4
Estratégia Orientada a Aspectos para Modelagem de Requisitos

Como descrevemos no Capítulo 2, durante a definição de requisitos os problemas de espalhamento e entrelaçamento de características dificultam a modelagem, rastreabilidade, evolução e a reutilização de requisitos. Visto que estes problemas têm sido abordados durante a implementação, por meio de linguagens de programação orientadas a aspectos, desenvolvemos uma estratégia baseada nos fundamentos deste paradigma para prover facilidades à modelagem de requisitos.

Esta estratégia consiste em um metamodelo para integração de características transversais. Este metamodelo define um conjunto mínimo de atividades para modelar requisitos considerando características transversais, e mecanismos para efetivá-las. A integração é centrada na definição de um elemento de modelagem que permite a descrição explícita de como as características estão entrelaçadas e espalhadas em modelos de requisitos. Este elemento é definido como parte de uma linguagem de modelagem de requisitos orientada a aspectos (LMROA), que pode ser instanciada para a utilização de diferentes métodos de modelagem.

A seguir, na Seção 4.1, apresentamos o metamodelo para integração de características transversais e seus componentes. Na Seção 4.2, definimos a linguagem de modelagem de requisitos orientada a aspectos. Na Seção 4.3, detalhamos como utilizar esta estratégia aplicando-a ao modelo V-Graph. Na Seção 4.4, descrevemos como aplicar nossa estratégia aos modelos de cenários (Leite, 1997), sentenças de requisitos ou léxico (Leite, 1990). Na Seção 4.5, apresentamos como esta estratégia afeta as atividades de elicitação e análise do processo de engenharia de requisitos. Na Seção 4.6, resumimos nossa abordagem.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.1. Metamodelo para Integração de Características Transversais

O metamodelo para integração de características transversais é uma estratégia para modelagem de requisitos que provê elementos para registrar, controlar e analisar a interação transversal entre eles. Desta forma, a estratégia oferece ao engenheiro uma nova maneira de raciocinar e modelar os requisitos e oferece, também, mecanismos que facilitam a visualização, análise e alteração do documento sendo criado (Silva, 2005c; 2005d).

A estratégia consiste de três atividades, denominadas de Separar, Compor e Visualizar, veja Figura 30. Colocamos estas três atividades como sendo essenciais durante o processo de modelagem de requisitos porque elas possibilitam, respectivamente, a modularização, a junção das partes modularizadas para formar o sistema e a análise de visões (opiniões, serviços e modelos) dos módulos e do sistema inteiro.

PUC-Rio - Certificação Digital Nº 0210666/CB

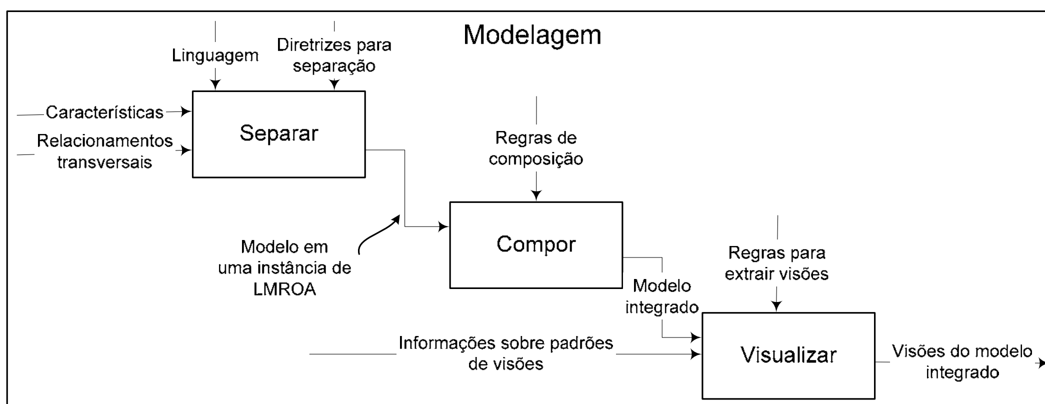


Figura 30. Metamodelo para integração de características transversais

Para Separação, provemos uma linguagem e diretrizes para que os requisitos sejam modelados levando em consideração sua natureza transversal, ou seja, explicitando quais e como alguns requisitos têm tendência a estar espalhados ou entrelaçados aos outros. Esta informação sobre transversalidade é registrada em um novo tipo de relacionamento, adicionado como uma extensão ao modelo de requisitos. Colocando em evidência a transversalidade dos requisitos podemos mais cuidadosamente observá-la, e assim entender como os requisitos afetam uns aos outros.

A Composição é um processo automatizado que combina os requisitos inicialmente modelados de maneira separada. Esta combinação é realizada pela interpretação da informação contida no relacionamento transversal e aplicação de

regras de composição. As regras de composição definem como os elementos do relacionamento transversal são transformados, i.e., sua semântica, propagando a informação nele contida.

O modelo resultante da composição e informações sobre quais visões podem ser extraídas de suas informações são necessários 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, no momento da modelagem, o mais interessante para o desenvolvedor não é ter apenas a representação do sistema completo, mas sim, diferentes visões parciais deste mesmo modelo.

As atividades de separação, composição e visualização compõem juntamente às atividades de elicitação e análise o processo iterativo de definição de requisitos. Na Seção 4.5 descrevemos qual o impacto de nossa estratégia nas atividades de elicitação e análise.

PUC-Rio - Certificação Digital Nº 0210666/CB

Componentes da Estratégia

Nossa estratégia para integração de características transversais define: uma linguagem de modelagem de requisitos orientada a aspectos (LMROA), diretrizes de como utilizá-la, um mecanismo de composição e um mecanismo de visualização, veja na Figura 31. Os mecanismos utilizam regras de composição e de transformação que dependem dos construtos de LMROA. Desta forma, a separação, a composição e a visualização são centradas na linguagem de modelagem de requisitos orientada a aspectos.

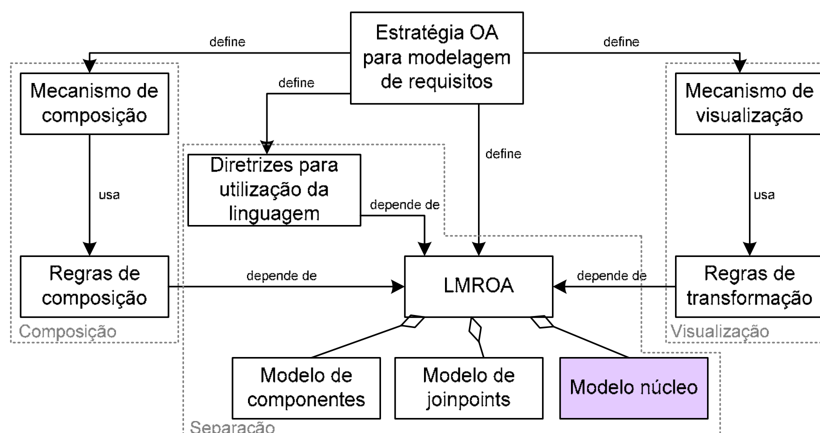


Figura 31. Componentes utilizados para integração de características transversais

LMROA é composta por: qualquer linguagem de modelagem de requisitos (modelo de componentes), tais como cenários, casos de uso ou modelo de metas; um relacionamento transversal (modelo núcleo), que foi definido com base em alguns conceitos da linguagem AspectJ; e um modelo de *joinpoints* que define como os modelos de componentes e núcleo estão relacionados. LMROA e as diretrizes para sua utilização constituem a infra-estrutura necessária para a atividade de separação, na qual as atividades de composição e visualização se baseiam.

É importante ressaltar que nos baseamos no metamodelo definido por Chavez (2003, 2004) para definir LMROA. Assim, utilizamos os nomes "modelo de componentes" e "modelo de joinpoints" com a mesma semântica descrita em (Chavez, 2003, 2004) apesar de estarmos trabalhando em um nível de abstração diferente.

Fazendo analogia às práticas utilizadas no nível de implementação temos que, veja a Figura 32:

- a atividade de separação é equivalente à modularização provida por linguagens de programação com sua sintaxe, semântica e diretrizes para melhor implementar diferentes características;
- a atividade de composição é equivalente ao *weaver* ou compilador das linguagens de programação; e
- o mecanismo de visualização é equivalente à realização de consultas (pattern match) em busca de diferentes características no código dos componentes (como acontece em Hyper/J) ou à visão representada pelo comportamento do programa em execução (como acontece em AspectJ).

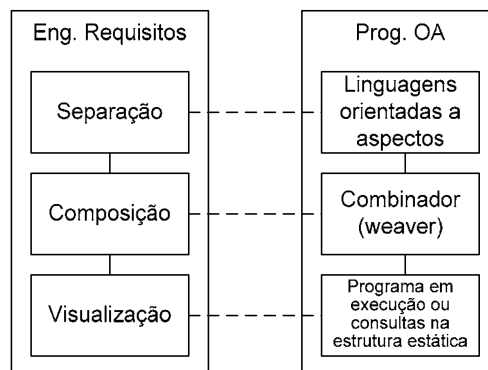


Figura 32. Analogia entre a estratégia para integração de características transversais durante a definição de requisitos e algumas práticas utilizadas no nível de implementação

PUC-Rio - Certificação Digital Nº 0210666/CB

A seguir, apresentamos um pequeno exemplo para ilustrar as atividades de Separação, Composição e Visualização. Detalhes de cada uma delas são apresentados nas próximas Seções.

Exemplo

Considere um sistema para edição de cenários, em que há dois principais grupos de requisitos: os específicos para edição, e os relacionados à persistência. Na Figura 33, representamos estes grupos de requisitos em V-graph, sem utilizar nossa abordagem. Neste grafo, podemos observar que as tarefas "Include [data]", "Update [data]", "Exclude [data]" e "Select [data]" afetam várias tarefas do grafo "Model [requirements]" (veja os relacionamentos em negrito). Estes relacionamentos dificultam a visibilidade do modelo e têm que ser desenhados, analisados, inspecionados e mantidos pelo engenheiro de requisitos quando o modelo muda.

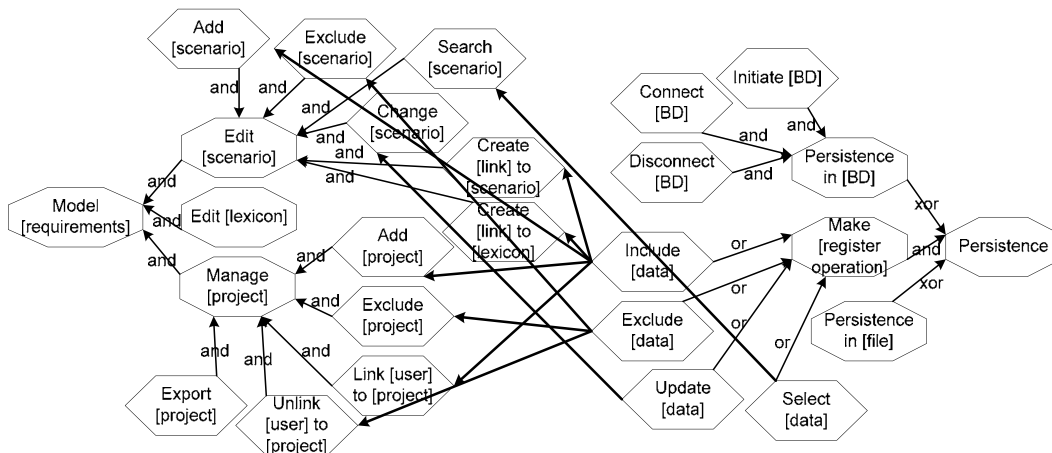


Figura 33. Exemplo ilustrativo da abordagem tradicional

A linguagem de modelagem e os mecanismos propostos por nossa abordagem possibilitam a separação, composição e visualização da seguinte maneira.

A atividade de separação é responsável por possibilitar a modelagem destes dois grupos de requisitos separadamente, bem como a modelagem de como cada um deles afeta os demais, por meio de relacionamentos transversais (elos em negrito na Figura 34 e descrição). Diretrizes indicando como e quando realizar esta separação são apresentadas na Seção 4.2.4.

O mecanismo de composição é responsável por processar estes modelos e realizar a propagação das informações contidas nos relacionamentos transversais, gerando, então o modelo ilustrado na Figura 33, o modelo complexo mostrado

PUC-Rio - Certificação Digital Nº 0210666/CB

anteriormente. Entretanto, com nossa abordagem, o responsável por desenhar e propagar mudanças nestes relacionamentos é o mecanismo de composição.

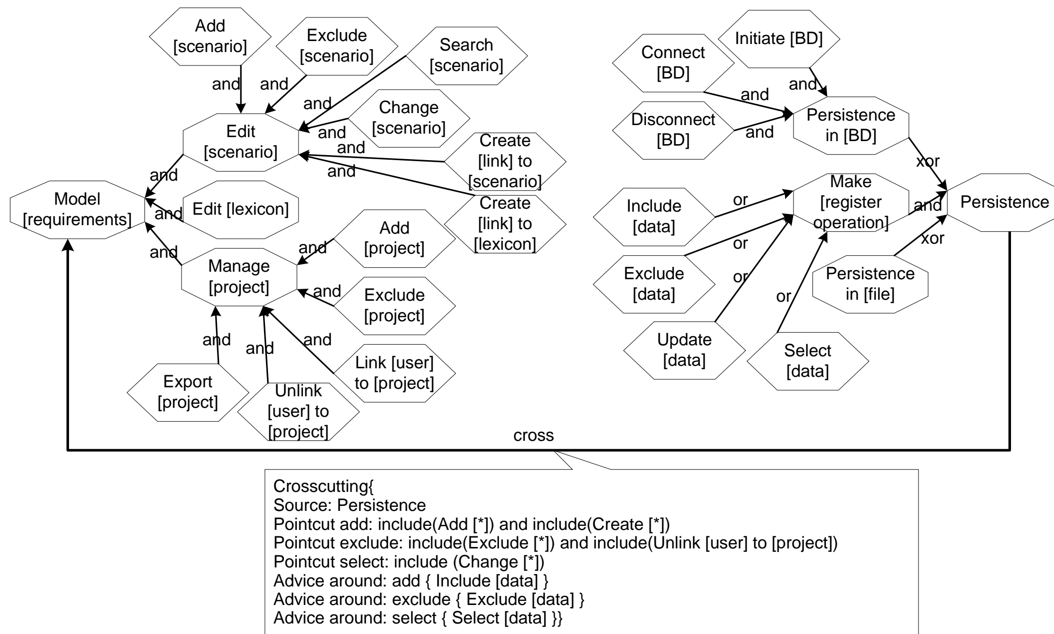


Figura 34. Exemplo ilustrativo - Separação

Além disso, com o mecanismo de visualização, o engenheiro de requisitos pode extrair diferentes visões, parciais ou totais do modelo integrado, por exemplo, matrizes de rastreabilidade, mapa de tópicos (*topic map)*, como a confidencialidade se espalha ou se entrelaça às demais características, dentre outras. Na Figura 35, mostramos uma visão parcial do modelo integrado (ilustrado na Figura 33), focando o serviço de persistência.

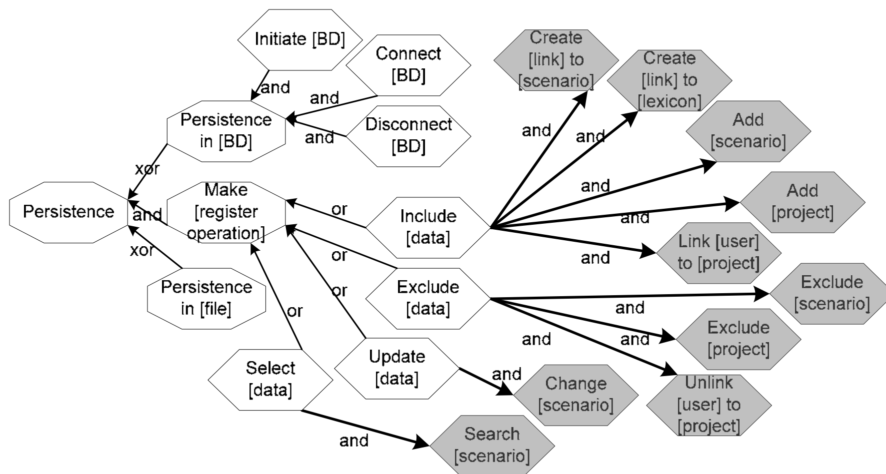


Figura 35. Exemplo ilustrativo - Visualização

PUC-Rio - Certificação Digital Nº 0210666/CB

Por deixar a informação de transversalidade centralizada no relacionamento transversal e deixar a cargo do mecanismo de composição espalhar esta informação:

- facilitamos a **modelagem** e **análise**, pois o engenheiro de requisitos pode se concentrar em um conjunto de requisitos por vez, não precisando manipular modelos grandes e com muitas interações. Isto diminui a complexidade de cada modelo;
- promovemos o **reuso**, porque alguns grupos de requisitos podem ser reaproveitados em outros projetos, sendo necessário modificar, apenas, a maneira na qual eles afetam os demais grupos;
- facilitamos a **rastreabilidade** entre requisitos, pois as informações de transversalidade estão centralizadas no relacionamento transversal. Estas informações são mais difíceis de rastrear quando utilizando a abordagem tradicional porque estão relacionadas a características que afetam ou são afetadas por muitas outras; e
- provemos facilidades para **modificar** os requisitos. Visto que eles foram modelados separadamente e o impacto que eles causam também está separado, é mais fácil modificar ambas as informações, pois fica a cargo do mecanismo de composição propagar estas mudanças.

No Capítulo 5, ilustramos estas contribuições nos estudos de caso. A seguir, na Seção 4.2, apresentamos LMROA e detalhamos seus modelos constituintes.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.2.
Linguagem de Modelagem de Requisitos Orientada a Aspectos

Estruturalmente, a linguagem de modelagem de requisitos orientada a aspectos (LMROA) consiste das seguintes partes: 1) a definição dos elementos da linguagem de modelagem de requisitos, i.e., o modelo de componentes da linguagem; 2) a definição dos elementos do modelo núcleo, neste caso, retratados pelo relacionamento transversal; e 3) o modelo de *joinpoints*.

```
1. <LMROA>:= <model>
2. <model>:= <component><relationship>
3. <relationship>:= <crosscuttingRel> | relacionamentos do modelo de componentes
4. <component>:= componentes do modelo de componentes
5. <joinpoint>:= elementos do modelo de componentes
```

Figura 36. Sintaxe de LMROA

Na Figura 36 e Figura 37, ilustramos os detalhes destes modelos. Os elementos em cinza representam os pontos fixos da linguagem, enquanto os elementos em branco representam pontos a serem instanciados.

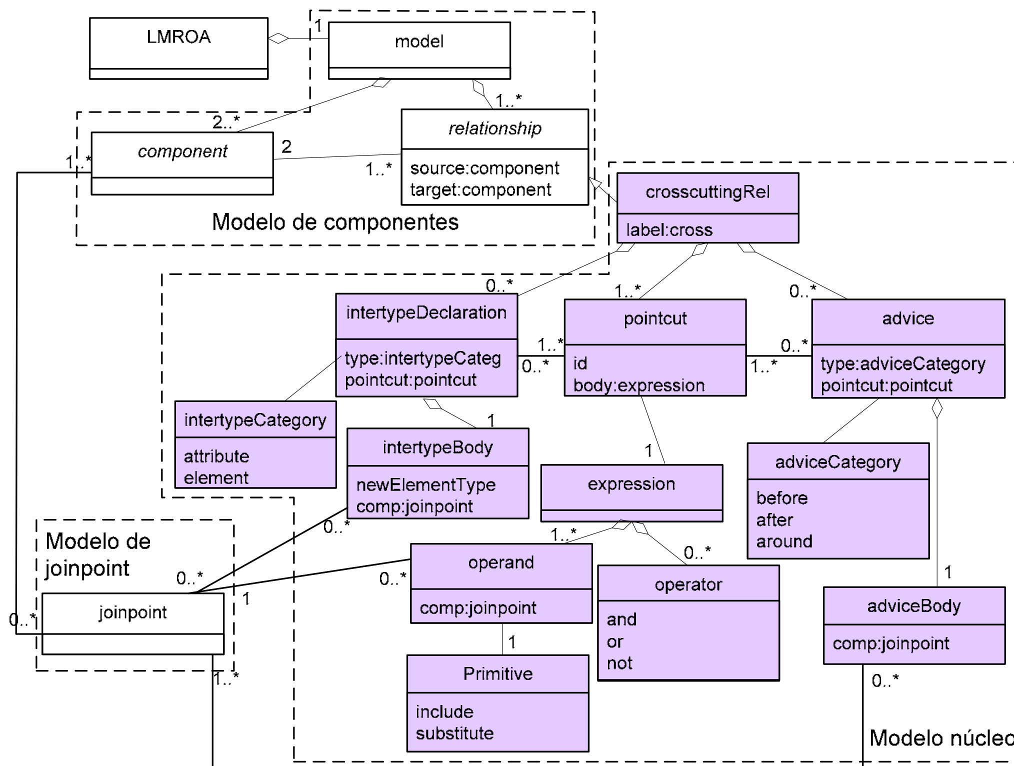


Figura 37 – Modelo conceitual da linguagem de modelagem de requisitos orientada a aspectos

4.2.1. Modelo de Componentes

No modelo conceitual de LMROA (Figura 37), o modelo de componentes representa uma linguagem de modelagem de requisitos descrita por componentes e relacionamentos (linhas 1 e 2 da Figura 36). Desta forma, *relationship* e *component* são elementos que devem ser instanciados (linhas 3-4 da Figura 36) de acordo com o modelo de requisitos a ser adotado, para dar origem a uma nova linguagem de modelagem de requisitos orientada a aspectos. Alguns exemplos de modelos de requisitos que podem ser instâncias de nosso modelo de componentes são os modelos de cenários, casos de uso, léxico, e os modelos de dependência e razão estratégica do i*, dentre outros. Na Seção 4.3, mostramos como instanciar o modelo de componentes.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.2.2. Modelo de *Joinpoints*

Joinpoints são elementos do modelo de componentes que podem ser afetados por aspectos (Chavez, 2003, 2004). Em linguagens de implementação, eles podem ser estáticos ou dinâmicos: um *joinpoint* estático é uma localização na estrutura de um elemento, enquanto um *joinpoint* dinâmico é uma localização na execução de um programa. Visto que, na maioria das vezes, linguagens de modelagem de requisitos são especificações não executáveis, i.e., estáticas, utilizamos apenas *joinpoints* estáticos.

Na Figura 36, o construto *joinpoint* (na linha 5) não tem valor definido porque depende do modelo de componentes a ser utilizado. Na Seção 4.3, instanciamos o modelo de *joinpoints* para o V-graph. Na descrição do relacionamento transversal, a seguir, referimos estes elementos por *joinpoint*.

4.2.3. Modelo Núcleo: Relacionamento Transversal

O relacionamento transversal representa, em nossa linguagem, o que o aspecto representa em AspectJ. Ele é um novo tipo de relacionamento a ser adicionado ao modelo de componentes (linha 3 da Figura 36). Optamos por não utilizar o conceito de Aspecto (Kiczales, 2001) ou candidato a aspecto (Rashid, 2002) como uma entidade de primeira ordem porque consideramos que desta maneira:

1) a extensão realizada é menos intrusiva à linguagem de modelagem de requisitos escolhida. Assim, as técnicas e diretrizes já definidas para a linguagem de modelagem continuam sendo válidas para todos os requisitos do sistema, podendo ou não estar entrelaçados e espalhados;

2) retardamos a decisão e descrição das informações relativas à transversalidade de requisitos. Isto significa que não é necessário saber a priori que uma característica é transversal. Quando percebemos esta propriedade, substituímos alguns relacionamentos por um relacionamento transversal, mas a estrutura e organização dos requisitos permanecem as mesmas. Por exemplo, a Figura 33, apresentada anteriormente, poderia ter sido criada pelo engenheiro, que ao perceber o espalhamento das atividades de "Persistence" modificaria o modelo,

PUC-Rio - Certificação Digital Nº 0210666/CB

trocando as dez contribuições por um relacionamento transversal (mostrado na Figura 34).

3) facilitamos o reuso de características que em alguns projetos têm esta propriedade de transversalidade e em outros não a têm, i.e, consideramos que características transversais, na verdade, “estão” transversais. Por isso, representamos todas elas, transversal ou não, da mesma maneira, o que diferencia é se ela interage com outras características usando um relacionamento transversal ou não; e

4) separamos as informações referentes a “o quê” é transversal, representada pelos elementos do modelo de componentes, das referentes a “como” é transversal, representadas no relacionamento transversal.

Como ilustrado na Figura 37, o relacionamento transversal tem o rótulo “cross” e herda as informações de *origem* e *destino* (*pointcuts*) de *relacionamento*. Além destas, a informação referente à transversalidade é descrita por meio de *pointcuts*, *advice* e *intertype declarations*.

PUC-Rio - Certificação Digital Nº 0210666/CB

```
1.  <crosscuttingRel> := crosscutting <crosscutting_label>{
2.                  source=<joinpoint>  <pointcut><advice><intertype_declaration>}
3.  <pointcut> := pointcut (<name>; <pointcut_id>) : <pointcut_expression> |
4.          <pointcut> <pointcut>
5.  <pointcut_expression> := <operand> |
6.                  not <operand> |
7.                  <pointcut_expression> and <pointcut_expression> |
8.                  <pointcut_expression> or <pointcut_expression>
9.  <operand> := <primitive> (<joinpoint>) | <primitive> (<regular_expression>)
10. <primitive> := include | substitute
11. <regular_expression> := <value>; “<joinpoint>”; <atribute_type>; <path>
12. <atribute_type> := id | name
13. <advice> := advice <advice_type>: <a_it_expression> {<advice_body>} |
14.         <advice> <advice>
15. <advice_type> := after | before | around
16. <a_it_expression> := <pointcut_id> |
17.             not <pointcut_id> |
18.             <a_it_expression> and <a_it_expression> |
19.             <a_it_expression> or <a_it_expression>
20. <intertype_declaration> := intertype <intertype_type>: <a_it_expression>
21.                         {<intertype_declaration_body>} |
22.             <intertype_declaration> <intertype_declaration>
23. <intertype_type> := attribute | element
24. <advice_body> := <joinpoint> | <advice_body>; <advice_body>
25. <intertype_declaration_body> := new_element_type {<new_element_type>} |
26.                         <joinpoint> |
27.                         <intertype_declaration_body> ; <intertype_declaration_body>
28. <new_element_type> := <name>=<value> | <new_element_type>; <new_element_type>
29. <crosscutting_label>:=cross
```

Figura 38. Sintaxe do relacionamento transversal

Na Figura 38, apresentamos a sintaxe do relacionamento transversal e a seguir detalhamos cada um de seus construtos. Utilizamos os nomes *pointcut*, *advice* e *intertype declaration* para ajudar o leitor a fazer uma associação direta a

elementos conhecidos no nível de desenho e implementação. Entretanto, para nossa abordagem, adaptamos a semântica destes elementos para que eles representem a transversalidade de características no nível de engenharia de requisitos.

Pointcuts

Pointcuts indicam os elementos afetados por uma determinada característica. Cada *pointcut* é definido por um nome e uma expressão (linhas 3-4 da Figura 38). Expressões consistem em sentenças que usam operandos, operadores e primitivas (linhas 5-8). Os operandos são elementos cujos tipos estão associados aos *joinpoints* (linha 9). Os operadores são utilizados para agrupar um ou mais *joinpoints*, eles são OR, AND e NOT.

As primitivas definem uma ação a ser realizada no *pointcut* (linha 10). Em nosso caso, definimos os seguintes tipos de primitivas: *include* - adiciona um conjunto de elementos ao local especificado; e *substitute* - substitui os elementos especificados. A primitiva *substitute* pode ser utilizada para apagar elementos quando o corpo do *advice* ou *intertype declaration* associado é vazio.

Na Figura 39, ilustramos três *pointcuts*, denominados *add*, *exclude* e *select*. Cada expresssão regular representa um operando associado à primitiva *include*. Nas expressões regulares é possível definir: a expressão (<value>); qual o tipo de elemento (“<joinpoint>”), neste caso são os tipos definidos em *joinpoints*; qual o atributo do elemento (<atribute_type>), podendo ser o identificador ou nome; e opcionalmente, qual o caminho (<path>) dos elementos a serem consultados. Operadores *and* são utilizados para agrupar os operandos.

Nome do pointcut   Primitiva   Expressão regular   Operador   Identificador

```
Pointcut add: include(Add [*]) and include(Create [*])
Pointcut exclude: include(Exclude [*]) and include(Unlink [user] to [project])
Pointcut select: include (Change [*])
```

Figura 39. Exemplo de *pointcut*

Advice

Um *advice* define quais elementos do modelo de origem se espalham ou se entrelaçam nos *pointcuts*. Ele registra o conjunto de elementos que representa um comportamento ou estrutura que se repete. Cada *advice* é formado por um tipo

PUC-Rio - Certificação Digital Nº 0210666/CB

(*after*, *before* e *around*) que indica a maneira como ele afeta os *pointcuts*; uma expressão de *pointcuts,* indicando quais são os *pointcuts* onde ele será aplicado; e o corpo que define o conjunto de elementos que se espalha ou se entrelaça nos *pointcuts*.

Na Figura 40, ilustramos três exemplo de *advice* do tipo *around.* O corpo de cada *advice* define os identificadores dos *pointcuts* (*add*, *exclude* e *select*) a serem afetados e as operações a serem incluídas ou substituídas (Include [data], Exclude [data] e Select [data], respectivamente).

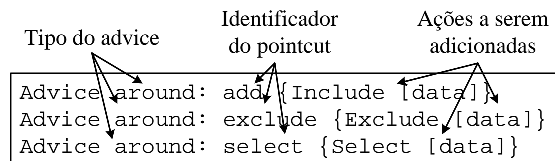


Figura 40. Exemplo de *advice*

Intertype declarations

Intertype declarations são utilizadas para adicionar novos tipos de dados (*attribute*) ou de elementos (*element*) ao modelo de requisitos escolhido, indicando que estes novos elementos são decorrentes da junção de duas características. Utilizar *intertype declarations* oferece informações para o mecanismo de visualização sobre novas visões a serem geradas, porque representam o interesse por estes novos elementos e pela interação deles com os demais.

Assim como o *advice*, cada *intertype declaration* é definida através de uma expressão de *pointcuts* indicando onde ela será aplicada, e um corpo que define os novos tipos de dados ou elementos a serem incluídos. Na Figura 41, ilustramos uma *intertype declaration*. Assim como no *advice*, o corpo da *intertype declaration* define os identificadores dos *pointcuts* a serem afetados (*add*, *exclude* e *select*) e novos tipos a serem incluídos no modelo (*actor=participant_user*).

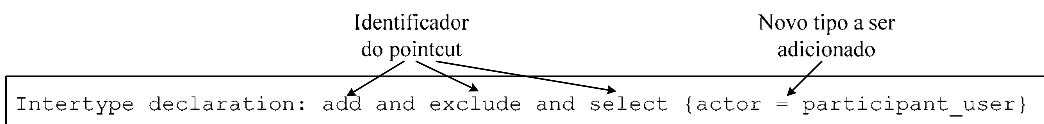


Figura 41. Exemplo de *intertype declaration*

PUC-Rio - Certificação Digital Nº 0210666/CB

4.2.4.
Diretrizes para Modelagem de Características Separadamente

A identificação de características transversais é na maioria das vezes realizada com base na experiência e intuição do desenvolvedor (Sousa, 2004b), tanto durante a definição de requisitos quanto nas atividades de desenho e implementação. Nesta tese, consideramos que qualquer característica pode ou não "estar" transversal. Não é necessário saber disto com antecedência porque elas são modeladas da mesma maneira (utilizando os elementos do modelo de componentes). É necessário sim, saber quando separá-las e quando utilizar o relacionamento transversal:

- Quando separá-las ? - qualquer característica que atenda a pelo menos uma das propriedades a seguir pode ser modelada separadamente:
  - Ela está repetida muitas vezes no modelo, ou possui alto *fan-in*;
  - Sua existência é decorrente da integração de requisitos semanticamente distintos;
  - Ela representa um ponto de variabilidade, podendo ou não estar presente no sistema, sem impedir a realização de seu objetivo global;
  - Ela pode ser reutilizada em outros domínios;
  - Ela é muito complexa, estando associada a muitas variáveis;
  - Ela é independente das demais, representando um conceito bem delimitado;
- Quando utilizar o relacionamento transversal ? - de maneira geral, nos três primeiros casos acima é recomendado o uso de relacionamentos transversais. Contudo, nossa abordagem considera que estas situações surgem durante o processo de modelagem, conforme a informação sobre o domínio e os requisitos é adquirida. Isto significa que, nem toda característica separada interage por meio de um relacionamento transversal com as demais. Por exemplo, quando a separação ocorre devido a apenas um dos três últimos casos, os relacionamentos providos pelo próprio modelo de componentes podem ser suficientes.

A seguir, na Seção 4.3, apresentamos como utilizar a estratégia definida na Seção 4.1, instanciando os pontos flexíveis de LMROA.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.3. Utilização da Estratégia – O V-Graph como Modelo de Componentes

Como relatamos na Seção 4.1, nossa estratégia para modelagem de requisitos é centrada na utilização de LMROA. LMROA é definida pelo modelo núcleo (apresentado na Seção 4.2), modelo de *joinpoints* e modelo de componentes. Nesta seção apresentamos como instanciar os elementos do metamodelo de integração, utilizando o V-graph como modelo de componentes de LMROA. Nas Seções 4.3.1, 4.3.2 e 4.3.3, definimos, respectivamente, as atividades de Separação, Composição e Visualização para esta instância.

4.3.1. Separação de Características Transversais

Nesta seção, considerando o modelo de requisitos V-graph, descrevemos uma instância de LMROA, denominada AOV-graph, e algumas diretrizes para utilização desta instância. Assim, AOV-graph consiste no modelo V-graph estendido, permitindo, além da descrição de *softmetas*, metas e tarefas, o controle dos problemas de entrelaçamento e espalhamento. Esta extensão consiste em adicionar o relacionamento transversal ao V-graph, e assim, uma nova maneira de pensar na separação e representação de RFs e RNFs (de características).

4.3.1.1. A Linguagem

Na Figura 42 e Figura 43 (em branco), apresentamos os elementos e relacionamentos do modelo *V-graph,* instância dos elementos de LMROA. Os elementos *softmetas*, *metas* e *tarefas* possuem *nomes*, *tipos* e *tópicos*. O relacionamento de decomposição, com os rótulos *And* e *Or*, e o de contribuição, com os rótulos *make*, *help*, *unknown*, *hurt*, e *break* estão representados através da agregação em *component*. Desta forma, cada componente tem um atributo denominado "decomposition_label" que indica qual o rótulo do relacionamento entre ele e seu componente pai (linhas 5, 7 e 9 da Figura 43). O relacionamento de correlação é definido através de *relCorrelation*.

PUC-Rio - Certificação Digital Nº 0210666/CB

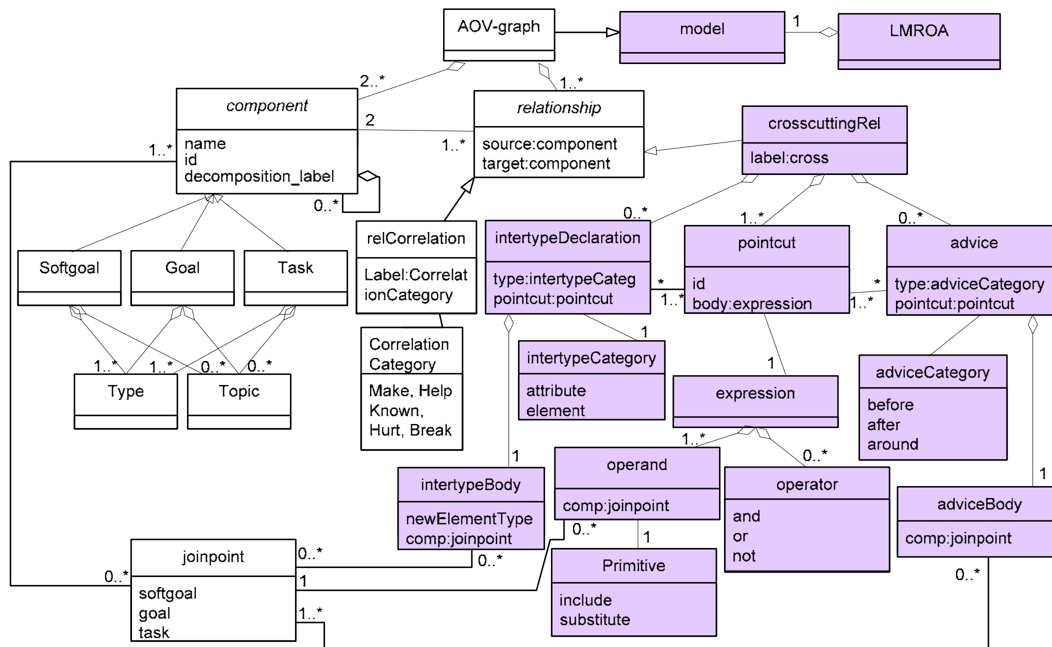


Figura 42. Instanciação de LMROA: AOV-graph

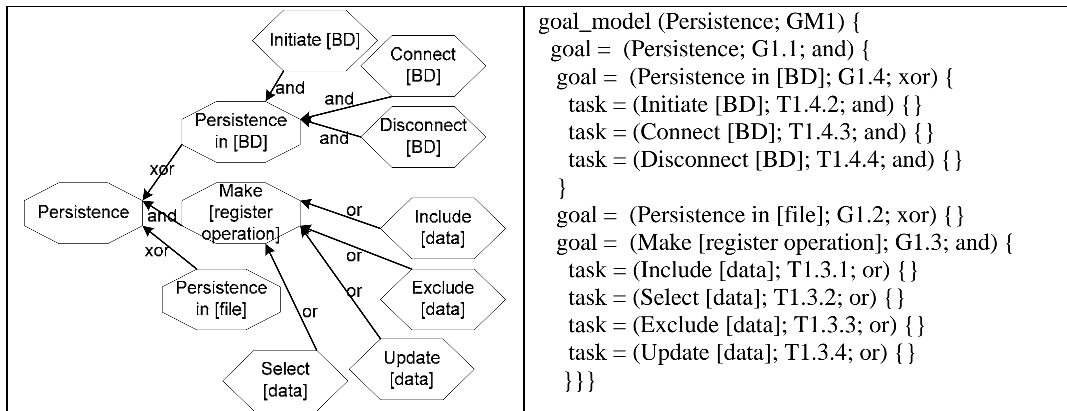
```
1.  <goal_model> := goal_model (<name>; <id>){<component> <relationship>} |
2.                  <goal_model> <goal_model>
3.  <component> := <softgoal> | <goal> | <task> |
4.                 <component> < component >
5.  <softgoal>:=softgoal(<name>;<softgoal_id>;<decomposition_label>;(<type>);(<topic>))
6.                 {<component><relationship>}
7.  <goal> := goal (<name>; <goal_id>; <decomposition_label> ; (<type>) ; (<topic>))
8.                 {<component><relationship>}
9.  <task> := task (<name>; <task_id>; <decomposition_label> ; (<type>) (<topic>))
10.                {<component><relationship>}
11. <topic> := <name> | <topic>; <topic>
12. <type> := <name> | <type>; <type>
13. <relationship> := < relcorrelation > | <crosscuttingRel> |
14.                 <relationship> <relationship>
15. <relcorrelation> :=
16.        correlation <correlation_label> {source=<goal_ref> target=<softgoal_ref>} |
17.        correlation <correlation_label> {source=<softgoal_ref> target=<softgoal_ref>} |
18.        correlation <correlation_label> {source=<task_ref> target=<task_ref>}
19. <correlation_label> := break | hurt | unknown | help | make
20. <decomposition_label>:= and | or | xor | break | hurt | unknown | help | make
21. <goal_ref> := <goal_id> <decomposition_label>
22. <softgoal_ref> := <softgoal_id> <decomposition_label>
23. <task_ref> := <task_id> <decomposition_label>
```

Figura 43. Sintaxe do *V-graph*

Exemplo

Na Figura 44, ilustramos o AOV-graph para o serviço de Persistência; em a) ilustramos sua representação gráfica e em b) sua descrição textual. Neste exemplo, vemos que “Persistence in [BD]” e “Persistence in [file]” são decomposições XOR de “Persistence”, enquanto “Make [register operation]” é uma decomposição AND.

PUC-Rio - Certificação Digital Nº 0210666/CB




```
goal_model (Persistence; GM1) {
  goal =  (Persistence; G1.1; and) {
   goal =  (Persistence in [BD]; G1.4; xor) {
     task = (Initiate [BD]; T1.4.2; and) {}
     task = (Connect [BD]; T1.4.3; and) {}
     task = (Disconnect [BD]; T1.4.4; and) {}
   }
   goal =  (Persistence in [file]; G1.2; xor) {}
   goal =  (Make [register operation]; G1.3; and) {
     task = (Include [data]; T1.3.1; or) {}
     task = (Select [data]; T1.3.2; or) {}
     task = (Exclude [data]; T1.3.3; or) {}
     task = (Update [data]; T1.3.4; or) {}
   }}}
```

Figura 44. Exemplo do modelo V-graph a) representação gráfica e b) representação em textual

4.3.1.1.1. Modelo de *Joinpoints*

No caso do V-graph os *joinpoints* podem ser elementos dos tipos *softmeta*, meta e tarefa, veja Figura 45. As instâncias destes tipos podem ser referenciadas como *pointcuts*. Elas podem ser identificadas: diretamente pelo nome ou identificador do elemento; ou indiretamente por expressões regulares indicando tipos e tópicos.

```
<joinpoint> := <softgoal_ref> | <goal_ref> | <task_ref>
```

Figura 45. Definição dos *joinpoints* no caso do V-graph como modelo de componentes

Instâncias destes tipos também são utilizadas na descrição do corpo do *advice* e *intertype declaration*. Como estes tipos possuem o atributo *decomposition_label* (veja linhas 21, 22 e 23 da Figura 43) então é possível descrever em *advice* e *intertype* quais os rótulos dos novos relacionamentos. Isto é equivalente ao “How much” da abordagem apresentada em (Leite, 2005).

4.3.1.1.2. Modelo Núcleo: Relacionamento Transversal

Como apresentamos na Seção 4.2.3, os elementos *source*, *operand*, *advice_body* e *intertype_declaration_body* estão associados aos tipos definidos nos *joinpoints* (veja na Figura 38). A seguir, descrevemos a semântica destes elementos na instância AOV-graph.

PUC-Rio - Certificação Digital Nº 0210666/CB

Pointcuts

Pointcuts podem fazer referência a quaisquer elementos dos tipos *softmetas*, metas e tarefas. Nos *pointcuts* do AOV-graph, o operador OR indica que se os operandos referem elementos filhos do mesmo pai então apenas um dos operandos precisa ser modificado; enquanto o operador AND indica que todos devem ser atingidos; e o operador NOT exclui um determinado operando dentre os elementos especificados no *pointcut*.

Exemplo

Na Figura 46, ilustramos três exemplos de *pointcuts*: no primeiro utilizamos o operador AND para indicar que os dois *pointcuts* serão afetados; no segundo utilizamos o operador OR, indicando que caso o *pointcut* já tenham sido afetado pelo mesmo *advice* ou *intertype* então ele não precisa ser afetado novamente; e no terceiro *pointcut* utilizamos o operador NOT e expressões regulares, indicando que qualquer meta, *softmeta* ou tarefa que atende a expressão regular e que não seja a tarefa "Exclude [data]", será afetada.

Pointcut exclude_data: include(Exclude [project]) and include(Exclude [scenario]) and
include(Unlink [user] to [project])
Pointcut BD: include(include [data]) or include(delete [data]) or include(update [data]) or
include (select [data])
Pointcut exclude: include(Exclude [*]) and not include (Exclude [data])

Figura 46. Exemplo de *pointcuts*

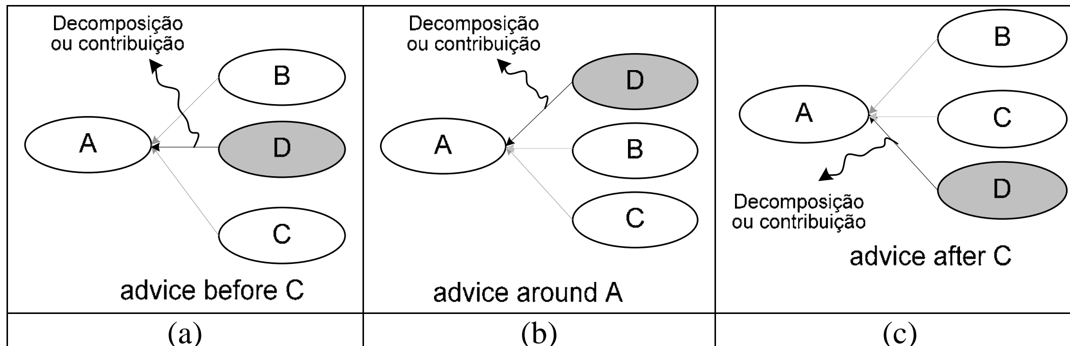
Advice

Cada *advice* define quais elementos do modelo de origem do relacionamento transversal se espalham ou se entrelaçam aos *pointcuts*. No caso do V-graph, podemos definir metas, *softmetas* e tarefas no *advice*. Como definimos na Seção 4.2.3, cada *advice* é um dos três tipos *before, after* ou *around*. Apesar do V-graph não realçar a seqüência de ações, normalmente tenta-se escrever o modelo da esquerda para direita ou de cima para baixo, indicando uma seqüência. Desta forma, utilizamos os tipos de *advice* para indicar:

- *Before* - inserir ANTES do *pointcut* como uma decomposição de seu elemento pai, veja a Figura 47(a);

PUC-Rio - Certificação Digital Nº 0210666/CB

- *Around* - inserir elementos como decomposições do elemento afetado, i.e., estes elementos são necessários para atingir o elemento afetado, veja Figura 47(b); e
- *After* - inserir DEPOIS do *pointcut* como uma decomposição de seu elemento pai, veja Figura 47(c).



(a)	(b)	(c)

Figura 47. Semântica dos tipos de *advice* a) before, b) around e c) after

Exemplo

No exemplo da Figura 48, cada *advice* indica quais elementos do modelo *Persistence* afetam os *pointcuts* indicados pelos nomes *BD,* "exclude_data" e "exclude". O primeiro *advice* indica a inclusão da tarefa "Exclude [data]" nos pontos indicados pelos pointcuts "exclude" e "exclude_data". O segundo *advice* indica que as tarefas "Initiate [BD]" e "Connect [BD]" devem acontecer antes dos pontos indicados pelo *pointcut* "BD". Como este *pointcut* é definido com o operador OR então estas duas operações serão incluídas apenas uma vez no mesmo nível da hierarquia. O terceiro *advice* indica que a tarefa "Disconnect [BD]" é incluída após os pontos definidos.

Advice around: exclude or exclude_data {Exclude [data]}
Advice before: BD {Initiate [BD]; Connect [BD];}
Advice after: BD {Disconnect [BD];}

Figura 48. Exemplo de *advice*

Intertype declarations

Intertype declarations são utilizadas para modificar a estrutura dos elementos do modelo de componentes. No V-graph, utilizamos *intertype declarations* para adicionar novos tipos de elementos ao modelo de componentes. Estes elementos podem ser novos atributos, tais como, atores, exceções, custo, dentre outros, ou podem ser novas tarefas, metas ou *softmetas*.

PUC-Rio - Certificação Digital Nº 0210666/CB

Exemplo

Na Figura 49, ilustramos dois exemplos de *intertype declaration.* No primeiro é definido um novo tipo de atributo denominado “exception”. Este elemento não pode ser visualizado na visão V-graph, mas pode ser utilizado por outros tipos de visão, tal como Cenário. No segundo exemplo, é definida uma nova tarefa denominada “Verify if [element] exists”, que é adicionada nos *pointcuts* e no elemento origem do relacionamento transversal.

intertype declaration attribute: exclude and exclude_data
{ exception = “This element doesn’t exist” ;
exception = “User doesn’t have permission”}
intertype declaration element: exclude and exclude_data
{ task = (Verify if [element] exists; T0; and) {}}

Figura 49. Exemplo de *intertype declarations*

4.3.1.2. Diretrizes para Modelagem Utilizando AOV-graph

No caso do AOV-graph, os relacionamentos de contribuição (indicam maior coesão) e correlação (indicam menor coesão) explicitam como os elementos podem ser modularizados. Assim, qualquer sub-árvore pode ser considerada uma característica modelada separadamente. Por exemplo, na Figura 50, a meta “Persistence in [BD]” com as tarefas necessárias para que ela seja satisfeita pode ser considerada uma característica separada das demais.

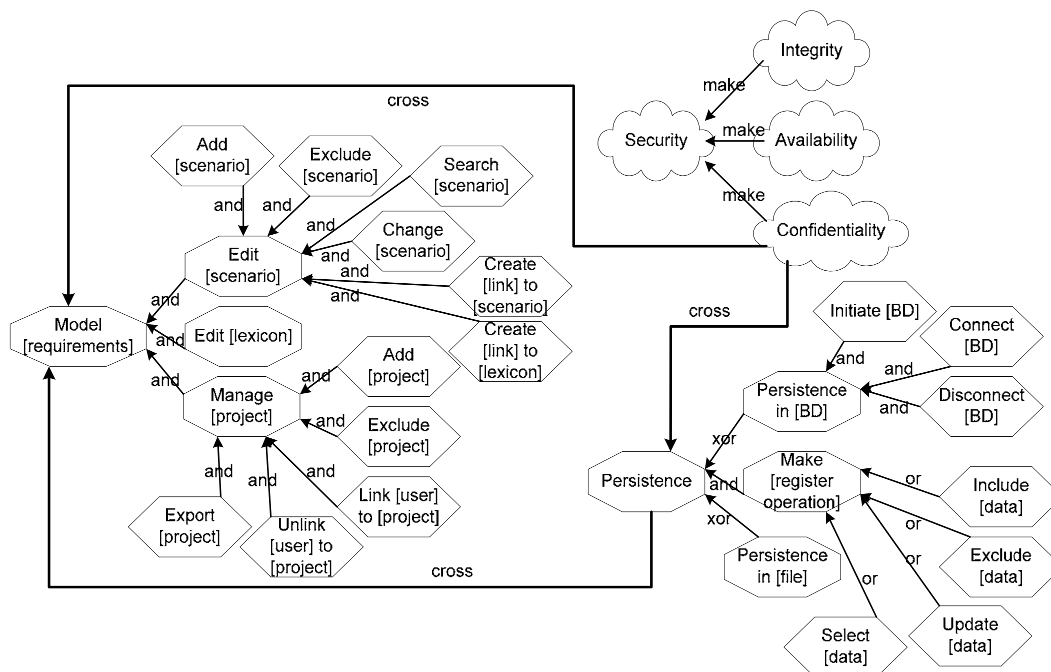


Figura 50. Exemplo da separação de modelos

PUC-Rio - Certificação Digital Nº 0210666/CB

Contudo, quando o fator decisivo para separação é somente um dos três últimos fatores citados na Seção 4.2.4 (alta complexidade, possibilidade de reuso, ou possuir baixo acoplamento com as demais), recomendamos que a separação seja realizada modelando cada característica como um modelo separado (um grupo de requisitos separado), por exemplo, os modelos "Persistence", "Security" e "Model [requirements]", ilustrados na Figura 50. Desta maneira é mais fácil delimitar o escopo de cada um.

O relacionamento transversal é fortemente indicado, no AOV-graph, quando *softmetas*, metas e tarefas:

- contribuem para satisfação de muitos elementos, indicando espalhamento. Por exemplo, na Figura 35, ilustramos que a tarefa "Include [data]" do serviço "Persistence" afeta cinco tarefas do serviço "Model [requirements]";
- contribuem para satisfação de elementos em árvores separadas (modelos separados), indicando entrelaçamento. Por exemplo, as tarefas relacionadas a meta "Make [register operation]" contribuem para as árvores relacionadas aos serviços de persistência e de modelar requisitos.

PUC-Rio - Certificação Digital Nº 0210666/CB

Cada *advice* e *intertype declaration* agrupa tarefas, metas e *softmetas* que afetam juntas um ou vários *pointcuts*. O número de relacionamentos representados por cada *advice* ou *intertype* é dado pela seguinte expressão (Nr = número de relacionamentos; Np = número de pontos a serem afetados; Ne = número de metas, *softmetas* ou tarefas definidas no corpo do *advice* ou *intertype*; Na = número de atibutos definidos no corpo do *advice* ou *intertype*):

- Nr = Np x Ne (no caso do *advice*);
- Nr = Np x Ne + 1 (no caso de *intertype* do tipo *element*);
- Nr = Np x Na + 1 (no caso de *intertype* do tipo *attribute*)

Desta forma, quanto mais elementos eles agrupam maior é o ganho obtido com o uso do relacionamento transversal porque há uma maior redução no número de relacionamentos que o engenheiro de requisitos precisa desenhar manualmente. Conseqüentemente, é mais fácil manipular os modelos e é maior a visibilidade de seus elementos.

Advice e intertype declarations são utilizadas nas seguintes situações:

- *Advice* é utilizado quando elementos do modelo de origem do relacionamento se espalham por outras características. Por exemplo, o relacionamento transversal de "Persistence" para "Model [requirements]"

representa relacionamentos de contribuição partindo das operacionalizações de "Persistence" para os elementos de "Model [requirements]", veja no exemplo mostrado na Figura 35.

- *Intertypes* (*element*) são utilizadas para criar novos elementos dos tipos tarefa, meta ou softmeta, que são oriundos da junção de características separadas, i.e., estes elementos não existem no modelo de origem do relacionamento; e
- *Intertypes* (*attribute*) são utilizadas para criar novos tipos de elementos, i.e., elementos que não são dos tipos meta, softmeta e tarefa. Com intertypes (*attribute*) criamos novas formas de separar, classificar e visualizar metas, *softmetas*, e tarefas. Neste caso, os relacionamentos criados não são visualizados no grafo de metas porque eles representam outra forma de decomposição.

Por definirem os elementos que se espalham e/ou entrelaçam, *advice* e *intertype* representam candidatos naturais a serem visualizados por meio de matrizes ou grafos, focando os relacionamentos nos quais eles fazem parte. Frente às vantagens oferecidas pelo relacionamento transversal, o engenheiro de requisitos deve avaliar qual o custo/benefício de utilizar relacionamentos de contribuição/correlação ou transversais.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.3.2. Composição de Características Transversais

A utilização do relacionamento transversal reduz a quantidade de relacionamentos entre metas, *softmetas* e tarefas porque "modulariza" muitas interações em poucos relacionamentos transversais. Esta modularização é possível porque muitas vezes um conjunto de relacionamentos é repetido para associar uma característica (origem) para muitos elementos (destino). Isto por si só diminui a complexidade do modelo ou, pelo menos, aumenta a visibilidade de seus elementos.

Visto que conseguimos agrupar relacionamentos, então é necessário um mecanismo de composição para processá-los. Além disto, *pointcuts* agrupam pontos atingidos da mesma maneira, *advice* e *intertype* agrupam estruturas que se repetem. Estes agrupamentos representam novas maneiras de modularização de elementos que estão espalhados e entrelaçados devido à decomposição dominante

do modelo de componentes, V-graph. Por exemplo, o *pointcut* "exclude_data" (Figura 46) agrupa todos os elementos afetados pela tarefa "Exclude [data]" de "Persistence in [DB]" (descrito no *advice* da Figura 48), bem como pela nova tarefa "Verify if [element] exists" e as exceções "This element doesn't exist" e "User doesn't have permission" (descritas no *intertype* da Figura 49).

Nesta Seção, descrevemos a atividade de composição de características transversais. Para esta etapa definimos um conjunto de regras de composição e implementamos um mecanismo para processá-las. As regras indicam como o mecanismo de composição transforma os modelos inicialmente criados em modelos integrados; elas representam a semântica dos construtos do relacionamento transversal e podem variar de acordo com o modelo de componentes utilizado.

Mecanismo de Composição

Em nossa abordagem, o mecanismo de composição é uma máquina de transformação guiada pelas regras semânticas do modelo para efetivar as mudanças que o relacionamento transversal descreve, i.e., ele é responsável por automaticamente espalhar e entrelaçar as características modeladas inicialmente de maneira separada. Ter um mecanismo automático para propagar informações num modelo de requisitos facilita a inclusão e exclusão de informações que podem estar separadas de acordo com o interesse do engenheiro de requisitos.

O mecanismo de composição recebe como entrada e gera como saída descrições estáticas de requisitos, i.e., modelos AOV-graph. A estratégia utilizada para composição não modifica o modelo AOV-graph inicial, ele cria um outro, desta maneira as versões inicial e integrada estão disponíveis. A composição ocorre sobre uma descrição estática e deve acontecer sempre que relacionamentos transversais são inseridos, modificados ou excluídos. Descrevemos a implementação do mecanismo de composição na Seção 4.3.4.2.

Regras de Composição

Na Tabela 2, apresentamos as regras de composição para AOV-graph. Há duas ações especificadas no relacionamento transversal por meio das primitivas, *include* e *substitute*. A combinação de operadores, primitivas, tipos de *advice* e *intertype declarations* determina a transformação a ser realizada em cada *pointcut*.

PUC-Rio - Certificação Digital Nº 0210666/CB

Tabela 2. Regras de composição

Operador	Semântica
AND	Afeta todos os operandos
OR	Afeta apenas um dos operandos se eles forem filhos do mesmo pai
NOT	Exclui o operando do conjunto de pontos a serem afetados

Primitiva	Tipo de *Intertype declaration* ou *advice*	Semântica
Include	*Advice* before	Inclui os elementos definidos no corpo do *advice* ANTES do *pointcut*, como filho de seu elemento pai
	Advice after	Inclui os elementos definidos no corpo do *advice* DEPOIS do *pointcut*, como filho de seu elemento pai
	Advice around	Inclui os elementos definidos no corpo do *advice* como filho do *pointcut*
	Intertype declaration element/attribute	Inclui os elementos definidos no corpo do *intertype* no elemento de origem do relacionamento transversal e no *pointcut*
Substitute	*Advice* Before/around/after	Substitui o elemento indicado no *pointcut* pelos elementos definidos no corpo do *advice*
	Intertype declaration element/attribute	Inclui os elementos definidos no corpo do *intertype* no elemento de origem do relacionamento transversal e no *pointcut*

Exemplo

Na Figura 51, apresentamos um exemplo de composição: (a) descrição inicial do modelo de metas, (b) relacionamento transversal e (c) resultado da composição, em negrito explicitamos os elementos adicionados por *advice* e *intertype declarations*.

```
goal_model (ModelRequirements; GM2) {
  goal = (Model [requirements]; G2.1; and) {
   goal = (Edit [scenario]; G2.4; or) {
    task = (Add [scenario]; T2.4.2; and) {}
    task = (Exclude [scenario]; T2.4.3; and) {}
    task = (Search [scenario]; T2.4.4; and) {}
    task = (Change [scenario]; T2.4.4; and) {}
    task = (Create [link] to [scenario]; T2.4.4; and) {}
    task = (Create [link] to [lexicon]; T2.4.4; and) {}
   }
   goal = (Edit [lexicon]; G2.2; or) {}
   goal = (Manage [project]; G2.3; and) {
    task = (Add [project]; T2.3.1; and) {}
    task = (Exclude [project]; T2.3.2; and) {}
    task = (Link [user] to [project]; T2.3.3; and) {}
    task = (Unlink [user] to [project]; T2.3.3; and) {}
    task = (Export [project]; T2.3.4; and) {}
} }}
```

(a)

```
Crosscutting {
Source = Persistence
Pointcut exclude_data: include(Exclude [project]) and
        include(Exclude [scenario) and include(Unlink [user] to [project])
Pointcut exclude: include(Exclude [*]) and not include (Exclude [data])
Advice around: exclude or exclude_data {(Exclude [data]; T1.3.3; and)}
Intertype declaration attribute: exclude and exclude_data
        { exception = "This element doesn't exist" ;
          exception = "User doesn't have permission"}
intertype declaration element: exclude and exclude_data
        { task = (Verify if [element] exists; T0; and) }
}
```

(b)

```
Goal_model (ModelRequirements; GM2) {
 goal = (Model [requirements]; G2.1; and) {
  goal = (Edit [scenario]; G2.4; or) {
   task = (Add [scenario]; T2.4.2; and) {}
   task = (Exclude [scenario]; T2.4.3; and) {
     task = (Exclude [data]; T1.3.3; and) {}
     task = (Verify if [element] exists; T0; and)
     exception = "This element doesn't exist" ;
     exception = "User doesn't have permission"}
   task = (Search [scenario]; T2.4.4; and) {}
   task = (Change [scenario]; T2.4.4; and) {}
   task = (Create [link] to [scenario]; T2.4.4; and) {}
   task = (Create [link] to [lexicon]; T2.4.4; and) {}
  }
  goal = (Edit [lexicon]; G2.2; or) {}
  goal = (Manage [project]; G2.3; and) {
   task = (Add [project]; T2.3.1; and) {}
   task = (Exclude [project]; T2.3.2; and) {
     task = (Exclude [data]; T1.3.3; and) {}
     task = (Verify if [element] exists; T0; and)
     exception = "This element doesn't exist" ;
     exception = "User doesn't have permission"}
   task = (Link [user] to [project]; T2.3.3; and) {}
   task = (Unlink [user] to [project]; T2.3.3; and) {
     task = (Exclude [data]; T2.3.3; and) {}
     task = (Verify if [element] exists; T0; and)
     exception = "This element doesn't exist" ;
     exception = "User doesn't have permission"}
   task = (Export [project]; T2.3.4; and) {}
}}}
```

(c)

Figura 51. Exemplo de composição

PUC-Rio - Certificação Digital Nº 0210666/CB

Nas Seções seguintes, utilizamos o exemplo ilustrado na Figura 52 como base para demonstrar algumas visões extraídas das informações descritas no V-graph.

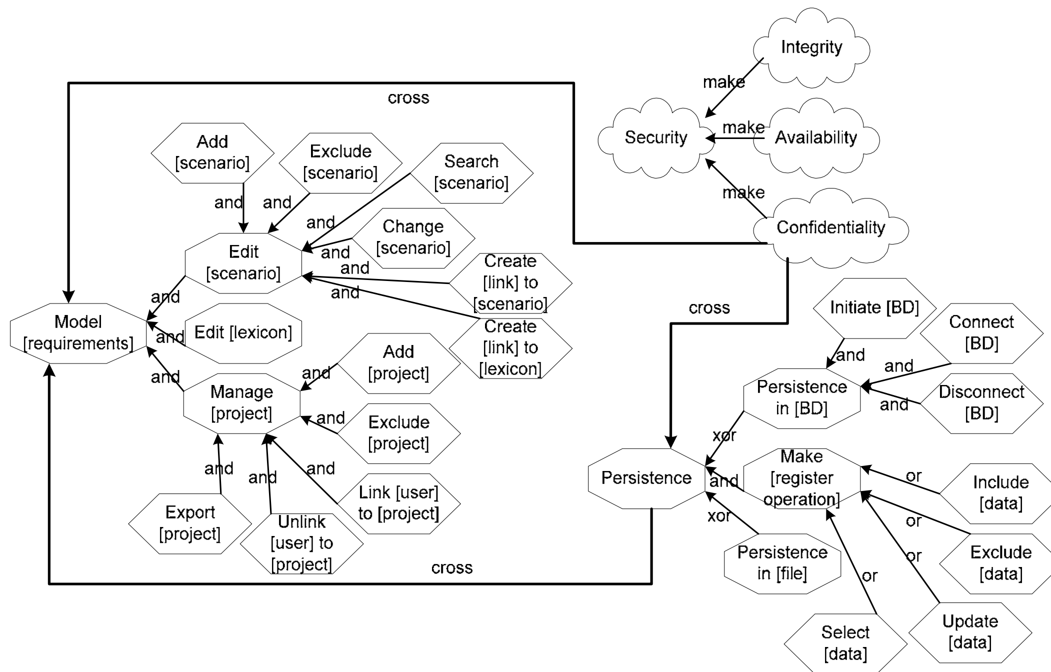


Figura 52. Exemplo base para extração de visões

4.3.3. Visualização de Características Transversais

Para a atividade de visualização definimos quais visões podem ser extraídas de informações modeladas em AOV-graph e as respectivas regras de transformação. A utilização de visões tem sido adotada em diferentes fases do desenvolvimento de software. Ela ajuda o desenvolvedor a diminuir o escopo de um problema e assim, sua complexidade, buscando a completude segundo algum interesse. Durante o processo de requisitos, bem como de projeto, i.e., durante a elaboração de soluções, é importante que o desenvolvedor possa adquirir diferentes visões do modelo base. Desta forma, ele pode analisar por diferentes ângulos a solução sendo criada.

Como definimos no Capítulo 2, uma visão é uma representação do todo ou de parte de uma arquitetura, segundo o foco em uma ou mais características, por um ou mais interessados. Desta forma, consideramos que um mecanismo de visualização é fundamental para a modelagem de requisitos. Um mecanismo

PUC-Rio - Certificação Digital Nº 0210666/CB

automático para gerar visões pode acelerar o processo de modelagem, porque diminui o re-trabalho e as inconsistências entre diferentes modelos.

Nosso mecanismo de visualização consiste de um componente de transformação que precisa dos seguintes conjuntos de informação: a sintaxe e semântica das linguagens de origem (AOV-graph) e de destino (representação a ser gerada), e as regras de transformação de uma linguagem para a outra, veja Figura 53.

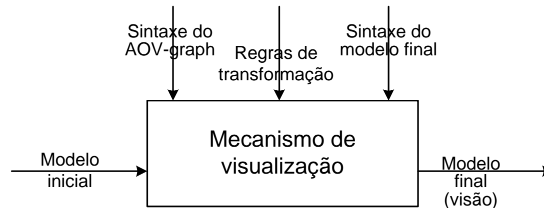


Figura 53. Mecanismo de visualização

Definimos um conjunto de visões que podem ser extraídas de AOV-graph. Estas visões compõem um conjunto mínimo de visões desejadas durante o processo de requisitos. Nas Seções 4.3.3.1 e 4.3.3.2, expomos, em duas categorias, as visões que compõem nosso mecanismo de visualização: visões com a conotação de serviço e visões no sentido de modelos, respectivamente. Estas categorias não são disjuntas, i.e. nós também podemos gerar visões no sentido de modelos apenas para alguns serviços. Na Figura 54, apresentamos, em um modelo de *features* (Czarnecki, 2000) , a variabilidade de possíveis visões.

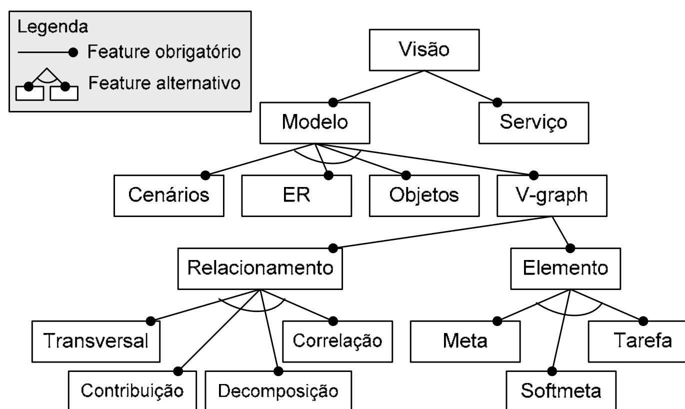


Figura 54. Modelo de *features* para configuração de visões de AOV-graph

4.3.3.1. Visões como Serviços

Como definido no Capítulo 2, visões como serviços representam partes do sistema que se complementam, tais como visão de segurança, visão de pagamento,

PUC-Rio - Certificação Digital Nº 0210666/CB

visão de características funcionais, dentre outras. Neste caso, temos **visões parciais** da modelagem, focando uma ou um conjunto de características por vez. No caso do AOV-graph, observamos a importância das seguintes visões:

1. Visão sem alguns tipos de relacionamento – em algumas situações o engenheiro de requisitos necessita observar, no modelo do sistema, os elementos que estão relacionados apenas por decomposição, por contribuição, por correlação ou pelo relacionamento transversal, podendo ser necessárias também visões que realcem os elementos relacionados com apenas um dos rótulos destes tipos de relacionamento. Na Figura 55, ilustramos um exemplo de AOV-graph com os relacionamentos de decomposição ocultados.

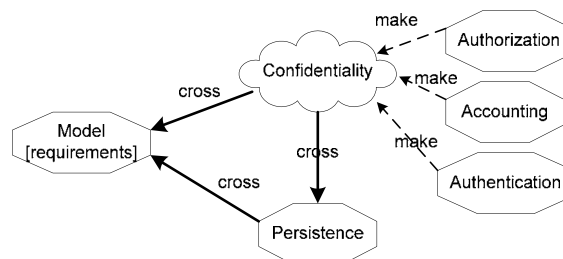


Figura 55. Exemplo de visão com os relacionamentos transversais resumidos e sem o relacionamento de decomposição

2. Visão com os relacionamentos transversais resumidos – visto que os relacionamentos transversais agrupam muitas contribuições e correlações, então quando eles estão expandidos a visibilidade da dependência entre elementos é maior, porém a visibilidade dos elementos é menor. Desta forma, é necessário ter uma visão onde os relacionamentos transversais apareçam resumidos, veja Figura 55.

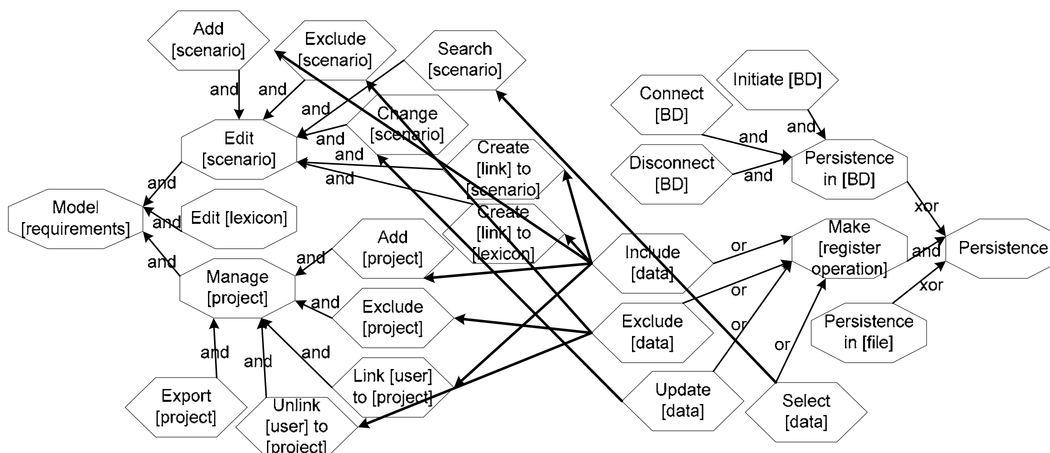


Figura 56. Exemplo de visão com o relacionamento transversal expandido

PUC-Rio - Certificação Digital Nº 0210666/CB

3. Visão com os relacionamentos transversais expandidos – os relacionamentos transversais modificam a modelagem do sistema por gerar novos relacionamentos, aumentando a visibilidade do impacto que os elementos exercem uns sobre os outros. Desta forma, uma visão mostrando o entrelaçamento e espalhamento dos elementos do sistema é necessária, veja a Figura 56.

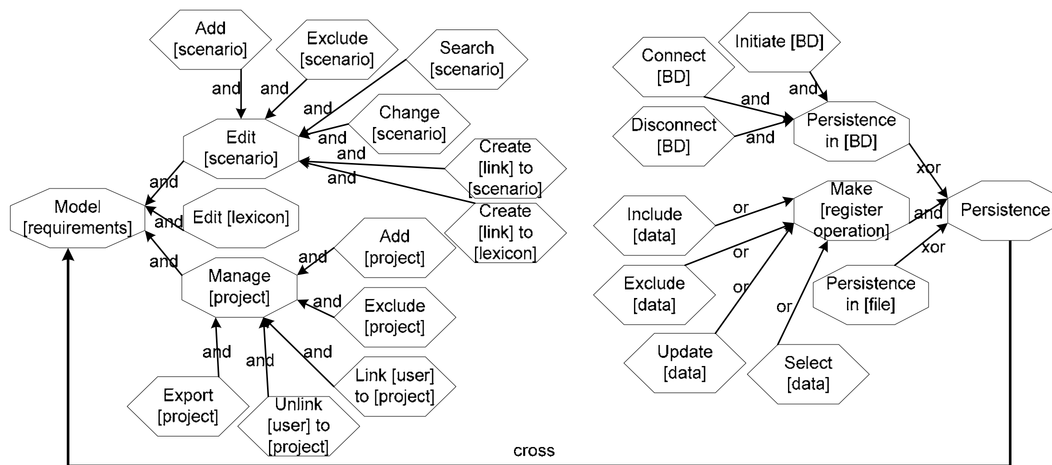


Figura 57. Exemplo de visão apenas com os elementos dos tipos meta e tarefa

4. Visão de um tipo específico de componente – cada tipo de elemento no AOV-graph representa um nível de abstração diferente. Uma visão focando apenas *softmetas*, metas ou tarefas ou combinações destes é interessante para que o engenheiro de requisitos possa analisar os relacionamentos existentes considerando o mesmo nível de abstração, veja Figura 57.
5. Visão de uma instância de *softmeta*, meta, tarefa, tipo ou tópico – cada sistema oferece serviços que devem ser analisados um a um de maneira que se possa garantir que ele será coberto pelo sistema sendo desenvolvido. Desta maneira, é importante ter visões com escopo delimitado por elementos relacionados a uma instância de *softmeta*, meta, tarefa, tipo ou tópico. A Figura 58, ilustra o modelo com foco no serviço “Persistence”, os elementos em cinza são elementos de outros modelos afetados por “Persistence”.

PUC-Rio - Certificação Digital Nº 0210666/CB

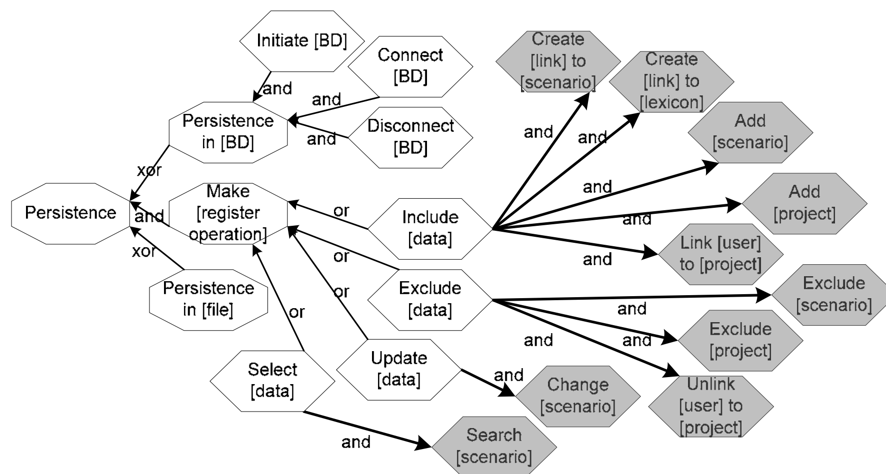


Figura 58. Exemplo de visão de uma instância de *softmeta*, meta, tarefa, tipo ou tópico

A combinação destas visões gera outras visões que são igualmente importantes. Por exemplo, uma visão que ilustre todos os elementos do tipo *softmeta* e meta que interagem por relacionamentos transversais, dentre outras. Poder fazer diferentes combinações de elementos e relacionamentos a serem focados fornece ao engenheiro de requisitos maior facilidade de entender, criar verificar e validar o modelo de requisitos.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.3.3.2. Visões como Modelos

Como definido no Capítulo 2, visões como modelos são representações cujos elementos realçam as características do sistema. De maneira geral, consideramos que as visões descritas nesta seção representam visões totais, representando a modelagem do sistema inteiro segundo a sintaxe e semântica de um tipo de modelo.

O *V-graph* é uma representação na qual podemos modelar requisitos funcionais e não funcionais explicitamente através de *softmetas*, metas e tarefas. Esta é sua forma de decomposição dominante. Entretanto, os tópicos destes elementos nos fornecem uma nova perspectiva do modelo, baseada em dados, assim como os relacionamentos nos fornecem uma perspectiva de interação, ou rastreabilidade, dentre outros. Utilizando estas informações definimos regras para transformar as informações modeladas em AOV-graph nos modelos: cenários (Leite, 1997), matriz de rastreabilidade (Sommerville, 2005), modelo entidade-relacionamento (Chen, 1976) e modelo de objetos (Booch, 1999). Nas seções seguintes, descrevemos cada um destes processos de transformação.

4.3.3.2.1. Cenários

A utilização de cenários durante o desenvolvimento de software é um tópico de destaque na comunidade de Engenharia de Software (Weidenhaupt, 1998; Rolland, 1998). O desenvolvimento de software baseado em cenários se apóia no conceito de que a utilização da linguagem do problema (domínio do usuário) é benéfica à interação entre usuários e desenvolvedores (Leite, 1995). Entendemos por cenários a descrição de situações comuns ao cotidiano dos usuários (Zorman, 1995). Eles devem levar em conta aspectos de usabilidade e permitir o aprofundamento do conhecimento do problema, a unificação de critérios, a obtenção do compromisso de clientes e/ou usuários, a organização de detalhes e o treinamento de pessoas (Carroll, 1994).

Cada cenário descreve, através de linguagem natural semi-estruturada, uma situação específica da aplicação, focando seu comportamento. Cenários podem ser detalhados e utilizados como desenho de maneira a auxiliar a programação. Existem várias propostas para a representação de cenários, desde a mais informal, em texto livre (Carroll, 1994) até representações formais (Hsia, 1994). Optamos por uma representação intermediária que, enquanto facilita a compreensão através da utilização de linguagem natural, força a organização da informação através de uma estrutura bem definida, proposta em (Leite, 1997).

PUC-Rio - Certificação Digital Nº 0210666/CB

Na Tabela 3, descrevemos a notação de cenários e as transformações que ocorrem entre AOV-graph e o modelo de cenários.

Tabela 3. Regras de transformação entre AOV-graph e Cenários

Notação	
Objetos: cenário	
Atributos: título, objetivo, contexto, atores, recursos, episódios, exceções	
Notação: tabular	Title Actor Resource Context Exception Episode
Relacionamento: elo	
Atributos: origem (cenário), destino (subcenário)	
Notação: representado através de elos de navegação	
Regras	
▪ Cada cenário tem um título diferente dos demais; ▪ cada cenário deve ter apenas um título e objetivo; ▪ cada cenário deve ter pelo menos um ator, recurso, contexto, episódio; ▪ cada ator e recurso devem aparecer em pelo menos um episódio ou exceção; ▪ o texto de episódios que é igual ao título de algum cenário deve conter uma referência (elo) para este cenário.	
Transformações	
▪ Cada meta e tarefa de AOV-graph que não é folha gera um cenário cujo título é o nome da meta ou tarefa; ▪ cada filho (and) de meta e tarefa é um episódio do cenário gerado pelo elemento pai; ▪ cada filho (or) de meta e tarefa é um episódio opcional do cenário gerado pelo elemento pai; ▪ cada episódio que se refere a elemento que não é folha gera uma referência (elo) para o cenário criado pelo elemento filho; ▪ cada elemento que possui um novo atributo (criado por *intertype*) cujo tipo é “ator” gera um ator no cenário gerado por este elemento; ▪ cada elemento que possui um novo atributo (criado por *intertype*) cujo tipo é “exceção” gera uma exceção no cenário gerado por este elemento; ▪ pré-condições são geradas de relacionamentos transversais com *advice* do tipo *before*; ▪ restrições são geradas de *softmetas* que contribuem ou se correlacionam com o elemento ou com seus pais.	

Na Figura 59, ilustramos um exemplo de cenários gerados a partir do modelo base. Neste exemplo há dois cenários, intitulados “Model [requirements]” e “Edit [scenario]”*;* os campos referentes a ator, recurso, contexto e exceção não foram gerados porque neste exemplo ilustrativo não há *intertypes* que incluam estes novos tipos no modelo; o campo referente a episódios é gerado com base nos elementos que decompõem “Model [requirements]” e “Edit [scenario]”. Cada episódio sublinhado indica um relacionamento para o cenário equivalente.

PUC-Rio - Certificação Digital Nº 0210666/CB

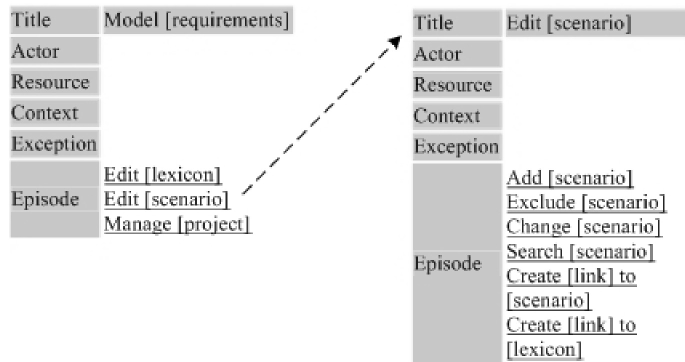


Figura 59. Exemplo da visão cenário para os grafos Model [requirements] e Edit [scenario]

4.3.3.2.2. Matriz de Rastreabilidade

Matrizes de rastreabilidade são tabelas utilizadas para registrar dependências e/ou relacionamentos entre os componentes de um sistema, sejam estes componentes, requisitos, interessados, componentes de projeto, componentes de implementação, dentre outros. Esta representação permite a fácil visualização destes relacionamentos e conseqüentemente a análise do impacto de mudanças em um dos componentes, i.e., rastreabilidade (Sommerville, 2000). Na Tabela 4, descrevemos a notação e as transformações para gerar matrizes de rastreabilidade a partir de AOV-graph.

Tabela 4. Regras de transformação entre AOV-graph e uma matriz de rastreabilidade

Notaçãos	
	Objetos: Coluna, linha, conteúdo da célula
	Atributos: rótulo
	Notação: tabular, veja Figura 60
Regras	
	▪ Cada coluna representa um elemento do modelo, sem repetição; ▪ cada linha representa um elemento do modelo, diferente ou igual aos da coluna, sem repetição; ▪ cada célula interior da tabela representa a interseção ou relacionamento que o elemento da coluna tem com o elemento da linha segundo uma das orientações linha->coluna ou coluna->linha.
Transformações	
	▪ Cada *softmeta*, meta e tarefa do modelo de metas gera uma linha e uma coluna na matriz de rastreabilidade; ▪ as células interiores da matriz são preenchidas com o relacionamento de correlação, decomposição e contribuição existente entre os elementos das linhas e os elementos das colunas; incluindo os novos relacionamentos descritos pelo relacionamento transversal.

Na Figura 60, ilustramos um exemplo de matriz de rastreabilidade indicando relacionamentos gerados pela especificação dos relacionamentos transversais

PUC-Rio - Certificação Digital Nº 0210666/CB

ilustrados no modelo base (Figura 52). Nesta matriz, os rótulos das colunas e linhas indicam as tarefas, metas e *softmetas* e as células centrais indicam os rótulos dos relacionamentos criados. Considere a direção coluna→linha, por exemplo, "Include [data]" é uma tarefa necessária para "Add [scenario]", e assim por diante.

	Select [data]	**Update [data]**	**Exclude [data]**	**Include [data]**	**...**	**Initiate [BD]**	**Connect [BD]**	**...**
Select [data]						And	And	
Update [data]						And	And	
Exclude [data]						And	And	
Include [data]						And	And	
...								
Add [scenario]				And				
Exclude [scenario]			And					
Search [scenario]	And							
Change [scenario]		And						
Create [lintk] to [scenario]				And				
Create [lintk] to [lexicon]				And				
Add [project]				And				
Link [user] to [Project]				And				
Unlink [user] to [Project]			And					

Figura 60. Exemplo da visão matriz de rastreabilidade

4.3.3.2.3. Modelo Entidade-Relacionamento

O modelo entidade-relacionamento (MER) é utilizado para descrever as principais entidades do sistema e as relações entre elas. MER é normalmente utilizado para descrever estruturas de bancos de dados (Somerville, 2005). Durante a definição de requisitos o MER é utilizado para registrar as entidades e relacionamentos levando em conta o vocabulário utilizado no Universo de Informação.

Na Tabela 5, descrevemos a notação e as transformações para gerar modelos entidade-relacionamento a partir de AOV-graph.

PUC-Rio - Certificação Digital Nº 0210666/CB

Tabela 5. Regras de transformação entre AOV-graph e o modelo de entidade-relacionamento

Notação		
	Objetos: entidade, atributo	
	Atributos: nome	
	Notação:	← Entidade ← Atributo
	Relacionamento: relacionamento	
	Atributos: origem, destino, cardinalidade	
	Notação:	—— Relacionamento
Regras		
	Cada entidade tem um nome diferente das demais;	
Transformações		
	▪ Cada tópico e novo atributo (criado por *intertype*) de *softmeta*, meta e tarefa que não é folha gera uma entidade cujo nome é o tópico ou o name do novo atributo; ▪ cada tópico e novo atributo (criado por *intertype*) de sofmeta, meta e tarefa que é folha gera um atributo na entidade gerada pelo elemento pai; ▪ cada relacionamento, no modelo de metas, entre elementos que geraram entidades gera relacionamento no modelo de entidade-relacionamento.	

Na Figura 61, ilustramos um exemplo de MER gerado para o grafo “Model [requirement model]” do modelo base. Com as informações contidas em AOV-Graph conseguimos identificar entidades, atributos e relacionamentos, mas não conseguimos extrair a cardinalidade destes relacionamentos.

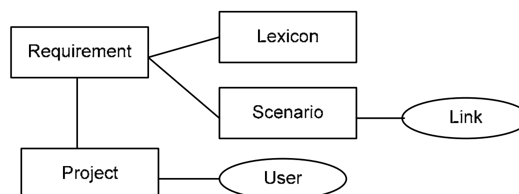


Figura 61. Exemplo da visão entidade-relacionamento para o grafo Model [requirements model]

4.3.3.2.4. Modelo de “objetos”

Objetos são entidades executáveis, instâncias de classes que definem seus atributos e serviços. Modelos de objetos ou de classes, normalmente, são utilizados quando se pretende adotar o paradigma de orientação a objetos nas atividades de desenho e implementação. Entretanto, estes modelos podem ser utilizados durante a definição de requisitos para representar os dados do sistema e as funcionalidades nas quais eles estão envolvidos (Sommerville, 2005). Neste

PUC-Rio - Certificação Digital Nº 0210666/CB

sentido, derivamos modelos de objetos do AOV-graph. Na Tabela 6, descrevemos a notação e as transformações para gerar modelos de objetos a partir de AOV-graph.

Tabela 6. Regras de transformação entre AOV-graph e o modelo de objetos

Notação		
	Objetos: objetos	
	Atributos: nome, atributo, métodos	
	Notação:	Nome Atributo Método
	Relacionamento: agregação, decomposição, associação	
	Atributos: origem, destino, cardinalidade	
	Notação:	Decomposição Agregação Associação
Regras		
	Cada objeto tem um nome diferente dos demais;	
Transformações		
	▪ Cada tópico e atributo (criado por *intertype*) de *softmeta*, meta e tarefa que não é folha gera um objeto cujo nome é o tópico ou o nome do atributo; ▪ cada tópico e atributo (criado por *intertype*) de sofmeta, meta e tarefa que é folha gera um atributo no objeto gerado pelo elemento pai; ▪ cada relacionamento de correlação, no modelo de metas, entre elementos que geraram objetos gera uma associação no modelo de objetos; ▪ cada *softmeta*, meta e tarefa gera um método no objeto gerado por seu respectivo tópico ou *intertype*.	

Na Figura 62, ilustramos um exemplo de modelo de objetos para o grafo "Model [requirements]" do modelo base. Neste exemplo temos os objetos "requirement model", "lexicon", "project" e "scenario", e os atributos e operações realizadas sobre estes objetos; os relacionamentos entre eles são gerados com base no relacionamento existente entre tarefas, metas e *softmetas* cujos tópicos deram origem a estes objetos.

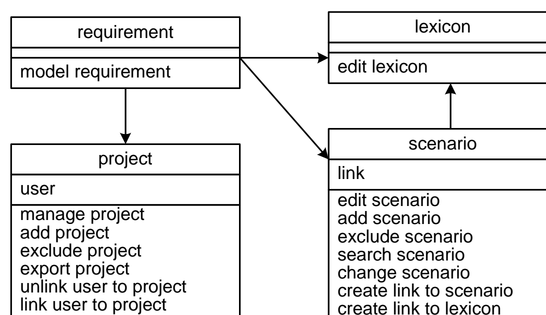


Figura 62. Exemplo da visão de objetos

PUC-Rio - Certificação Digital Nº 0210666/CB

4.3.4. Implementação

Implementamos a estratégia aplicada ao V-graph utilizando o padrão XML. AOV-graph foi definida em uma DTD (*Document Type Definition*), que é obedecida pelos modelos escritos na atividade de separação (modelo inicial) e os modelos gerados pela atividade de composição (modelo integrado). Os mecanismos de composição e visualização foram implementados utilizando XSLT (*eXtensible Stylesheet Language Transformation*). Na Figura 63, ilustramos como utilizamos o padrão XML para implementar nossa abordagem. O código desta implementação pode ser consultado nos Apêndices A e B.

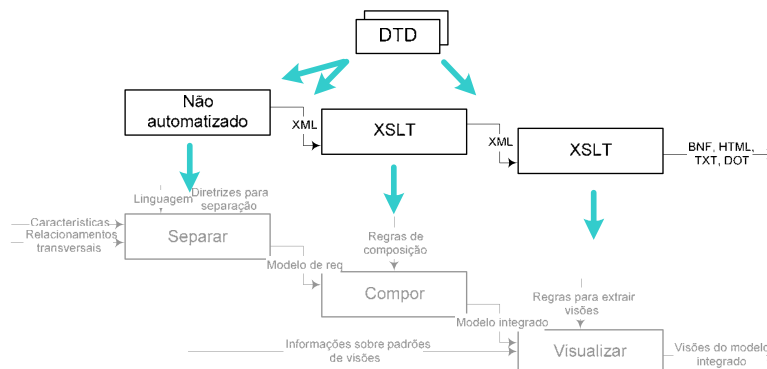


Figura 63. Implementação da estratégia para integração de características transversais

4.3.4.1. Separação

Especificamos a linguagem descrita na Seção 4.3 em uma DTD, ilustrada na Figura 64. A primeira parte da DTD é referente ao *V-graph* e a segunda parte, ao relacionamento transversal (em cinza). Como optamos por definir uma única DTD para os modelos escritos antes e depois da composição então flexibilizamos um pouco mais alguns dos elementos do modelo de componentes V-graph:

- Os elementos do *V-graph* – *Softmetas*, metas e tarefas possuem um identificador e nome como atributos, e os elementos tipo, tópico, referências para outros elementos do *V-graph*, novo tipo, e os relacionamentos de correlação e transversal (veja as linhas 7, 13 e 19 na Figura 64).
- Os relacionamentos do *V-graph* – o relacionamento de correlação é um elemento XML que possui o atributo *label*, podendo ser *make, help, known, hurt* e *break*, e os elementos *source* e *target*. Os relacionamentos de

PUC-Rio - Certificação Digital Nº 0210666/CB

contribuição e decomposição são representados pela hierarquia de metas, *softmetas* e tarefas. Desta forma, cada *softmeta*, meta e tarefa possui um atributo denominado rótulo do relacionamento (*decomposition_label*), podendo ser *And, Or, Make, Help, Unknown, Hurt* e *Break*, eles indicam o relacionamento entre filho e pai (veja as linhas 12, 18 e 24 na Figura 64).

- O relacionamento transversal – o relacionamento transversal é descrito pelos elementos origem, *pointcut*, *advice* e *intertype declaration*. Os *pointcuts* são descritos por operandos que podem definir um elemento específico do modelo *V-graph* através de seu identificador, ou um conjunto de elementos através de expressões regulares, podendo selecionar um conjunto de elementos do mesmo tipo (*softmeta*, meta e tarefa) ou qualquer dos tipos anteriores (*any*). Veja as linhas 44 a 79 na Figura 64.

```
1.  <!ELEMENT aspect_oriented_model (goal_model)*>
2.  <!ELEMENT goal_model ((softgoal | goal | task | softgoal_ref | goal_ref | task_ref)*,
3.  (correlation_relationship | crosscutting_relationship)*)>
4.  <!ATTLIST goal_model
5.      id ID #REQUIRED
6.      name CDATA #REQUIRED>
7.  <!ELEMENT softgoal ( type*, topic*, (softgoal | task | softgoal_ref | task_ref | new_element_type)*,
8.  (correlation_relationship | crosscutting_relationship)*)>
9.  <!ATTLIST softgoal
10.     id ID #REQUIRED
11.     name CDATA #REQUIRED
12.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
13. <!ELEMENT goal ( type*, topic*, (goal | task | goal_ref | task_ref | new_element_type)*,
14. (correlation_relationship | crosscutting_relationship)*)>
15. <!ATTLIST goal
16.     id ID #REQUIRED
17.     name CDATA #REQUIRED
18.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
19. <!ELEMENT task (type*, topic*, ( task | task_ref | new_element_type)*, (correlation_relationship |
20. crosscutting_relationship)*)>
21. <!ATTLIST task
22.     id ID #REQUIRED
23.     name CDATA #REQUIRED
24.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
25. <!ELEMENT topic (#PCDATA)>
26. <!ELEMENT type (#PCDATA)>
27. <!ELEMENT correlation_relationship (source, target)>
28. <!ATTLIST correlation_relationship
29.     label (break | hurt | unknown | help | make) "help">
30. <!ELEMENT target (softgoal_ref | goal_ref | task_ref)>
31. <!ELEMENT source (softgoal_ref | goal_ref | task_ref)>
32. <!ELEMENT softgoal_ref EMPTY>
33. <!ELEMENT goal_ref EMPTY>
34. <!ELEMENT task_ref EMPTY>
35. <!ATTLIST softgoal_ref
36.     id IDREF #REQUIRED
37.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
38. <!ATTLIST goal_ref
39.     id IDREF #REQUIRED
40.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
41. <!ATTLIST task_ref
42.     id IDREF #REQUIRED
43.     decomposition_label (and | or | break | hurt | unknown | help | make) "and">
```

PUC-Rio - Certificação Digital Nº 0210666/CB

```
44.  <!ELEMENT crosscutting_relationship (source, (pointcut | advice | intertype_declaration)*)>
45.  <!ELEMENT pointcut (pointcut_expression)>
46.  <!ATTLIST pointcut
47.       id ID #REQUIRED
48.       name CDATA #REQUIRED>
49.  <!ELEMENT pointcut_expression (operand | ((operand | pointcut_expression), (and | or), (operand |
50.  pointcut_expression)) | (not, (operand | pointcut_expression)))>
51.  <!ELEMENT and EMPTY>
52.  <!ELEMENT or EMPTY>
53.  <!ELEMENT not EMPTY>
54.  <!ELEMENT operand (softgoal_ref | goal_ref | task_ref | regular_expression)>
55.  <!ATTLIST operand
56.       primitive (include | substitute) "include">
57.  <!ELEMENT regular_expression (target*)>
58.  <!ATTLIST regular_expression
59.       text CDATA #REQUIRED
60.       type (goal | sotgoal | task | any) "any"
61.       param (id | name) "name"
62.             path CDATA>
63.  <!ELEMENT advice (a_it_expression, advice_body)>
64.  <!ATTLIST advice
65.       type (after | before | around) "after">
66.  <!ELEMENT a_it_expression (pointcut_ref | ((pointcut_ref | a_it_expression), (and | or), (pointcut_ref
67.  | a_it_expression)) | (not, (pointcut_ref | a_it_expression)))>
68.  <!ELEMENT pointcut_ref EMPTY>
69.  <!ATTLIST pointcut_ref
70.       id IDREF #REQUIRED>
71.  <!ELEMENT intertype_declaration (a_it_expression, intertype_declaration_body)>
72.  <!ATTLIST intertype_declaration
73.       type (attribute | element) "attribute">
74.  <!ELEMENT advice_body (softgoal_ref | goal_ref | task_ref)*>
75.  <!ELEMENT intertype_declaration_body (new_element_type | softgoal | goal | task | softgoal_ref |
76.  goal_ref | task_ref)*>
77.  <!ELEMENT new_element_type (new_element_type)*>
78.  <!ATTLIST new_element_type
79.       name CDATA #REQUIRED
          value CDATA #REQUIRED>
```

Figura 64. DTD referente à AOV-graph

4.3.4.2. Composição

Utilizamos XSLT para implementar o mecanismo de composição. Devido a algumas limitações da linguagem XSLT referentes ao sequenciamento na aquisição e escrita de dados, nós dividimos a composição em quatro partes: processamento de expressões regulares, identificação de transformações, composição da primitiva *substitute* e composição da primitiva *include*, veja Figura 65.

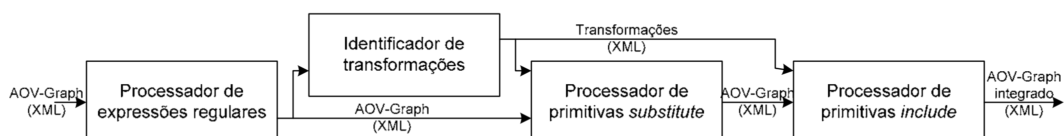


Figura 65. Componentes do mecanismo de composição

PUC-Rio - Certificação Digital Nº 0210666/CB

1. O processador de expressões regulares é responsável por criar os tipos e tópicos de todas as metas, *softmetas* e tarefas e por criar elementos *target* em *pointcuts* definidos por expressões regulares. Este processamento resulta em um arquivo XML que também está de acordo com a DTD definida anteriormente. Este componente exige a versão 2.0 de XSLT, pois esta inclui o tratamento de expressões regulares.
2. O identificador de transformações é responsável por ler os relacionamentos transversais e gerar um arquivo XML contendo onde e o que deve ser adicionado em cada *pointcut* segundo a ordem em que estas transformações devem ser aplicadas.
3. O processador dos elementos a serem substituídos recebe os dois arquivos XMLs anteriores (passos 1 e 2) e apaga todos os elementos (e seus filhos) indicados pelos *pointcuts* com primitiva *substitute*.
4. O processador dos elementos a serem incluídos também recebe os dois arquivos XMLs anteriores (passos 2 e 3) e inclui todos os elementos definidos em *advice* e *intertype declaration* no local indicado pelos *pointcuts* com ambas as primitivas.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.3.4.3. Visualização

O mecanismo de visualização gera visões de AOV-graph. Cada visão é gerada por um arquivo XSLT que implementa separadamente as transformações definidas na Seção 4.3.3. Na Figura 66, ilustramos os componentes implementados para a atividade de visualização. O formato DOT é utilizado juntamente com o software GraphViz (Graphviz, 2006) para gerar figuras (JPG).

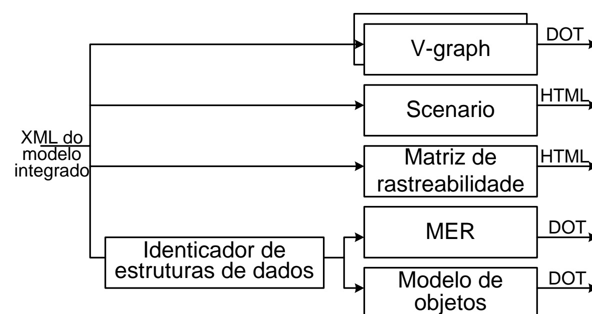


Figura 66. Componentes do mecanismo de visualização

4.4. Aplicação da Estratégia a outros Modelos de Requisitos

A estratégia definida nas Seção 4.1 pode ser aplicada a diferentes modelos de componentes, tais como lista de requisitos, casos de uso, léxico, cenários, i*, dentre outros. Para utilizá-la é necessário instanciar (Figura 31, página 65): o modelo de componentes, o modelo de *joinpoints*, os mecanismos de composição e de visualização. A seguir, descrevemos quais os benefícios e como aplicar nossa estratégia aos modelos de Cenários, Léxico e sentenças de requisitos. Definimos os modelos de componentes e de *joinpoints* para estas linguagens e apenas apontamos como os mecanismos de composição e de visualização poderiam ser definidos para estes modelos de componentes. Assim como apresentamos na Seção 4.3, seria necessário definir e implementar as regras de composição e de transformação para estes mecanismos.

4.4.1. Cenários

Em (Leite, 1997) é definido um modelo para especificação de cenários estruturados. Cada cenário é descrito pelas informações: título, objetivo, contexto, atores, recursos, episódios. Esta linguagem é focada na seqüência de cenários, ou de funções para cumprir uma funcionalidade. Como a decomposição dominante é funcional então as informações sobre o contexto, atores e recursos encontram-se espalhadas e entrelaçadas no conjunto de cenários que retrata o sistema.

A abordagem em (Leite, 1997) define um esquema de rastreabilidade entre estes elementos através de um léxico e da utilização de hiper-elos (*hiperlinks*), permitindo a navegação entre definições (no léxico) e cenários, mas que abordam apenas parcialmente estes problemas. Além disto, os relacionamentos entre cenários através da decomposição funcional muitas vezes agrupam funções (episódios) pouco coesas. Nestes casos, mudar a direção do relacionamento, tal como em AspectJ, e ter um mecanismo de composição seria interessante para incluir e excluir episódios e outras informações menos coesas de cada cenário.

PUC-Rio - Certificação Digital Nº 0210666/CB

Título	Adicionar cenário
Objetivo	Inserir um cenário no modelo de cenários
Ator	Usuário participante, sistema
Recurso	BD, informações do cenário
Contexto	Pre-condição: Usuário participante tem uma sessão, um projeto está aberto
Exceção	Se cenário já existe então enviar mensagem de erro Se usuário participante fechar janela do sistema então fechar sessão Se usuário participante fechar janela do sistema então desconectar BD
Episódio	Usuário participante preenche as informações do cenário Sistema verifica se informações do cenário estão completas Sistema verifica se sintaxe do cenário está correta Se BD está desconectado então conectar BD Se cenário não existe no projeto então incluir cenário Desconectar BD

Figura 67. Exemplo de espalhamento e entrelaçamento no modelo de cenários

Na Figura 67, ilustramos o cenário “Adicionar cenário”, nele as informações em cinza são referentes à infra-estrutura de armazenamento de dados (banco de dados) e controle de acesso (através de sessões). Tais informações são invasivas e não há controle sobre como elas estão espalhadas e entrelaçadas. Desta forma, ao modificar a infra-estrutura de armazenamento de banco de dados, todos os cenários têm que ser novamente inspecionados para retratar esta modificação.

PUC-Rio - Certificação Digital Nº 0210666/CB

```
1.  <aspect_oriented_model> := <scenario_model>
2.  <scenario_model>:=scenario_model(<name>;<id>){<component><relationship>} |
3.                          <scenario_model> <scenario_model>
4.  <component> := <scenario> |  <component> <component>
5.  <scenario>:=scenario (<title>;<scenario_id>)
6.       {<goal>;<context>;<actor>;<resource>;<exception>;<episode>;<relationship>}
7.  <goal> := goal <name><goal_id> <relationship>
8.  <actor> := actor <name><actor_id> <relationship> | <actor> <actor>
9.  <resource>:= resource <name><resource_id> <relationship> | <resource> <resource>
10. <context> := context <sentence> <context_id> <relationship> | <context> <context>
11. <exception>:=exception<sentence><exception_id><relationship>|<exception><exception>
12. <episode> := episode <sentence><episode_id> <relationship> | <episode> <episode>
13. <sentence> := <name>|<scenario_ref>| <sentence><sentence>
14. <scenario_ref> := <scenario_id>
15. <goal_ref> := <goal_id>
16. <actor_ref> := <actor_id>
17. <resource_ref> := <resource_id>
18. <exception_ref> := <exception_id>
19. <episode_ref> := <episode_id>
20. <context_ref> := <context_id>
21. <title>:=”a..z” <name>:=”a..z” <scenario_id>:=”a..z” <goal_id>:=”a..z” <actor_id>:=”a..z”
22. <resource_id>:=”a..z” <exception_id>:=”a..z”
23. <episode_id>:=”a..z”  <context_id>:=”a..z”
24. <relationship> := <crosscuttingRel> |  <relationship> <relationship>
25.
26. <joinpoint> := <scenario_ref> | <goal_ref> | <context_ref> | <actor_ref> |
27.                    <resource_ref> | <exception_ref> | <episode_ref>
```

Figura 68. Sintaxe do modelo de componentes: Cenários

Para utilizar nossa estratégia com o modelo de cenários definimos os itens da Figura 31 (página 65) como a seguir:

- Modelo de componentes – é definido pelos itens do modelo de cenários: cenário, título, objetivo, atores, recursos, contexto, episódios e exceções

(linhas 1 a 24 da Figura 68). O relacionamento transversal (em cinza na Figura 68) tem exatamente a mesma sintaxe do definido na Seção 4.2.3.

- Modelo de *joinpoints* – é definido pelos mesmos itens do modelo de componentes, com exceção do título. Eles expõem informações importantes que podem ser utilizadas para identificar os cenários que são afetados (linhas 26 e 27 da Figura 68).
- Mecanismo de composição – pode incluir, alterar ou excluir episódios em um cenário e juntamente incluir ou excluir os elementos contexto, atores, recursos e exceções intrínsecos àqueles episódios. Na Figura 69(d), apresentamos um exemplo.
- Mecanismo de visualização – a estruturação das informações descritas em cenários permite a extração de várias visões, tais como: modelo de classes; tutorial para os usuários; visão gráfica do relacionamento entre cenários, estes relacionamentos podem ser devido à utilização dos mesmos atores, recursos, exceções ou episódios ou de acordo com o tipo de interação que ocorre entre eles (Breitman, 2000); dentre outras. A Figura 69(d) ilustra uma visão parcial dos cenários após a composição, enquanto a Figura 69(a) e Figura 69(b) ilustram cenários antes da composição.

Na Figura 69, ilustramos um exemplo de descrição em cenários utilizando nossa abordagem: (a) é o cenário "Adicionar cenário" sem qualquer referência a infra-estrutura de banco de dados, (b) é o cenário "Persistência em BD", e (c) é a descrição do relacionamento transversal de (b) para (a). A composição destes cenários resulta na descrição ilustrada em (d).

Na descrição do relacionamento transversal (Figura 69(c)), há dois *pointcuts*: em PC1 indicamos que o cenário intitulado "Adicionar cenário" é afetado e em PC2 indicamos que qualquer episódio cujo nome contenha a palavra "Incluir" é afetado. Podemos observar também a utilização de *advice* around, before e after associados a *pointcuts* com a primitiva include, indicando inclusão em, respectivamente, qualquer lugar, antes e depois do ponto definido pelo *pointcut*. Veja o resultado da composição na Figura 69(d).

PUC-Rio - Certificação Digital Nº 0210666/CB

Título	Adicionar cenário
Objetivo	Inserir um cenário no modelo de cenários
Ator	Usuário participante, sistema
Recurso	Informações do cenário
Contexto	Pre-condição: Usuário participante tem uma sessão, um projeto está aberto
Exceção	Se cenário já existe então enviar mensagem de erro Se usuário participante fechar janela do sistema então fechar sessão
Episódio	Usuário participante preenche as informações do cenário Sistema verifica se informações do cenário estão completas Sistema verifica se sintaxe do cenário está correta Se cenário não existe no projeto então incluir cenário

(a)

Título	Persistência em BD
Objetivo	Armazenar e recuperar informações em BD
Ator	Sistema
Recurso	BD
Contexto	...
Exceção	...
Episódio	Se BD está desconectado então Conectar BD [Incluir dados] [Excluir dados] [Atualizar dados] [Selecionar dados] Desconectar BD

(b)

```
Crosscutting {
  Source: Persistência em BD
  Pointcut PC1: include (Adicionar cenário)
  Pointcut PC2: include (.*incluir cenário.*;episode)
  Advice aroud: PC1 { recurso = BD }
  Advice before: PC2 { Se BD está desconectado
                  então Conectar [BD] }
  Advice after: PC2 { Desconectar BD }
  Intertype declaration element:
    {Exceção = Se usuário participante fechar
     janela do sistema então Desconectar BD }
}
```

(c)

Título	Adicionar cenário
Objetivo	Inserir um cenário no modelo de cenários
Ator	Usuário participante, sistema
Recurso	BD, informações do cenário
Contexto	Pre-condição: Usuário participante tem uma sessão, um projeto está aberto
Exceção	Se cenário já existe então enviar mensagem de erro Se usuário participante fechar janela do sistema então desconectar BD Se usuário participante fechar janela do sistema então fechar sessão
Episódio	Usuário participante preenche as informações do cenário Sistema verifica se informações do cenário estão completas Sistema verifica se sintaxe do cenário está correta Se BD está desconectado então conectar BD Se cenário não existe no projeto então incluir cenário Desconectar BD

(d)

Figura 69. Exemplo do uso de nossa abordagem no modelo de cenários

4.4.2. Léxico Ampliado da Linguagem (LAL)

Em (Leite, 1990) é definido um modelo baseado em linguagem natural semi-estruturada para especificação de termos utilizados no universo de informações (UdI). Este modelo registra cada termo como um símbolo, que consiste de tipo, sinônimos, noção e impacto. O relacionamento entre símbolos ocorre quando o nome de um símbolo é citado na descrição de outro símbolo, podendo representar um relacionamento de decomposição, de abstração ou associação. Além disto, é requerido que o léxico seja circular, i.e., a maximização do uso dos símbolos definidos. Quando um símbolo é incluído ou excluído os relacionamentos para aquele símbolo são atualizados, inseridos ou excluídos. No caso de exclusão, isto significa apenas que o símbolo não é mais uma entrada do LAL, ele continua sendo referido na descrição dos outros símbolos.

O relacionamento transversal pode ser utilizado para indicar uma relação mais forte entre o nome de um símbolo e as referências a ele na descrição dos outros termos, indicando que se ele existe como entrada do LAL implica em referências a ele, e sua inexistência implica na ausência daquelas referências. Na Figura 70, ilustramos a descrição de um símbolo do léxico, nesta descrição os elementos sombreados indicam a existência do "léxico" não apenas como símbolo

PUC-Rio - Certificação Digital Nº 0210666/CB

do modelo, mas como documento existente no sistema modelado. Desta forma, caso o “léxico” não exista no sistema, então é necessária a inspeção de todos os símbolos para exclusão de referências a ele.

Nome	Cenário
Tipo	Objeto
Sinônimo	Cenários
Noção	Descrição em linguagem natural semi-estruturada representando uma situação que ocorre no UdI ou a interação entre usuários e sistema. Contém título, objetivo, atores, recursos, contexto, episódios, exceções. Utiliza símbolos do léxico.
Impacto	Adicionar cenário Excluir cenário Modificar cenário Criar referências para cenários Excluir referências para cenários Criar referências para símbolos do léxico Excluir referências para símbolos do léxico Verificar cenário

Figura 70. Exemplo de espalhamento e entrelaçamento no LAL

```
1.  <aspect_oriented_model> := <lexicon_model>
2.  <lexicon_model> := lexicon_model (<name>; <id>){<component> <relationship>} |
3.                         <lexicon_model> <lexicon_model>
4.  <component> := <symbol> |  <component> <component>
5.  <symbol>:= symbol (<name>;<symbol_id>)
6.               {<synonym>; type <type>;<notion>;<behavior_response><relationship>}
7.  <synonym> := synonym <name><goal_id>
8.  <type> := object | state | verb | subject
9.  <notion> notion <sentence><notion_id> <relationship> | <notion> <notion>
10. <sentence>:= <name> | <symbol_ref> | <sentence><sentence>
11. <behavior_response> :=
12.                   behavior_response <sentence><behavior_response_id> <relationship> |
13.                   <behavior_response> <behavior_response>
14. <symbol_ref> := <symbol_id>
15. <synonym_ref> := <synonym_id>
16. <notion_ref> := <notion_id>
17. <behavior_response_ref> := <behavior_response_id>
18. <name>:=”a..z” <symbol_id>:=”a..z” <synonym_id>:=”a..z”
19. <notion_id>:=”a..z” <behavior_response_id>:=”a..z”
20. <relationship> := <crosscuttingRel> |  <relationship> <relationship>
21.
22. <joinpoint>:=<symbol_ref> | <notion_ref> | <synonym_ref> |<behavior_response_ref>
```

Figura 71. Sintaxe do modelo de componentes: LAL

Para utilizar nossa estratégia com o LAL, definimos os itens da Figura 31 (página 65) como a seguir:

- Modelo de componentes – é definido pelos itens do LAL: símbolo, nome do símbolo, sinônimos, tipo, noção e impacto (linhas 1 a 20 da Figura 71).
- Modelo de *joinpoints* – é definido pelos itens noção e impacto (linha 22 da Figura 71).
- Mecanismo de composição – pode incluir ou excluir sentenças na noção e no impacto de outros símbolos, registrando a referência naqueles para o “símbolo transversal”.

PUC-Rio - Certificação Digital Nº 0210666/CB

- Mecanismo de visualização – o LAL registra informações sobre objetos, verbos, estados e sujeitos e desta maneira permite a extração destas visões isoladamente ou em conjunto, por exemplo, quais os atores do sistema e como estão relacionados entre si, quais atores estão relacionados a quais ações, dentre outras.

A Figura 72 ilustra um exemplo de descrição de símbolos do léxico utilizando nossa abordagem: (a) representa o símbolo "Cenário", (b) representa o termo "Símbolo", (c) é a descrição do relacionamento transversal existente entre (b) e (a), e (d) representa o resultado da composição. Desta forma, se o sistema sendo desenvolvido não tem funcionalidades a respeito de "Símbolos do léxico", então a exclusão do termo "Símbolo" implicaria não apenas na remoção do sublinhado em símbolos do léxico na descrição de "Cenário" (d) mas também na remoção das frases em cinza deste mesmo símbolo (d).

Nome	Cenário
Tipo	Objeto
Sinônimo	Cenários
Noção	Descrição em linguagem natural semi-estruturada representando uma situação que ocorre no UdI ou a interação entre usuários e sistema. Contém título, objetivo, atores, recursos, contexto, episódios, exceções.
Impacto	Adicionar cenário Excluir cenário Modificar cenário Criar referências para cenários Excluir referências para cenários Verificar cenário

(a)

Nome	Símbolo
Tipo	Objeto
Sinônimo	Símbolo do léxico, Símbolos do léxico, símbolos
Noção	Descrição em linguagem natural semi-estruturada representando os termos utilizados no UdI. Contém nome, tipo, sinônimo, noção, impacto.
Impacto	Adicionar símbolo Excluir símbolo Modificar símbolo Criar referências para símbolos Excluir referências para símbolos Verificar símbolo

(b)

```
Crosscutting {
  Source: Símbolo
  Pointcut PC1: include (Cenários)
  Advice aroud: PC1 { Impacto = Criar referências para
        símbolos; Impacto = Excluir referências para
        símbolos }
  Intertype declaration element: PC1
     { Noção = Utiliza símbolos do léxico }
}
```

(c)

Nome	Cenário
Tipo	Objeto
Sinônimo	Cenários
Noção	Descrição em linguagem natural semi-estruturada representando uma situação que ocorre no UdI ou a interação entre usuários e sistema. Contém título, objetivo, atores, recursos, contexto, episódios, exceções. Utiliza símbolos do léxico.
Impacto	Adicionar cenário Excluir cenário Modificar cenário Criar referências para cenários Excluir referências para cenários Verificar cenário Criar referências para símbolos Excluir referências para símbolos

(d)

Figura 72. Exemplo do uso de nossa abordagem no Léxico

4.4.3. Sentenças de Requisitos

Sentenças de requisitos é a maneira mais usual de documentar requisitos. Elas são, geralmente, escritas livremente em linguagem natural, permitindo sua leitura tanto por pessoal técnico quanto usuários e clientes. Costuma-se numerar

PUC-Rio - Certificação Digital Nº 0210666/CB

os requisitos para que sejam identificados e referidos. Entretanto, como apresentamos no Capítulo 2 (Figuras 6 e 7), algumas informações e características estão espalhadas e entrelaçadas em muitas sentenças de requisitos, dificultando a identificação e atualização delas.

O relacionamento transversal pode ser utilizado para mapear a interação entre sentenças, registrando onde elas se sobrepõem ou estão repetidas. Desta maneira, pode-se criar uma lista de requisitos com ou sem repetição, separando ou compondo as "sentenças transversais". Neste caso, o relacionamento transversal pode ser entendido como uma matriz de rastreabilidade estendida, que permite o mecanismo de composição gerar as interações registradas nele.

```
1. <aspect_oriented_model> := <requirementSentence_model>
2. <requirementSentence_model> := <component> | <component> <component>
3. <component> := <requirementSentence> |  <component> <component>
4. <requirementSentence> := (<name>; <requirementSentence_id>; <relationship_label>){<component>
5. <relationship>}
6. <requirementSentence_ref> := <requirementSentence_id>; <relationship_label>
7. <relationship> := <crosscuttingRel> |  <relationship> <relationship>
8. <name>:="a..z" <requirementSentence_id>:="a..z" <relationship_label>:="a..z"
9. <joinpoint> := <requirementSentence_ref>
```

Figura 73. Sintaxe do modelo de componentes: sentenças de requisitos

Para utilizar nossa estratégia com sentenças de requisitos, definimos os itens da Figura 31 (página 65) como a seguir:

- Modelo de componentes – é definido por sentenças, cada sentença pode ser decomposta em outras sentenças (linhas 1 a 8 da Figura 73).
- Modelo de *joinpoints* – é definido, também, pelas sentenças de requisitos, indicando a interação entre quaisquer sentenças (linha 9 da Figura 73).
- Mecanismo de composição – pode incluir ou excluir sentenças de requisitos como parte da hierarquia de outras sentenças, espalhando e/ou entrelaçando as informações. Cada sentença a ser afetada pode ser identificada por sua numeração ou através de expressões regulares.
- Mecanismo de visualização – cada hierarquia de requisitos pode ser vista separadamente ou não; matrizes de rastreabilidade podem ser facilmente geradas; expressões regulares podem ser utilizadas para recuperar sentenças que mencionam o mesmo sujeito, verbo, ou objeto, dentre outras visões.

Na Figura 74, ilustramos: em (a) um lista de requisitos em que separamos restrições de formatação (requisito 5) e funções de edição (requisito 4) das demais porque ele se repetem no documento; em (b) a descrição de dois relacionamentos transversais cujas origens são os requisitos de número 4 e 5 respectivamente; o

PUC-Rio - Certificação Digital Nº 0210666/CB

primeiro espalha os requisitos 4.1, 4.2 e 4.3, e o segundo espalha uma restrição quanto ao formato de hora/data; e em (c) está a descrição de requisitos resultante da composição.

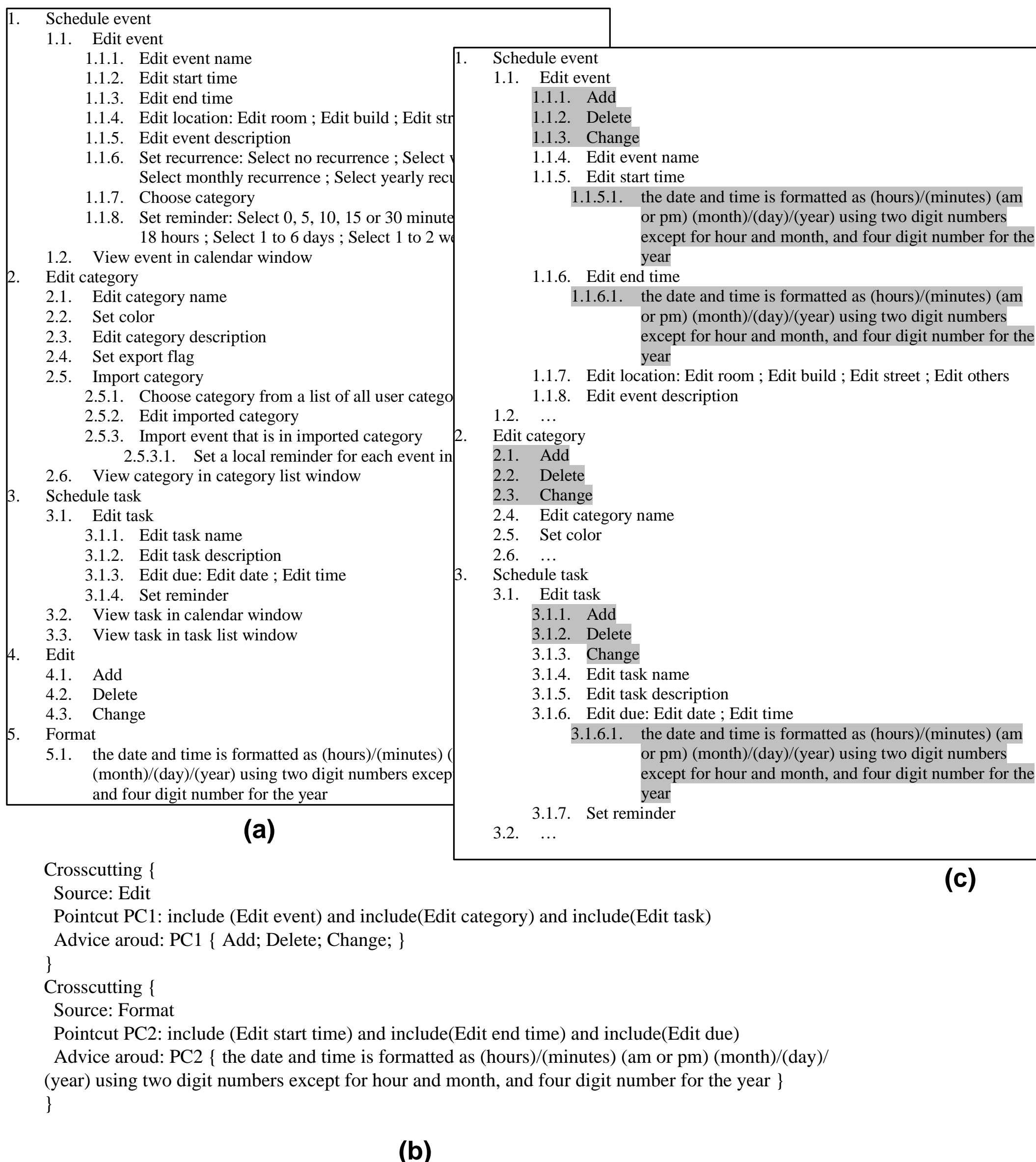


Figura 74. Exemplo do uso de nossa abordagem em sentenças de requisitos

4.5. Processo de Engenharia de Requisitos Utilizando a Estratégia Orientada a Aspectos

Como apresentamos no Capítulo 2, o processo de engenharia de requisitos é formado pelas atividades: elicitação, modelagem e análise. O foco deste trabalho é a atividade de modelagem, mas como não há uma delimitação clara entre estas atividades, então a elicitação e análise são fortemente afetadas. Consideramos que

PUC-Rio - Certificação Digital Nº 0210666/CB

utilizando nossa estratégia tornamos a atividade de modelagem menos árdua e mais ágil, e desta maneira, auxiliamos a elicitação e a análise porque o engenheiro consegue obter visões mais dirigidas à atividade em questão, veja Figura 75.

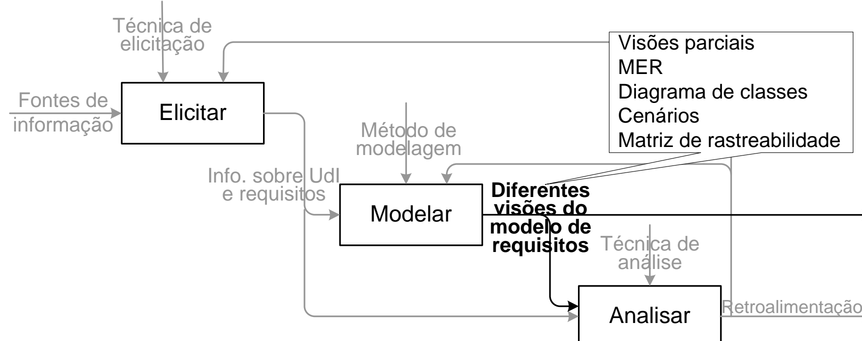


Figura 75. Processo de engenharia de requisitos modificado pela estratégia de integração de características transversais

O ciclo elicitação-modelagem-análise ocorre em muitas iterações. Propomos que a primeira iteração seja realizada da maneira convencional, i.e.:

- Elicitação – os engenheiros elicitam requisitos utilizando alguma técnica de elicitação;
- Modelagem – os engenheiros especificam os requisitos elicitados utilizando uma instância de LMROA. Inicialmente, o engenheiro deve se preocupar apenas em encontrar a melhor decomposição para os requisitos, utilizando os construtos definidos no modelo de componentes. Tendo apenas uma parte do modelo desenhado, já é possível extrair diferentes visões para serem analisadas.
- Análise – com o apoio de várias visões o engenheiro pode analisar o modelo do sistema sob diferentes ângulos, possivelmente com menor complexidade. Verificando se ele está correto e claro de acordo com o que os requerentes esperam, o engenheiro procura melhorar a especificação, tanto na decomposição e modularização dos requisitos quanto modificando ou adicionando novas informações, i.e, ele volta à atividade de modelagem e elicitação. O engenheiro deve estar atento a características que estão espalhadas e entrelaçadas, elas normalmente dificultam a compreensão do modelo de requisitos. Neste caso, as características transversais são aquelas que se repetem, que estão fortemente dependentes, mas não possuem um forte relacionamento conceitual (semântico) ou aquelas que são candidatas a muitas evoluções ou a reuso (veja na Seção 4.2.4).

PUC-Rio - Certificação Digital Nº 0210666/CB

Voltando à atividade de modelagem, o engenheiro deve procurar modularizar estas características e desenhar as interações delas para o restante do sistema, usando relacionamentos transversais. Neste caso, pode ser necessário, novamente, voltar aos requerentes e elicitar mais requisitos ou esclarecer pontos ambíguos.

Note que, a atividade de identificar características transversais é adiada para quando o engenheiro tem alguma parte do modelo desenhada, sendo parte da atividade de análise. Entretanto, muitas vezes a distância entre modelar e analisar se aquela é a melhor modularização e decomposição é tão pequena que, nestes casos, identificar problemas e buscar a melhor representação é parte do processo de modelagem.

Esperamos que por oferecer visões automaticamente, agilizamos a modelagem e, assim, permitimos que este ciclo ocorra em menos tempo e que ocorram mais iterações. Para a atividade de elicitação, esta estratégia de modelagem pode oferecer visões mais apropriadas para os usuários a serem abordados, mostrando os serviços que lhes interessam. Para a atividade de análise, esta estratégia oferece diferentes visões, parciais ou não, que podem ajudar o engenheiro a analisar a corretude, consistência, omissões, impacto de mudanças, custo, dentre outros.

PUC-Rio - Certificação Digital Nº 0210666/CB

4.6. Resumo

A estratégia apresentada neste Capítulo tem o objetivo de facilitar a modelagem de requisitos. As atividades de Separação e Composição foram baseadas nos fundamentos do paradigma de aspectos e modificadas para aplicação no processo de requisitos. A atividade de Visualização foi baseada em trabalhos sobre visões e pontos-de-vista, considerados fundamentais para o processo de requisitos (Nuseibeh, 1994; Kotonya, 1996; Leite, 1996).

Esta abordagem pode ser aplicada a diferentes modelos, sendo necessário definir os elementos mostrados na Seção 4.1. Consideramos que separação, composição e visualização constituem o conjunto mínimo de atividades necessário para tratar os problemas de espalhamento e entrelaçamento durante a definição de requisitos, quando a especificação é muito volátil devido à constante mudança na demanda e no entendimento sobre o contexto e problema em questão.

Constatamos que o trio separação-composição-visualização é importante em conjunto porque:

- Se somente separamos não temos a visão do sistema inteiro, e muito trabalho é realizado para manter a consistência entre as partes;
- Se não separamos, o modelo é demasiadamente complexo tornando-se difícil analisá-lo e mantê-lo;
- Se separamos e combinamos, diminuímos o problema de inconsistências, mas o problema em analisar o modelo complexo resultante da composição permanece;
- Se separamos e visualizamos, aumentamos o poder de análise do modelo, mas cada visão é apenas parte do modelo do sistema;
- Se não separamos e geramos várias visões, podemos obter visões parciais ou totais, i.e., o problema de complexidade pode ser controlado, entretanto esta abordagem se preocupa a posteriori com a melhor modularização e o reuso das partes, sendo necessário um trabalho de refatoração.

Assim, aplicamos nossa abordagem ao modelo de metas V-graph, instanciando LMROA e definindo os mecanismos de composição e visualização para este modelo de componentes. Além disso, apontamos como esta abordagem pode ser aplicada aos modelos: cenários, LAL e sentenças de requisitos. Finalizamos, mostrando qual o impacto de aplicar nossa abordagem orientada a aspectos no processo de definição de requisitos.

PUC-Rio - Certificação Digital Nº 0210666/CB