Uma Linguagem de Modelagem de Requisitos Orientada a Aspectos

Lyrene Fernandes da Silva[1,2], Julio Cesar Sampaio do Prado Leite[1]

[1]Departamento de Informática - PUC-Rio - Brasil

{lyrene,julio}@inf.puc-rio.br

Resumo. *Durante a modelagem de requisitos percebemos que muitos dos requisitos estão fortemente relacionados, entrelaçados ou sobrepostos, influenciando ou restringindo uns aos outros. Todos estes relacionamentos tornam a rastreabilidade entre requisitos fator crucial para modelagem e para evolução de software. Neste trabalho, para abordar os problemas de entrelaçamento e espalhamento de requisitos, apresentamos uma linguagem de modelagem que utiliza conceitos da programação orientada a aspectos, de maneira a registrar o impacto que os requisitos exercem uns sobre os outros. Esta linguagem é parte de um método para separação e composição de características transversais, que visa facilitar a modelagem, entendimento, reuso e evolução de requisitos.*

Palavras-chave. *Requirement modeling, crosscutting concerns, early aspects, goal models.*

1. Introdução

Para facilitar a separação e composição de características transversais[3], nos últimos anos houve um investimento no paradigma de orientação a aspectos [Kiczales 97]. Linguagens de programação orientadas a aspectos dão suporte à composição de características separadas e encapsuladas em aspectos, tornando a tarefa de composição transparente para o programador. Entretanto, realizar este tratamento neste nível de abstração não é suficiente, pois muitas decisões de planejamento e de desenho já foram tomadas sem levar em consideração esta natureza transversal [Nuseibeh 04]. Além disto, muitas vezes, já não se tem mais o rastro entre os aspectos e o que deu origem a eles, dificultando o entendimento de onde novos requisitos terão impacto.

---

[2] Este trabalho foi realizado com o apoio do CNPq.

[3] Utilizamos o termo características transversais como tradução para crosscutting concerns. Consideramos que um requisito é uma característica a ser provida pelo software.

Têm surgido linguagens e métodos de modelagem considerando estes novos elementos e relacionamentos de desenho de maneira a permitir a separação e composição de características transversais em um nível mais alto de abstração [Chavez 04]. Identificar, separar e compor características transversais de alto nível de abstração provê melhor entendimento do problema e da solução, rastreabilidade entre os diversos modelos construídos durante o desenvolvimento e entre as diversas características, reusabilidade de soluções de maior granularidade, e diminuição do *gap* entre as fases de desenvolvimento [Rashid 04][Baniassad 04][Brito 04]. Esta abordagem tem se estendido não só à fase de desenho, mas também à fase de definição de requisitos, denominada por alguns de *early-aspects* [Rashid 02].

Entretanto, estas abordagens são ainda imaturas [Bakker 05] e pesquisas neste sentido são necessárias. O problema é como tratar, cedo, características que são naturalmente entrelaçadas e sobrepostas entre si? Como modelá-las? Como manter a rastreabilidade entre elas? Como visualizá-las?

Para facilitar a modelagem de requisitos propomos um método de integração de características transversais [Silva 05]. Este método é fundamentado na idéia implementada em linguagens de programação orientadas a aspectos, tais como AspectJ [Kiczales 01] e Hyper/J [Tarr 00]. Assim como nestas linguagens, nosso método de integração aborda a separação e composição de características transversais. Nossa contribuição é trazer os benefícios de linguagens orientadas a aspectos para linguagens de modelagem de requisitos, através de um mecanismo que tornará possível o gerenciamento de requisitos transversais.

A separação é realizada pela modelagem de características através de Modelos de Metas [Mylopoulos 92]. A composição é realizada por um mecanismo (weaver) que gera um modelo único do sistema sendo desenvolvido, facilitando a inclusão e exclusão de características deste modelo. Além disto, o método aborda também a extração de diferentes visões dos modelos separados e compostos. Nesta etapa, isto é essencial, visto que esta é a etapa de elaboração da solução, então ora é importante visualizar as características juntas, ora separadas.

Nossa premissa é que conhecer a natureza transversal dos requisitos nesta fase, pode ajudar não só nas tarefas de priorizar, modelar, validar e gerenciar requisitos, mas também prover reuso da especificação e conhecimento para desenhar a arquitetura do sistema e manter a rastreabilidade entre características transversais e entre as fases de definição de requisitos e arquitetura.

Neste artigo detalhamos a primeira parte do método de integração de características transversais, a Separação. Para dar suporte à separação,

definimos uma linguagem baseada em modelos de metas e em conceitos de um modelo de aspectos [Chavez 04] [Han 04]. Esta linguagem é uma extensão para o modelo de metas. Ela provê um componente que registra como os requisitos afetam uns aos outros, facilitando a rastreabilidade entre eles.

As seções seguintes deste artigo estão organizadas da seguinte maneira: na Seção 2 descrevemos o contexto em que este trabalho está inserido e resumimos os principais conceitos que fundamentam nossa abordagem; na Seção 3 apresentamos a linguagem de modelagem de metas orientada a aspectos e seus benefícios; na Seção 4 apresentamos os principais trabalhos que estão relacionados a este; por fim, na Seção 5 relatamos nossas conclusões e futuros trabalhos.

2. Um Método de Integração de Características Transversais

Os princípios de separação de composição têm sido usados para abordar os problemas de espalhamento e entrelaçamento de características. O princípio de separação é importante para diminuir a complexidade em certos momentos, permitindo o entendimento e análise de uma única característica por vez. Entretanto, um sistema é naturalmente composto de diferentes características, e entendê-las em conjunto é igualmente importante. Se por um lado, precisamos separar as características, por outro precisamos compô-las, pois muitas vezes estas características são fortemente relacionadas, entrelaçadas e/ou sobrepostas, influenciando ou restringido umas às outras, sendo chamadas de características transversais [Tekinerdoðan 04]. Isto torna difícil a separação e análise das partes do sistema bem como a análise do impacto que umas exercem sobre as outras.

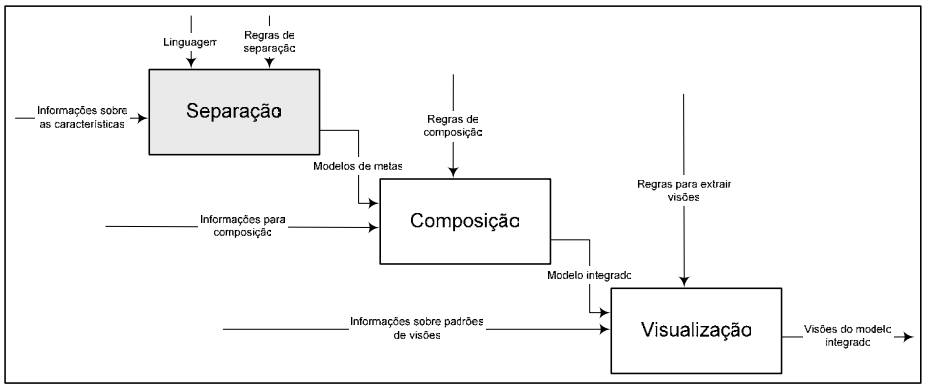

Figura 1 – Método de integração de características transversais

Para abordar estes problemas, nós propomos um método de integração de características transversais na fase de definição de requisitos, veja Figura 1. Este método é composto de três atividades principais:

1) Separação - provê técnicas e linguagem para representação das características de um sistema, dando apoio à modelagem de requisitos. Por exemplo, podemos separar as funcionalidades específicas da aplicação dos requisitos de segurança e de tratamento de exceções. Além de dar suporte à separação, a linguagem de modelagem provê elementos que indicam como e onde cada grupo de requisitos afeta. Esta informação é registrada nos elementos do modelo de aspectos que faz parte da linguagem de modelagem. Detalhamos a linguagem de modelagem na Seção 3.

2) Composição - provê regras de composição (transformação) e um interpretador, que gera um modelo integrado a partir de um conjunto de modelos definidos segundo a linguagem provida pela atividade de Separação. As informações de como e onde compor são registradas no elemento "relacionamento transversal", detalhado na Seção 3. O modelo resultante da composição contempla um conjunto de visões que pode ser derivado dele.

3) Visualização - provê regras para recuperar diferentes visões do modelo integrado. O modelo resultante da composição e informações sobre algumas visões comuns são as entradas para a atividade de visualização. Esta atividade provê alguns modelos parciais do modelo integrado de maneira a facilitar o entendimento das características do sistema e da composição delas. Nós consideramos que o mais interessante para o desenvolvedor não é ter apenas a representação do sistema completo, mas sim diferentes visões deste mesmo modelo. Cada visão retrata o sistema inteiro por um certo ângulo [Leite 91], por exemplo: uma visão de quais partes do sistema utilizam um mecanismo de autenticação ou sofrem impacto dele, uma visão que mostre quais usuários tem autorização para utilizar quais partes do sistema, o impacto que o sistema sofre se uma outra característica transversal for adicionada, dentre outros.

Neste trabalho detalhamos a linguagem de modelagem criada para dar suporte à separação de características. Esta linguagem utiliza: um modelo de componentes, retratando os elementos da linguagem de modelagem de requisitos e suas restrições, em nosso caso o modelo de metas; e um modelo de aspectos, representando como conjuntos de requisitos afetam (influenciam ou restringem) outros. A seguir resumimos alguns conceitos referentes a modelos de metas e linguagens orientadas a aspectos.

2.1 Modelo de metas

Modelos de metas representam requisitos funcionais ou não funcionais através da decomposição de metas [Giorgini 02][Yu 04]. Esta decomposição indica como satisfazer uma determinada meta e por outro lado indica a razão pela qual as sub-metas são necessárias. Na literatura há algumas variações de modelos de metas [Yu 04] [Lamsweerde 01]. Em geral, usam-se árvores *and/or* para representar a decomposição de metas e definir um espaço de soluções alternativas para satisfazer a meta raiz. Em [Mylopoulos 92],

softmetas são propostas como meio para modelar e analisar requisitos não funcionais.

Para os nossos propósitos, estamos interessados no *V-graph*, um tipo de modelo de metas [Yu 04]. Este modelo é composto de 3 elementos: soft-metas, metas e tarefas. Os possíveis relacionamentos entre eles são *make, help, unknow, hurt* e *break*. Este modelo permite a descrição de nós intencionais (metas e *softmetas*) e nós operacionais (tarefas). Cada elemento do *V-graph* é composto de duas partes: um tipo e um tópico. O tipo descreve uma função genérica ou um requisito não funcional genérico. O tópico captura a informação contextual para o elemento. Esta informação contextual é similar ao *subject* definido em [Zave 97].

Nós escolhemos este modelo por algumas razões: com ele temos três níveis de abstração, *softmetas*, metas e tarefas – estes diferentes níveis permitem que pensemos em três níveis de variabilidade e configurabilidade, indicam características diferentes, permitem que separemos características de maneiras diferentes e enriquecem os relacionamentos transversais entre diferentes árvores de metas. Além disto, o tipo e o tópico dos elementos permitem que tenhamos uma visão de dados e de funções no mesmo modelo, sem tornar a visibilidade afetada, sendo outra maneira de ver as características.

2.2 Modelo de Aspectos

Linguagens de programação orientadas a aspectos, tais como AspectJ e Hyper/J, estendem linguagens orientadas a objetos para prover a separação e composição de características transversais. No caso da linguagem AspectJ [Kiczales 01], a separação é realizada através da inserção de uma nova abstração denominada aspecto à programação orientada a objetos. A composição é realizada através de um combinador denominado *weaver*, que é responsável por pré-processar o código orientado a objetos, inserindo ou modificando os objetos com o comportamento dos aspectos.

Além dos métodos e atributos, em AspectJ, um aspecto é composto por *pointcuts*, *advices* e *inter-type declarations*. Eles podem alterar a estrutura estática ou dinâmica de um programa. A estrutura estática é alterada adicionando, por meio das *inter-types declarations*, membros (atributos, métodos ou construtores) a uma classe, modificando assim a hierarquia do sistema. Já a alteração na estrutura dinâmica de um programa ocorre em tempo de execução por meio dos *join points*, os quais são selecionados por *pointcuts*, e através da adição de comportamentos (*advices*) antes ou depois dos *join points*.

Um *pointcut* indica os pontos onde um determinado comportamento será inserido. Ele é descrito por um nome e um corpo. O corpo indica os

joinpoints onde serão aplicados os *advices* associados, podendo haver vários *joinpoints* aninhados através dos operadores *or*, *and* e *not*. Cada *pointcut* está associado a um ou mais *advices*. *Advices* descrevem o comportamento (trecho de código) a ser inserido nos *joinpoints*. Existem três formas de *advice*, são elas: *before*, *around* e *after*.

Nossa abordagem utiliza estes conceitos para representar o impacto entre as características de um sistema. Não distinguimos características transversais de características não-transversais, todas são representadas por modelos de metas. Entretanto, as características que se espalham pelo sistema estão ligadas às outras por relacionamentos transversais. Um relacionamento transversal possui as informações de *pointcuts*, *advices* e *inter-types declarations*. Nosso mecanismo de composição (assim como o combinador em AspectJ) utiliza estas informações para inserir o comportamento de um modelo de metas em outro, criando *links* de rastreabilidade entre os diversos modelos de metas. Na Seção 3 detalhamos a linguagem de modelagem definida.

3. Linguagem de Modelagem de Metas Orientada a Aspectos

Para separar características transversais nós definimos uma linguagem de modelagem. Esta linguagem é uma extensão do modelo de metas *V-graph* [Yu 04]. Na Figura 2 ilustramos o modelo de dados da linguagem. Há duas partes principais neste modelo: definição dos elementos da linguagem de modelagem de requisitos, neste caso o modelo *V-graph*; e a definição dos elementos do modelo de aspectos, neste caso definida pelo relacionamento transversal, nas cores cinza claro e cinza escuro na Figura 2, respectivamente.

A linguagem de modelagem, na Figura 2, é composta de modelos e relacionamentos. Os modelos (neste caso modelo de metas) também são constituídos de elementos e relacionamentos. Os elementos são *softgoals*, *goals* e *tasks*. Cada um deles possui topic e type. Os relacionamentos possuem ponto de origem (*source*), ponto de destino (*target*) e um rótulo (*label*). Há quatro tipos de relacionamentos: 1) relacionamento de decomposição, ilustrado pela agregação entre *component* e *compoundComponent*; 2) relacionamento de contribuição, podendo ser *make*, *help*, *unknown*, *hurt* e *break*; 3) relacionamento de combinação, podendo ser AND, OR e XOR; e 4) relacionamento transversal. Além de rótulo, destino e origem, os relacionamentos transversais possuem *pointcuts*, *advices* e *intertype-declarations*.

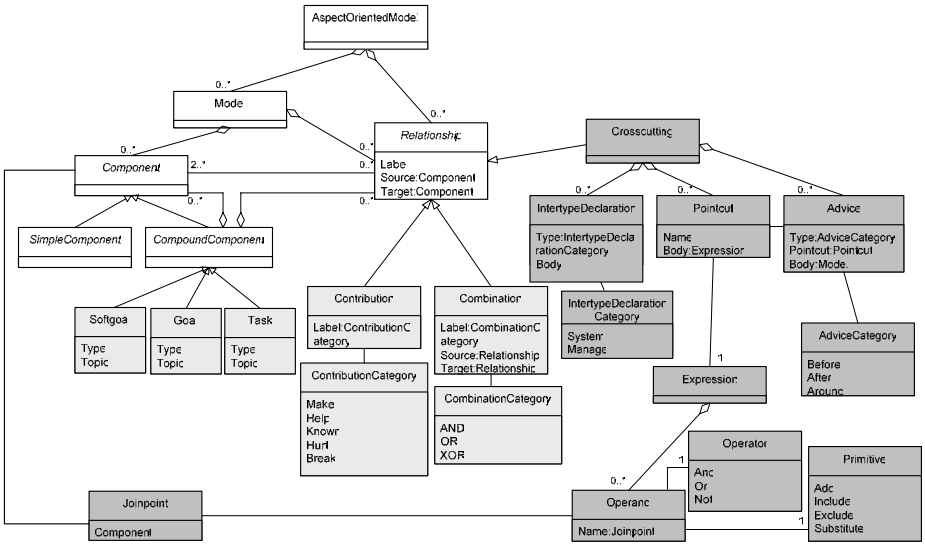

Figura 2 – Modelo de componentes da linguagem de modelagem de requisitos

O relacionamento transversal representa em nossa linguagem o que o aspecto representa em *AspectJ*. Consideramos que desta forma podemos mais facilmente reutilizar modelos de metas, visto que as informações de como um modelo afeta o outro estão somente nos relacionamentos. Estas informações são descritas através dos *pointcuts*, *advices* e *intertype declarations*. Em linguagens de implementação *pointcuts* e *advices* são utilizados para modificar a estrutura dinâmica de classes, e *intertype declarations* para modificar a estrutura estática. Em modelos de metas, consideramos que a estrutura dinâmica é definida pelos elementos e seus relacionamentos enquanto a estrutura estática é representada pelo tópico dos elementos. Desta forma, utilizamos *pointcuts* e *advices* para inserir ou modificar os elementos de um modelo de metas, e utilizamos *intertype declarations* para inserir e ou modificar novos tópicos ou dados aos elementos de um modelo de metas.

Cada *pointcut* é definido através de um nome e uma expressão. Expressões consistem em sentenças que usam operandos, operadores e primitivas. Os operandos são elementos dos tipos associados aos *joinpoints* (soft-metas, metas e tarefas). Os operadores são utilizados para agrupar um ou mais *joinpoints*, eles são OR, AND e NOT. As primitivas definem como um *pointcut* será afetado. Em nosso caso definimos os seguintes tipos de primitivas: *add* - adiciona um conjunto de elementos ao local especificado; *include* - adiciona um conjunto de elementos como sub-elementos do elemento especificado; *exclude* - exclui um conjunto de elementos da sub-árvore especificada; e *substitute* - substitui elementos da sub-árvore especificada.

Na Figura 3, exemplificamos as informações de um relacionamento transversal. O pointcut denominado *encrypt* indica que as tarefas Get[data] e Register[user] são afetadas. Neste pointcut utilizamos o operador AND para agrupar pontos que serão afetados da mesma maneira. As primitivas add e include são utilizadas para definir que elementos da sub-árvore Cryptography serão adicionados no mesmo nível que Get[data] está, e serão incluídos como sub-elementos da tarefa Register[user], respecticamente.

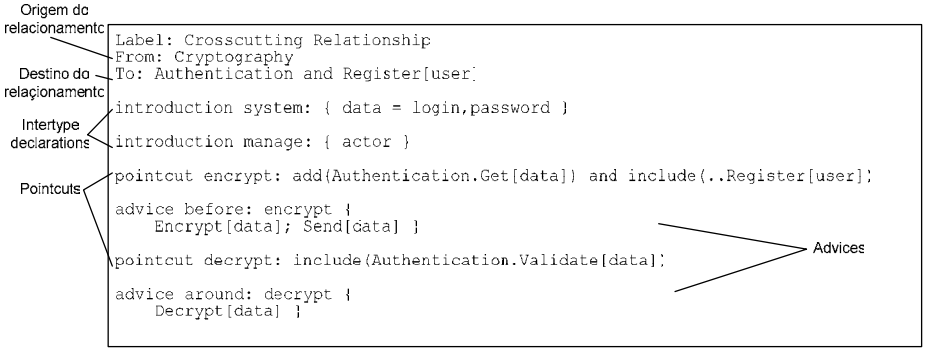

Figura 3. Exemplo de relacionamento transversal

Advices estão associados a *pointcuts*. Eles definem o conjunto ou sub-conjunto de elementos de um certo modelo de metas que deve afetar um outro modelo de metas. *Advices* também são descritos por expressões. Estas expressões são formadas por: tipo (*after*, *before* e *around*); *pointcuts* que serão afetados; e corpo que define como os *pointcuts* serão afetados. Por definição o corpo de um *advice* é constituído de toda uma árvore ou sub-árvore de metas, mas podemos também definir o comportamento apenas com partes da árvore. Na Figura 3, os *advices* indicam quais elementos do modelo *Cryptography* afetaram os *pointcuts* indicados pelos nomes *encrypt* e *decrypt*.

Intertype declarations são utilizadas para adicionar novos tópicos ou dados aos modelos de metas. *Intertype declarations* são definidas através de *introductions*. Definimos dois tipos de *introduction*: *manage* (gerencial) e *system* (de sistema). A *introduction* gerencial registra alguns atributos de propósito gerencial aos modelos de metas, por exemplo, qual equipe está desenvolvendo cada parte do modelo ou usuários interessados, dentre outros. A *introduction* de sistema adiciona novos tópicos à modelagem. Na Figura 3, a *introduction* do tipo gerencial acrescenta um atributo denominado *actor* a todos os pontos afetados. Isto significa que o engenheiro de requisitos pode indicar quais os usuários interessados em quais requisitos. Desta forma, após a composição (segunda atividade do método descrito na Seção 2) podemos extrair uma visão do modelo integrado com esta informação.

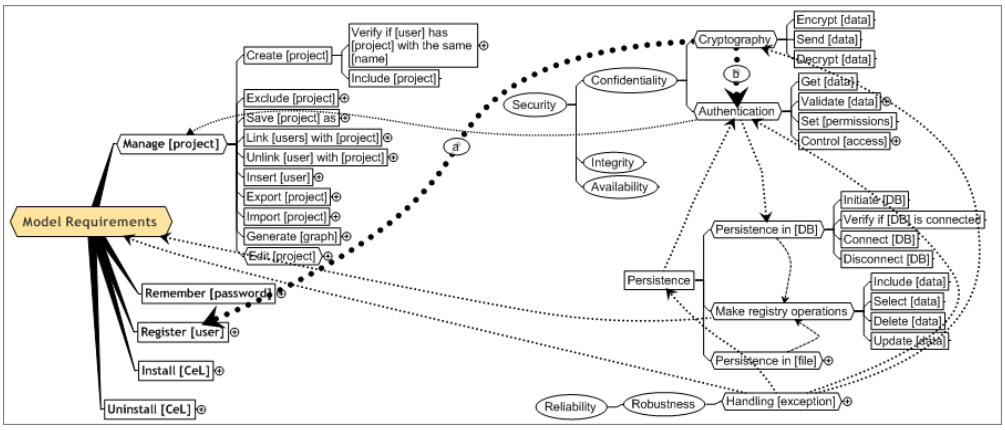

Figura 4. Relações transversais entre os modelo de metas

A Figura 4 é um exemplo ilustrativo de modelos de metas. Este estudo de caso é composto por quatro modelos de metas: sobre a aplicação (um editor de léxico e cenários), sobre segurança, sobre persistência e sobre confiabilidade. Eles foram modelados separadamente, mas levando em consideração o mesmo contexto, ou seja, o modelo de segurança criado é o modelo utilizado especificamente por este sistema. As elipses representam softmetas, os hexágonos representam metas e as tarefas são representadas por retângulos. As linhas contínuas representam o relacionamento de decomposição e as linhas tracejadas representam relacionamentos transversais. O relacionamento transversal ilustrado na Figura 3 está em destaque na Figura 4. Semanticamente o relacionamento (b) indica que algumas tarefas de Cryptography são incluídas antes de Authentication.Get[data] e é uma sub-tarefa de Authentication.Validate[data], resultando no modelo ilustrado na Figura 5, os elementos na cor cinza são elementos da sub-árvore Cryptography.

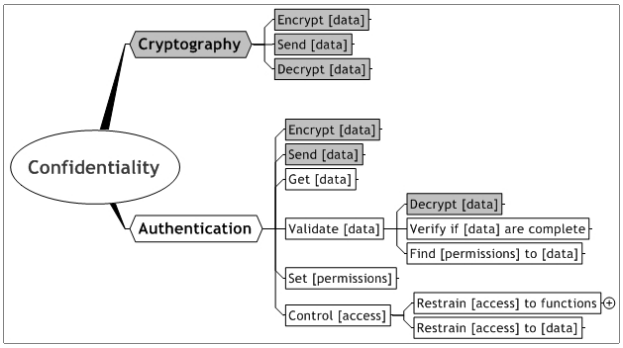

Figura 5. Modelo resultante de um relacionamento transversal

Os elementos definidos pelo relacionamento transversal não devem adicionar complexidade ao modelo de requisitos, pois graficamente são

representados, simplesmente, por um relacionamento e textualmente, as informações são acrescentadas à medida que surgir a necessidade, sem gerar muitas mudanças ao modelo. Entretanto, o uso demasiado de relacionamentos transversais pode dificultar um pouco a visualização do modelo. Desta forma, as etapas de Composição e Visualização, ilustradas na Figura 1, se tornam ainda mais importantes para amenizar a complexidade e proporcionar facilidades durante a modelagem de requisitos.

O modelo de componentes na Figura 2 está sendo utilizado para gerarmos um editor, gráfico e textual, de modelos de metas segundo esta hierarquia. Este editor estará integrado aos componentes de composição e de visualização que estamos desenvolvendo para dar suporte ao método de integração de características transversais definido na Seção 2.

4. Trabalhos Relacionados

O crescente interesse pelo desenvolvimento orientado a aspectos tem levado pesquisadores a definirem linguagens de modelagem de projeto que considerem os conceitos definidos por modelos de aspectos [Chavez 04][Han 04]. Enquanto estas linguagens estendem linguagens de modelagem de objetos para retratar os mesmos conceitos em níveis de abstrações diferentes, nós estendemos as linguagens de modelagem de requisitos para registrar o impacto que eles exercem uns sobre os outros.

Na fase de definição de requisitos há algumas propostas para tratar características transversais, mas o foco tem sido dado à identificação de "candidatos a aspectos" [Bakker 05]. Para representação dos aspectos candidatos, [Moreira 02], [Brito 04], [Rashid 03] e [Sousa 04] utilizam templates descrevendo como e onde cada aspecto candidato exercerá impacto. Alguns destes trabalhos utilizam o modelo de casos de uso para modelar os requisitos funcionais e usam as relações *extend* para representar os aspectos candidatos no mesmo modelo. Entretanto, com exceção de [Rashid 03], estas informações são registradas em linguagem natural, dificultando a atividade de composição. Assim como as linguagens de modelagem de projeto estas extensões tem o único propósito de representar as características transversais no nível de Requisitos. Nós não temos este único propósito, mas sim dar apoio à modelagem destas características por registrar a rastreabilidade entre elas, facilitando a inclusão e exclusão delas no modelo.

Em [Rashid 03], há uma interessante abordagem para separar e compor características transversais utilizando modelos em XML. Entretanto, os requisitos funcionais e as características transversais são sentenças de requisitos e o propósito da composição é gerar um único modelo. Consideramos que nas fases anteriores à implementação, a composição de características transversais não tem seus reais benefícios se não prover

maneiras de extrair visões das características integradas, pois o modelo integrado pode ser tão complexo que não proverá sua principal vantagem que é facilitar o entendimento do sistema.

Em [Leite 04], os autores propõem um processo para viabilizar o reuso de requisitos não funcionais. Esta abordagem define uma linguagem baseada no framework 5W2H para armazenar e recuperar requisitos da biblioteca. Além disto, eles sugerem que um mecanismo de composição poderia ser utilizado para juntar um modelo de uma aplicação ao requisito não funcional recuperado da biblioteca, mas a linguagem utilizada é natural, não permitindo mecanização.

O trabalho [Leite 04] motiva nossa idéia de definir um mecanismo automatizado de separação e composição de características transversais. Com o modelo de metas nós podemos utilizar as técnicas de análise de metas [Lamsweerde 00][Giorgini 02][Gonzáles 04] para derivar novas visões do modelo de metas separado ou integrado. Desta forma, adicionamos os benefícios dos modelos de metas às técnicas de separação e composição de características transversais e vice-versa.

5. Conclusões e Trabalhos Futuros

Dado que os requisitos estão continuamente mudando, é necessário prover meios para facilmente modelá-los. Nosso método de integração de características transversais propõe um mecanismo para separação, composição e visualização delas [Silva 05]. Neste artigo, detalhamos uma linguagem de modelagem que registra o relacionamento transversal entre estas características. Através deste relacionamento podemos mais facilmente modificar, incluir e excluir requisitos da modelagem. O relacionamento transversal representa como um requisito (ou um conjunto de requisitos) afeta os outros.

Não é nosso foco identificar ou analisar características transversais, nem definir o que são, qual a origem delas ou se elas devem ser implementadas como aspectos, mas consideramos importante tratá-las durante todo o processo de desenvolvimento de software. Não distinguimos características transversais de não transversais, nem discutimos a relação entre características transversais, requisitos não-funcionais e aspectos, consideramos que não precisamos desta informação a priori e que o mais importante é prover um meio mais fácil de modelá-las. Desta forma, criamos um ambiente propício para que possamos estudar estes conceitos mais profundamente.

Acreditamos que nossa abordagem tem um impacto positivo em todo o processo de desenvolvimento de software. Ela facilita a modelagem de requisitos, o reuso de características transversais, o reuso do conhecimento do

comportamento de características transversais, a estimativa de custos e prazos, as decisões de desenho e a evolução de software.

Para tornar esta abordagem eficaz estamos trabalhando: na implementação do conjunto de ferramentas para dar apoio ao nosso método; no aprimoramento e flexibilização da linguagem de modelagem; no desenvolvimento de regras de composição e regras para extração de um conjunto de visões comuns que podem ser derivadas do modelo de metas; e na modelagem de um conjunto preliminar de características transversais reusáveis.

6. Referências

[Bakker 05] Jethro Bakker, Bedir Tekinerdogan, Mehmet Aksit; Characterization of early aspects approaches; Proceedings of the Early Aspects Workshop at AOSD; 2005.

[Baniassad 04] Elisa Baniassad, Siobhán Clarke; Theme: An approach for aspect-oriented analysis and design; 26th International Conference on Software Engineering (ICSE'04); Scotland, pp. 158-167, 2004.

[Brito 04] Isabel Brito, Ana Moreira; Integrating the NFR framework in a RE model; Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, In 3rd International Conference on Aspect-Oriented Software Development (AOSD 2004); Inglaterra; 2004.

[Chavez 04] Cristina Chavez; A Model-Driven Approach to Aspect-Oriented Design; PhD Thesis, Computer Science Department; PUC-Rio; Rio de Janeiro; Brazil, April 2004.

[Giorgini 02] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, Roberto Sebastián; Reasoning with goal models; Proceedings of the 21st International Conference on Conceptual Modeling; p.167-181; 2002.

[Gonzáles 04] Bruno Gonzáles, Miguel Laguna, Julio Leite; Visual Variability Analysis with Goal Models; Proceedings of Requirements Engineering Conference, 12th IEEE International (RE'04); pp. 38-47; Japan; 2004.

[Han 04] Yan Han, Gunter Kniesel e Armin B. Cremers; A meta model for AspectJ; Technical Report IAI-TR-2004-3, Computer Science Department III, University of Bonn. ISSN 0944-8535. October 2004.

[Kiczales 97] G. Kiczales et al.; Aspect-Oriented Programming; Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97); LNCS (1241), Springer-Verlag, Finland, June 1997.

[Kiczales 01] G. Kiczales et al.; An Overview of AspectJ; Proceedings of the European Conference on Object-Oriented Programming (ECOOP'01); Budapest, Hungary; 2001.

[Lamsweerde 00] Axel van Lamsweerde, E. Letier; Handling obstacles in goal-oriented requirements engineering; IEEE Trans. Software Eng.; 26(10):978–1005; 2000.

[Lamsweerde 01] Axel van Lamsweerde; Goal-Oriented Requirements Engineering: A Guided Tour; Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering; Toronto; August 2001; 249-263.

[Leite 91] Julio Leite and Peter Freeman; Requirements Validation Through Viewpoint Resolution; IEEE Transactions on Software Engineering: Vol. 17, N. 12, 1991, pp 1253-1269.

[Leite 04] Julio Leite, Yijun Yu, Lin Liu, John Mylopoulos; Non-functional driven reusability; Relatório interno, 2004.

[Moreira 02] Ana Moreira, João Araújo, Isabel Brito; Crosscutting quality attributes for requirements engineering; Proceeding of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM Press; Italy; 2002.

[Mylopoulos 92] J. Mylopoulos, L. Chung, and B. Nixon; Representing and using nonfunctional requirements: A process-oriented approach; IEEE Transactions on Software Engineering; 18(6):483–497; June 1992.

[Nuseibeh 04] Bashar Nuseibeh; Crosscutting requirements; Proceedings of the 3rd International Conference on Aspect-Oriented Software Development (AOSD 2004); Lancaster-UK; 2004.

[Rashid 02] Awais Rashid, Peter Sawyer, Ana Moreira, João Araújo; Early aspects: a model for aspect-oriented requirements engineering; Proceedings of IEEE Joint Conference on Requirements Engineering; Germany; pp. 199-202; 2002.

[Rashid 03] Awais Rashid, Ana Moreira, João Araújo; Modularization and composition of aspectual requirements; Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, ACM; pp. 11-20; 2003.

[Silva 05] Lyrene Silva, Julio Leite; Integração de Características Transversais Durante a Modelagem de Requisitos; Submetido para o 20º Simpósio Brasileiro de Engenharia de Software a ser realizado de 3 a 7 de Outubro de 2005 em Uberlândia-MG, Brasil.

[Sousa 04] Geórgia Sousa, Sérgio Soares, Paulo Borba, Jaelson Castro; Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach; Proceedings of the Early Aspects Workshop at AOSD; 2004.

[Tarr 00] P. Tarr, H. Ossher; Hyper/J User and Installation Manual; 2000. http://www.alphaworks.ibm.com/tech/hyperj

[Yu 04] Yijun Yu, Julio Leite, John Mylopoulos; From goals to aspects: discovering aspects from requirements goal models; Proceedings of Requirements Engineering Conference, 12th IEEE International (RE'04); pp. 38-47; Japan; 2004.

[Zave 97] P. Zave, M. Jackson; Four dark corners of requirements engineering; ACM Transaction Software Engineering Methodologics; 6(1):1–30; 1997.