenticação, outro a *logging*, etc. O modelo de negócio, por sua vez, é composto pelas entidades, classes de serviços, relacionamentos, restrições, diagramas de classes e interação e demais elementos que representam toda a modelagem do processo de negócio.

A fase 2 é responsável por mapear os tipos de relacionamentos entre os aspectos e os elementos do modelo de negócio. Essa fase inicia-se com a atividade (3) que permite ao projetista selecionar os pacotes de interesses transversais que são relevantes ao domínio da aplicação. Em seguida, é iniciado o processo repetitivo de definição de relacionamento que engloba as atividades (4), (5) e (6). A atividade (4) é responsável pela seleção dos elementos aspectuais; a atividade (5) é responsável pela seleção dos elementos do negócio e; a atividade (6) realiza o mapeamento final do relacionamento, armazenando os elementos selecionados nas atividades (4) e (5) juntamente com o tipo designador do **ponto de junção** (*join point*) e os tipos de **adendo** (*advice*) selecionados no modelo de mapeamento.

A fase 3 é a responsável por realizar a composição do novo modelo, incluindo todos os elementos do modelo de negócio existentes e os novos elementos que representam as instâncias dos aspectos mapeados em um nível já dependente de plataforma computacional (PSM). É composta por 4 atividades que representam a **combinação de modelos** (*weaving*) e a **transformação**. A fase é iniciada com as atividades (7) e (8) do **combinador** (*weaver*). A atividade (7) é responsável por gerar um modelo intermediário a partir dos relacionamentos mapeados da fase 2. O modelo intermediário é uma representação que contém a hierarquia de composição de uma instância de uma classe aspecto e a sua dependência com o elemento de negócio ao qual se relaciona. Em seguida, a atividade (8) é iniciada, cuja responsabilidade é transformar o modelo intermediário em uma especificação formal através da geração de um programa de transformação baseada na especificação MOF QVT da OMG [OMG 2006c]. As atividades (9) e (10) representam as funções do transformador de modelos, e consistem respectivamente em compilar e executar o programa de transformação gerado pelo combinador de modelos.

2.1. Serviços do CrossMDA

Esta seção descreve os principais serviços providos por CrossMDA e que formam a base de trabalho para permitir a realização das atividades componentes do processo oferecido pelo arcabouço. Os serviços são: (i) persistência de modelos; (ii) mapeamento de elementos; (iii) combinador e; (iv) transformador de modelos. Para mostrar o funcionamento e a implementação desses serviços utilizamos um exemplo ilustrativo de um sistema de vendas ao qual deve ser aplicado um *interesse* de autenticação para controlar o acesso a qualquer informação do cliente.

Conforme descrito anteriormente, na *fase 1* o projetista seleciona os modelos PIM fontes que serão utilizados durante todas as fases do processo CrossMDA. Na Figura 2 é apresentado um modelo simplificado de aspectos, organizados em pacotes, utilizado como uma das fontes de entrada para o processo, no presente exemplo ilustrativo.

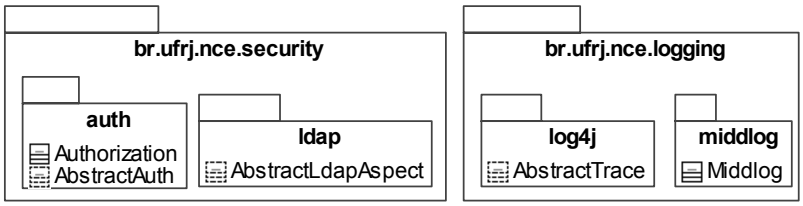

Figura 2. Modelo PIM de aspectos organizados em pacotes de interesses transversais.

Nesse exemplo de modelo, o pacote denominado *br.ufrj.nce.security.auth* contém os aspectos relacionados a autenticação. A Figura 3 ilustra uma classe representando um aspecto abstrato de autenticação. CrossMDA provê suporte para a representação de aspectos abstratos e não abstratos, onde as diferenças típicas entre eles são listadas a seguir. Aspectos abstratos podem possuir **pontos de atuação (***pointcut*) definidos como abstratos e estar ou não associados a um adendo. Um ponto de atuação abstrato não tem conhecimento dos pontos de junção que serão afetados e é um tipo de construção utilizada na POA para a criação de aspectos reutilizáveis. Nesse caso, na realização da classe do aspecto só é necessário definir os designadores de ponto de atuação (*pointcut designator* - PCD) e os pontos de junção. No caso de pontos de atuação abstratos não associados a um adendo, estes são combinados com outros pontos de atuação. Ainda, um adendo pode ser definido sobre um ponto de atuação abstrato e com isso pode-se implementar um comportamento transversal no aspecto abstrato. Aspectos abstratos são a base de desenvolvimento que permitem o seu reuso em diferentes cenários. Já os aspectos não abstratos, por sua vez, podem ter pontos de atuação não associados a um adendo, devendo-se também indicar o tipo do adendo (*after, before, around*) ao se configurar o ponto de atuação.

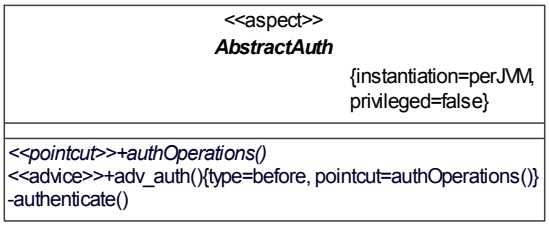

Figura 3. Classe representando um aspecto abstrato de autenticação para o modelo PIM (adaptado da proposta de [Stein 2002]).

O modelo de aspectos pode ser desenvolvido pelo próprio projetista do sistema ou pode-se utilizar algum modelo previamente desenvolvido para tal finalidade, seguindo as características de modelagem apresentadas na seção anterior. É importante notar que esse modelo representa aspectos independentes de plataforma. Ainda, a organização em pacotes torna-se um facilitador no momento da escolha de que tipo de interesse transversal será utilizado, restringindo a quantidade de aspectos a serem apresentados ao projetista durante o processo de mapeamento dos relacionamentos. O outro modelo fonte de entrada é o modelo de negócio que, para o exemplo utilizado, é apresentado na Figura 4.

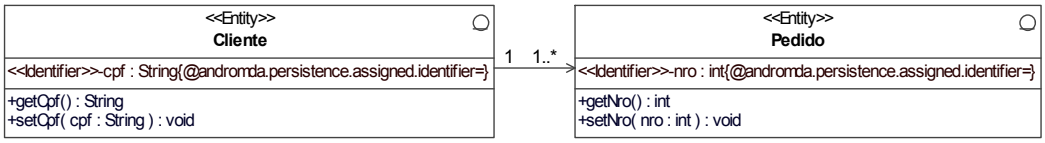

Figura 4. Fragmento do modelo PIM de classes do sistema de vendas utilizando perfil UML da ferramenta AndroMDA [AndroMDA 2006].

Os modelos selecionados são então carregados para o repositório de metadados que é o responsável pela persistência de todos os elementos que compõem os modelos.

2.1.1 Serviço de Persistência de Modelos

Este serviço é o responsável por implementar as operações básicas para permitir a carga e persistência dos modelos e as operações para navegar, recuperar e instanciar novos elementos em modelos existentes. A realização desta tarefa é feita por um serviço de repositório para persistência de metadados. Na implementação do CrossMDA, o repositório escolhido para gerenciar e persistir os elementos do modelo é o *NetBeans Metadata Repository* [NetBeans-MDR 2007]. Tal escolha deve-se principalmente ao fato desse repositório ser uma implementação popular e aberta do padrão OMG MOF (*Meta Object Facility*)[OMG 2006b]. Como o repositório é uma implementação externa ao ambiente de CrossMDA, o arcabouço provê um serviço (Figura 5) responsável pela interação com o repositório.

```
public interface IRepository  {
        public org.omg.uml.UmlPackage getUmlPackage();
        public Object getRepository();
        public void loadModel(String[] fileXmi, String searchRef) throws Exception;
        ....
```

Figura 5. Interface para o serviço de manipulação do repositório.

2.1.2. Serviço de Mapeamento de Elementos

Este serviço provê mecanismos para gerenciar o mapeamento dos relacionamentos entre os aspectos e os elementos de negócio, que é uma atividade chave do processo de CrossMDA. Os mapeamentos suportados em CrossMDA são de dois tipos: (i) pontos de atuação e (ii) intertipos. Porém, nesse artigo trataremos especificamente dos mapeamentos de pontos de atuação. Os mapeamentos de pontos de atuação seguem o padrão de especificação de pontos de atuação da abordagem POA e da linguagem AspectJ [AspectJ 2006, Laddad 2003], devido a essa especificação ser utilizada também por outras linguagens e arcabouços orientados a aspectos (Figura 6).

```
[visibilidade] [abstract] palavra-chave nome([args]) : tipo-pointcut ( assinatura-join point )
```

Figura 6. Definição de um ponto de atuação.

A especificação de um ponto de atuação é realizada através do uso de um tipo de ponto de atuação primitivo e da assinatura de um ponto de junção. Um tipo de ponto de atuação primitivo ou PCD, provê uma definição ao redor dos pontos de junção, por exemplo: um PCD do tipo chamada (*call*) corresponde a uma chamada para um método ou para um construtor. Existem vários PCDs disponíveis [Laddad 2003, 74pp] que são suportados no processo de mapeamento do CrossMDA. Os PCDs também podem ser

combinados através do uso de operadores lógicos, o que permite gerar especificações mais complexas para um ponto de atuação.

Com o objetivo de facilitar o mapeamento entre os elementos de negócio e os interesses transversais selecionados, o CrossMDA oferece para o projetista um processo e um serviço para armazenar os elementos do mapeamento. O processo é composto dos seguintes passos: (i) selecionar o aspecto e, se disponível, uma especificação de ponto de atuação pré-definido, podendo esse ponto de atuação ser ou não abstrato; (ii) selecionar uma ou mais entidades de negócio ou selecionar um ou mais métodos das entidades de negócio; (iii) indicar o tipo do ponto de atuação e; (iv) indicar o tipo do adendo (*before*, *after* ou *around*) [Laddad 2003, 81pp]. No exemplo do sistema de vendas é estabelecida uma regra de negócio em que toda a informação do cliente deve ser controlada via uma autenticação prévia. Para ilustrar os passos acima vamos aplicar um interesse de autenticação para controlar o acesso ao atributo *Cpf* da classe *Cliente*, conforme apresentado na Tabela 1.

Tabela 1: Passos e as informações selecionadas pelo projetista para o mapeamento

Passo	Informação selecionada		
	Tipo		Proprietário
(i)	ponto de atuação abstrato	método authOperations	classe AbstractAuth
(ii)	ponto de junção	método setCpf	classe Cliente
(iii)	tipo do PCD	chamada (*call*)	-
(iv)	tipo do adendo	-	-

Esse interesse é representado no modelo de aspectos por uma classe chamada *AbstractAuth* que especifica um método ponto de atuação abstrato chamado *authOperations* associado a um método adendo do tipo *before* chamado *adv_auth*. Este adendo implementa a programação necessária para efetuar a autenticação antes da chamada ou execução do ponto de junção. A Figura 3 representa a especificação UML desse interesse. Após a execução desses passos, tem-se então, a realização de um mapeamento (Figura 7). Tal mapeamento é persistido através do serviço de mapeamento seguindo um modelo específico do CrossMDA (Figura 8). Neste modelo, os relacionamentos entre os elementos *Aspect* e *Pointcut* possuem cardinalidade (1..n), permitindo assim que um mesmo aspecto possa relacionar vários pontos de atuação com tipos de adendos e tipos de PCD diferentes, da mesma forma que um mesmo ponto de junção pode ser referenciado por vários pontos de atuação com diferentes tipos PCD e, conseqüentemente, por vários aspectos. Cada elemento do modelo representa uma parte da especificação do ponto de atuação e também auxilia no mapeamento final de como o ponto de atuação será aplicado por um adendo do aspecto.

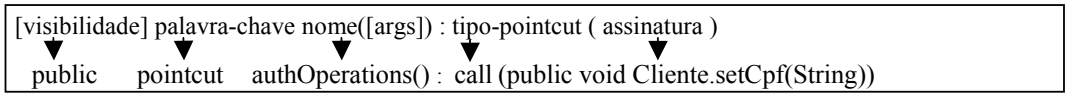

Figura 7. Definição de um ponto de atuação para controlar o acesso ao atributo Cpf da classe Cliente.

Conforme apresentado anteriormente, o serviço de mapeamento prevê ainda através do seu modelo (Figura 7) a capacidade de armazenar mapeamentos de declarações intertipos (*Intertype declaration*). Intertipos permite a declaração de membros e relações que afetam a estrutura e hierarquia de outros tipos. O elemento *Introduction* é o responsável por armazenar os membros (atributos, métodos e construtores) a serem adicionados para um tipo (incluindo outro aspecto) e o elemento

Parents armazena as informações necessárias para que o aspecto possa declarar que outros tipos implementam novas interfaces ou estende uma nova classe.

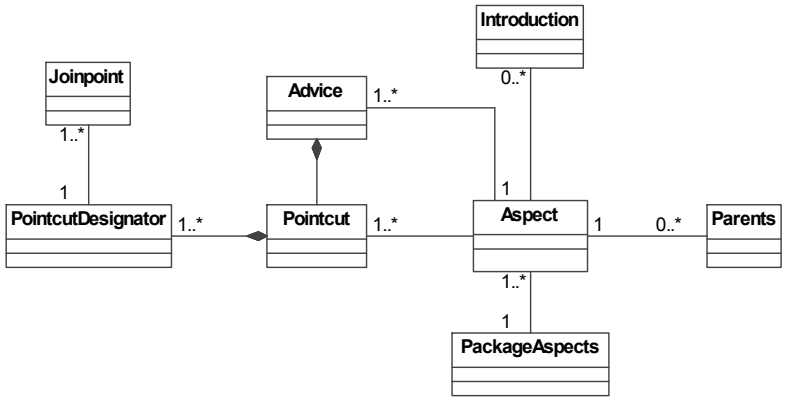

Figura 8. Modelo de mapeamento.

2.1.3. Serviços de Composição do modelo

A composição do modelo é uma fase que finaliza o processo de mapeamento, gerando um modelo que especifica o entrelaçamento entre os aspectos e os elementos de negócios. Esta fase é dividida em 2(duas) subfases que são: (i) combinação (*weaving*) e; (ii) transformação. Para cada sub-fase o CrossMDA provê um serviço.

2.1.3.1 Combinação (*weaving*)

A combinação consiste em integrar o modelo de aspectos ao modelo de negócio gerando as instâncias dos aspectos selecionados e as associações destas com os elementos de negócio. O serviço de combinação do arcabouço CrossMDA é provido por uma classe, chamada de combinador cuja responsabilidade é gerar o programa de transformação. O programa de transformação representa a implementação da especificação formal do processo de composição de modelos, ou seja, a integração entre elementos do modelo de negócio e as instâncias de aspectos mapeados para o novo modelo, o modelo PSM com os aspectos.

A atividade inicia-se quando o combinador recebe um conjunto de instâncias de mapeamentos e gera, internamente no arcabouço CrossMDA, o modelo intermediário. Com base nessa representação intermediária, é iniciada então a geração do programa de transformação. O programa de transformação é gerado através do uso de arquivos de *templates* de códigos (Figura 9) os quais são combinados, isto é, é feita uma fusão dos vários *templates* para a criação de um único *template*, e suas *tags* são substituídas pelas informações dos aspectos, oriundas do modelo de mapeamento, para geração do código final do programa de transformação. Por exemplo, a *tag* <*ASPECT_NAME*> durante a geração do programa é substituída pelo nome de um aspecto. Os *templates* são codificados utilizando a *ATLAS Transformation Language* (ATL) [Jouault e Kurtev 2005], linguagem de transformação proposta pelo ATLAS group (INRIA e LINA, Universidade de Nantes) para a especificação MOF QVT da OMG [OMG 2006c].

O modelo PSM de aspectos gerado em CrossMDA é uma adaptação do modelo de aspectos proposto por [Stein 2002], em que os aspectos são representados por classes da UML marcadas com o estereótipo <<*aspect*>>. Os pontos de atuação são representados por métodos da classe aspecto marcados com o estereótipo <<*pointcut*>.

Os adendos são representados por métodos com o estereótipo <<*advice*>> e com as etiquetas (*TaggedValue*) *type* e *pointcut,* identificando o tipo do adendo (*before, after* ou *around*) e o ponto de atuação respectivamente. Porém, quando realiza-se um aspecto em que um dos pontos de atuação é abstrato, essa definição do adendo não é necessária porque já está definida na classe aspecto abstrata correspondente. As especificações do ponto de atuação, dos PCDs e da assinatura do ponto de junção são mapeados como etiquetas dos métodos pontos de atuação. Ainda, informações relevantes de identificação do aspecto, tais como o seu *namespace* indicando o pacote aonde o aspecto encontra-se no modelo PIM, e o *extends* para indicar a abstração da qual o aspecto deriva, são configuradas como etiquetas da classe.

```
lazy rule newClass  {
from className : String, namespace : String
to t : UML!Class (
  name <-  className,
  visibility <- #vk_public,
  isAbstract <- false,
  namespace <-
    if thisModule.packageExists(namespace) then
       thisModule.getPackage(namespace)
   else
       thisModule.newPackage(thisModule.pckPSM)
   endif,
  stereotype <- thisModule.getStereotype('aspect'))
}
lazy rule newOperation {
from c : UML!Class, s : UML!Operation,
       stereotypeName : String
to t : UML!Operation (
  owner <- c,
  visibility <- #vk_public,
  name <- s.name,
  stereotype <-
    thisModule.getStereotype(stereotypeName),
  ...
} ...
```

```
thisModule.umlClass<-
if thisModule.classExists('<ASPECT_NAME_IMPL>','aspect')
   then thisModule.getClass('<ASPECT_NAME_IMPL>','aspect')
   else thisModule.newClass('<ASPECT_NAME_IMPL>',
                '<ASPECT_OWNER>')
endif;
if thisModule.taggedValueExists(thisModule.umlClass,
      'namespace') then true
   else  thisModule.newTaggedValue(thisModule.umlClass,
            'namespace', Sequence{'<ASPECT_OWNER>'})
endif;
thisModule.umlOperation <-
   if thisModule.operationExists('<ASPECT_NAME_IMPL>',
            '<POINTCUT_NAME>','pointcut')
    then  thisModule.getOperation('<ASPECT_NAME_IMPL>',
            '<POINTCUT_NAME>','pointcut')
   else thisModule.newOperation( thisModule.umlClass,
            thisModule.getOperation('<ASPECT_NAME>',
            '<POINTCUT_NAME>','pointcut'), 'pointcut')
   endif;
if thisModule.taggedValueExists(thisModule.umlOperation,
          'base')  then true
 else thisModule.newTaggedValue(thisModule.umlOperation,
              'base', Sequence{<POINTCUT_VALUE>})
endif;
```

Figura 9. Fragmentos de *templates* de código ATL para criação de classes e métodos.

Como exemplo, a Figura 10 apresenta um fragmento do programa de transformação que representa a integração do aspecto de autenticação com a classe de negócio *Cliente* de acordo com os mapeamentos definidos na seção 2.1.2.

```
thisModule.umlClass<-  if thisModule.classExists('AbstractAuth_Impl','aspect')
                          then  thisModule.getClass('AbstractAuth_Impl','aspect')
                          else  thisModule.newClass('AbstractAuth_Impl',  'br.ufrj.nce.security')  endif;
if thisModule.taggedValueExists(thisModule.umlClass, 'namespace') then true
        else  thisModule.newTaggedValue(thisModule.umlClass, 'namespace', Sequence{'br.ufrj.nce.security'})
endif;
if thisModule.taggedValueExists(thisModule.umlClass, 'extends') then true
        else thisModule.newTaggedValue(thisModule.umlClass,'extends', Sequence{'AbstractAuth'})  endif;
thisModule.umlOperation <-  if thisModule.operationExists('AbstractAuth_Impl', 'authOperations','pointcut')
    then thisModule.getOperation('AbstractAuth_Impl', 'authOperations','pointcut')
    else thisModule.newOperation( thisModule.umlClass,
                thisModule.getOperation('AbstractAuth', 'authOperations','pointcut'),'pointcut')   endif;
if thisModule.taggedValueExists(thisModule.umlOperation, 'base') then true else
  thisModule.newTaggedValue(thisModule.umlOperation,'base', Sequence{'call(public void Cliente.setCpf(String))'})
endif;
```

Figura 10. Fragmentos de uma regra gerada em linguagem ATL a partir de *templates* para a criação de instância da classe aspecto e seu ponto de atuação.

Para uma melhor ilustração da geração do programa de transformação, a Tabela 2 apresenta os elementos mapeados na fase do relacionamento e as *tags* dos *templates* a serem substituídas.

Tabela 2: Mapeamento dos elementos aspectuais nas *tags* dos *templates* para criação de instância da classe aspecto e seu ponto de atuação com seu respectivo ponto de junção referente aos relacionamentos da seção 2.1.2

Nome da Tag	Descrição	Valor
<ASPECT_NAME>	Nome da instância para o aspecto	AbstractAuth
<ASPECT_NAME_IMPL>	Nome de implementação para uma instância de aspecto abstrato	AbstractAuth_Impl
<ASPECT_OWNER>	Identifica o elemento ao qual o aspecto pertence	br.ufrj.nce.security.auth
<POINTCUT_NAME>	Nome da instância para o ponto de atuação	authOperations
<ADVICE_TYPE>	Tipo de ligação para o adendo	-
<POINTCUT_VALUE_ID>	Identificador das regras (valor) para um ponto de atuação	PointcutValueID_1
<POINTCUT_VALUE>	Tipo do designador (PCD) junto a assinatura do ponto de junção	call(public void Cliente.setCpf(String))

2.1.3.2. Transformação do modelo

A atividade de transformação do modelo é iniciada quando um programa de transformação, gerado pelo combinador, necessita ser compilado e executado. CrossMDA provê um serviço (Figura 11) para compilar e executar o programa de transformação e dessa forma gerar o novo modelo, através do uso do motor de transformação da ATL (*ATL engine*) [ATL 2007].

```
public interface IScriptCompiler {
    public void compile (String fileName);
}
```

```
public interface IScriptExecute {
   public int parseArgs(String[] args);
   public String[] setParameters(String script, String in, String out, String libs);
   public void run(); }
```

Figura 11. Interface para serviços de compilação e execução do motor de transformação.

O motor de transformação é um arcabouço que inclui uma máquina virtual (*ATLvm*) e um compilador. Também é fornecido um conjunto de classes escritas usando linguagem Java que oferece, entre outros serviços: (i) *parser,* para realizar a análise sintática do programa de transformação; (ii) *compilador*, para gerar o *byte-code* (arquivo com extensão *asm*) da máquina virtual (*vm)* e; (iii) *executor*, responsável por realizar a carga e execução do programa de transformação.

O resultado final da execução da transformação é um novo modelo que contém as adaptações previstas nas regras declaradas no programa de transformação. Assim, o resultado da execução do programa para o exemplo utilizado é o modelo de negócio adaptado com novas instâncias de classes que representam os aspectos e seus relacionamentos com os elementos de negócio (Figura 12).

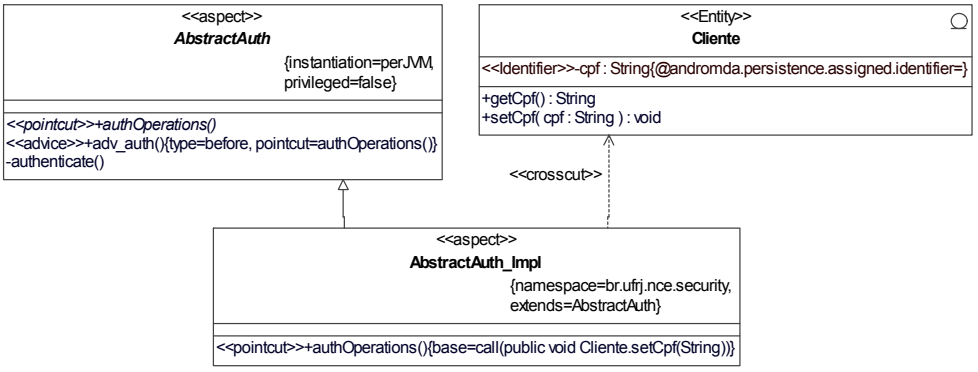

Figura 12. Modelo PSM aplicando-se o aspecto de autenticação na classe Cliente.

3. Trabalhos relacionados

Na área de Desenvolvimento de Software Orientado a Aspecto (DSOA) alguns trabalhos buscam integrar os conceitos da POA na modelagem de sistemas orientados a objetos. Tais trabalhos usam os mecanismos de extensibilidade da UML para adicionar novos elementos de modelagem que representam os conceitos da POA. Em [Aldawud et al. 2003] propõe-se a criação de um perfil UML para DSOA, que oferece aos projetistas meios comuns de representar visualmente os artefatos da POA, criando-se um metamodelo aspectual. Outra abordagem semelhante, denominada Modelo de Projeto Orientado a Aspecto, é proposta por [Stein 2002], na qual elementos da UML são estendidos para representar a semântica dos elementos da linguagem AspectJ [AspectJ 2006]. Esses trabalhos têm a vantagem de utilizar somente os mecanismos de extensibilidade da UML sendo, portanto, facilmente integrados às ferramentas de modelagem existentes. CrossMDA relaciona-se com estas abordagens por utilizar perfis UML na definição de elementos aspectuais que irão compor os modelos PSM gerados.

O trabalho de [Chavez 2004] propõe um arcabouço integrador dos conceitos de POA denominado teoria dos aspectos. Com base nessa teoria foi desenvolvida a linguagem *aSideML* para modelagem de sistemas orientados a aspectos, juntamente com o metamodelo *aSide,* que define a semântica de modelos estruturais e comportamentais representados nessa linguagem. Esse trabalho é um avanço significativo na área, por definir um metamodelo amplo que formaliza a semântica dos elementos relacionados à modelagem orientada a aspectos com UML. Como a linguagem *aSideML* é baseada em um metamodelo específico, faz-se necessária a criação de ferramentas que suportem esse novo metamodelo. Quando tais ferramentas estiverem disponíveis, CrossMDA poderá ter transformadores que gerem modelos no padrão *aSideML*. Dessa forma, o presente trabalho pode ser considerado como complementar ao que foi proposto em [Chavez 2004].

Ainda na linha de modelagem, uma discussão sobre uso de aspectos em modelos de domínio foi proposto por [Steimann, 2005]. Ele argumenta a favor de ter modelos de domínio livres de aspectos baseado na observação da falta de exemplos para aspectos no nível de domínios. Sua hipótese é que a idéia de aspectos em um desenvolvimento orientado a aspectos é um conceito de alto nível. Porém, [Rashid e Moreira, 2006] apresentam argumentos e exemplos que contradizem o trabalho de Steimann mostrando que modelos de domínio podem ter representações de aspectos. Em CrossMDA, os aspectos são representados em modelos separados do modelo de domínio no nível do PIM sendo representados no mesmo modelo somente no nível PSM após o entrelaçamento entre aspectos e negócio realizado pelo processo de transformação.

Na área de MDA, os trabalhos concentram-se na criação de modelos de transformação para facilitar a junção de aspectos com modelos do negócio (os modelos primários). Em [Chaves e Zancanella 2004] é apresentado um conjunto de extensões orientadas a aspectos para UML, chamado *Libra*, que possibilita a especificação de modelos tanto na parte estrutural como comportamental. O modelo de classes é incrementado de forma a representar aspectos e seus relacionamentos com o modelo primário, enquanto para definir os comportamentos é oferecida uma linguagem de ações usando sintaxe XML, com capacidades reflexivas e baseada na semântica de ações da UML. *Libra* utiliza a abordagem de transformação MDA para combinar o modelo de aspectos com os elementos do modelo primário. Apesar desse trabalho evidenciar a viabilidade da junção de POA e MDA para melhor integrar aspectos, por não ser o seu

foco principal, ele não apresenta nenhuma formalização de como essa combinação é realizada e tampouco trata de questões importantes como a especificação e gerenciamento de modelos de composição. Ambos os aspectos são tratados no CrossMDA e a sua formalização é feita através do programa de transformação escrito em uma linguagem formal de transformação [Jouault e Kurtev 2005].

[Reina e Torres 2005, Simmonds et al. 2005] utilizam a linguagem QVT [OMG 2006c] da abordagem MDA para realizar a transformação de modelos. [Reina e Torres 2005] usam a transformação para entrelaçar aspectos de AspectJ e elementos básicos ao nível de PSM antes da geração de código. CrossMDA tem uma abordagem semelhante, porém trabalha com modelos no nível do PIM. Já em [Simmonds et al. 2005] é apresentado um arcabouço que realiza a transformação de modelos de negócio e de aspectos do nível PIM para PSM. O arcabouço trabalha com dois modelos como entrada (primário e genérico de aspectos), os quais são especificados como diagramas de interação da UML. A ligação entre os modelos e a composição do novo modelo é feita através de transformações em QVT baseadas em metamodelos e especificadas pelo projetista. O modelo PSM do aspecto é dependente da plataforma na qual o aspecto será implementada, sendo o modelo de negócio marcado com estereótipos indicando a atuação do aspecto. Da mesma forma, a abordagem CrossMDA também separa os modelos de entrada (primário ou negócio e aspectos). Porém, segue outra abordagem na composição dos modelos, onde os modelos de aspectos são compostos por aspectos modelados em classes da UML [Stein 2002]. O modelo PSM gerado é a realização de uma classe aspecto do modelo PIM junto com a definição dos relacionamentos com o modelo de negócio. Assim, não existe alteração no modelo PIM de negócio, ao contrário da abordagem de Simmonds et. al, que realiza marcações no modelo de negócio para indicar os relacionamentos com aspectos. O modelo PSM gerado no CrossMDA está alinhado com a abordagem de modelagem de ferramentas de transformação modelo-texto existentes, como o AndroMDA [AndroMDA 2006], permitindo assim a continuidade do processo até a geração de código.

4. Conclusão

CrossMDA é um arcabouço que incorpora um processo de transformação para integração de interesses transversais em sistemas orientados a modelos explorando a sinergia entre as abordagens POA e MDA. Tal integração é realizada quando um modelo é transformado de nível PIM para o nível PSM, permitindo, desta forma, que a modelagem de interesses transversais seja ortogonal à modelagem dos processos de negócio. O uso das técnicas de POA auxilia nas atividades de mapeamento dos relacionamentos e na composição do modelo. Já da proposta MDA foram utilizadas a abordagem de transformação, como a base para automatizar o processo de integração dos interesses transversais, e a adoção de uma linguagem de transformação baseada no recente padrão MOF QVT da OMG para geração do programa de transformação. O uso dessa linguagem, aliado à forma de implementação através de *templates* de código, tornam o CrossMDA uma poderosa ferramenta na geração de programas de transformação baseada em aspectos.

Uma preocupação no projeto do CrossMDA é promover um alto grau de reuso de artefatos. No nível de artefatos de transformação, o reuso é alcançado através da utilização de *templates* em ATL capazes de gerar programas de transformação para diferentes sintaxes de linguagens formais de transformação, por exemplo em QVT ou MWDL [Milewski e Roberts 2005]. O uso de *templates* facilita a manutenção e permite

que novas implementações do programa de transformação sejam realizadas sem alterar o código do CrossMDA. Quanto a artefatos de modelo, o reuso é favorecido tanto no nível de PIM quanto PSM. O emprego de modelos PIM de aspectos permite que estes sejam desenvolvidos por qualquer projetista e reutilizados em várias transformações. Por outro lado, modelos PIM de negócio podem ser reaproveitados e entrelaçados com diferentes modelos de aspectos de forma a gerar sistemas que necessitem de diferentes requisitos computacionais. No nível PSM, o CrossMDA gera modelos PSM de aspectos seguindo as tecnologias MDA padrão, no caso XMI, fazendo com que esses modelos sejam utilizáveis por qualquer ferramenta MDA de transformação modelo-texto para geração do código fonte do aspecto. Outra característica importante do CrossMDA é o baixo grau de acoplamento entre os modelos PIM e PSM de aspectos e o modelo de negócio. Essa característica permite um certo grau de independência entre os modelos PIM de negócio e modelos de aspectos, fazendo com que esses modelos (PIM e PSM) possam ser evoluídos sem interferência mútua.

O CrossMDA foi implementado utilizando linguagem de programação Java e inclui as ferramentas necessárias para automatizar todas as atividades do processo proposto através de uma interface gráfica simples e de fácil operação[CrossMDA 2007].

Referências

Aldawud, O.; Elrad, T. and Bader, A. (2003). UML Profile for Aspect-Oriented Software Development. In Third Workshop on Aspect-Oriented Modeling with UML, AOSD'03. Boston, Massachussets, March, 2003.

AndroMDA (2006). Disponível em: http://www.andromda.org. Acesso em: 05/04/2006.

AOM (2006). Aspect-Oriented Modeling Workshop. Disponível em: http://www.aspect-modeling.org. Acesso em: 01/03/2006.

AspectJ (2006), a Java implementation of AOP. Disponível em: http://www.eclipse.org/aspectj. Acesso em: 05/04/2006.

ATL (2007), ATL Home Page. Disponível em: http://www.eclipse.org/m2m/atl/. Acesso em: 11/01/2007.

Baniassad, E. and Clarke, S. (2004). "Theme: An Approach for Aspect-Oriented Analysis and Design" In Proceedings of the 26th ICSE, Edinburgh, Scotland, May 2004.

Chaves, R.; Zancanella, L. C. (2004). Modelos Executáveis Baseados em Aspectos, apresentado no WASP'04 - Primeiro Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos, 18 de Outubro de 2004, Brasília, Brasil.

Chavez, C. F. G. (2004). A Model-Driven Approach for Aspect-Oriented Design. Rio de Janeiro, 2004. 304p. Tese de Doutorado. DI/PUC, Rio de Janeiro, Brasil.

CrossMDA (2007), Disponível em: http://labdist.dimap.ufrn.br/projetos/crossmda.

Graziadei, T. R (2005). Aspect-Oriented Model Weaver, 2005. 127p. Dissertação de Mestrado, Fachhochschule Vorarlberg, Dornbirn, Austria. Disponível em: http://www.sciences.univ-nantes.fr/lina/atl/bibliography/ext_GRAZIADEI05.

Jouault, F. and Kurtev, I. (2005) Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda C.; Lopes, C.; Loingtier, J.M.; and Irwin, J. (1997). Aspect-Oriented Programming. Published In ECOOP, Finland, 1997. Springer-Verlarg LNCS 1241.

Laddad, R. (2003). AspectJ in Action, Pratical Aspect-Oriented Programming. Manning Publications CO, 2003, ISBN 1930110936.

Milewski, M. and Roberts, G. (2005). The Model Weaving Description Language (MWDL) - towards a formal Aspect Oriented Language for MDA model transformations. First Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD at the 19th ECOOP.

NetBeans-MDR. (2007). Disponível em: http://mdr.netbeans.org.Acesso em 10/01/2007.

OMG (2006a). MDA Guide version 1.0.1. Formal Document: 03-06-01. Disponível em: http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf. Acesso em: 01/03/2006.

OMG (2006b). OMG Meta-Object Facility (MOF). Formal Document: 2002-04-03. Disponível em: http://www.omg.org/technology/documents/formal/mof.htm. Acesso em: 11/04/2006.

OMG (2006c) MOF QVT. Disponível em: http://www.omg.org/cgi-bin/doc?ptc/2005-11-01. Acesso em: novembro de 2006.

Rashid, A. and Moreira, A. (2006) Domain Models are NOT Aspect Free. Proceedings of MoDELS/UML. Springer, Lecture Notes in Computer Science. Volume 4199, Pages 155-169.

Reina, A.M; Torres, J. (2005). Weaving AspectJ Aspects by means of transformations. First Workshop on Models and Aspects-Handling Crosscutting Concerns in MDSD at ECOOP 2005.

Simmonds D., Solberg A., Reddy R., France R., Ghosh, S. (2005). "An Aspect Oriented Model Driven Framework", Accepted to Ninth IEEE "The Enterprise Computing Conference" (EDOC 2005), Enschede, Netherlands, 19-23 September, 2005.

Solberg, A.; Simmonds, D.; Reddy, R.; Ghosh, S.; France, R. (2005). Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development. 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1 pp. 121-126.

Suzuki, J. and Yamamotto, Y (1999). Extending UML with Aspects: Aspect Support in the Design Phase. 3rd Aspect-Oriented Programming Workshop at the 13th ECOOP. Lisbon, Portugal, 1999.

Stein, D. (2002). An Aspect-Oriented Design Model Based on AspectJ and UML. 2002. 186p. Dissertação de Mestrado, Universidade de Essen, Alemanha.

Stein, D.; Hanenberg, S.; Unland, R. (2002). Designing Aspect-Oriented Crosscutting in UML; 1st International Workshop on Aspect-Oriented Modeling with UML, AOSD 2002, Enschede, The Netherlands, April 22, 2002.

Steimann, F. (2005). Domain models are aspect free. In: MoDELS 2005, 8th International Conference on Model Driven Engineering Languages and Systems. 171185.

Tekinerdogan, B.; Moreira, A.; Araújo, J.; Clements, P. (2004). Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. Workshop Proceedings, University of Twente, TR-CTIT-04-44, 119 pp, October 2004.

Wampler, D. (2003). The Role of Aspect-Oriented Programming in OMG's Model-Driven Architecture. Disponível em: http://aspectprogramming.com/papers/AOP and MDA.pdf. Acesso em:Outubro/2005.