Um Estilo Arquitetural para Linhas de Produtos de Software

Eiji Adachi[1], Thais Batista[1], Christina Chavez[2], Uirá Kulesza[1], Alessandro Garcia[3]

[1]Departamento de Informática – Universidade Federal do Rio Grande do Norte (UFRN)

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)

[3]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

eijiadachi@gmail.com, thais@ufrnet.br, uirakulesza@gmail.com, flach@ufba.br, afgarcia@inf.puc-rio.br

Abstract. *This paper proposes an architectural style for the definition of software product lines architectures and also a strategy for the instantiation of specific products derived from the architecture. This style is defined as a seamless extension of AspectualACME, characterizing the PL-AspectualACME architectural description language, which uses the original abstractions of ACME and Armani's formal constraints in order to guarantee well-formed descriptions. Specific product line architectures can be derived from the style. As a case study, this paper presents the specialization of the architectural style for the Mobile Media product line, resulting in one product line architecture, and defines the derivation of a specific product.*

Resumo. *Esse artigo propõe um estilo arquitetural para a definição de arquiteturas de linhas de produtos de software e também uma estratégia para instanciação de produtos específicos derivados da arquitetura. Esse estilo é definido como extensão a linguagem AspectualACME, caracterizando a linguagem de descrição arquitetural PL-AspectualACME que aproveita os elementos originais de ACME e restrições formais de Armani de modo a assegurar descrições bem-formadas. Arquiteturas de linhas de produtos específicas podem ser derivadas do estilo. Como estudo de caso, o artigo apresenta a especialização do estilo arquitetural para a linha de produtos do Mobile Media, resultando em uma arquitetura de linha de produtos, e define a derivação de um produto específico a partir desta arquitetura.*

1. Introdução

Ao longo dos últimos anos, métodos e técnicas de linhas de produto de software [Clements 2001] têm sido propostos como uma estratégia técnica e organizacional para melhorar a produtividade e qualidade de produção de famílias de sistemas, a partir do reuso de artefatos comuns. Linhas de produto de software (LPSs) definem famílias de sistemas que atendem um determinado segmento de Mercado [Clements 2001]. Elas são concebidas, desde seu início, através da modelagem e implementação separadas de suas *features* comuns e variáveis. Uma *feature* [Czarnecki 2000] é uma funcionalidade ou propriedade relevante para algum interessado (*stakeholder*), e é usada para distinguir

similaridades (*commonalities*) de variabilidades (*variabilities*) da linha de produto. Durante o desenvolvimento de linhas de produto, uma das atividades centrais é a definição de uma arquitetura flexível que atenda as *features* comuns e variáveis definidas para toda a linha de produto. Tal arquitetura é então usada como referência para o projeto e implementação individual de cada um dos componentes/módulos e respectivas interações, de forma a contemplar o nível de flexibilidade e customização necessários para produzir diferentes produtos (sistemas) pertencentes à linha de produto (família de sistemas).

Diversas linguagens de descrição de arquitetura (ADLs – *architecture description languages*) têm sido propostas recentemente com o objetivo de promover uma melhor especificação de arquiteturas de LPSs [Van Ommering, R. et al. 2000] [Bashroush, R. et al. 2005] [Van der Hoek 2001] [Dashofy et al. 2001]. Através do uso de tais linguagens, é possível deixar claro como similaridades e variabilidades de uma LPS são atendidas por elementos arquiteturais (componentes, conectores, configurações). Isso pode trazer benefícios, tais como: (i) garantir o rastreamento entre artefatos de requisitos (modelo de *features*), projeto (modelos arquiteturais e de projeto) e implementação; e (ii) promover o uso de tal especificação em atividades como derivação automática de produtos e reconfiguração dinâmica da arquitetura. Entretanto, as ADLs para especificação de arquiteturas de LPSs propostas até então, possuem limitações no que diz respeito a: (i) oferecer um mapeamento claro e direto entre tipos de *features* modelados e construções de ADLs que atendem o nível de flexibilidade demandado por tais *features*; e (ii) permitir a especificação explícita de interdependências entre componentes que reflitam restrições (*constraints*) definidas no modelo de *features*.

Neste contexto, este artigo propõe um novo estilo arquitetural para definição de arquiteturas de LPSs. O estilo é definido usando o arcabouço conceitual da ADL AspectualACME, a saber, *Family*, *Type* e elementos arquiteturais como *Component*, *Connector*, *Port*, *Role* e *Representation*, sendo portanto, tratado como uma extensão a AspectualACME para a representação de arquiteturas de linhas de produtos – denominada PL-AspectualACME. O objetivo do estilo proposto é oferecer um conjunto de elementos arquiteturais capazes de modelar os diferentes tipos de variabilidades que aparecem em arquiteturas de LPS, permitindo a descrição de características e restrições comuns a LPSs. São definidos tipos de elementos arquiteturais para descrição de pontos de variação e de variantes, bem como para abstração do mecanismo de seleção das variantes. Restrições formais são incorporadas aos tipos de elementos de modo a assegurar a criação de modelos bem-formados. Arquiteturas de linhas de produtos específicas podem então ser especificadas usando o estilo arquitetural para LPS como um *template*, estendendo os tipos básicos definidos no vocabulário do estilo de modo a adequar-se as suas necessidades.

Para ilustrar a definição de uma arquitetura para LPS com o estilo arquitetural proposto, bem como a instanciação de um produto específico a partir da arquitetura obtida, a linha de produtos MobileMedia (MM) [Figueiredo et al. 2008] é utilizada. Produtos da MobileMedia permitem o gerenciamento (criar, excluir, visualizar / reproduzir, enviar) de diferentes tipos de mídia (foto, música) em dispositivos móveis. A arquitetura definida para a MM reflete as características de seu modelo de *features* e o produto MM instanciado a partir desta arquitetura dá suporte apenas à mídia foto e

inclui algumas funcionalidades opcionais. Apesar de as características comuns e variáveis do MobileMedia terem sido modeladas através de um modelo de *features*, o uso do estilo arquitetural aqui proposto é independente da estratégia adotada para representar tais características. O uso de outras estratégias para representar as características de uma linha de produtos de software não inviabiliza o uso do estilo arquitetural para linhas de produtos de software, desde que essa outra estratégia também categorize as características em obrigatórias, opcionais e alternativas.

Esse artigo está estruturado da seguinte forma. A seção 2 introduz conceitos básicos das linguagens ACME/Armani e AspectualACME. A seção 3 apresenta PL-AspectualACME e o estilo arquitetural proposto. A seção 4 apresenta a descrição da arquitetura do MobileMedia. A seção 5 trata da instanciação de produtos usando PL-AspectualACME e apresenta a instanciação de um produto específico do MobileMedia. A seção 6 apresenta alguns trabalhos relacionados. A seção 7 apresenta as conclusões.

2. Conceitos

Nesta seção, as linguagens de descrição arquitetural ACME [Garlan 1997], AspectualACME [Batista et al. 2006] e Armani [Monroe 2000] – duas extensões da linguagem ACME – são apresentadas, já que provêem o arcabouço conceitual para a definição do estilo arquitetural usado para arquitetura de linhas de produtos de software.

2.1. ACME

ACME é uma linguagem de descrição arquitetural que permite a definição de: (i) uma *estrutura arquitetural*, isto é, a organização de um sistema em suas partes constituintes; (ii) *propriedades*, que incluem anotações sobre o sistema e suas partes constituintes; (iii) *tipos e estilos*, que definem classes de elementos e estilos arquiteturais; e (iv) *restrições*, estabelecem como arquiteturas podem evoluir com o tempo.

A estrutura arquitetural é descrita através de *Components*, *Connectors*, *Systems*, *Attachments*, *Ports* e *Roles* [Garlan 1997]. Além destes elementos arquiteturais, ACME dá suporte a *Representations* e *Bindings*. *Representations* são descrições mais detalhadas de um dado elemento (*Component*, *Connector*, *Port* ou *Role*). Essa descrição pode corresponder a múltiplas visões arquiteturais alternativas de um elemento ou a uma decomposição hierárquica detalhada de um elemento. Elementos em ACME podem conter mais de um *Representation*, cada um correspondendo a uma visão conceitual diferente do elemento ou a uma forma dentre diversas formas alternativas de decompor o elemento. *Bindings* associam *Ports* nos elementos mais abstratos a *Ports* de suas representações mais detalhadas. *Properties* são triplas <*nome, tipo, valor*> que podem ser associadas a qualquer elemento ACME, exceto *Attachments*. São mecanismos de anotação para descrever informações adicionais. Além dos elementos arquiteturais básicos, ACME permite definir estilos arquiteturais através dos construtores *Type* e *Family*. Um estilo arquitetural descreve uma família de sistemas em termos de um padrão de organização estrutural, definindo um vocabulário de tipos de elementos e um conjunto de restrições que indica como os elementos são combinados [Shaw and Clements 1996]. *Type* é usado para definir um vocabulário de tipos abstratos de *Components*, *Connectors*, *Ports e Roles*; o construtor *Family* é usado para definir estilos arquiteturais em termos de tipos abstratos de elementos. No estilo arquitetural

pipes e filtros, por exemplo, os componentes são chamados de filtros e possuem, no mínimo, uma porta de entrada e uma de saída. Os conectores são chamados de *pipes* e servem de condutores unidirecionais de dados, levando as saídas de um filtro para a entrada de outro. A especificação ACME completa do estilo *pipes* e filtros, bem como de sistemas que aderem a este estilo, pode ser encontrada em [Adachi 2009].

2.2. Armani

Armani [Monroe 2000] é uma extensão de ACME que consiste em uma linguagem de predicados baseada em lógica de primeira ordem, usada para expressar restrições arquiteturais sobre elementos ACME. Restrições são definidas em termos de invariantes (*invariants)* ou heurísticas (*heuristics*). Invariantes são restrições de projeto que devem ser seguidas e heurísticas são sugestões que podem ou não ser seguidas. Armani é basicamente composta de um conjunto de funções primitivas, um conjunto de operadores, um conjunto de quantificadores e um mecanismo para definição de novas funções. As funções primitivas são usadas na definição de invariantes, heurísticas e criação de novas funções. As funções primitivas podem ser agrupadas em funções de tipos, funções de grafo, funções de propriedades e funções de conjunto. O conjunto de operadores é composto de operadores de comparação, operadores lógicos, operadores aritméticos, dentre outros. O conjunto de quantificadores possui o quantificador universal (*forall)* e o quantificador existencial (*exists)*. Finalmente, o construtor *analysis* permite que novas funções sejam definidas usando funções primitivas ou funções previamente definidas pelo usuário.

2.3. AspectualACME

AspectualACME [Batista et al. 2006] estende ACME para dar suporte à especificação modular de aspectos arquiteturais. AspectualACME define basicamente duas extensões: (i) um tipo especial de conector arquitetural, o *aspectual connector*, que encapsula interações transversais entre componentes, e (ii) um mecanismo de quantificação que simplifica sintaticamente a referência a um conjunto de pontos de junção em uma descrição arquitetural. O mecanismo de quantificação é utilizado na seção de configurações (*attachments*) através do uso de wildcards (*), que representam zero ou mais caracteres quando se referenciam nomes de componentes ou portas. Neste artigo, não usaremos os elementos arquiteturais específicos de AspectualACME, mas trataremos as extensões propostas como uma extensão a AspectualACME, já que fazem parte de uma abordagem mais ampla para descrição de famílias de sistemas que também inclui modularização de aspectos arquiteturais já explorado em [Adachi et al. 2009].

3. Um Estilo Arquitetural para Linhas de Produto

Esta seção apresenta PL-AspectualACME, uma extensão de AspectualACME que atende a modelagem de arquiteturas de linhas de produto e suas variações. A extensão proposta não adiciona novos elementos à linguagem AspectualACME, apenas enriquece semanticamente alguns elementos existentes e define um novo estilo arquitetural para linhas de produto. A Tabela 1 resume os conceitos suportados por PL-AspectualACME.

Tabela 1 – Principais conceitos de PL-AspectualACME

Conceito	Elemento Base	Semântica relacionada
Family ProductLineT	*Family (ACME)*	Estilo arquitetural para linhas de produtos de software

Component Type OptionalT	*Component Type (ACME)*	Encapsula pontos de variação opcionais
Component Type AlternativeT	*Component Type (ACME)*	Encapsula pontos de variação alternativos
Component Type InclusiveOrT	*Component Type (ACME)*	Encapsula pontos de variação *inclusive-or*
Representation	*Representation (ACME)*	Encapsula as variantes
SelectPort	*Port (ACME)*	Abstrai o mecanismo de seleção das variantes
selected_variants	*Property (ACME)*	Indica as variantes escolhidas para uma SelectPort
analysis requires	*Analysis (Armani)*	Define a restrição *feature A* requer *feature B*
analysis excludes	*Analysis (Armani)*	Define a restrição *feature A* exclui *feature B*

O construtor *Family* é usado para definir um estilo arquitetural para linhas de produtos de software, denominado de ProductLineT, que agrega vocabulário de tipos de elementos e um conjunto de restrições de projeto para linhas de produto. O vocabulário de tipos de elementos é definido através do construtor *Type* e as restrições de projeto são definidas através de predicados de lógica de primeira ordem na linguagem Armani. O estilo *ProductLineT* define uma infra-estrutura básica que serve de *template* para a definição de arquiteturas de linhas de produto específicas. Ele deve ser especializado, de modo que a arquitetura obtida incorpore *commonalities*, variabilidades e restrições de um modelo de *features* associado a linha de produtos em questão. Desse modo, assegura-se que os modelos arquiteturais obtidos a partir deste estilo serão bem formados, aderindo a tipos e restrições formais associados a linha de produtos.

No estilo arquitetural ProductLineT suportado por PL-AspectualACME, tipos de *features* são modelados como tipos de componentes. Variantes são modeladas usando-se o elemento arquitetural *Representation*. Em PL-AspectualACME, o construtor *Representation* é usado para modularizar as alternativas para se concretizar uma variabilidade específica. Desta forma, as partes em comum de uma LPS (*commonalities*) são modeladas como componentes ou como portas regulares, pontos de variação são modelados como tipos de componentes e os elementos *Representation* encapsulam as variantes. Relações de dependências entre as variantes podem ser especificadas no nível arquitetural através de predicados Armani. Dependências do tipo *A requer B*, ou ainda, *A exclui B*, são bastante comuns. O estilo arquitetural para LPS provê funções próprias para modelar esses tipos de restrições. A Figura 1 mostra a descrição do estilo arquitetural para LPS provido por PL-AspectualACME.

```
01 Family ProductLineT = {
02   Component Type ContainsOptionalT = {...}
03   Component Type AlternativeT = {...}
04   Component Type InclusiveOrT = {...}
05   Port Type SelectPort = {
06     Property selected_variants : Set {string};
07   }
08   analysis requires(a : String, b : String, p : SelectPort) : boolean =
09     contains(a, p.selected_variants) -> contains(b, p.selected_variants);
10   analysis excludes(a : String, b : String, p : SelectPort) : boolean =
11     contains(a, p.selected_variants) -> !contains(b, p.selected_variants);
12 }
```

Figura 1. Descrição do estilo para linhas de produtos

O estilo arquitetural *ProductLineT* define um componente e três tipos de componentes. Os três tipos de componentes são *ContainsOptionalT* (linha 02), *AlternativeT* (linha 03) e *InclusiveOrT* (linha 04), que modelam os pontos de variação dos tipos opcional, alternativo ou *inclusive-or*, respectivamente. Pontos de variação opcionais representam *features* que possuem *subfeatures* opcionais. Pontos de variação alternativos encapsulam conjuntos de variantes dentre as quais uma e somente uma é

selecionada. Pontos de variação *inclusive-or* encapsulam conjuntos de variantes dentre as quais no mínimo uma é selecionada. O tipo de porta *SelectPort* (linhas 05-07) abstrai o mecanismo responsável pela seleção das variantes para um produto específico e sua propriedade *selected_variants* (linha 06) indica as variantes escolhidas. A função *requires* (linhas 08-09) é usada para modelar a relação de dependência do tipo *a requer b*. Ela aceita como parâmetros duas strings e uma *SelectPort* e verifica que, se a primeira string pertence ao conjunto de variantes selecionadas pela *SelectPort*, então obrigatoriamente, a segunda string também deve pertencer. A função *excludes* (linhas 10-11) é usada para modelar a relação de dependência do tipo *a exclui b*. Ela aceita como parâmetros duas strings e uma *SelectPort* e verifica que, se a primeira string pertence ao conjunto de variantes selecionadas pela *SelectPort*, então obrigatoriamente, a segunda string não deve pertencer a este mesmo conjunto.

```
01 Component Type InclusiveOrT = {
02   Property variants : Set{String};
03   Port select-inclusive-or : SelectPort = new SelectPort;
04   design invariant
05     isSubset(self.select-inclusive-or.selected_variants, self.variants);
06   design invariant size(self.select-inclusive-or.selected_variants) >= 1;
07   design invariant forall s : String in self. variants |
08      exists r : Representation in self.REPRESENTATIONS | name(r) == s;
09 }
```

Figura 2. Descrição de pontos de variação *inclusive-or* em PL-AspectualACME

A Figura 2 mostra a descrição de pontos de variação *inclusive-or* em PL-AspectualACME. O tipo de componente *InclusiveOrT* (linhas 01-09) modela pontos de variação do tipo *inclusive-or*. Neste caso, a porta *select-inclusive-or* (linha 03) do tipo *SelectPort* abstrai o mecanismo de seleção das variantes *inclusive-or*. Em seguida, definem-se três predicados Armani que definem as restrições comuns a pontos de variação do tipo *inclusive-or*. O primeiro invariante (linhas 04-05) define que a propriedade *selected_variants* deve ser subconjunto próprio da propriedade *variants*. O segundo invariante (linha 06) define que o tamanho da propriedade *selected_variants* da porta *select-inclusive-or* deve ser no mínimo igual a 1, i.e., no mínimo uma variante dentre as disponíveis deve ser selecionada. O último invariante (linhas 07-08) define que para toda string da propriedade *variants* deve existir uma representação cujo nome é igual a esta string. Ou seja, para cada variante definida como disponível na propriedade *variants* deve existir um respectivo elemento *Representation*, que encapsula a variante.

```
01 Component Type ContainsOptionalT = {
02   Property variants : Set{String};
03   Port select-optional = : SelectPort = new SelectPort;
04   design invariant isSubset(self.select-optional.selected_variants, self.variants);
05   design invariant forall s : String in self.variants |
06      exists r : Representation in self.REPRESENTATIONS | name(r) == s;
07 }
```

Figura 3. Descrição de pontos de variação opcional em PL-AspectualACME

A Figura 3 ilustra a descrição textual em PL-AspectualACME para pontos de variação do tipo opcional. O componente do tipo *ContainsOptionalT* (linhas 01-07) modela pontos de variação opcionais. Neste caso, a porta *select-optional* (linha 03) abstrai o mecanismo de seleção das opções. Os dois predicados Armani (linhas 04-06) definem as restrições relativas a pontos de variação opcionais. O primeiro invariante (linha 04) indica que o conjunto de strings atribuído à propriedade *selected_variants* deve ser subconjunto próprio do conjunto atribuído à propriedade *variants*. O segundo invariante (linhas 05-06) indica que para cada string da propriedade *variants* deve existir uma representação cujo nome é igual a esta string.

4. Arquitetura do MobileMedia

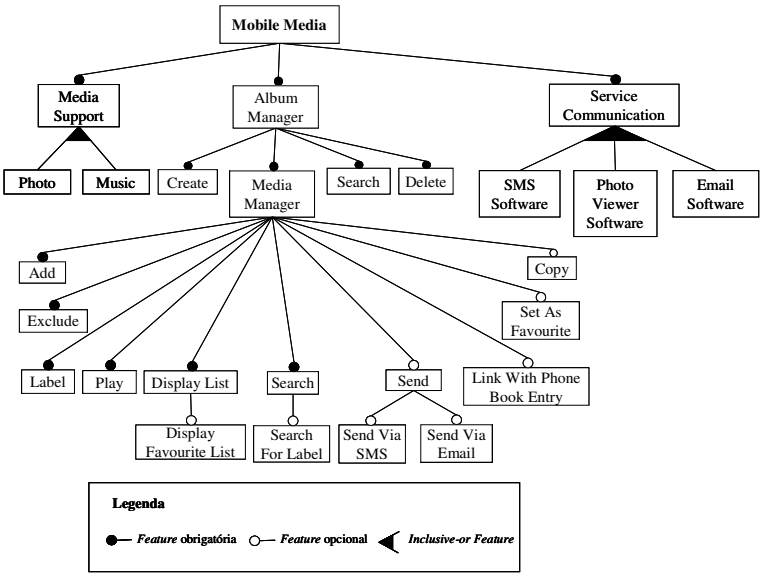

Figura 4. Modelo de *Features* do Mobile Media

Esta seção apresenta a descrição da arquitetura da linha de produtos MobileMedia (MM) obtida a partir de um modelo de *features* (seção 4.1). A arquitetura da linha de produtos MM, chamada de MobileMediaT (seção 4.2) é descrita como especialização do estilo arquitetural *ProductLineT*. A descrição da arquitetura da MM é bem formada, pois adere aos tipos e às restrições definidos pelo estilo ProductLineT, e servirá de modelo base para a instanciação de diversos produtos. Na seção 4.3 são discutidas algumas questões relacionadas ao estilo proposto.

4.1. Modelo de *features* do MobileMedia

O MobileMedia inclui diversas *features* obrigatórias, opcionais e alternativas. Há diferentes produtos derivados da MobileMedia, implementados em Java, AspectJ e CaesarJ. Cada implementação tem 10 versões e cada nova versão contém funcionalidades adicionais em relação à versão anterior. A Figura 4 ilustra o modelo de *features* da versão 6 da linha de produtos MobileMedia. Este modelo contém *features* obrigatórias, opcionais e *inclusive-or*.

4.2. Especificação da arquitetura do MM

A especificação usada nesse trabalho é relativa à versão 6 do MM. Todas as *commonalities* e variabilidades desta versão estão presentes no modelo de *features* da Figura 4. Por questão de simplicidade e limitação de espaço, algumas *features* não serão mostradas nos exemplos a seguir. A descrição textual completa do MM em PL-AspectualACME pode ser encontrada em [Adachi 2009].

A Figura 5 mostra a especificação da arquitetura do MobileMedia. São definidos os tipos de componentes *MediaSupportT* (linha 02), *AlbumManagerT* (linha 03), e *MediaManagerT* (linha 04) que dão suporte a diferentes mídias, gerenciamento de álbuns e, gerenciamento de mídias, respectivamente.

```
01 Family MobileMediaPL extends ProductLineT with {
02   Component Type MediaSupportT extends InclusiveOrT with {...}
```

```
03   Component Type AlbumManagerT = {...}
04   Component Type MediaManagerT extends ContainsOptionalT with {...}
05   // Component Type para a feature Service Communication não será mostrado.
06 }
```

Figura 5. Descrição da arquitetura do MobileMedia

A Figura 6 mostra a especificação do ponto de variação *inclusive-or* relacionado ao suporte do MM a diferentes tipos de mídia. Na versão 6 são suportadas as mídias música e foto. Para descrever o ponto de variação relacionado ao suporte de mídias, é definido o tipo de componente *MediaSupportT* (linhas 01-16), o qual estende o tipo de componente *InclusiveOrT*. Atribui-se à propriedade *variants*, o conjunto de strings *{"Music", "Photo"}*, indicando que as variantes *Music* e *Photo* são disponibilizadas. Os elementos *Representation Music* (linhas 03-09) e *Photo* (linhas 10-16) encapsulam as variantes através dos subsistemas *MusicSupport* (linhas 04-07) e *PhotoSupport* (linhas 11-14). O componente *Music* (linha 05-07) modela a variante relacionada ao suporte a música e oferece seu serviço através da porta *play* (linha 06). O componente *Photo* (linhas 12-14) modela a variante relacionada ao suporte a foto e oferece seu serviço através da porta *display* (linha 13). Nas seções de *bindings* (linha 08 e linha 15), acopla-se a porta *select-inclusive-or* às portas *play* do componente *Music* e *display* do componente *Photo*. A porta *select-inclusive-or*, apenas abstrai o mecanismo de seleção das variantes. As portas *play* e *display* são as que, de fato, provêem serviços. A propriedade *variants* (linha 02) define o conjunto de variantes que podem ser escolhidas. A definição de quais variantes são escolhidas ocorre no momento da instanciação (Seção 5).

```
01 Component Type MediaSupportT extends InclusiveOrT with {
02   Property variants = {"Music","Photo"};
03   Representation Music = {
04     System MusicSupport = {
05       Component Music  = {
06         Port play; }
07     }
08     Bindings { select-inclusive-or to Music.play; }
09   }
10   Representation Photo = {
11     System PhotoSupport = {
12       Component Photo  = {
13         Port display; }
14     }
15     Bindings { select-inclusive-or to Photo.display; }
16   } ... }
```

Figura 6 – Descrição do tipo MediaSupportT

O componente do tipo *AlbumManagerT* (Figura 7) modela a *feature* obrigatória do MM relacionada ao gerenciamento de álbuns de mídias. As três *subfeatures* obrigatórias do gerenciamento de álbuns são modeladas pelas portas *CreateAlbum* (linha 02), *ExcludeAlbum* (linha 03) e *SearchAlbum* (linha 04). O elemento *Representation MediaManager* (linhas 05-08) é usado para representar o subcomponente *MediaManager* (linha 07) do tipo *MediaManagerT*, o qual é responsável por modelar a *feature* obrigatória do MM relacionada ao gerenciamento de mídias.

```
01 Component Type AlbumManagerT = {
02   Port CreateAlbum;
03   Port ExcludeAlbum;
04   Port SearchAlbum;
05   Representation MediaManager = {
06     System MediaManagerSys : MobileMedia_SPL = new MobileMedia extended with {
07       Component MediaManager : MediaManagerT = new MediaManagerT;
08 } } }
```

Figura 7 – Descrição do tipo AlbumManagerT

O componente do tipo *MediaManagerT* (Figura 8) atende a *feature* obrigatória do MM relacionada ao gerenciamento de mídias. Suas *subfeatures* obrigatórias são modeladas pelas portas *AddMedia* (linha 02), *ExcludeMedia* (linha 03) e *DisplayMediaList* (linha 04). A propriedade *variants* (linha 06) define as opções que são providas pelo ponto de variação. Os elementos *Representation SetAsFavouriteMedia* (linhas 07-10) e *DisplayFavouriteMediaList* (linhas 11-14) encapsulam as opções providas. O predicado Armani (linhas 16-17) define uma restrição de projeto que indica que a seleção da opção *DisplayFavouriteMediaList* requer a seleção da opção *SetAsFavouriteMedia*.

```
01 Component Type MediaManagerT extends ContainsOptionalT with {
02   Port AddMedia;
03   Port ExcludeMedia;
04   Port DisplayMediaList;
05   ...
06   Property variants = {"SetAsFavouriteMedia", "DisplayFavouriteMediaList"};
07   Representation SetAsFavouriteMedia = {
08     System SetAsFavouriteMedia = {
09       Component SetAsFavouriteMedia = {...};
10   } } ...
11   Representation DisplayFavouriteMediaList = {
12     System DisplayFavouriteMediaList = {
13       Component DisplayyFavouriteMediaList = {...};
14   } } ...
15 } ...
16   design invariant requires("DisplayFavouriteMediaList", "SetAsFavouriteMedia",
17     self.select-optional);
18 }
```

Figura 8 – Descrição do tipo MediaManagerT

4.3. Discussões

O estilo arquitetural para linhas de produtos de software *ProductLineT* disponibilizado por PL-AspectualACME descreve um conjunto de elementos comuns a LPS, servindo de *template* para que linhas de produtos específicas sejam modeladas. Ele provê um vocabulário de tipos e um conjunto de restrições de projeto próprios da descrição de linhas de produto. São disponibilizados tipos de componentes para modelar pontos de variação opcionais, *inclusive-or* e alternativos e também um tipo de porta especial, que abstrai o mecanismo de seleção das variantes. Predicados de lógica de primeira ordem na linguagem Armani são incorporados aos tipos de componente para assegurar formalmente que os modelos derivados sejam bem formados e que a semântica de cada tipo de ponto de variação seja respeitada. Em um ponto de variação alternativo, por exemplo, há um predicado para assegurar que apenas uma variante será selecionada. O estilo *ProductLineT* também disponibiliza as funções Armani *requires* e *excludes* para modelar no nível arquitetural relações de dependência entre *features* dos tipos *feature A requer feature B,* e *feature A exclui feature B*, respectivamente.

Arquiteturas de LPSs podem então ser descritas através da combinação do uso de: (i) uma *Family* que estende o estilo *ProductLineT*, para descrever uma dada arquitetura de LPS; (ii) extensões aos tipos *AlternativeT, InclusiveOrT* e *ContainsOptionalT* para descrever os pontos de variação da LPS; (iii) elementos *Representation* para encapsular as variantes dos pontos de variação; e (iv) propriedades e a *SelectPort* para abstrair o mecanismo de seleção. Desta forma, arquitetos de software estão aptos a modelar arquiteturas de linhas de produto sem a necessidade de acrescentar novas abstrações à linguagem. Além disso, os modelos obtidos a partir do estilo *ProductLineT* de PL-AspectualACME serão bem formados, visto que seguem obrigatoriamente as restrições formais de projeto definidas por este. Entretanto,

predicados Armani são limitados a quantificações sobre propriedades de elementos que, por sua vez, só podem ser definidas em termos dos tipos primitivos float, integer, boolean e string. Desta forma, os predicados que asseguram as restrições de projeto do estilo *ProductLineT* são quantificados sobre propriedades definidas como conjuntos de *strings*, o que os torna dependentes de especificações textuais sintáticas bastante específicas. Extensões às linguagens PL-AspectualACME e Armani para permitir a definição de propriedades em termos de elementos arquiteturais de primeira ordem, poderiam tornar os mecanismos de restrições mais robustos.

5. Instanciação de Produtos em PL-AspectualACME

Essa seção descreve como um produto pode ser instanciado a partir de uma arquitetura especificada em PL-AspectualACME (5.1) e, mais especificamente, apresenta a instanciação de um produto a partir da arquitetura do MobileMedia (seção 5.2).

5.1. Instanciação em PL-AspectualACME

Todos os membros de uma linha de produtos de software compartilham um conjunto mínimo de *features* comuns, as *commonalities*. O que difere um produto do outro é o conjunto de *features* variáveis que cada um possui. O processo de instanciação (ou derivação) baseia-se basicamente na escolha de quais *features* variáveis serão disponibilizadas em determinado produto, considerando também a interdependência entre eles.

Em PL-AspectualACME, as variabilidades de uma LPS são modularizadas através de *Representation*. Uma porta do tipo *SelectPort* abstrai o mecanismo de seleção no nível arquitetural, mas não o implementa de fato. Usam-se anotações nos elementos *Property* para especificar quais elementos *Representation* serão selecionados ou não, considerando uma determinada descrição, ficando a cargo de uma ferramenta específica interpretar as informações contidas nestas anotações e, de fato, acrescentar ou retirar os elementos *Representation* da descrição arquitetural do produto.

O processo de instanciação proposto consiste na instanciação de membros da família de sistemas definida pela arquitetura associada à linha de produto. Durante a instanciação de um sistema membro da família, deve-se: (i) instanciar elementos arquiteturais (componentes, etc.) a partir dos tipos definidos pela arquitetura referenciada LPS, a fim de se construir o sistema em questão; e (ii) definir quais variantes dos tipos serão selecionadas. Esta definição é feita através do uso da propriedade *selected_variants* de portas do tipo *SelectPort*. Durante a instanciação dos elementos, atribuem-se a essa propriedade as variantes que devem ser selecionadas de fato. Portanto, a propriedade *selected_variants* indica os elementos *Representation* que devem ser selecionados para um produto específico.

5.2. Instanciação de um produto MobileMedia

A Figura 9 mostra a descrição da instanciação de um produto específico da LPS MobileMedia. O sistema *Product* é uma instância da família de sistemas *MobileMediaPL*. *Product* possui o componente *AlbumManager* (linhas 02-10), instância do tipo *AlbumManagerT*, que, por sua vez, possui o sub-componente *MediaManager* (linhas 05-10). Acessando os sub-elementos de *MediaManager*, atribui-se à propriedade

selected_variants da porta *Select-Optional* quais variantes devem ser selecionadas: *SetAsFavouriteMedia* e *DisplayFavouriteMediaList*. Na instanciação do componente *MediaSupport*, instância do tipo *MediaSupportT*, atribui-se o valor *{"Photo"}* à propriedade *selected_variants* da porta *Select-Inclusive-Or*. Em outras palavras, o produto instanciado *Product* possui suporte à mídia foto e possui as funcionalidades opcionais *SetAsFavouriteMedia* e *DisplayFavouriteMediaList.* A Figura 10 mostra esquematicamente o processo de instanciação para este exemplo.

```
01 System Product : MobileMediaPL = new MobileMediaPL extended with {
02   Component AlbumManager : AlbumManagerT = new AlbumManagerT extended with {
03     Representation MediaManager = {
04       System MediaManagerSys : MobileMediaPL = new MobileMediaPL extended with {
05         Component MediaManager = {
06           Port select-optional = {
07             Property selected_variants =
08               {"SetAsFavouriteMedia","DisplayFavouriteMediaList"};
09           }
10   } } } } ...
11   Component MediaSupport : MediaSupportT = new MediaSupportT extended with {
12     Port select-inclusive-or  = {
13       Property selected_variants = {"Photo"};
14   } } ... }
```

Figura 9 – Descrição de instanciação de um produto com PL-AspectualACME

6. Trabalhos Relacionados

Nesta seção, PL-AspectualACME é comparada com outras ADLs segundo 5 dimensões: domínio representado, representação de elementos alternativos, representação de elementos opcionais, suporte a versões, suporte a definição de restrições e relacionamento do modelo arquitetural com modelo de *features*. Os resultados estão sintetizados na Tabela 2.

6.1. ADLs para linhas de produtos de software

Algumas das ADLs para LPS encontradas na literatura foram desenvolvidas para domínios específicos. Koala [Van Ommering, R. et al. 2000] é direcionada para modelagem de sistemas embutidos e ADLARS [Bashroush, R. et al. 2005] para modelagem de sistemas em tempo real. Mae [Van der Hoek 2001], xADL 2.0 [Dashofy et al. 2001] e PL-AspectualACME são ADLs de propósito geral. Todas possuem mecanismos para definição de vocabulários de tipos de elementos arquiteturais. Também é comum a todas elas a possibilidade de representar pontos de variação nos modelos arquiteturais, através da representação de elementos alternativos ou opcionais.

Elementos alternativos são representados por novos tipos de abstrações nas linguagens xADL 2.0 e Mae. Em xADL 2.0, os tipos dos elementos regulares são definidos através do schema XML *Structure & Type,* enquanto os tipos dos elementos alternativos são definidos no schema *Variants*. Em Mae, componentes regulares são definidos através do construtor *Component Type*, enquanto os componentes alternativos são definidos através do construtor *Variant Component Type*. Em PL-AspectualACME,

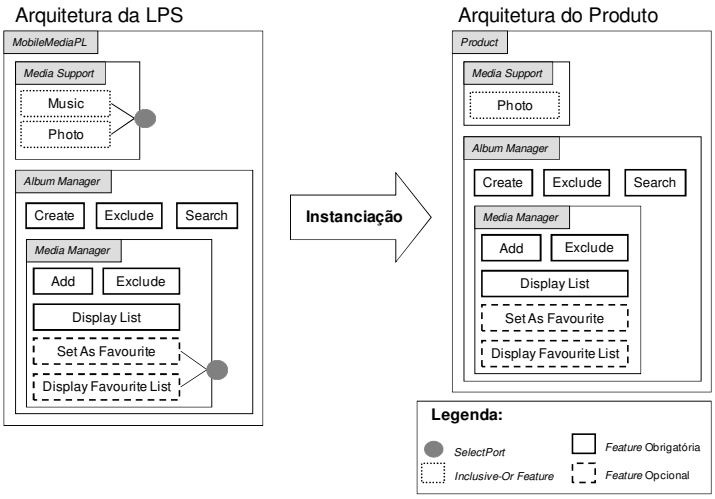

Figura 10 – Ilustração esquemática da instanciação

componentes regulares são definidos através do construtor *Component* e elementos alternativos e *inclusive-or* são definidos através de extensões aos tipos de componente *AlternativeT e InclusiveOrT*, respectivamente. As variantes disponibilizadas por cada ponto de variação são encapsuladas por elementos *Representation*. Já as linguagens Koala e ADLARS não fazem distinção entre seus elementos regulares e elementos alternativos. Elementos opcionais são representados por um novo tipo de abstração apenas na linguagem xADL 2.0. Em xADL 2.0, os tipos dos elementos regulares são definidos através do schema XML *Structure & Type,* enquanto os tipos dos elementos opcionais são definidos no schema *Options*. Em PL-AspectualACME, os componentes opcionais são definidos através de extensões ao tipo *ContainsOptionalT*, ficando encapsulados nos elementos *Representation*. Nas demais ADLs não há distinção entre componentes regulares e componentes opcionais.

O mecanismo de seleção das variabilidades também difere entre as ADLs. As linguagens xADL 2.0, Mae e ADLARS anotam seus elementos alternativos e opcionais com expressões condicionais que são avaliadas no momento da instanciação. Caso estas expressões sejam atendidas, os elementos opcionais são selecionados, ou, no caso de elementos alternativos, uma variante própria é selecionada. Koala modela a seleção de seus elementos alternativos e opcionais através de um construtor específico da linguagem, o switch. O construtor *switch* implementa um padrão de projeto responsável por selecionar, no momento da instanciação, quais elementos, dentre os que ele se relaciona, devem ser selecionados. Em PL-AspectualACME, o mecanismo de seleção é modelado pela S*electPort*. A S*electPort* é um tipo de porta especial responsável por se conectar aos pontos de variação, definindo em sua propriedaed *selected_variants*, no momento da instanciação, quais elementos opcionais e quais variantes dos elementos alternativos devem ser selecionados.

xADL 2.0, Mae e Koala suportam a captura da evolução de seus modelos arquiteturais. xADL 2.0 representa as informações de versões dos elementos arquiteturais em um schema XML – *versions* – no qual as versões são organizadas em forma de um grafo. Já Mae representa as versões de seus elementos diretamente na definição dos tipos abstratos de elementos, definindo uma árvore de versões para cada tipo abstrato. Koala possui repositórios de tipos de interfaces e componentes, que são

gerenciados por sistemas padrão de gerenciamento de versão. ADLARS e PL-AspectualACME não possuem mecanismos próprios para suportar a evolução de seus modelos arquiteturais. Finalmente, ADLARS propõe o relacionamento entre o modelo de *features* e o modelo arquitetural. Modelos arquiteturais descritos em ADLARS são anotados com informações sobre o modelo de *features*, relacionando explicitamente, no nível arquitetural, elementos arquiteturais com *features* do modelo de *features* através de expressões condicionais. No entanto, não há mecanismos formais para assegurar que restrições entre *features* são respeitadas. PL-AspectualACME é a única ADL que incorpora a suas descrições uma notação formal para definir restrições de projeto. Predicados de lógica de primeira ordem em Armani permitem restrições sobre propriedades dos elementos, assegurando a correção dos modelos arquiteturais. Desta forma, o relacionamento entre o modelo de *features* e o modelo arquitetural de uma LPS pode ser representado através de anotações nos elementos *Property*. Os predicados Armani asseguram que as mesmas restrições que existem entre *features* no modelo de *features* sejam respeitadas no nível arquitetural. Os resultados das comparações entre as ADLs estão sintetizados na Tabela 2.

Tabela 2. Comparação entre ADLs

	Koala	***xADL 2.0***	***Mae***	***ADLARS***	***PL-AspectualACME***
Domínio dos sistemas	Sistemas embutidos	Propósito geral	Propósito geral	Sistemas de tempo real	Propósito geral
Representação de elementos alternativos	Mesma de elementos regulares	Schema XML *Variants*	Construtor *Variant Component Type*	Mesma de elementos regulares	Extensões ao tipo de componente *AlternativeT* e *InclusiveOrT*
Representação de elementos opcionais	Mesma de elementos regulares	Schema XML *Options*	Mesma de elementos regulares	Mesma de elementos regulares	Extensão ao tipo de componente *OptionalT*
Suporte a versões	Sim	Sim	Sim	Não	Não
Suporte a definição de restrições	Não	Não	Não	Não	Sim
Relacionamento com modelos de features	Não	Não	Não	Sim	Sim

7. Conclusões

Esse artigo apresentou um estilo arquitetural para descrição de linhas de produtos de software que compõe PL-AspectualACME – uma linguagem de descrição arquitetural orientada a aspectos para linha de produtos de software. PL-AspectualACME aproveita as abstrações arquiteturais de AspectualACME/Armani, estendendo algumas delas para permitir tanto a definição da arquitetura da linha de produtos quanto da instanciação de um produto específico. PL-AspectualACME define poucas extensões às abstrações existentes em ACME/Armani, de forma a evitar introduzir complexidade nas descrições arquiteturais. O uso de portas *select* juntamente com o conceito de *property,* que define as variantes selecionadas, aliadas a predicados Armani que definem restrições para a seleção das variantes de acordo com o tipo da variabilidade (*alternative*, *incluse-or* e *optional*), forma um poderoso mecanismo para se definir, na arquitetura de linhas de produtos, como poderão ser instanciados os produtos derivados da arquitetura. Para exemplificar a aplicabilidade de PL-AspectualACME, a linguagem foi usada na definição da arquitetura para a versão 6 da linha de produto MobileMedia e instanciação de um produto específico. Como trabalhos futuros planejamos aplicar PL-AspectualACME em outros estudos de caso mais complexos e que utilizem outras

estratégias de modelagem de características de linhas de produtos de software. Além disso, planejamos disponibilizar a ferramenta PL-AspectualACME Studio, uma extensão de ACME Studio. Finalmente, pretendemos também comparar a abordagem apresentada neste artigo, com outras estratégias de instanciação [Adachi et al. 2009] de especificações PL-AspectualACME baseadas em VML4Arch [Loughran et al. 2008].

Referências

Adachi, E. (2009) http://www.dimap.ufrn.br/plaspectualacme

Adachi, E. et al. (2009) "Variability Management in Aspect-Oriented Architecture Description Languages: An Integrated Approach." a ser publicado no SBES 2009, Fortaleza-CE.

Bashroush, R. et al. (2005) "ADLARS: An Architecture Description Language for Software Product Lines." Proceedings of the 2005 Annual IEEE/NASA Software Engineering Workshop (SEW).

Batista, T. et al. (2006) "Reflections on Architectural Connection: Seven Issues on Aspects and ADLs." Workshop on Early Aspects - Aspect-Oriented Requirements Engineering and Architecture Design, ICSE'06, Shanghai, China, May, p. 178-187.

Clements, P ; Northrop, L. (2001). Software product lines: practices and patterns. Addison-Wesley.

Czarnecki, K., Eisenecker, U. (2000) "Generative Programming: Methods, Tools, and Applications", Addison-Wesley.

Dashofy, E. M., van der Hoek, A. Taylor, R. N. (2001) "A highly-extensible, XML-based architecture description language". Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), Amsterdam, Aug., p. 103-112.

Figueiredo, et al (2008) "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability". Proceedings of the 30th International Conference on Software Engineering (ICSE'08) , Leipzig.

Garlan, D., Monroe, R., Wile, D. (1997) "ACME: An Architecture Description Interchange Language". Proc. of the CASCON '97, Toronto, Nov., p. 169-183.

Loughran, et al. (2008) "Language Support for Managing Variability in Architectural Models". Proc. of the 7th Intl. Symposium on Software Composition, SC2008. Berlin: Springer Berlin / Heidelberg, v. 4954., p. 36-51.

Monroe, R. (2000) "Capturing Software Architecture Design Expertise With Armani", CMU School of Computer Science Technical Report CMU-CS-98-163.

Shaw, M. and Clements, P. (1996) "Toward boxology: preliminary classification of architectural styles." In Joint Proceedings of the Second international Software Architecture Workshop (ISAW-2) Workshops

Van der Hoek, A. et al. (2001) "Taming Architectural Evolution" in Proceedings of the Joint 8th European Software Engineering Conference.

Van Ommering, R. et al. (2000) The Koala Component Model for Product Families in Consumer Electronics Software. IEEE Computer, 2000. 33(2): pp. 78-85.