Requisitos e Arquitetura de Software Orientada a Aspectos: Uma Integração Sinérgica

Ana Luisa Medeiros[1], Lyrene F. Silva[2], Thais Batista[1], Leonardo Minora[3]

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

[2] Universidade Estadual do Rio Grande do Norte (UERN)

[3] Centro Federal de Educação Tecnológica do Rio Grande do Norte (CEFET-RN)

analuisafdm@gmail.com, lyrene@gmail.com, thais@ufrnet.br, minora@cefetrn.br

Resumo. *A integração entre diferentes fases de desenvolvimento de software é essencial para garantir a correspondência entre os modelos e artefatos produzidos em cada uma das fases. Esse artigo apresenta uma integração sinérgica entre um modelo de requisitos orientado a aspectos, AOV-graph, e uma linguagem de descrição arquitetural orientada a aspectos, AspectualACME. A integração sinérgica é representada por um conjunto de regras de mapeamento bidirecional, que mapeia modelos em AOV-Graph para AspectualACME e vice-versa. As regras de mapeamento são avaliadas através de um estudo de caso tradicionalmente usado para avaliar estratégias e linguagens orientadas a aspectos, o Health Watcher System.*

Abstract. *The integration between different software engineering phases is essential to guarantee the correspondences between models and artifacts produced by each phase. This paper presents a synergic integration between an aspect-oriented requirement model, AOV-graph, and an aspect-oriented architectural description language, AspectualACME. The synergic integration is represented by a set of bi-directional mapping rules, mapping from AOV-Graph to AspectualACME and vice-versa. The mapping rules are evaluated through a case study commonly used to assess aspect-oriented strategies and languages, the Health Watcher System.*

1. Introdução

Engenharia de requisitos e arquitetura de software são duas atividades iniciais no processo de desenvolvimento de software. O mapeamento entre modelos de requisitos e especificações arquiteturais e vice-versa é necessário para que as decisões tomadas em ambas às atividades de modelagem sejam propagadas de uma para a outra, bem como para outras atividades do ciclo de vida do software. Com isto facilita-se o desenvolvimento de software, pois: (i) diminui-se o *gap* entre estas etapas de desenvolvimento e entre os modelos construídos nelas; (ii) facilita-se o rastreamento de mudanças tanto em requisitos quanto na arquitetura; (iii) diminui-se o retrabalho de modelagem e modificação; (iv) facilita-se o acesso às informações relativas às atividades do processo de desenvolvimento; dentre outros.

Silva (2007) define um mapeamento denominado simbiôntico entre requisitos e arquitetura. A idéia do mapeamento simbiôntico é a possibilidade de considerar que requisitos e arquitetura sejam artefatos associados a atividades que se beneficiam uma da outra, de forma que seja possível propagar mudanças entre estes artefatos e navegar de uma fase para outra. Neste artigo estamos acrescentando ao mapeamento simbiôntico o conceito de *sinergia*, que significa que além deles se beneficiarem um do outro para atingir suas metas específicas, há uma ação sinérgica entre eles fazendo com que haja possibilidade de mapeamento bidirecional e o efeito resultante dessa sinergia seja maior que a soma de seus efeitos individuais.

A primeira parte deste mapeamento sinérgico, o mapeamento de requisitos para arquitetura, está definida em Silva (2007) e contém as regras de mapeamento entre um modelo de requisitos orientado a aspectos, AOV-Graph definido por Silva (2006), e uma linguagem de descrição arquitetural orientada a aspectos, AspectualACME definida por Batista (2006). Esse artigo define as regras de mapeamento de AspectualACME para AOV-graph, mostra a aplicação das regras em um estudo de caso tradicionalmente usado em sistemas orientados a aspectos, o sistema *HealthWatcher* (HW), e apresenta uma análise qualitativa dos resultados obtidos. HealthWatcher, definido por Soares (2002), é um sistema de informações disponibilizado via WEB que oferece aos usuários serviços relacionados ao sistema público de saúde, como o suporte ao registro de reclamações e guia de saúde para o usuário.

AOV-graph e AspectualACME são estratégias orientadas a aspectos representativas de requisitos e arquitetura, respectivamente. V-graph é um tipo de modelo de metas que provê representações para requisitos funcionais (*goals* e *tasks*) e não funcionais (*softgoals*) bem como captura múltiplas formas de relacionamentos entre eles. Modelos de metas explicitam a complexidade inerente desses relacionamentos e como alguns conceitos entrecortam outros. ACME é uma ADL de propósito geral que fornece um conjunto de elementos básicos e representativos de ADLs e um flexível mecanismo de anotação usado para incluir descrição adicional que os elementos básicos não conseguem expressar.

Apesar de AOV-Graph e AspectualACME terem sido desenvolvidas independentemente, elas compartilham algumas características: (i) qualquer elemento (requisito ou componente) pode entrecortar qualquer outro elemento, transversal ou não-transversal; (ii) ambas são estratégicas simétricas orientadas a aspectos e, consequentemente, elementos transversais ou não transversais são representados pela mesma abstração; (iii) não há uma abstração exclusiva para aspectos, sendo a distinção entre conceitos transversais estabelecida com base na maneira como eles interagem (ou compõem) com o restante do sistema; e (iv) há um relacionamento especial ou conector que estende as linguagens de modelagem como forma de dar suporte à composição de conceitos que estão espalhados e entrelaçados pelo sistema. Apesar das similaridades, a integração sinérgica entre AOV-graph e AspectualACME não é trivial pois, apesar de haver intersecções sintáticas e semânticas entre seus elementos estruturais, há elementos e informações particulares em cada uma destas linguagens.

Esse artigo está estruturado da seguinte forma. A Seção 2 descreve, brevemente, AOV-Graph e AspectualACME. A Seção 3 define as regras de mapeamento nos dois sentidos, de requisitos para arquitetura e de arquitetura para requisitos e apresenta uma

análise qualitativa que valida o mapeamento proposto. O sistema *HealthWatcher* é utilizado como estudo de caso, e os exemplos aqui apresentados foram criados com o apoio da ferramenta MARISA que operacionaliza as regras de mapeamento, disponibilizada por Medeiros (2007). A Seção 4 contém uma comparação com trabalhos relacionados. A Seção 5 apresenta os comentários finais e trabalhos futuros.

2. Conceitos Básicos

2.1 AOV-graph

AOV-graph, definida por Silva (2006), é uma extensão do modelo de metas V-graph, definido em Yu (2004). Esta extensão adiciona à linguagem V-graph um novo tipo de relacionamento denominado relacionamento transversal (*crosscutting relationship*). Além disso, AOV-graph consiste de metas (*goals*), softmetas (*softgoals*) e tarefas (*tasks*). Metas representam objetivos ou estados concretos a serem alcançados pelo sistema; softmetas representam objetivos abstratos, freqüentemente associados aos requisitos não-funcionais; e tarefas representam ações (ou requisitos funcionais) a serem realizadas como meio para atingir metas e softmetas. A linguagem AOV-graph também contém os seguintes relacionamentos: contribuições (*contributions*) que representam decomposição e correlações (*correlations*) que representam associações positivas ou negativas entre metas e softmetas, Figura 1(a).

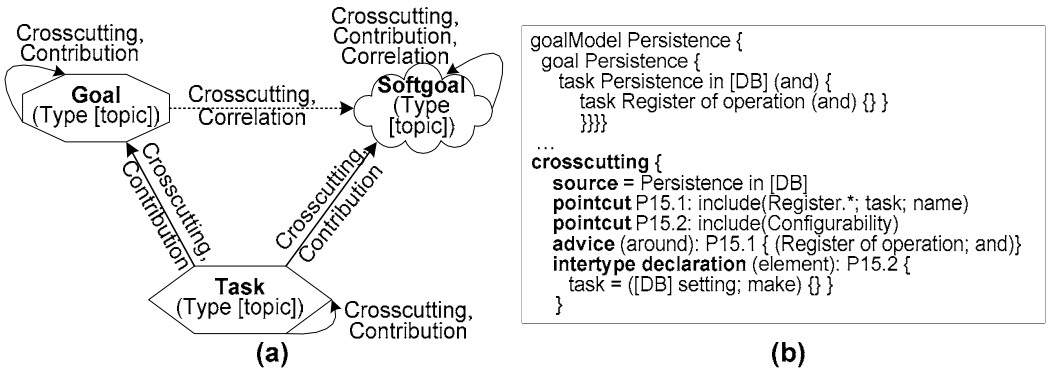


Figura 1. (a) Relacionamentos entre metas, softmetas e tarefas e (b) exemplo de relacionamento transversal em AOV-graph

O relacionamento transversal foi criado com base em elementos tradicionais de AOSD (Aspect-oriented Software Development) propostos por AspectJ, são eles: *pointcut*, *advice* e *intertype declaration*. Em AOV-graph, tais elementos possuem uma semântica diferenciada e apropriada para modularizar características transversais no nível de requisitos. Como mostrado na Figura 1(b), o relacionamento transversal é descrito por: (i) *source* - indica o ponto de origem do relacionamento; (ii) *pointcut* – indica os pontos de destino do relacionamento, os pontos que são afetados por uma característica transversal; (iii) *advice* - indica elementos pertencentes ao ponto de origem que estão transversais aos pontos de destino, a característica transversal em si. No AOV-graph, os tipos *before*, *around* e *after* podem indicar, respectivamente, pré-condição, decomposição e pós-condição; e (iv) *intertype declaration* – define novos tipos de elementos, também transversais, que surgem com a integração das características indicadas nos pontos de origem e destino do relacionamento transversal.

Há dois tipos de *intertype declaration*: *element* – usado quando os novos tipos referem-se a instâncias de elementos de tipos existentes no modelo de requisitos, no caso de AOV-graph referem-se a metas, softmetas e tarefas; e *attribute* – quando os novos tipos referem-se a atributos inexistentes no modelo de requisitos, tais como os atributos *ator* e *custo* no modelo AOV-graph.

Na Figura 1(b), exemplificamos o uso do relacionamento transversal. O ponto de origem do relacionamento transversal é *Persistence in [DB]* e os pontos afetados são *Configurability* e *Register.**. Através do *advice* e *intertype declaration*, indicamos que *Register of operation* e *[DB] setting* afetam todas as tarefas iniciadas pela *string Register* e *Configurability*, respectivamente. O uso do *intertype declaration* indica que *DB setting* é um elemento criado devido à junção dos conceitos *Persistência* e *Configurabilidade*.

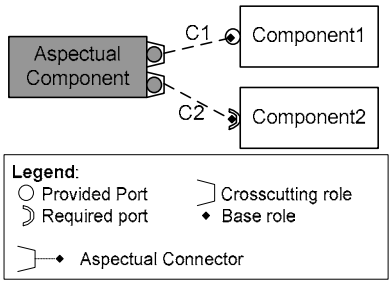
2.2 AspectualACME

AspectualACME, definida por Batista (2006), é uma extensão da Linguagem de Descrição de Arquitetura (ADL) ACME. Esta extensão adiciona um novo tipo de conector, chamado conector aspectual (*aspectual connector*), à linguagem ACME. Além disso, AspectualACME consiste de: componentes (*components*), conectores (*connectors*) e configurações (*attachments*). Componentes e conectores possuem interfaces associadas. A interface de um componente é o conjunto de pontos de interação entre o componente e o mundo externo. Tal interface especifica os serviços (mensagens, operações e variáveis) que um componente fornece e também os serviços requisitados de outros componentes e são comumente chamadas de portas. Conectores especificam regras que governam a interação entre componentes. A interface do conector especifica os pontos de interação entre o conector e os componentes ligados a ele. *Configurações* definem a conexão de componentes com conectores.

O conector aspectual define uma nova interface que distingue os papéis dos elementos internos do conector na interação transversal. Uma das partes está ligada ao componente afetado (denominado componente base) por outro componente (denominado componente aspectual). A interface do conector aspectual contém: (i) uma *base role*, (ii) uma *crosscutting role,* e (iii) uma cláusula *glue*. A Figura 2 contém a especificação textual e gráfica do Conector Aspectual.

```
System Example = {
Component aspectualComponent = { Port aPort }
Connector aConnector = {
  baseRole aBaseRole;
  crosscuttingRole aCrosscuttingRole;
  glue glueType;
  }
Attachments {
  aspectualComponent.aPort to aConnector.aCrosscuttingRole;
  aConnector.aBaseRole to *.prefix* } }
```

(a) Notação Textual



(b) Notação Gráfica

Figura 2. Conector aspectual

A Figura 2(b) mostra um exemplo de composição entre um componente (aspectual) com dois outros componentes. C1 e C2 são exemplos de conectores aspectuais. É importante ressaltar que não há uma abstração distinta para representar

"aspectos" arquiteturais. Estes são representados por componentes. As cores diferentes na Figura 2 são usadas apenas para diferenciar qual componente está fazendo o papel de componente aspectual na interação transversal.

Uma *base role* deve ser conectada à porta do componente afetado pela interação transversal – componente base - e uma *crosscutting role* deve ser conectada à porta do componente aspectual. O par *base role--crosscutting role* não impõe a restrição, comum em ADLs, de conectar portas de entrada (*provided ports*) e portas de saída (*required ports*). A *crosscutting role* define o local no qual o componente aspectual conecta-se ao conector. Na Figura 2(b), o conector aspectual C1 conecta a porta de entrada do componente aspectual à porta de entrada do componente 1. A cláusula *glue* especifica os detalhes da conexão aspectual como, por exemplo, em que momento a interação aspectual acontece – *after, before, around*. O conector aspectual provê um mecanismo de quantificação que simplifica, sintaticamente, a referência a conjuntos de pontos de junção em uma descrição arquitetural. O mecanismo de quantificação é usado na configuração através do uso de *wildcards* (*) que representam parte do nome de componentes ou portas. A Figura 2(a) ilustra um exemplo em AspectualACME onde o componente *aspectualComponent* é conectado, via o conector aspectual *aConnector*, à todos os componentes que contenham uma porta com nome começando com *prefix*.

3. Integração Sinérgica

A integração sinérgica entre AOV-graph e AspectualACME significa que elementos destas duas linguagens estão associados de forma que é possível propagar mudanças entre eles rapidamente e navegar do modelo de requisitos para a arquitetura, e vice-versa. Este mapeamento bidirecional é possível devido a algumas similaridades entre essas duas linguagens: ambas focam na modelagem de características transversais e propõem modelos de propósito geral, baseados em algumas extensões simétricas da orientação a aspectos. Em ambas linguagens, considera-se que o espalhamento e entrelaçamento de características transversais são representados pela maneira com que essas características interagem umas com as outras. Tais informações sobre a interação estão nos relacionamentos transversais, no caso de AOV-graph, e nos conectores aspectuais e *attachments*, no caso de AspectualACME. Assim, cada relacionamento transversal é mapeado para conectores aspectuais e attachments, e vice-versa.

Além disso, os conceitos do sistema são representados em AOV-graph por *softmetas, metas* e *tarefas,* e em AspectualACME por *componentes* e *portas*. O mapeamento entre estes elementos é possível porque há uma certa correspondência semântica entre *softmetas/metas/tarefas* e *componentes/portas*: todos eles podem ser vistos como serviços a serem providos ou requeridos pelo sistema ou partes dele. Assim, embora softmetas, metas e tarefas estejam em diferentes níveis de abstração, eles são mapeados para serviços providos ou requeridos na arquitetura de software. Por outro lado, componentes representam serviços providos pelo sistema e portas representam serviços providos ou requeridos pelos componentes para alcançarem seus objetivos. Apresentamos os detalhes destes mapeamentos nas próximas seções: a Seção 3.1 resume o mapeamento de modelos em AOV-graph para descrições em AspectualACME (veja Silva (2007) para informações mais detalhadas); a Seção 3.2 apresenta as regras de mapeamento de AspectualACME para AOV-graph; e a Seção 3.3 relata a análise realizada nos resultados obtidos com esta integração.

3.1. Mapeamento de AOV-Graph para AspectualACME

Em resumo, o processo de mapeamento de AOV-graph para AspectualACME ocorre em três etapas:

(i) Mapeamento da hierarquia de metas, softmetas e tarefas de AOV-graph para componentes, portas e representações (representations) de AspectualACME - Cada elemento (meta/*softmeta*/tarefa) de AOV-graph é mapeado para um componente ou porta de AspectualACME. Essa decisão é baseada na posição em que cada meta/*softmeta*/tarefa está na hieraquia: elementos que são raízes da árvore de metas são mapeados para componentes; subelementos que não são folhas (na árvore AOV-graph) são mapeados para componentes em *representações, e* subelementos que são folhas são mapeados para portas e são inseridos no componente pai correspondente. Para complementar, quando um componente e uma porta são gerados algumas propriedades (*properties)* também são inseridas para registrar a origem de cada um deles. Essas propriedades registram algumas informações sobre requisitos no modelo arquitetural e possibilitam o mapeamento inverso – de AspectualACME para AOV-graph. Por exemplo, na Figura 3(a) é descrito o modelo de metas *Reliability com a* meta *Handling Exception* (linhas 2 a 4). Sua hierarquia de metas e tarefas é mapeada para o componente *Exception_handling,* e para as portas *Detection_of_exception e Selection_of_Dealer* (linhas 1-3 da Figura 3(b)) seguindo o processo de mapeamento de metas, *soft*metas e tarefas do AOV-graph para componentes ou portas do AspectualACME.

(a) AOV-graph

```
1  goal_model Reliability {
2  goal Exception handling  (and) {
3    task Detection of [exception] (and) {}
4    task Selection of Dealer (and) {} }
5  crosscutting {
6    source = Detection of [exception]
7    pointcut PC5.3: include(Persistence in [DB])
8    pointcut PC5.4: include(Authentication by Login)
9    pointcut PC5.5: include(Availability)
10 intertype declaration (element): PC5.3 {  task Detection of [persistence exception] (and) {} }
11 intertype declaration (element): PC5.4 { task Detection of [authentication exception] (and) {} }
12 intertype declaration (element): PC5.5 { task Detection of [availability exception] (and) {} } } }
```

(b) AspectualACME

```
1  Component Exception_handling={
2  Port Detection_of_exception;
3  Port Selection_of_dealer;
4  Representation{
5    System details ={
6      Component Detection_of_exception = {
7        Port self;
8        Port Detection_of_persistence_exception;
9        Port Detection_of_authentication_exception;
10       Port Detection_of_availability_exception;    };}
11   Bindings {Detection_of_exception to Detection_of_exception.self};}
12 Connector Detection_of_exceptionCon1, Detection_of_exceptionCon2, Detection_of_exceptionCon3 ={
13   baseRole sink
14   crosscuttingRole source
15   glue source around sink
16   properties{comesFrom: intertype declaration (element); crossRelSource = Detection_of_exception;};};
17 attachment Detection_of_exception.Detection_of_persistence_exception to Detection_of_exceptionCon1.source;
18 attachment Detection_of_exceptionCon1.sink to Persistence.Persistence_in_DB;
19 attachment Detection_of_exception.Detection_of_authentication_exception to     Detection_of_exceptionCon2.source;
20 attachment Detection_of_exceptionCon2.sink to Authentication.Authentication_by_login;
21 attachment Detection_of_exception.Detection_of_availability_exception to Detection_of_exceptionCon3.source;
22 attachment Detection_of_exceptionCon3.sink to Security.Availability;
```

Figura 3. Exemplo do mapeamento de especificação em (a) AOV-graph para (b) AspectualACME

(ii) Mapeamento de *pointcuts* e *intertype-declarations (element)* de AOV-graph para portas e ligações (*bindings)* de AspectualACME - Alguns conceitos nos

relacionamentos tranversais também podem ser mapeados para portas: (i) cada elemento do corpo de uma *intertype declaration;* (ii) cada elemento do corpo de um *advice*; e (iii) cada elemento de um *pointcut.* Semanticamente, relacionamentos transversais agrupam diversos relacionamentos entre metas*,* tarefas *e soft*metas; eles representam as interações que, na arquitetura, são representadas por conectores ligando portas. Por exemplo, na Figura 3(a) observamos que há um relacionamento transversal cuja origem (*source*) é *Detection_of_exception* e há três *intertype declarations.* Como os elementos definidos no corpo das *intertype declarations* são elementos novos, então realiza-se o processo de mapeamento de *pointcuts e intertype declarations (element)* para portas e ligações de AspectualACME. Dessa forma, *Detection_of_exception* é mapeada para um subcomponente de *Exception_handling* e os elementos do corpo das *intertype declarations* são mapeados para portas desse subcomponente (linhas 4 a 10). Além disso, são criadas uma porta *self* dentro do mesmo subcomponente e uma ligação da porta *Detection_of_exception* para essa porta *self* (linha 11 da Figura 3(b)). Da mesma forma, cada elemento dos *pointcuts* é mapeado para uma porta nos componentes gerados pelos seus elementos pais, por exemplo: *Persistence in [DB]* (no primeiro *pointcut*) é mapeado para uma porta do componente *Persistence,* e assim sucessivamente.

(iii) Mapeamento de relacionamentos transversais para conectores e configurações (*attachments)* - Conectores e configurações *também são* gerados a partir do relacionamento transversal: (i) *intertype declarations* e *advice* são mapeados para conectores aspectuais, sendo que o tipo da cláusula *glue* é determinado pelo tipo do advice (podendo ser around, after ou before), ou pelo tipo de intertype declaration (sendo around se a intertype for do tipo *element* e caso contrário assumirá o valor definido pelo atributo criado); (ii) cada relação *intertype declaration* → *pointcut* e *advice* → *pointcut* é mapeada para uma associação de configuração de elementos, do *intertype declaration* para uma *crosscutting role,* e uma outra associação, de *base role* para elementos dos *pointcuts*. Por exemplo, na Figura 3(b) as duas primeiras configurações (attachments) estão associando *Detection_of_exception.Detection_of_persistence_exception to Detection_of_exceptio nCon1.source e Detection_of_exceptionCon1.sin to Persistence.Persistene_in_DB* (linhas 17 a 18).

Tabela 1. Mapeamento de AOV-graph para AspectualACME

AOV-graph		AspectualACME
Goal, sofgoal, task	→	Component ou port + property elementType
Contribution	→	Representation + property contributions
Correlation	→	Property correlation
Topic	→	Property topic
Advice	→	Aspectual connector + properties comesFrom e crossRelSource
Intertype declaration	→	Aspectual connector + properties comesFrom e crossRelSource
Advice type	→	glueType
Intertype declaration do tipo element	→	glueType = around
Intertype declaration do tipo attribute	→	glueType = valor do atributo
Source (crosscutting relationship)	→	Attachment
Pointcut	→	Attachment, port self + binding
Property (elemento criado para registrar informações da arquitetura)	→	Property

A Tabela 1 resume como os elementos de AOV-graph são transformados nos elementos de AspectualACME. Com essas regras de mapeamento é possível gerar especificações em AspectualACME a partir de descrições em AOV-graph.

3.2. Mapeamento de AspectualACME para AOV-Graph

No mapeamento de AspectualACME para AOV-graph precisamos considerar duas situações: (i) a arquitetura em AspectualACME foi gerada com base em um modelo AOV-graph, desta forma há informações da modelagem de requisitos na especificação da arquitetura, através das propriedades; (ii) a arquitetura foi gerada independentemente de um modelo de requisitos em AOV-graph, assim, não há informações sobre o modelo de metas. Isto pode ocorrer, por exemplo, na engenharia reversa ou quando outro modelo de requisitos deu apoio à modelagem da arquitetura.

Na primeira situação as informações registradas em propriedades (*properties*) guiam o mapeamento, dando apoio às decisões acerca do: tipo de elemento, se é softmeta, meta ou tarefa; como os conceitos estão transversais, se via *advice* ou *intertype declaration*; correlações existentes no modelo de metas; e rótulos das contribuições e correlações. A Tabela 2 resume o mapeamento de AspectualACME para *AOV-graph.* Com estas regras é possível gerar modelos em AOV-graph a partir de especificações em AspectualACME.

Tabela 2. Mapeamento AspectualACME para AOV-graph

AspectualACME		AOV-graph
Component, port	→	Goal ou sofgoal ou task + property
Representation	→	Contributions
Aspectual connector	→	Advice, intertype declaration + property
glueType	→	AdviceType, intertypeDeclarationType
Attachment	→	Pointcut, source, contribution (se não está associado à um conector aspectual)
Connector	→	Contribution + Property
Property topic	→	Topic
Property elementType	→	Determina se o elemento é uma meta, softmeta ou tarefa
Property correlations	→	Correlations
Property comesFrom	→	Determina se os conceitos transversais devem ser descritos em advice ou intertype declaration
Property crossRelSource	→	Auxilia na identificação da origem do relacionamento transversal
Property	→	Property

Para a segunda situação, visto que as informações citadas acima não estão disponíveis então algumas observações precisam ser feitas: (i) a hierarquia de componentes e portas deve ser mapeada para uma hierarquia de metas, softmetas e tarefas, mas não há como determinar precisamente qual o tipo de cada elemento, assim, se a propriedade *elementType* não estiver disponível, portas são mapeadas para tarefas e componentes são mapeados para softmetas; (ii) o rótulo das contribuições entre metas, softmetas e tarefas não podem ser identificados precisamente, mas como determinado no próprio AOV-graph, o rótulo *and* é tido como *default*; (iii) o tipo de *glue* contido nos conectores aspectuais determina se aquele conector é mapeado para um advice, uma intertype declaration do tipo *attribute* ou do tipo *element* – intertype declarations do tipo *attribute* são geradas por *glue* que são diferentes de *around, before* e *after*, advice dos tipos *before* e *after* também podem ser diretamente identificados, mas advice do tipo *around* e intertype declarations do tipo *element* não podem ser distinguidos, pois possuem o *glue* do mesmo tipo. Assim, determinamos que estes geram advice ao invés

de intertype declarations já que os primeiros foram mais utilizados em nossos estudos de caso do que intertype declarations; e (iv) as correlações do AOV-graph não podem ser geradas, visto que elas representam associações abstratas entre softmetas e metas.

Além disso, assim como no mapeamento de AOV-graph para AspectualACME, em que as informações específicas deste modelo de requisitos são registradas em um elemento de AspectualACME (nas propriedades), é necessário que AOV-graph seja capaz de registrar informações arquiteturais especificadas em AspectualACME. Por exemplo, informações tais como aquelas relativas ao estilo arquitetural utilizado, ou o tipo de conexão existente entre os componentes. Assim, adicionamos um novo elemento à linguagem AOV-graph denominado *property*, que é idêntico ao elemento de mesmo nome em AspectualACME. Estas propriedades podem estar associadas ao *modelo de metas* (goal_model), *softmeta*, *meta*, *tarefa*, *correlações* e *contribuições*. Dado que tais informações arquiteturais são registradas em propriedades de AspectualACME então os valores de tais propriedades em AOV-graph são determinados pelos próprios valores registrados em AspectualACME (veja a última linha das Tabelas 1 e 2).

O processo que mapeia especificações em AspectualACME para modelos em AOV-graph consiste das seguintes etapas:

(i) Mapeamento de conectores aspectuais e *attachments* para relacionamentos transversais - cada conector aspectual é mapeado para um relacionamento transversal com uma *intertype declaration* ou *advice* em AOV-graph. O tipo da cláusula *glue* é verificado e usado para especificar o tipo deste *advice* e *intertype declaration,* enquanto seus *pointcuts* e corpo são determinados pelos *attachments*. Para cada *attachment* que possui o mesmo conector ligando portas e papéis: (i) portas relacionadas com o *crosscuttingRole*, são mapeadas para o corpo do *intertype declaration* ou *advice* (dependendo do tipo da cláusula *glue*); (ii) o componente que possui esta porta é mapeado para a origem do relacionamento transversal no AOV-graph; (iii) enquanto portas ligadas ao *BaseRole* são mapeadas para *pointcuts*. No final do mapeamento é necessário reunir todos os relacionamentos transversais que possuem a mesma origem, bem como todos os pontos que são atingidos pelo mesmo *intertype declaration* ou *advice*. Por exemplo, na Figura 4, podemos observar que o conector *CryptographyCon* e os *attachements* em que ele faz parte geram o relacionamento transversal cuja origem é *Cryptography*.

(ii) Mapeamento de componentes, portas, representações *(representations)* e propriedades para metas, tarefas ou softmetas de AOV-graph – (i) A partir das propriedades é possível identificar precisamente qual o tipo do elemento derivado de componentes ou portas, através da propriedade *elementType* e através da propriedade *topics* identificamos quais *strings* do nome da meta, softmeta ou tarefa deve ser posta entre colchetes. (ii) Quando informações sobre o AOV-graph não estão disponíveis – a hierarquia de componentes e portas gera uma hierarquia de softmetas e tarefas, sendo as tarefas derivadas de portas e as softmetas derivadas de componentes. Entretanto, é necessário analisar se cada um destes componentes e portas faz ou não parte do corpo de alguma *intertype declaration*, pois se eles fizerem, então eles não são mapeados para softmeats e tarefas, eles existirão apenas no corpo da intertype declaration. Na Figura 4(a), a hierarquia de componentes, portas e representações é mapeada para a hierarquia de metas, e tarefas da Figura 4(b), neste caso as propriedades *elementType* dos

componentes são consideradas para a determinação do tipo de componente em AOV-graph.

```
System Security = { …
 Component Cryptography = {
   properties = {
     elementType=goal;
     correlations=[(Security (help);]}
   port Assymmetric_cryptography = {};
   port Symmetric_cryptography = {};
   representation {
     system details = {
      component Cryptography_operation_maker ={
        properties = {elementType=task;}
        port Encrypter_data={};
        port Decrypter_data={};};};};};
 Connector CryptographyCon={
    baseRole sink
    crosscuttingRole source
    glue source around sink };
  attachment Cryptography.Symmetric_cryptography
      to CryptographyCon.source;
  attachment CryptographyCon.sink to
      Authentication.Authentication_by_Login;
```

(a) AspectualACME

```
Goal_model Security = {
…
 goal Cryptography = {
   task Assymmetric_cryptography (or) = {};
   task Symmetric_cryptography (or) = {};
   task cryptograph_operation_maker (and) ={
      task Encrypter_data (and) ={};
      task Decrypter_data (and) ={};};};};
 Correlation = {
    source = Cryptography;
    target = Security;}
 Crosscutting ={
    source = Cryptography;
    pointcut P1: include (Authentication_by_Login);
    advice (around): P1 {(Symmetric_cryptography)};}
};
```

(b) AOV-graph

Figura 4. Exemplo de mapeamento de AspectualACME para AOV-graph

(iii) Mapeamento da propriedade *correlation* de AspectualACME para *correlations* de AOV-graph – Em AspectualACME as correlações da modelagem em AOV-graph são especificadas em propriedades. Por exemplo, na Figura 4(a), o componente Cryptography possui a propriedade *correlations=[(Security (help));]*, indicando que este componente está associado ao conceito segurança. Esta propriedade é mapeada para uma correlação de *Cryptography* para *Security* na Figura 4(b).

(iv) Mapeamento de conectores de AspectualACME para contribuições de AOV-graph – conectores não-aspectuais de AspectualACME representam relacionamentos entre portas que requerem serviços e portas que provêem serviços. Em AOV-graph tais relacionamentos são representados por contribuições, assim um serviço provido contribui para duas ou mais softmetas, metas e tarefas. Tendo em vista a maneira como AOV-graph foi implementado, a tarefa derivada da porta provedora é distinta da tarefa derivada da porta requeredora, porque a primeira é mapeada para uma tarefa de fato, enquanto a segunda é derivada para uma referência à primeira (task_ref). A Figura 5 ilustra uma parte da arquitetura do HealthWatcher apresentada por Batista (2006). Neste exemplo, o conector C3 e os *attachements* associados são mapeados para a hierarquia de metas e tarefas da Figura 5(b).

```
Component Distribution = {
  Port RemoteUpdateStateControl
  Port RemoteAccess
  Port UseBusinessServices }
Component Business = {
  Port BusinessService
  Port SaveInfo
  Port RestoreInfo }
Connector C3 = {
  Roles caller, callee }
Attachements {
   Distribution.UseBusicessServices to C3.caller
   C3.callee to Business.BusinessService}
```

(a) AspectualACME

```
Softgoal Distribution = {
  task RemoteUpdateStateControl (and);
  task RemoteAccess (and);
  task UseBusinessServices (and);
  task_ref BusinessService (and){
     property {comesFrom=connector (C3);};};};
}
Softgoal Business = {
  task BusinessService (and);
  task SaveInfo (and);
  task RestoreInfo (and);}
```

(b) AOV-graph

Figura 5. Exemplo de mapeamento de conectores de AspectualACME para contribuições de AOV-graph

3.3. Análise dos Resultados Obtidos

A integração sinérgica propõe que as atividades referentes à modelagem de requisitos e arquitetura atuem cooperativamente, de modo que o efeito resultante seja maior que a soma dos efeitos individuais destas. Dessa forma, tanto os artefatos de requisitos quanto a arquitetura desenvolvida beneficiam-se desta ação integrada, como mostramos nas Seções 3.1 e 3.2. Ambos os modelos provêem elementos estruturais que registram informações além daquelas providas por sua decomposição dominante, por exemplo, os modelos AOV-graph podem conter informações arquiteturais registradas em *properties*, tanto quanto o modelo AspectualACME registra informações de requisitos.

Nesta seção relatamos nossa experiência com o mapeamento de AOV-graph para AspectualACME e vice-versa, mostrando as vantagens obtidas, as dificuldades e os desafios identificados. Analisamos o estudo de caso realizado com o sistema *HealthWatcher*, avaliando qualitativamente os seguintes parâmetros: rastreabilidade, modularidade, corretude e completude, veja as Seções 3.3.1, 3.3.2 e 3.3.3.

3.3.1 Avaliação da arquitetura gerada a partir da modelagem em AOV-graph

(1) Corretude – os elementos de AspectualACME foram adequadamente modelados? Componentes e portas são gerados a partir de softmetas, metas e tarefas enquanto conectores são gerados a partir dos relacionamentos de contribuição do AOV-graph. Assim, geramos para cada conceito uma hierarquia de componentes e portas que representam os serviços a serem providos pelo sistema em desenvolvimento. Por outro lado, embora as correlações do AOV-graph sejam registradas como propriedades de AspectualACME, elas não relacionam os componentes e portas associados por este tipo de relacionamento. Assim, alguns dos conceitos do *HealthWatcher* são representados por componentes não conectados na arquitetura. Este é o caso do conceito de Segurança, Criptografia (*Cryptography*) e Autenticação (*Authentication*), que estão correlacionados à *Security* (no AOV-graph), mas não há nenhum conector ou hierarquia entre eles (veja a propriedade *correlations* do componente *Cryptography* na Figura 6(b)). Um outro problema neste exemplo é que o componente *Security* possui apenas uma porta cujo nome é *Availability*. Como *Availability* é um conceito abstrato e não uma funcionalidade então o componente *Security* não representa um serviço que realmente possa ser requerido e provido na arquitetura.

```
Goal_model Security = {
 softgoal Security ={
   softgoal Availability ={}}
 goal Authentication = {...}
 goal Cryptography = {
  task Assymmetric_cryptography (or) = {};
  task Symmetric_cryptography (or) = {};
  task cryptograph_operation_maker (and) ={
    task Encrypter_data (and) ={};
    task Decrypter_data (and) ={};};};};
 Correlation (help) = {
   source = Cryptography;
   target = Security;}
 Correlation (help) = {
   source = Authentication;
   target = Security;}
```

(a) AOV-graph

```
System Security = { …
 Component Security = {
   port Availability;}
 Component Authentication = {
   properties = {correlations=[(Security (help));]}…}
 Component Cryptography = {
  properties = {
    elementType=goal;
    correlations=[(Security (help));]}
  port Assymmetric_cryptography = {};
  port Symmetric_cryptography = {};
  representation {
    system details = {
     component Cryptography_operation_maker ={
       port Encrypter_data={};
       port Decrypter_data={};};};};};
```

(b) AspectualACME

Figura 6. Exemplo de mapeamento de correlações

Um outro problema, causado pela ausência de um relacionamento entre elementos correlacionados em AOV-graph, é o conceito *Performance*. No *HealthWatcher* há duas metas correlacionadas à *Performance*, são elas: "*Response time must not exceed five seconds*" e "*Capability to handle twenty simultaneous users*", veja a Figura 7(a) e (b). Neste caso surgem dois problemas: o primeiro é que estes três elementos geram três componentes desconectados na arquitetura, e o segundo problema é que os dois últimos não representam serviços providos, mas propriedades.

Para o exemplo de Security propomos acrescentar uma restrição às regras de mapeamento: qualquer meta e softmeta é mapeada para um componente ou porta (dependendo da hierarquia) desde que: 1) haja tarefas que operacionalize-as - existam tarefas que contribuam para aquela meta e softmeta, diretamente ou indiretamente; ou 2) ela seja origem de um relacionamento transversal. Com isto, evita-se que componentes inteiramente abstratos sejam gerados na arquitetura.

Para o exemplo de *Performance* percebemos, através da arquitetura, que seria mais adequado modelá-la no AOV-graph utilizando um relacionamento transversal como mostrado na Figura 7(c) e (d). Assim, *Performance* gera um componente que está conectado, via conectores aspectuais, à todas os componentes/portas cujos nomes sejam *Complaint manager* e *Information divulger* ou iniciam com a string *Display*. No primeiro caso, o conector aspectual tem, em sua clausula *glue,* a restrição "*constraint: response time must not exceed 5 seconds*"; no segundo caso, ele possui a restrição "*constraint: Capability to handle 20 simultaneous users*".

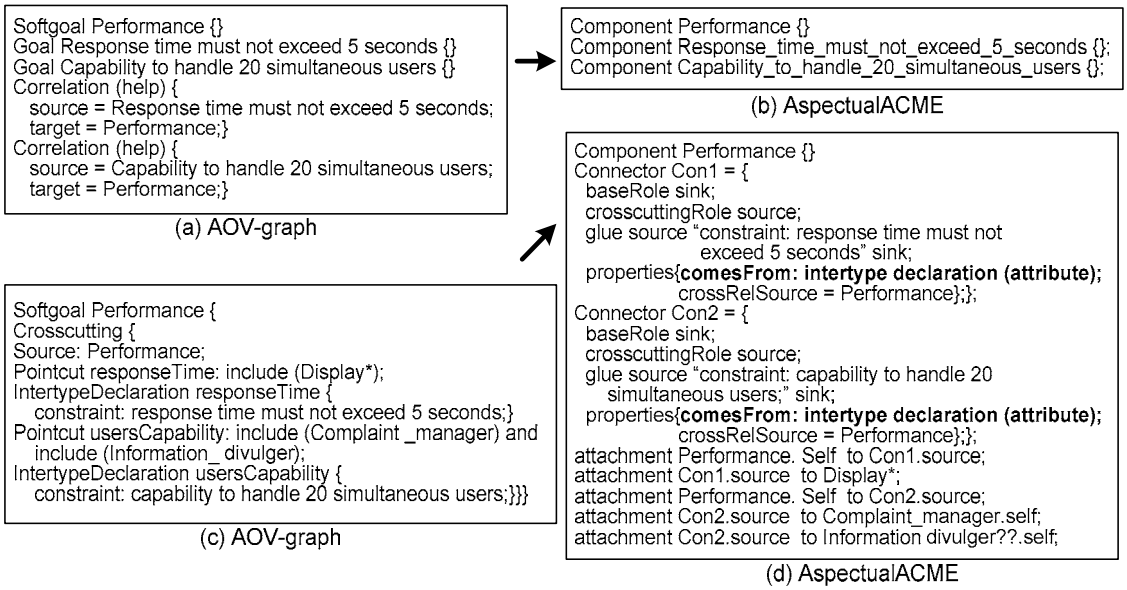


Figura 7. Exemplo do mapeamento de especificação em (a) AspectualACME para (b) AOV-graph

(2) Completude – há elementos em AspectualACME que não tenham sido modelados? O mapeamento aproveita os principais elementos estruturais de AspectualACME, que são *components, ports, connectors, attachments, bindings, representations* e *properties*. Além disso, semanticamente, é possível obter uma estruturação inicial da arquitetura que é valiosa por trazer as informações levantadas nas atividades de requisitos. Porém, informações arquiteturais tais como plataforma a ser utilizada, estilo arquitetural mais

adequado, protocolos de conexão, dentre outras, são informações que não estão presentes na arquitetura, pois geralmente os requisitos não as descrevem.

(3) Rastreabilidade: a arquitetura em AspectualACME possui elementos que representem todos os requisitos modelados e suas características?; *Rationale* – é possível identificar quais elementos nos requisitos deram origem aos elementos arquiteturais? Como podemos observar na Seção 3.1, todos os elementos estruturais de AOV-graph são mapeados para algum elemento estrutural de AspectualACME. Dessa forma, há continuidade das informações definidas em AOV-graph, pois elas continuam existindo em AspectualACME. É possível, por exemplo, identificar quais componentes estão correlacionados à uma certa softmeta a partir da arquitetura gerada, bem como saber se esta correlação é positiva ou negativa ou saber se determinados componentes e portas foram gerados de softmetas, metas ou tarefas. Podemos observar, por exemplo, na Figura 6, que embora os componentes *Authentication* e *Cryptography* sejam componentes desconectados, eles são derivados de elementos correlacionados à softmeta de segurança (*Security*). Esta informação pode ser útil ao arquiteto para que ele possa mais facilmente identificar como alguns conceitos devem ser modularizados na arquitetura.

(4) Modularidade: Coesão - como os conceitos levantados nos requisitos estão modularizados na arquitetura?; Acoplamento – como componentes da arquitetura estão conectados para atender os conceitos levantados nos requisitos? Nosso estudo de caso, o *HealthWatcher*, possui sete conceitos: *Complaint manager*, *Information divulger, Security, Persistence, Performance, Usability e Exception handling*. Estes sete conceitos foram identificados e modelados em AOV-graph separadamente e assim, eles geram componentes distintos na arquitetura. Além disso, a conexão gerada entre os componentes se dá somente entre portas provedoras, não sendo necessário que um determinado componente requisite os serviços intrusivos (isto ocorre apenas quando há referências para elementos, como citamos no item 4 da Seção 3.2). Dessa forma há um menor acoplamento entre os componentes. Por outro lado, em nosso estudo de caso, apenas os conectores aspectuais foram gerados.

3.3.2 Avaliação do modelo de requisitos gerado a partir da modelagem em AspectualACME

Chamemos de Situação1 aquela que se refere ao mapeamento quando há informações do AOVgraph nas propriedades (*properties*) de AspectualACME, e Situação2 aquela que se refere ao mapeamento quando tais informações não estão disponíveis.

(1) Corretude – os elementos de AOV-graph foram adequadamente modelados? Considerando a Situação1, todos os elementos de AOV-graph são diretamente recuperados. Porém, na Situação2, não é possível determinar com exatidão quais componentes são mapeados para metas, softmetas ou tarefas. Pela restrição que adicionamos anteriormente é possível saber apenas que todas as portas são mapeadas para tarefas. Assim, estabelecemos que componentes e portas que não são mapeadas diretamente para tarefas são mapeadas para softmetas (é necessário que o arquiteto analise a semântica de cada um deles). Quanto ao relacionamento transversal, apenas não é possível determinar quais elementos são transversais via *advice* ou *intertype*

declaration (do tipo *element*) - como estabelecemos que todos eles são mapeados para *advice*, então *instertype declarations* não são geradas.

(2) Completude – há elementos em AOV-graph que não tenham sido modelados? Na Situação1 todos os elementos estruturais de AOV-graph são gerados. Na Situação2 correlações não podem ser geradas, nem *intertype declarations*, como citado anteriormente.

(3) Rastreabilidade: o modelo de requisitos em AOV-graph possui elementos que representem todos os elementos arquiteturais modelados e suas características? É possível identificar quais elementos na arquitetura deram origem aos elementos no modelo AOV-graph? Um novo atributo, denominado *property,* foi adicionado aos elementos de AOV-graph (softmeta, meta, tarefa, correlação, contribuição, crosscutting). Com isto, na Situação1 e Situação2 todas as informações específicas da arquitetura são mapeadas para propriedades em AOV-graph, tal como fizemos de AOV-graph para AspectualACME. Dentre tais informações é possível saber quais os componentes ou portas geram uma determinada meta, ou se uma certa contribuição foi gerada por um conector não-aspectual, por exemplo.

(4) Modularidade – como os conceitos levantados na arquitetura estão modularizados nos requisitos? Nos estudos de caso realizados tanto na Situação1 quanto na Situação2, os conceitos estão modularizados em componentes distintos e faz-se o uso de conectores aspectuais para representar o entrelaçamento entre tais componentes (serviços). Cada um destes componentes gera árvores separadas no AOV-graph e a transversalidade de serviços é representada por relacionamentos transversais. Entretanto, quando conectores não-aspectuais são usados, é possível gerar metas, softmetas e tarefas com altos *fan-out*, isto significa que determinadas metas, softmetas e tarefas contribuem para muitos elementos, representando acoplamento entre eles. Isto pode ser um indicativo que algum serviço foi mal modularizado na arquitetura.

3.3.3 Avaliação da sinergia entre requisitos e arquitetura

Quais os benefícios da integração para os engenheiros de requisitos e arquitetos de software? Com esta integração simbiôntica entre as linguagens AOV-graph e AspectualACME, conseguimos prover, de forma automática, versões iniciais da arquitetura e do modelo de requisitos, bem como propagar mudanças realizadas em ambos os modelos. Além disso, os arquitetos de software podem mais facilmente recorrer às informações de requisitos, assim como os engenheiros de requisitos também podem recuperar mais agilmente as informações arquiteturais. Ressaltamos que embora informações extras estejam sendo geradas como elementos da própria linguagem de requisitos e de arquitetura elas devem ser geradas e mantidas por uma ferramenta e não pelo engenheiro de requisitos e arquiteto. Assim, este mapeamento gera mais do que modelos de requisitos e arquiteturas, ele gera associações entre estes artefatos possibilitando uma ação coordenada (sinérgica) nestes modelos.

4. Trabalhos Relacionados

O mapeamento entre requisitos e arquitetura e a rastreabilidade de modelos desenvolvidos nessas fases tem sido assunto de vários trabalhos analisados nesta seção. Entretanto, há três problemas gerais nesses trabalhos: (i) artefatos baseados em

diferentes meta-modelos e com diferentes propósitos (requisitos retratam o domínio do problema e arquiteturas representam o domínio da solução) são usados, assim todas estas abordagens geram apenas modelos preliminares; (ii) os métodos propostos são específicos para certas linguagens (meta-modelos). Portanto, é difícil generalizar as diretrizes para outras linguagens; e (iii) o rastro é definido somente de requisitos para arquitetura, mas não vice-versa. Assim, o arquiteto pode se beneficiar das informações contidas no modelo de requisitos, mas engenheiros de requisitos não podem facilmente acessar as decisões e limitações arquiteturais, nem propagar mudanças na arquitetura para os requisitos.

A abordagem apresentada nesse artigo apresenta o primeiro destes problemas, porque também gera apenas uma especificação preliminar em AspectualACME ou em AOV-graph. Informações arquiteturais ou de requisitos têm que ser adicionadas após o mapeamento. Entretanto, uma vez adicionadas tais informações, elas podem ser propagadas de um artefato para o outro, facilitando a manutenção e o acesso à tais informações. Quanto aos dois outros problemas, nossa abordagem está preocupada com a generalização das regras de mapeamento para aplicação em outras linguagens e também com o rastreamento bidirecional. Como mostramos na Seção 3, parte das regras de mapeamento pode ser reusada se as linguagens usam uma abordagem simétrica para representar conceitos transversais, mas se abstrações diferentes forem utilizadas para modularizar tais conceitos, então estas regras podem ser usadas como guia para mapeá-las.

Abordagens não orientadas a aspectos – A abordagem CBSP (*Component-Bus-System, and Properties*) de Egyed (2001) e Grunbacher (2001) ajuda no refinamento de um conjunto requisitos em uma arquitetura preliminar, que captura "decisões arquiteturais". Para isto, este mapeamento é uma atividade intensivamente manual, diferindo de nossa abordagem que é automática e provê regras para o mapeamento em ambos os sentidos. Jani (2005) e Lamsweerde (2003) definem um método para derivar arquiteturas de modelos de metas KAOS. Neste método especificações vão sendo gradualmente refinadas até reunir restrições arquiteturais específicas do domínio. As principais diferenças desta abordagem em relação à nossa é que ela usa um modelo de metas formal como entrada do processo, o mapeamento se dá apenas dos requisitos para a arquitetura, mas por outro lado, este método provê diretrizes para o refinamento da arquitetura, aplicando estilos e padrões, enquanto nossa abordagem provê somente uma primeira versão da arquitetura.

Abordagens orientadas a aspectos – Rashid (2003) e Sousa (2003) definem o mapeamento de aspectos candidatos para funções, decisões arquiteturais ou "aspectos" na arquitetura. Esse mapeamento é baseado apenas na experiência do desenvolvedor e não há diretrizes para mapear também os conceitos não-transversais. Jacobson (2005) define que casos de uso podem ser refinados para projetar diagramas de componentes (UML). Neste caso, nem os casos de uso, nem os diagramas de componentes explicitam quais os conceitos transversais, mas por outro lado, os diagramas UML são bem conhecidos e há vários métodos e ferramentas que apóiam a transição de um modelo para o outro. Sanchez (2006) define um processo automatizado baseado em MDD (*Model-Driven Development*) que deriva arquiteturas orientadas a aspectos (*UML 2.0 Profile for CAM,* em Pinto (2005)) de especificações de requisitos orientadas a aspectos (UML Profile). Kulesza (2004) define uma abordagem generativa orientada a aspectos e

mapeia uma extensão do modelo de *features* para uma extensão de diagrama de componentes UML. Esta extensão explicita os conceitos transversais, mas o modelo de *features* não explicita quais conceitos são funcionais e não-funcionais como AOV-graph representa. Além disso, o diagrama de componentes da UML é somente uma linguagem gráfica.

A vantagem do mapeamento de modelos de requisitos para ADLs, em vez de mapear para uma linguagem puramente gráfica como UML, é a possibilidade de usar ferramentas que incluem facilidades de verificação. ACME, por exemplo, fornece o ACME Studio que é um ambiente gráfico de desenvolvimento, associado a uma ferramenta de verificação formal que verifica as restrições especificadas na descrição arquitetural. ACME usa uma linguagem de predicados de primeira-ordem (FOPL) para especificar restrições às descrições arquiteturais.

5. Conclusões

Este artigo acrescenta o conceito de sinergia à integração simbiôntica entre requisitos e arquitetura, definida por Silva (2007), e especifica o mapeamento inverso – de arquitetura para requisitos, permitindo um fácil mapeamento entre as duas fases de desenvolvimento e a propagação de mudanças realizadas em ambos os modelos. Além disso, apresentamos um estudo de caso que ilustra o uso do mapeamento e discutimos os problemas identificados, apontando algumas soluções que podem ser reaproveitadas em outros modelos de requisitos e arquitetura orientados a aspectos que sigam uma abordagem simétrica similar a AOV-graph e AspectualACME .

As contribuições desse trabalho incluem: (i) a definição de regras de mapeamento de AspectualACME para AOV-Graph considerando tanto arquiteturas geradas com base em um modelo AOV-graph quanto arquiteturas geradas independentemente de AOV-graph; (ii) a modelagem e mapeamento de um estudo de caso que valida as regras; (iii) uma análise detalhada da arquitetura gerada a partir dos requisitos e dos requisitos gerados a partir da arquitetura. Trabalhos futuros incluem a aplicação das regras de mapeamento em outros estudos de caso, o mapeamento do modelo arquitetural para um modelo de projeto detalhado, uma avaliação quantitativa do mapeamento e o desenvolvimento de uma ferramenta integrada que dê apoio ao mapeamento nos dois sentidos e ao gerenciamento de mudanças nestes modelos.

6. Referências

Batista, T. et al. (2006) “Reflections on Architectural Connection: Seven Issues on Aspects and ADLs”. In: Workshop on Early Aspects – Aspect-Oriented Requirements, ICSE'06, May 2006, Shanghai, China.

Egyed, A., Grunbacher, P. and Medvidovic, N. (2001) “Refinement and Evolution Issues in Bridging Requirements and Architecture - the CBSP Approach”, In: From Requirements to Architecture Workshop (co-located with ICSE 2001), pp. 42-47.

Filman, R.et al. (2005) Aspect-Oriented Software Development. Addison-Wesley.

Garlan, D., Monroe, R., Wile, D. (1997) “ACME: An Architecture Description Interchange Language”. In: Proc. of the CASCON '97, Nov. 1997.

Grunbacher, P. et al. (2001) "Reconciling Software Requirements and Architectures: the CBSP Approach", In: Requirements Engineering, IEEE Computer Society, pp. 202-211.

Jacobson, I., NG, P. (2005) Aspect-Oriented Software Development with Use Cases. Addison-Wesley.

Jani, D et al. (2005) "Deriving Architecture Specifications from KAOS Specifications: A Research Case Study", In: European Workshop on Software Architecture (EWSA), LNCS 3527, pp.185-202.

Kulesza, U., Garcia, A., Lucena, C. (2004) "Towards a method for the development of aspect-oriented generative approaches", In: Workshop on Early Aspects, OOSPLA'04, Vancouver, Canadá .

Lamsweerde, A. v. (2003) "From System Goals to Software Architecture, Formal Methods for Software Architectures", In: Springer, Lecture Notes in Computer Science 2804, No., pp. 25 - 43.

Medeiros, A. L. (2007) MARISA - Mapping Requirements to Software Architecture, http://www.ppgsc.ufrn.br/~analuisa/marisa/. Acesso em 10 de Junho de 2007.

Pinto, M., Fuentes, L., Troya, J. (2005) A Dynamic Component and Aspect Platform, The Computer Journal, 48(4):401-420.

Rashid, A., et al. (2003) "Modularization and composition of aspectual requirements", In: Proc. of the 2nd Int. Conf. on Aspect-Oriented Software Development, ACM. p. 11-20.

Sanchez, P., et al. (2006) "Towards MDD Transformations from AO Requirements into AO Architecture", In: EWSA 2006, LNCS 4344, pp. 159–174.

Silva, L. (2006) "Uma Estratégia Orientada a Aspectos para Modelagem de Requisitos". Rio de Janeiro, 2006. 220p. Tese de Doutorado em Engenharia de Software - PUC-Rio.

Silva, L. et. al. (2007) "On the Symbiosis of Aspect-Oriented Requirements and Architectural Descriptions". In: Proc. of the EarlyAspects co-located with AOSD 2007, Vancouver, Canada, 2007.

Soares, S. et al. (2002) "Implementing Distribution and Persistence Aspects with AspectJ". In: Proc. of the OOPSLA'02,. pp. 174-190.

Sousa, G.; Silva, G.; Castro, J (2003) "Adapting the NFR framework to aspect-oriented requirements engineering". In: Anais do Simpósio Brasileiro de Engenharia de Software (SBES), Brazil.

Yu, Y., Leite, J., Mylopolous, J. (2004) "From goals to aspects: discovering aspects from requirements goal models", In: Proc. of IEEE Int. Symp. on Requirements Engineering (RE'04), Japan, pp. 38-47.