3
Processo para Desenvolvimento de Software Mais Transparente (PDS+T)

Este capítulo define e explica o processo proposto para construção de software mais transparente, com foco em software para Web, o que permite a criação de heurísticas mais específicas. Este processo utiliza como alicerce dois artefatos de requisitos: LAL e cenários. A partir destes artefatos, a organização modular do software é obtida e, posteriormente, sua arquitetura. A partir da arquitetura, heurísticas guiam a atividade de escrita do código fonte do software. Os cenários, em diferentes níveis de abstração, permeiam todo o processo de desenvolvimento, atuando inicialmente como guia e, posteriormente, como explicação de diferentes componentes do software, tais como arquitetura e código. Para demonstrar o uso do processo, ao longo do seu detalhamento mostraremos como ele foi empregado na construção do software C&L. Ao final, demonstramos como as operacionalizações que compõem o processo foram anexadas ao Catálogo de Transparência de Software, com o intuito de promover o seu reuso.

3.1.
O Processo PDS+T

O processo PDS+T se inicia a partir dos artefatos de requisitos Léxico Ampliado da Linguagem (LAL) e cenários, que descrevem, respectivamente, a linguagem e as situações do UdI. A partir destes artefatos iniciais, utilizando um método que pode ser parcialmente automatizado, derivamos a organização modular do software. Esta organização é evoluída com o uso do padrão de arquitetura MVC, a fim de se obter a arquitetura do software, que é descrita através dos cenários e seus relacionamentos. A última etapa do processo consiste no refinamento dos cenários que compõem a arquitetura e, posteriormente, sua operacionalização. Como produto final desta atividade, obtemos um software com código fonte anotado através de cenários operacionais. Estes cenários podem ser rastreados até os cenários que documentam a arquitetura que, por sua vez, podem ser rastreados até os cenários de requisitos. Esta rastreabilidade funciona em ambas as direções, isto é, podemos rastrear um cenário de requisitos até o código e vice-versa.

Escolhemos focar em software Web, pois estes se adéquam naturalmente a arquitetura MVC proposta. Além disto, esta escolha nos permitiu construir heurísticas específicas, pois consideramos as tecnologias tradicionalmente utilizadas no desenvolvimento destes software, como por exemplo, linguagem de marcação HTML para definição de interface e linguagem de programação específica para lidar com requisições provenientes da interface.

A interseção entre a transparência de software e o MPS. BR (MPS.BR, 2012) já foi percebida e explorada em trabalho anterior (Sousa et al., 2013). O que foi constatado é que as qualidades relacionadas com Transparência também estão presentes nos enunciados e resultados esperados do MPS.BR. Assim, ao aplicar operacionalizações visando uma das abordagens, alcança-se, como efeito colateral, a contribuição à outra. Ou seja, o desenvolvimento de um processo transparente beneficia a organização que deseja certificar-se com o MPS.BR, sendo que o pensamento inverso também é verdade. Com relação ao PDS+T, em uma análise preliminar, percebemos que algumas operacionalizações propostas pelo processo atendem aos resultados esperados estabelecidos pelo modelo MPS. Para exemplificar, utilizamos o resultado esperado GR3, pertencente ao nível G do MPS, o qual estabelece que a rastreabilidade bidirecional entre os requisitos e os produtos de trabalho deve ser estabelecida e mantida. Esta rastreabilidade é proporcionada pelo processo PDS+T, uma vez que, como consequência do processo, temos a rastreabilidade, em ambas as direções, entre requisitos, arquitetura e código.

A construção de um modelo de domínio, a partir dos requisitos, de onde possa se extrair o modelo de dados do software, não foi objeto de estudo deste trabalho. Nos sistemas construídos com o PDS+T, o modelo de dados foi obtido através da identificação das necessidades de armazenamento do software, obtidas a partir de uma análise dos cenários de requisitos e seus relacionamentos. No entanto, este problema já foi explorado em outros trabalhos (Leonardi et al., 2001; Bidle et al., 2002; Liang, 2003; Fortuna et al., 2008). O trabalho de Leonardi, que define uma estratégia para obtenção de um modelo conceitual de objetos a partir de requisitos em linguagem natural (Leonardi et al., 2001), se baseia nos mesmos artefatos de requisitos explorados nesta proposta: cenários e LAL.

Utilizamos o modelo SADT[2] proposto por Ross (Ross e Schoman, 1977) para descrever em detalhes as atividades compreendidas no processo, assim como as entradas e saídas de cada etapa. Esta técnica permite uma decomposição hierárquica das atividades do processo. Em cada novo diagrama, as atividades do nível anterior são decompostas em pelos menos três outras. O diagrama SADT da Figura 12 apresenta, de forma abstrata, as atividades que compõem o processo PDS+T, que são: AGRUPAR, ORGANIZAR e OPERACIONALIZAR. Nas seções subsequentes, cada uma destas atividades é decomposta através de um novo diagrama SADT, fornecendo mais detalhes a cerca do que é feito durante sua realização.

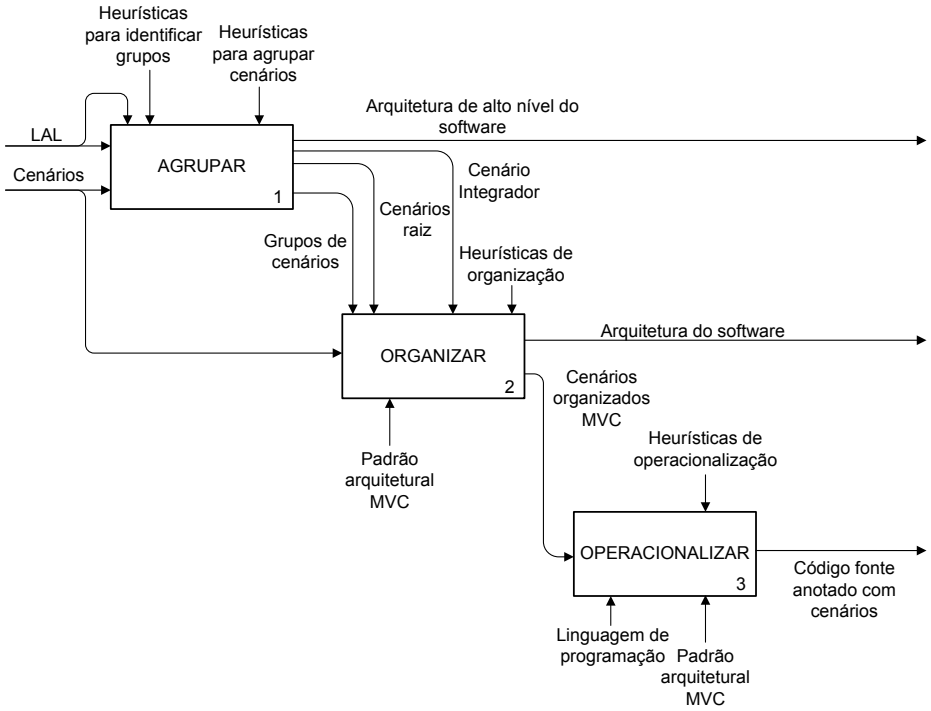

Figura 12. SADT do processo proposto.

A fim de exemplificar a utilização do PDS+T, apresentaremos como ele foi utilizado durante a construção do software C&L (C&L, 2013a). Este software permite a organização de cenários e símbolos de léxicos através de projetos. O conceito de projeto dentro do C&L serve para representar um UdI específico. O C&L permite a adição de cenários (requisitos) e símbolos (linguagem) aos

[2] No modelo SADT as caixas representam as atividades, as setas à esquerda as entradas requeridas pela atividade, as setas para baixo os controles, as setas para cima os mecanismos e as setas à direita as saídas da atividade.

projetos e cria, de forma automática, links para implementar o relacionamento entre eles. Desta forma, o C&L permite a navegação entre cenários, entre símbolos do léxico e entre cenários e símbolos do léxico do mesmo projeto. Além disto, o software também oferece um ambiente colaborativo para a elaboração dos projetos.

Na Tabela 4 enumeramos os cenários de requisitos do C&L, que são utilizados ao longo do processo para demonstrar a execução das atividades do PDS+T.

1. Criar símbolo do léxico	10.Editar cenário	19. Gerar grafo do projeto
2. Criar cenário	11. Editar usuário	20.Adicionar colaborador ao projeto
3. Criar usuário	12. Editar projeto	21.Remover colaborador do projeto
4. Criar projeto	13. Remover símbolo do léxico	22.Visualizar sugestões para o cenário
5. Visualizar símbolo do léxico	14. Remover cenário	23.Visualizar sugestões para o símbolo do léxico
6. Visualizar cenário	15. Remover projeto	24.Aprovar sugestões para o cenário
7. Visualizar usuário	16. Exportar projeto	25.Rejeitar sugestões para o cenário
8. Visualizar projeto	17. Autenticar usuário	26.Rejeitar sugestões para o símbolo do léxico
9. Editar símbolo do léxico	18. Sair do software	27.Aprovar sugestões para o símbolo do léxico

Tabela 4. Cenários de requisitos do C&L.

3.2. Descrição das Atividades do PDS+T

A seguir, explicaremos as atividades apresentadas no modelo SADT da Figura 12. Para um melhor detalhamento, cada atividade é decomposta em três ou mais atividades, que são representadas em um novo modelo SADT.

3.2.1. Agrupar

O objetivo principal desta atividade (Almentero et al., 2014) é definir a arquitetura inicial do software. Para atingir este objetivo, os cenários são divididos em grupos, de forma que cada grupo representa um módulo do software. Como os grupos formados aqui são utilizados como a base para a arquitetura do software, esta atividade é guiada por princípios de modularização. Do ponto de vista da modularização, algumas das métricas mais utilizadas para avaliar a qualidade da organização do software são: acoplamento e coesão (Meyer, 1988; Pressman, 1992; Jacobson et al., 1992; Sommerville, 2001; Ghezzi et al., 2002). Geralmente, baixo acoplamento intermodular e alta coesão intramodular são características desejadas para qualquer organização modular do software. Desta forma, o foco nesta atividade é separar os cenários em grupos, garantindo, por construção, que a distribuição realizada atenda as características supracitadas.

Os requisitos não funcionais serão descritos na arquitetura através das restrições dos cenários, que, como vimos anteriormente, podem estar vinculadas a contexto, recursos e episódios.

Esta atividade pode ser dividida em três etapas bem delimitadas, são elas: identificação dos grupos, distribuição dos cenários entre os grupos identificados e verificação do acoplamento e coesão da organização obtida. A Figura 13 apresenta um modelo SADT que detalha a atividade AGRUPAR, de acordo com a divisão em etapas apresentada anteriormente. Este modelo é composto das seguintes atividades: IDENTIFICAR, DISTRIBUIR, VERIFICAR e INTEGRAR. Nas próximas seções, descreveremos as tarefas compreendidas em cada uma destas atividades.

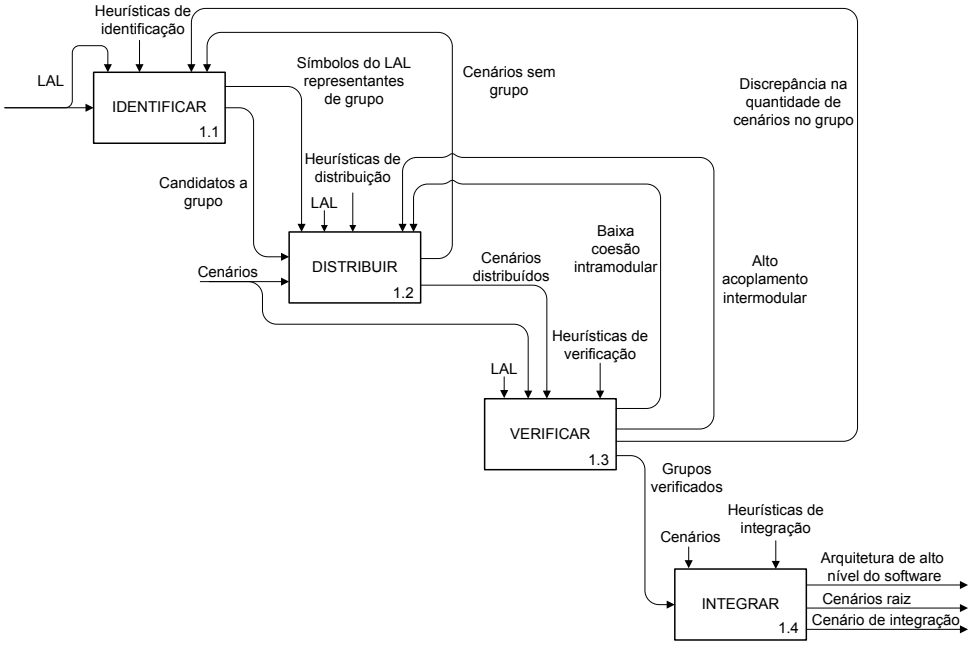

Figura 13. Atividade AGRUPAR detalhada através do SADT.

O resultado obtido através da execução destas atividades é uma divisão modular, que chamamos de arquitetura de alto nível do software. Esta arquitetura é composta de três níveis. O mais abstrato é composto dos cenários iniciais, que descrevem as situações do UdI. O segundo nível é representado pelos cenários raiz, que são cenários iniciais promovidos para representar os grupos. O último nível, e mais abstrato, é representado pelo cenário de integração, que provê uma breve descrição do software (Figura 14). Esta descrição é feita através da identificação do objetivo do software, das situações em que deve ser utilizado e da listagem de suas funcionalidades, respeitando a ordem em que são executadas.

É interessante destacar que uma abordagem semelhante, baseada no mesmo artefato (LAL), foi utilizada no com sucesso no contexto de linhas de produto (Clements e Northrop, 2005), para identificação de *features* (Almentero et al., 2013a).

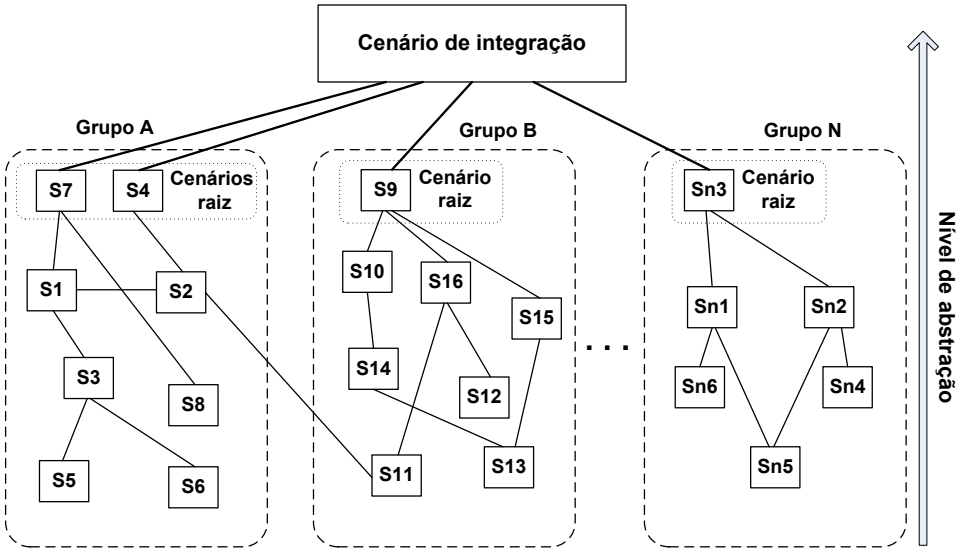

Figura 14. Arquitetura de alto nível (modelo conceitual).

3.2.1.1. Identificar

Esta atividade compreende a identificação de candidatos a grupo a partir do LAL do UdI. Qualquer um dos símbolos que compõe o LAL pode ser um candidato. A ideia é que os símbolos do LAL com grande número de relacionamentos, o que indica sua ampla utilização pelos atores do UdI, representam conceitos importantes dentro do domínio. Portanto, entendemos que seria natural utilizar os conceitos descritos através destes símbolos para representar módulos do software. A tarefa de encontrar estes símbolos é simplificada quando o LAL é interpretado através de um grafo não direcionado. Podemos considerar $G(V,E)$ um grafo que representa um UdI específico, composto por $|V|$ vértices e $|E|$ arestas, no qual cada vértice representa um símbolo do LAL e cada aresta representa o relacionamento entre dois símbolos. Consequentemente, o número de relacionamentos de um símbolo, isto é, o grau k_i de um vértice, pode ser calculado através da seguinte fórmula:

$$k_i = \sum_j e_{ij}$$

Onde e_{ij} significa uma aresta entre os vértices i e j. Portanto, a tarefa real consiste em selecionar os n símbolos de maior valor k_i do grafo. O número de grupos, representado pelo valor n, é variável, e dependerá do tamanho e

complexidade do software. Este valor deve ser determinado pelo engenheiro que está aplicando o método. Cada símbolo selecionado nesta etapa será utilizado para representar um grupo. Esta atividade deve ser executada com cuidado, pois o cálculo errado da quantidade de relacionamentos do símbolo pode levar a construção de uma arquitetura que não atenda aos princípios de mais aceitos de modularização (baixo acoplamento e alta coesão).

No inicio da realização desta atividade no software C&L, foi decidido, com base na quantidade de cenários, que o número de grupos candidatos, seria seis (Tabela 5), pois, desta forma, os grupos conteriam, na média, sete cenários. Através da formula apresentada acima, foi realizado o cálculo do grau k_i de todos os vértices do grafo que representa o LAL do software C&L. Na Tabela 5 exibimos os seis vértices com maior grau, em ordem decrescente.

Símbolo do LAL	**Número de relacionamentos**
Usuário	24
Cenário	19
Símbolo do LAL	16
Projeto	15
Colaborador	6
Sugestão ao projeto	6

Tabela 5. Número de relacionamentos dos símbolos do léxico.

3.2.1.2. Distribuir

Esta atividade consiste em distribuir os cenários que descrevem as situações do software, de forma que cada um seja alocado em um único grupo. Como vimos anteriormente, cada grupo é representado por um símbolo do LAL, que pode ser classificado como objeto, sujeito, estado ou verbo. Para determinar se um cenário pertence a um grupo específico são necessárias comparações, que dependem da classificação do símbolo que representa o grupo. Abaixo descrevemos as heurísticas de comparação que devem ser executadas para cada tipo de símbolo. Estas comparações devem ser feitas na seguinte ordem: objeto, sujeito, estado e verbo. Esta ordem foi definida, pois foi verificado, ao longo da aplicação em diferentes UdIs, que os símbolos do tipo objeto são mais

propensos a representar um módulo, seguidos dos símbolos sujeito, estado e, por último, símbolos do tipo verbo.

- **Objeto.** É necessário procurar por ocorrência do símbolo no objetivo do cenário, recursos e episódios. Se o cenário depender do objeto para alcançar seu objetivo, então deve ser colocado no grupo representado por este símbolo.

- **Sujeito.** É preciso procurar por ocorrência símbolo nos atores e episódios do cenário. Se o ator desempenha alguma função no cenário, contribuindo para que seu objetivo seja alcançado, então o cenário deve ser colocado no grupo representado por este símbolo.

- **Estado.** A busca por ocorrência do símbolo deve ser feita no objetivo e episódios do cenário. Se o estado descrito pelo símbolo é alcançado dentro do cenário ou faz parte de seu objetivo, então o cenário deve ser colocado no grupo representado pelo símbolo.

- **Verbo.** Ocorrências dos símbolos devem ser buscadas no objetivo e episódios do cenário. Se a ação descrita pelo símbolo deve se executada para que o objetivo do cenário seja alcançado, então devemos colocá-lo no grupo representado por este símbolo.

Ao final do procedimento descrito anteriormente, pode ocorrer à situação em que um ou mais cenários iniciais fiquem sem grupo. Neste caso, é necessário executar novamente a atividade IDENTIFICAR, de modo que a lista de símbolos mais citados seja ampliada, e, então, executar a distribuição dos símbolos que ficaram sem grupo novamente. Ao final desta atividade, teremos todos os cenários de requisitos alocados em um grupo representado por um dos símbolos mais citados, identificados anteriormente.

Apesar de necessitar de uma análise mais complexa, que requer a interação humana, a atividade DISTRIBUIR pode ser apoiada através de heurísticas automatizadas. Um exemplo de automatização é disponibilizar para o engenheiro os símbolos mais citados pelo cenário, ordenados de acordo com o critério de tipos (objeto, sujeito, estado e verbo) a ser seguido. Esta heurística pode ser facilmente implementada a partir da interpretação do LAL como um grafo.

Abaixo, apresentamos a distribuição dos cenários do software C&L entre os grupos identificados (Tabela 5), de acordo com as heurísticas apresentadas anteriormente.

- **usuário** = {criar usuário, visualizar usuário, editar usuário, autenticar usuário, sair do software}
- **cenário** = {criar cenário, visualizar cenário, editar cenário, remover cenário, visualizar sugestões para o cenário, aprovar sugestão para o cenário, rejeitar sugestão para o cenário}
- **símbolo do léxico** = {criar símbolo do léxico, visualizar símbolo do léxico, editar símbolo do léxico, remover símbolo do léxico, visualizar sugestões para o símbolo do léxico, aprovar sugestão para o símbolo do léxico, rejeitar sugestão para o símbolo do léxico}
- **projeto** = {criar projeto, visualizar projeto, editar projeto, exportar projeto, gerar grafo do projeto, adicionar colaborador ao projeto}
- **colaborador** = {}
- **sugestão ao projeto** = {}

Após a distribuição, os dois últimos grupos (colaborador e sugestão ao projeto) ficaram sem cenários. Consequentemente, eles foram descartados e apenas quatro módulos foram confirmados: usuário, cenário, símbolo do léxico e projeto.

3.2.1.3. Verificar

A intenção nesta atividade é detectar possíveis anomalias nos grupos formados anteriormente. Estas anomalias podem ser: alto acoplamento intermodular, baixa coesão intramodular e discrepância na quantidade de cenários. Estas anomalias afetam a qualidade da modularização do software que será construído. Portanto, as heurísticas de identificação propostas estão diretamente relacionadas com as métricas de coesão e acoplamento. A identificação de algumas destas anomalias é baseada nos relacionamentos existentes entre os cenários. Por isso, antes de executar esta atividade, é essencial que estes relacionamentos sejam identificados. Para identificar os quatro tipos de relacionamentos possíveis (precondição, subcenário, restrição e exceção), é preciso executar as seguintes comparações entre os cenários:

- **Precondição.** Procurar no contexto do cenário o título de todos os outros cenários.

- **Restrição:** (i) Procurar no atributo restrição do contexto do cenário o título de todos os outros cenários, (ii) procurar no atributo restrição dos recursos de um cenário o título de todos os outros cenários e (iii) procurar no atributo restrição de cada um dos episódios o título de todos os outros cenários.

- **Subcenário.** Procurar nos episódios do cenário o título de todos os outros cenários.

- **Exceção.** Procurar na exceção do cenário o título de todos os outros cenários.

Com o conhecimento dos relacionamentos entre os cenários, podemos iniciar a busca pelas anomalias na organização dos grupos. Abaixo, descrevemos as possíveis anomalias e suas respectivas heurísticas de identificação. Também detalharemos o tratamento a ser adotado quando uma delas é identificada.

- **Alto acoplamento intermodular.** Esta anomalia acontece quando temos um ou mais cenários com muitos relacionamentos fora do grupo. É importante destacar que é esperado que a camada de controle de um software, com arquitetura MVC, possua alto acoplamento. Entretanto, neste caso, estamos avaliando artefatos (cenários) que ainda não foram organizados de acordo com o padrão MVC, logo, não devem possuir tal característica. Com o intuito de amenizar este problema, a atividade DISTRIBUIR deve ser realizada novamente. Quando a distribuição for refeita, apenas os cenários com estas características (muitos relacionamentos intergrupo) devem ser realocados, de modo que os relacionamentos intergrupo sejam minimizados. Neste caso, durante a distribuição, devemos tentar alocar o cenário em outro grupo, representado por símbolo do mesmo tipo do anterior, ou alterar a ordem de comparação utilizada. Então, por exemplo, se

tivermos este problema com um cenário pertencente ao grupo de um símbolo objeto, teríamos que tentar alocá-lo em grupos de outros símbolos objeto. Se não houvesse tais grupos, ou se as comparações não forem bem sucedidas, então é necessário avançar na lista e iniciar comparações com símbolos do tipo sujeito e assim por diante. A descoberta deste tipo de anomalia pode ser feita de forma automática. Para tanto, é preciso contabilizar os relacionamentos intergrupo de cada cenário.

- **Baixa coesão intramodular.** Esta anomalia é caracterizada pela existência de um ou mais cenários dentro de um grupo, que descrevem situações sem relação conceitual com a maioria dos outros cenários do grupo. Neste caso, a atividade de DISTRIBUIÇÃO deve ser executada novamente, objetivando encontrar um grupo onde o cenário tenha uma coesão mais alta. A distribuição deve ser feita da mesma forma descrita acima, com a tentativa de alocação em outros grupos do mesmo tipo, avançando no critério de ordem entre os tipos caso necessário. Apesar de esta anomalia requerer um julgamento difícil de ser automatizado para ser identificada, é possível reconhecer casos potenciais através de uma análise automática. Esta análise é baseada na quantidade de relacionamentos dos cenários dentro do grupo. A ideia é que cenários com poucos relacionamentos intragrupo podem representar falta de coesão.

- **Discrepância na quantidade de cenários.** Nesta anomalia temos um ou mais grupos com muito mais cenários que os demais. Neste caso a atividade IDENTIFICAR deve ser refeita, de modo que mais grupos sejam formados. O objetivo é obter uma distribuição mais equilibrada de cenários entre os grupos. Esta anomalia pode ser facilmente identificada de forma automatizada, através da contabilização e comparação do número de cenários de cada grupo.

3.2.1.4. Integrar

Nesta última atividade, o objetivo é integrar os grupos obtidos, produzindo uma visão mais abstrata do UdI descrito pelos cenários. Esta atividade está divida em três etapas, que são (i) ordenar cenários, (ii) identificar cenários raiz e (iii) construir cenário integrador. A seguir descreveremos as tarefas executadas em cada uma destas etapas.

Ordenar cenários

Como vimos anteriormente, cenários estão conectados entre si através de quatro tipos de relacionamentos: subcenário, precondição, restrição e exceção. Através da análise destes relacionamentos, é possível estabelecer uma ordem entre os cenários. Por exemplo, se o cenário X tem, dentre suas precondições, o cenário Y, então este deve ser executado primeiro. Se o cenário Y, por sua vez, tem Z como um de seus subcenários, então Y deve ser executado primeiro. A mesma ideia pode ser utilizada para os relacionamentos de exceção e restrição. Se W for exceção de Z, então este deve ser executado primeiro. Igualmente, se K for restrição de Z, este deve ser executado primeiro. Assim, teríamos a seguinte ordem de execução entre estes cenários:

$$Y \rightarrow X \rightarrow Z \rightarrow W \rightarrow K$$

A tarefa de ordenar os cenários não é complexa e pode ser feita de forma automática, através da identificação dos tipos de relacionamentos entre cenários.

Identificar cenários raiz

Os cenários raiz são aqueles que não têm nenhum antecessor (precondição) dentro do grupo ao qual pertence, isto é, nenhum cenário do grupo precisa ser executado antes para que ele possa ser executado. Utilizando o exemplo acima, o cenário Y seria o cenário raiz de um grupo formado pelos cenários Y, X, Z, W e K. Um grupo pode ter mais de um cenário raiz. Caso haja mais de um cenário sem antecessor dentro do grupo, todos nesta condição são considerados cenário raiz. Este caso não tem nenhuma implicação no

desenvolvimento do software, apenas na construção do cenário de integração, como será explicado adiante.

Empregando as heurísticas acima ao software C&L, obtivemos o resultado exposto abaixo, onde os símbolos sublinhados são os cenários raiz identificados em cada grupo.

A representação abaixo indica que no grupo usuário o cenário raiz é “Criar usuário”. Este cenário é precondição para o cenário “Autenticar usuário”, que por sua vez é precondição para os cenários “Visualizar usuário” e “Sair do software”. O cenário “Visualizar cenário”, por sua vez, é precondição do cenário “Editar usuário”.

Usuário:

- criar usuário → autenticar usuário → visualizar usuário → editar usuário
- criar usuário → autenticar usuário → sair do software

No caso abaixo, "Criar cenário" é o cenário raiz do grupo. Este cenário é precondição para "Visualizar cenário" e "Visualizar sugestões para o cenário". O cenário "Visualizar cenário", por sua vez, é precondição para "Editar cenário" e "Remover cenário". Os dois últimos não possuem uma ordem definida entre si, por isso estão representados entre colchetes. Por fim, temos os cenários "Aprovar sugestão para o cenário" e "Rejeitar sugestão para o cenário", que não possuem uma ordem definida entre si, e tem como precondição o cenário "Visualizar sugestões para o cenário".

Cenário:

- criar cenário → visualizar cenário → {editar cenário, remover cenário}
- criar cenário → visualizar sugestões para o cenário → {aprovar sugestão para o cenário, rejeitar sugestão para o cenário}

Como podemos ver abaixo, o grupo símbolo do léxico tem o cenário "Criar símbolo do léxico" como raiz. Este cenário é precondição para "Visualizar símbolo do léxico" e "Visualizar sugestões para o símbolo do léxico". Os cenários "Editar símbolo do léxico" e "Remover símbolo do léxico" não possuem uma ordem definida entre si, e tem como precondição o cenário "Visualizar símbolo do léxico". Já os cenários "Aprovar sugestão para o símbolo do léxico" e

"Rejeitar sugestão para o símbolo do léxico" tem como precondição o cenário "Visualizar sugestões para o símbolo do léxico" e também não possuem uma ordem definida entre si, por isso são representados entre colchetes.

Símbolo do léxico:

- criar símbolo do léxico → visualizar símbolo do léxico → {editar símbolo do léxico, remover símbolo do léxico}
- criar símbolo do léxico → visualizar sugestões para o símbolo do léxico → {aprovar sugestão para o símbolo do léxico, rejeitar sugestão para o símbolo do léxico}

A seguir, são explicitados os relacionamentos entre cenários do grupo projeto. Como é possível ver, o cenário "Criar projeto" é raiz, pois não tem nenhuma precondição dentro do grupo e é precondição para os cenários "Visualizar projeto", "Exportar projeto" e "Gerar grafo do projeto". O cenário "Visualizar projeto", por sua vez, é precondição para os cenários "Editar projeto", "Adicionar colaborador ao projeto" e "Remover colaborador do projeto". Estes últimos são representados entre colchetes, pois não possuem uma ordem definida entre si.

Projeto:

- criar projeto → visualizar projeto → {editar projeto, adicionar colaborador ao projeto → remover colaborador do projeto}
- criar projeto → exportar projeto
- criar projeto → gerar grafo do projeto

Construir cenário integrador

O cenário integrador é um cenário criado artificialmente para relacionar e organizar os cenários raiz identificados, permitindo uma visão mais abstrata das situações e o conhecimento da sequência em que são executadas no UdI. Portanto, para criar este cenário, temos que estabelecer uma ordem entre os grupos formados nas atividades anteriores. Esta ordem é estabelecida através do mesmo raciocínio utilizado na identificação do cenário raiz, só que desta vez analisaremos os relacionamentos entre os cenários raiz. Cada episódio do cenário integrador será composto de um cenário raiz, mantendo-se a sequência

entre eles obtida anteriormente. O título do cenário raiz deve identificar o UdI que está sendo descrito. Seu objetivo deve ser composto de um resumo dos objetivos que se deseja alcançar com o software que será produzido. O contexto deve descrever em que situações o software é adequado para ser utilizado. Como este é um cenário artificial, não é necessário especificar os atores e recursos. Também não há necessidade de detalhar exceções e restrições**.** Com isso, pretendemos fornecer informações de alto nível que são essenciais para a decisão de utilizar o software, de forma semelhante ao que foi feito em (Leal et al., 2012).

Ao aplicar as heurísticas para ordenar os cenários raiz identificados no exemplo do software C&L, obtivemos o seguinte resultado:

criar usuário → criar projeto → {criar símbolo do léxico, criar cenário}

A representação acima nos diz que o cenário criar usuário deve ser executado primeiro, pois é precondição para o cenário "Criar projeto", que, por sua vez, é precondição para "Criar símbolo do léxico" e "Criar cenário". Estes dois últimos cenários não possuem uma ordem definida entre si.

Com base nesta ordem e empregando as heurísticas apresentadas acima, criamos o cenário integrador do software C&L, que pode ser visto na Tabela 6. Os caracteres "**#**", que aparecem nos episódios do cenário integrador, significam que os episódios compreendidos entre eles podem ser executados em qualquer ordem. Esta representação atende a situação identificada acima, de que não há ordem de execução definida entre os cenários "Criar cenário" e "Criar símbolo do léxico".

Título	Software C&L
Objetivo	Permitir a colaboração entre diversos usuários no desenvolvimento de projetos, utilizando cenários e símbolos do léxico, fornecendo meios para facilitar a integração com outras ferramentas.
Contexto	O usuário deseja modelar a linguagem e as situações de um UdI específico.
Atores	Usuário
Recursos	Conta, projeto, cenário e símbolo do léxico.
Episódios	1. Se o usuário não possuir uma conta, então CRIAR

	USUÁRIO, senão AUTENTICAR USUÁRIO. 2. O usuário pode CRIAR PROJETO ou selecionar um criado anteriormente. 3. #O usuário pode CRIAR CENÁRIO. 4. O usuário pode CRIAR SÍMBOLO DO LÉXICO.#

Tabela 6. Cenário integrador do software C&L.

3.2.2. Organizar

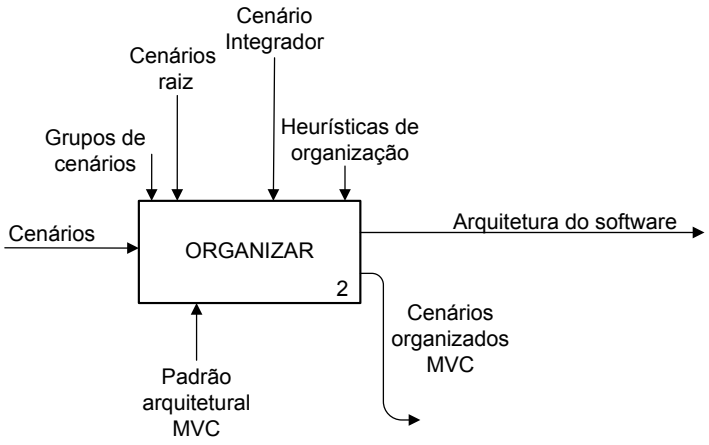

Figura 15. Atividade ORGANIZAR.

O objetivo desta atividade é derivar a arquitetura do software, a partir dos cenários iniciais, seguindo o padrão arquitetural MVC. Cada camada da arquitetura será descrita por novos cenários especializados, a fim de fornecer uma explicação mais detalhada. Estes cenários serão criados a partir dos iniciais, de forma que seja mantido um rastro entre eles. No intuito de proporcionar uma melhor descrição, decompomos a atividade ORGANIZAR em três novas atividades, que são: DIVIDIR, DESCREVER e VERIFICAR. O SADT contendo estas atividades é apresentado na Figura 16. Nas próximas seções, detalharemos as tarefas realizadas em cada uma destas atividades.

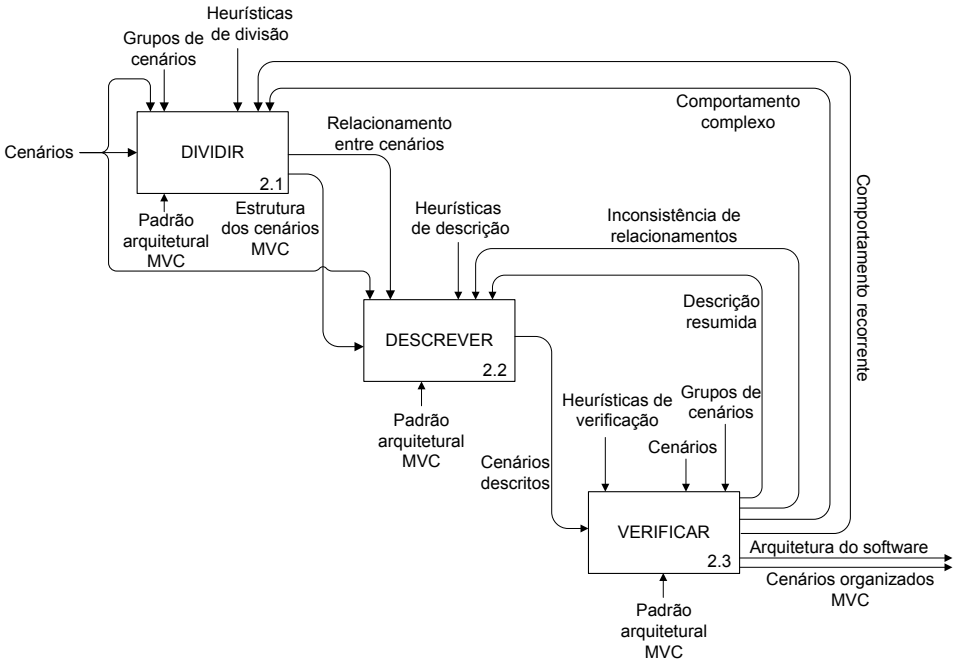

Figura 16. Atividade ORGANIZAR detalhada através do SADT.

3.2.2.1. Dividir

A divisão dos cenários iniciais é guiada pela estrutura do padrão arquitetural MVC. Como resultado desta atividade, cenários de visão, controle e modelo serão identificados e sua estrutura inicial será criada (Figura 17). A tarefa de criação destes cenários é realizada através de heurísticas, que dependem da camada em que estamos trabalhando. Descreveremos estas heurísticas nas próximas seções.

Visão

Os cenários de visão são responsáveis por descrever a interface do software para interação com o usuário. Nesta atividade, nosso foco principal é identificar estes cenários e criar apenas sua estrutura. Sua descrição completa será feita em uma atividade posterior.

Podemos presumir que todo cenário inicial possui pelo menos um ator, pois esta condição faz parte de suas regras de construção. Embora este ator possa ser um software, parte de um ou uma máquina, normalmente pelo menos um deles é humano. Isto acontece porque os cenários são escritos por usuários, que normalmente não conhecem o comportamento interno de um software.

Desta forma, sua descrição compreende a forma com que ele interage ou irá interagir com o software. No entanto, podemos ter cenários que descrevem exclusivamente uma situação interna do software. Nestes casos, embora tenhamos atores, eles são internos do software. Portanto, não será necessária uma interface para interação com o usuário.

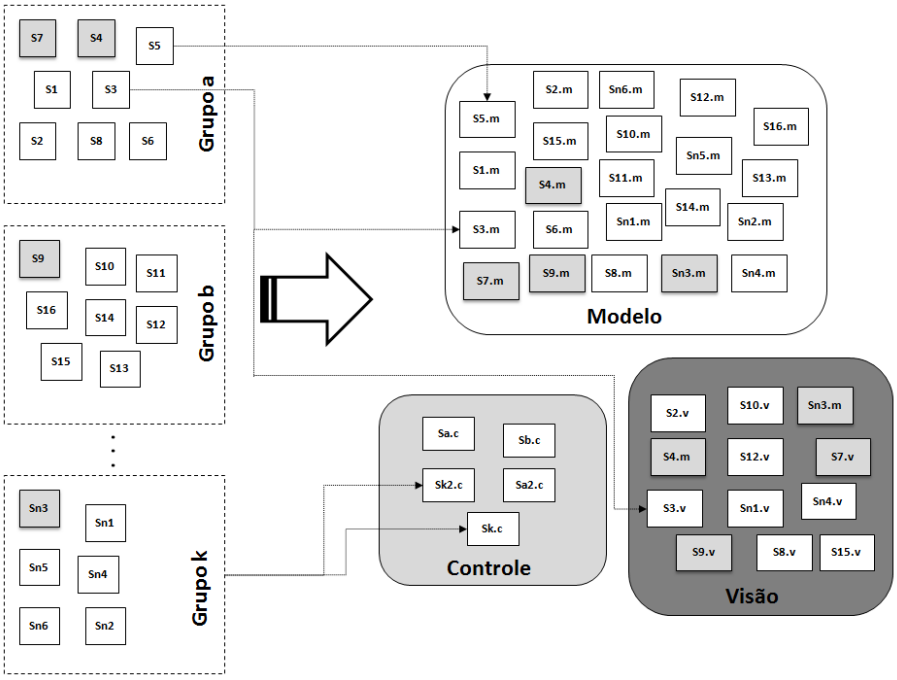

Figura 17. Tarefa de divisão dos cenários iniciais.

Toda interação com usuário requerer uma interface para possibilitá-la. A fim de detalhar esta interface, pelo menos um cenário de visão deve ser criado. O objetivo destes cenários é descrever, em detalhes, como a interface funciona. Esta especificação deve incluir os componentes da interface, as mensagens trocadas entre o software e o usuário, requisitos de qualidade (RNF) associados à interface e comportamentos excepcionais previstos *a priori*.

A criação destes cenários começa com a análise de cada cenário inicial (requisitos). Primeiro, iremos verificar os atores destes cenários. Aqueles que possuem pelo menos um humano dentre seus atores serão separados. Para garantir que todos os cenários que interagem com atores humanos foram identificados, é necessária uma busca mais refinada. Esta busca consiste em procurar por atores humanos nos episódios dos cenários iniciais. Isto é essencial, pois quando um ator humano tem um papel passivo dentro do cenário, como por exemplo, receber notificações do software, eles não são elencados na

lista de atores do software. Porém, mesmo nestes casos, o software deve possuir uma interface para interagir com eles, por mais simples que seja. Portanto, os cenários com esta característica também devem ser separados.

Após a identificação dos cenários que possuem interface com o usuário, iremos iniciar a criação dos cenários de visão para descrevê-las. Para determinar quais cenários de visão devem ser criados, iremos analisar novamente os episódios dos cenários separados. Cada interação com o usuário deve estar descrita através de um ou mais episódios. Após verificar estes episódios, é preciso decidir quais cenários de visão serão criados. Uma interface simples pode ser descrita através de apenas um cenário. Já uma interface mais complexa, pode exigir vários cenários para descrevê-la. Neste último caso, os cenários criados devem estar conectados através dos possíveis relacionamentos entre cenários. Neste momento, criamos apenas a estrutura dos cenários, descrevendo seus títulos e identificado os relacionamentos entre eles. O restante dos componentes destes cenários será descrito na próxima atividade.

Para que seja possível rastrear qual cenário inicial deu origem ao de visão, incluiremos o título do cenário inicial no componente contexto de cada cenário de visão. Desta forma, estamos criando um relacionamento de precondição entre os cenários iniciais e os de visão derivados a partir deles. O que não deixa de ser verídico, já que estes últimos não existiriam sem os primeiros.

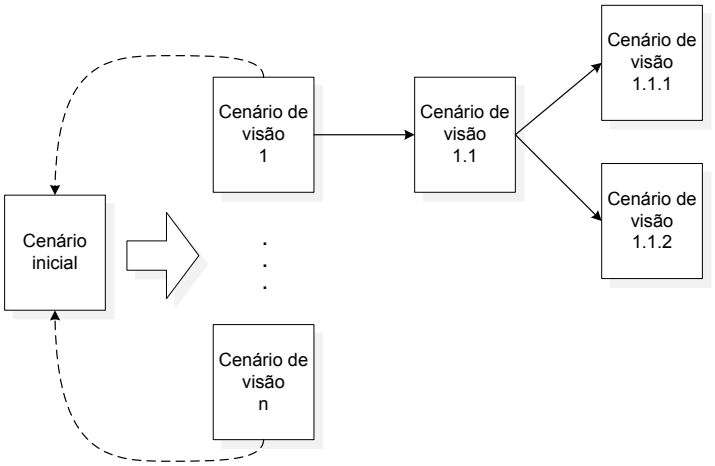

Figura 18. Tarefa de criação de cenários de visão.

A Figura 18 apresenta o que é realizado na tarefa de criação de cenários de visão. As setas normais representam os relacionamentos entre cenários de visão. As setas tracejadas representam o relacionamento de precondição, que passa a existir entre o primeiro nível de cenários de visão e o cenário inicial.

Como podemos observar, cada cenário inicial pode dar origem a vários cenários de visão, de forma que uma rede de cenários é criada. Entretanto, um cenário de visão está relacionado a apenas um cenário inicial.

Através da Figura 19, apresentamos um exemplo da atividade de divisão aplicada ao cenário "*criar cenário*" do software C&L. Neste exemplo, foram criados quatro cenários de visão. Um deles é responsável por apresentar a interface para inserção dos dados do cenário (*exibir formulário para cadastro de cenário*), outros dois, que tem como precondição a execução do primeiro, são utilizados para exibir a mensagem do resultado da operação para o usuário (*exibir mensagem de sucesso e exibir mensagem de erro*). O último, que tem como precondição o cenário "*exibir mensagem de sucesso*", é utilizado para exibir os dados do cenário cadastrado (*exibir cenário cadastrado*). Um relacionamento de precondição também é criado entre os cenários "*criar cenário*" e "*exibir formulário para cadastro de cenário*".

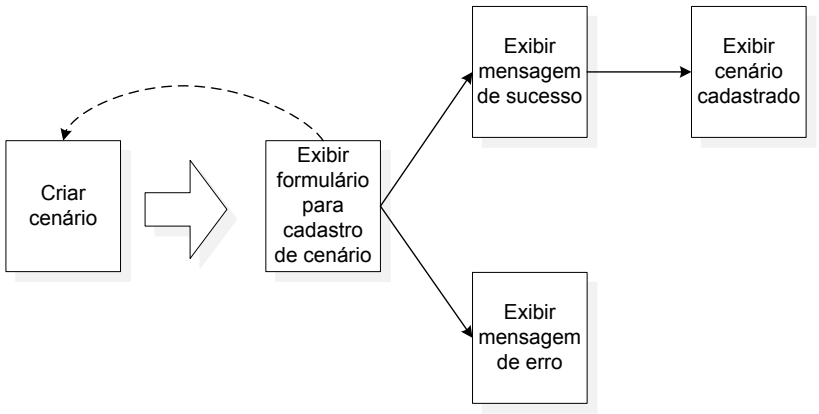

Figura 19. Exemplo de divisão de um cenário de visão do C&L.

Controle

Os cenários de controle são responsáveis por: (i) receber as requisições dos cenários de visão, (ii) aplicar as regras de negócio necessárias, (ii) encaminhar a informação para os cenários de modelo apropriados, (iv) coletar os resultados dos cenários de modelo e (v) atualizar a camada de visão adequadamente. Nesta etapa, o foco principal é identificar as requisições que podem vir da camada de visão e associá-las aos cenários de modelo apropriados.

É possível criar um cenário de controle para lidar com cada requisição proveniente da camada de visão. Porém, esta não é a melhor alternativa, pois geraria uma grande quantidade de cenários, tornando o software mais complexo. A alternativa que adotamos é agrupar as requisições que manipulam dados similares, de forma que seja possível criar um único cenário de controle para lidar com elas. Desta forma, simplificamos a camada de controle, tornando-a mais fácil de entender e manter.

Porém, determinar quais requisições devem ser colocadas no mesmo grupo não é uma tarefa simples, pois requer análise minuciosa de todas as requisições. No entanto, a partir da arquitetura de alto nível, construída na atividade anterior, é possível extrair uma informação que nos ajudará nesta tarefa. Os grupos de cenários, formados na atividade anterior, são uma boa base para utilizar a fim de agrupar as requisições que manipulam dados similares. Por estarem associados ao mesmo conceito, isto é, mesmo símbolo do LAL, temos um forte indicador que os cenários de um grupo lidam com conjunto de dados semelhante. Portanto, pelo menos um cenário de controle deve ser criado para lidar com as requisições dos cenários de um grupo específico. Com o objetivo de manter a rastreabilidade, o cenário de controle deve ter em seu contexto uma referência ao nome do símbolo do LAL que representa o grupo que o originou. Em software mais complexos, pode ser necessário criar mais de um cenário de controle para cada grupo. Esta decisão é subjetiva e fica sob responsabilidade do engenheiro de software. Entretanto, é preciso sempre levar em conta a complexidade que isto acarretará ao código do software, uma vez que mais elementos serão adicionados à camada de controle, dificultando o seu entendimento e manutenção.

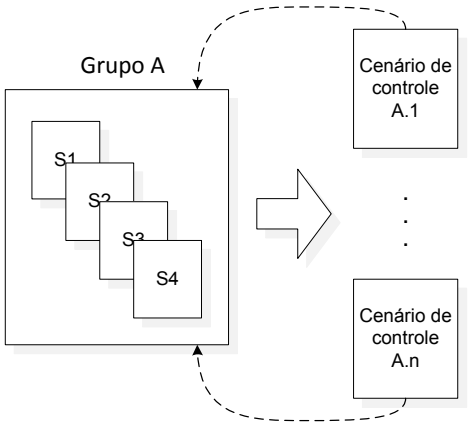

Figura 20. Tarefa de criação de cenários de controle.

A tarefa de criação de cenários de controle descrita é apresentada na Figura 20. As setas tracejadas representam a referência que cada cenário de controle terá em seu contexto ao grupo que o originou.

Na Figura 21, exibimos um exemplo de divisão aplicado ao controle do grupo "*cenário*" do software C&L. Neste caso, apenas um cenário foi criado (*controle do grupo cenário*) para gerenciar todo o fluxo dos cenários do grupo.

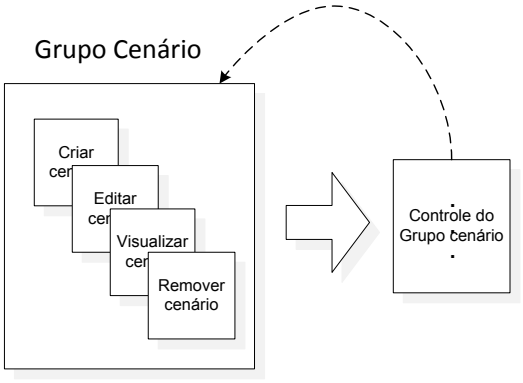

Figura 21. Exemplo de divisão de um cenário de visão do C&L.

Modelo

Os cenários de modelo encapsulam conhecimento sobre o modelo conceitual do software. Também são responsáveis por manter o estado do software e devem prover métodos para recuperar e alterar estes estados. A finalidade desta tarefa é apenas identificar estes cenários. Os detalhes de seu comportamento interno serão adicionados em uma atividade posterior.

Requisições da camada de visão, que acarretem manipulação de dados ou mudança/consulta de estado, envolvem pelo menos um cenário de modelo. Desta forma, para cada uma destas requisições, identificadas nos cenários de requisitos, pelo menos um cenário de modelo deverá ser criado. Requisições simples podem ser tratadas por um cenário apenas. As mais complexas, ou recorrentes, podem necessitar de mais de um cenário. Para manter o rastro entre os cenários iniciais e os de modelo, todo cenário de modelo deve conter em seu contexto o título do cenário de controle que o originou.

A Figura 22 apresenta a tarefa de criação de cenários de modelo. Neste caso, o cenário 1.1 foi criado para destacar um comportamento recorrente a vários cenários, provendo desta forma o reuso interno. As setas normais

representam o relacionamento entre os cenários de modelo. As setas tracejadas representam o relacionamento de precondição, estabelecido entre o primeiro nível dos cenários de modelo e os cenários iniciais.

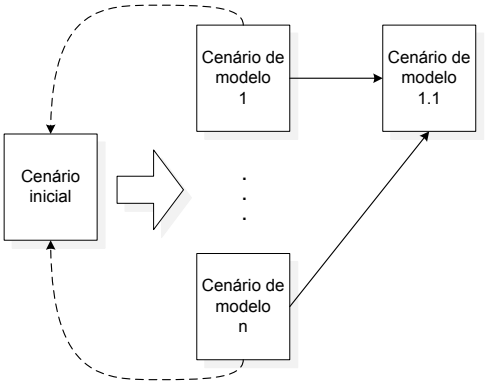

Figura 22. Tarefa de criação de cenários de modelo.

Exibimos na Figura 23 um exemplo de divisão de um cenário de modelo do software C&L. Neste exemplo, dois cenários de modelo foram criados: "*inserir cenário no projeto*" e "*executar consulta de inserção no banco de dados*". Neste caso, o segundo cenário tem como precondição a execução do primeiro, que por sua vez, tem como precondição o cenário de requisitos que deu origem a ele ("*criar cenário*").

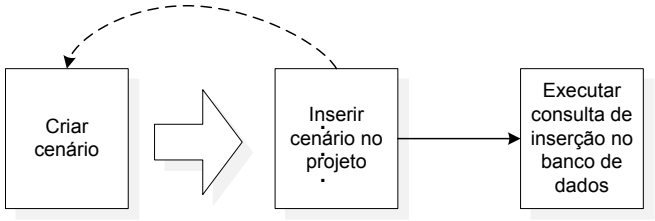

Figura 23. Exemplo de divisão de um cenário de modelo do C&L.

3.2.2.2. Descrever

Esta atividade compreende o detalhamento dos cenários, cuja estrutura inicial foi criada na atividade anterior. Todo relacionamento identificado anteriormente deve ser mantido. Especificamos as tarefas de detalhamento desta atividade através de um conjunto de heurísticas, que dependem da camada a qual o cenário pertence. Com o intuito de fundamentar a proposta de

detalhamento deste trabalho, iremos compará-la com o trabalho definido em (Staa, 2000).

Em seu trabalho, Staa[3] (Staa, 2000) afirma que a especificação de uma função deve conter as seguintes informações:

- **Objetivo.** Detalha a finalidade da função.

- **Acoplamento.** Identifica os dados requeridos pela função e seus tipos. Os resultados produzidos, juntamente com seus tipos, também devem ser identificados.

- **Interface com o usuário.** Identifica toda a comunicação realizada entre o software e o usuário feita através da função. O que inclui comandos, mensagens e relatórios.

- **Precondição.** Identifica as condições que os dados, arquivos e estados devem cumprir para que a função seja executada corretamente. É responsabilidade de quem utiliza a função assegurar a validade destas condições.

- **Poscondições.** Identifica as condições que os dados, arquivos e estados devem assumir após a execução da função. É responsabilidade do código da própria função assegurar a validade destas condições, desde que as precondições sejam atendidas.

- **Requisitos.** Estabelece as condições de aceitação da função. São utilizados para especificar características de performance, que devem ser satisfeitas pela função. Como, por exemplo, tempo de resposta e capacidade de armazenamento.

- **Hipótese.** Identifica as características do ambiente que são esperadas pela função, a fim de assegurar que as condições de aceitação são satisfeitas. Exemplos destas características seria a quantidade de memória RAM e acesso a internet.

[3] O livro do Prof. Arndt von Staa é utilizado como referência principal nos cursos de engenharia de software da PUC-Rio e em muitas outras universidades.

Nas seções subsequentes nos apoiamos no trabalho de Staa (Staa, 2000) para explicar como os cenários são utilizados para descrever o comportamento do software. Embora esta especificação tenha sido criada para descrever funções, acreditamos que sua estrutura contém as informações necessárias para permitir o entendimento do software como um todo.

Visão

Através do objetivo dos cenários de visão, iremos detalhar como a troca de informação entre usuário e software acontece. As condições e características do ambiente necessárias, para que a interface funcione corretamente, serão explicadas no contexto. Nos atores do cenário de visão, elencamos os usuários que interagem com a interface e os papeis que estes podem assumir. Os recursos do cenário são os componentes que constituem a interface. A ocorrência de cada elemento na interface é descrita pelos episódios, que também são utilizados para relacionar os cenários de visão com os de controle.

Normalmente, as precondições de um cenário devem ser garantidas pelo cenário que o chama. Porém, os cenários de visão, que possuem o usuário como ator, podem ser chamados diretamente por ele, sem nenhum intermediário. Neste caso em particular, as precondições do cenário devem ser verificadas em seus episódios e tratadas através de exceções. O detalhamento das heurísticas de descrição dos cenários de visão é apresentado na Tabela 7. É importante ressaltar que nenhum dos elementos da descrição é opcional. Todos são essenciais para o detalhamento adequado das interfaces do software.

Staa (Staa, 2000)	**Cenários (Leite et al., 1997)**	**Propósito**
Objetivo	Título	- Identifica o cenário que está descrevendo a interface. Dever ser único.
	Objetivo	- Descreve quais informações podem ser enviadas para o software pelo usuário através da interface. Também descreve quais informações podem ser enviadas para o usuário pelo software. Uma breve descrição de como a troca de informações é realizada deve

		ser fornecida. Como, por exemplo, através de formulários e relatórios.
Precondição	Contexto	- Identificar a condição inicial dos dados para o funcionamento correto da interface. Como, por exemplo, se é necessária à existência de um objeto de sessão contendo o id do usuário autenticado. - Especificar as características do ambiente que são requeridas para o funcionamento adequado da interface. - Identifica onde ocorre o relacionamento do tipo precondição com outros cenários de visão. - Identifica que o cenário pertence à camada de visão e o arquivo em que est que arquivo o cenário se encontra.
Hipótese		- Identifica as especificações técnicas do ambiente, requeridas para atingir o critério de qualidade estabelecido Como, por exemplo, internet de 1Mb para visualizar vídeo de alta definição.
Interface com usuário	Atores	- Identifica os perfis de usuário que interagem com a interface. Como, por exemplo, usuário comum e administrador.
Acoplamento	Recursos	- Lista os componentes da interface que recebem ou enviam informações para o usuário, incluindo aqueles que são criados dinamicamente durante a execução do cenário. O tipo associado a cada componente também deve ser identificado. Como por exemplo, idade - números naturais. - Lista arquivos e componentes externos utilizados pela interface. Como, por exemplo, arquivos JavaScript e CSS ou vídeos do youtube.
Poscondição	Episódios	- Identifica os componentes da interface exibidos pelo software.

		- Descreve as ações que os usuários podem aplicar nos componentes da interface. O estado assumido pelos componentes após as ações deve ser identificado. - Identifica onde ocorre o relacionamento do tipo subcenário com os outros cenários de visão e os de controle. Como, por exemplo, na submissão de um formulário ou chamada a uma função Java Script.
	Exceção	- Descreve o comportamento a ser tomado caso uma das precondições do cenário não seja cumprida. Como, por exemplo, se o navegador utilizado não dá suporte ao componente `<video>` do HTML 5, então o software deve exibir uma mensagem para o usuário informando o problema. - Descreve o comportamento excepcional relacionado aos componentes da interface e seu tratamento. * As condições, que os componentes manipulados durante a execução de comportamentos excepcionais assumirão, devem ser informadas. * O tratamento de exceções não precisa satisfazer o objetivo do cenário.
Requisitos	Restrição	- Descreve os atributos associados com a interface, que podem restringir a qualidade com que o objetivo do cenário é alcançado. Estes atributos não impedem que o objetivo seja alcançado. Eles podem estar associados com o contexto, recursos e episódios. Como, por exemplo, o formulário apresentado deve ser amigável. -Identifica onde ocorre o relacionamento do tipo restrição com outros cenários de visão.

Tabela 7. Heurísticas para descrever cenários da camada de visão.

A Tabela 8 apresenta um exemplo de cenário de visão extraído do software C&L. O cenário em questão descreve a interface utilizada para cadastrar cenários em um projeto. Os termos sublinhados, escritos em letra minúscula, representam símbolos do LAL do UdI. Os escritos com letra maiúscula destacam a referência a outros cenários do UdI. Neste caso específico, há uma referência ao cenário inicial (cadastrar cenário no projeto), dois da camada de visão (exibir página inicial do projeto e exibir formulário para login) e um da camada de controle (controle do grupo cenário).

Título: Exibir formulário para cadastro de cenário

Objetivo: Exibe um formulário para o usuário, contendo os campos e controles necessários para que ele informe os dados do cenário que será cadastrado no software.

Contexto:

Precondições:

- Cenário inicial relacionado: CADASTRAR CENÁRIO NO PROJETO.

- Usuário deve estar autenticado e ter permissão de adicionar cenário no projeto selecionado.

- EXIBIR PÁGINA INICIAL DO PROJETO.

Localização: camada de visão

Atores: software C&L, usuário administrador, gerente, colaborador.

Recursos: campos para entrada de dados do tipo texto, botões para controle, textos informativos.

Episódios:

1. Software C&L apresenta campo para usuário informar o título do cenário.
2. Software C&L apresenta campo para usuário informar o objetivo do cenário.
3. Software C&L apresenta campo para usuário informar o contexto do cenário.
4. Software C&L apresenta campo para usuário informar os atores do cenário.
5. Software C&L apresenta campo para usuário informar os recursos do cenário.
6. Software C&L apresenta campo para usuário informar os episódios do cenário.
7. Software C&L apresenta campo para usuário informar as exceções do cenário.
8. Software C&L apresenta botão para usuário confirmar o cadastro do cenário.

Restrição: Este botão deve ser destacado.

9. Software C&L apresenta botão para usuário cancelar o cadastro do cenário. 10. Se o usuário pressionar o botão para cadastrar o cenário, então o Software C&L envia requisição de cadastrar cenário para o CONTROLE DO GRUPO CENÁRIO. 11. Se o usuário pressionar o botão de cancelar, então o Software C&L redireciona para EXIBIR PÁGINA INICIAL DO PROJETO. 12. Software C&L apresenta mensagem para o usuário informando se o cenário foi cadastrado ou não. **Exceções:** - Se o usuário não estiver autenticado, então ele é redirecionado para EXIBIR FORMULÁRIO PARA LOGIN. - Se o usuário não estiver permissão para adicionar cenário no projeto, então uma mensagem é apresentada informando que ele deve entrar em contato com um dos administradores do projeto.

Tabela 8. Exemplo de cenário de visão do software C&L.

Controle

O objetivo dos cenários de controle deve conter a descrição do tipo de requisição tratada no cenário, e uma breve descrição de como são atendidas. A condição necessária dos dados, que são recebidos da camada de visão, deve ser detalhada no contexto do cenário de controle. As especificações mínimas do ambiente, para que o objetivo possa ser alcançado, também devem ser descritas no contexto. Como estes cenários não interagem diretamente com o usuário, seus atores se resumem ao próprio software. Os recursos do cenário são os dados manipulados para atender as requisições. Os artefatos externos, como bibliotecas, também devem ser identificados nos recursos do cenário. Os episódios devem identificar as requisições recebidas, e detalhar as ações tomadas para atendê-las. Como os cenários de modelo serão chamados a partir dos de controle, suas precondições devem ser validadas por eles. O comportamento do sistema, quando uma destas condições não é atendida, deve ser detalhado através de uma exceção.

O tratamento dado quando uma requisição desconhecida é recebida, também deve ser descrito através de uma exceção. Finalmente, os atributos de qualidade, relacionados com a precondição, recursos ou episódios do cenário de

controle, são detalhados como restrições do cenário de controle. Na Tabela 9[4], detalhamos as heurísticas para criação de cenários de controle, fazendo um comparativo com o trabalho de Staa (Staa, 2000). Nenhum dos elementos da descrição apresentada pode ser omitido. Todos são essenciais para o detalhamento adequado da camada de controle do software.

Staa (Staa, 2000)	**Cenários (Leite et al., 1997)**	**Propósito**
Objetivo	Título	Identifica o cenário de controle. Deve ser único.
	Objetivo	- Descreve os tipos de requisições que podem ser atendidas pelo cenário. Uma breve descrição de como estas requisições serão atendidas deve ser fornecida.
Precondição	Contexto	- Identifica as condições dos dados necessárias para que a requisição seja atendida. Como, por exemplo, se os dados obrigatórios foram informados.
Hipótese		- Identifica as especificações técnicas do ambiente, requeridas para atender os critérios de qualidade estabelecidos. Como, por exemplo, a exigência de um servidor com, no mínimo, 4Gb de memória RAM para abrigar o cenário de controle. - Identifica o grupo do qual foi derivado, a camada a qual o cenário pertence e o arquivo em que ele se encontra.
Interface com usuário	Os cenários desta camada não interagem diretamente com o usuário.	
Acoplamento	Atores	- O único ator nestes cenários é o próprio software. Componentes do software podem ser utilizadas para um maior detalhamento.

[4] Apesar de estarmos utilizando o mesmo formato da Tabela 7, cada caso, neste contexto, é instanciado de forma diferente. A mesma observação vale para as tabelas subsequentes, que utilizam o mesmo formato.

	Recursos	- Lista os dados que podem ser recebidos pelo cenário, identificando seu tipo. Como por exemplo, nome do usuário - *string*. - Lista as requisições que podem ser recebidas pelo cenário e o comando associados a elas. Como, por exemplo, requisição para adicionar usuário através do comando `adc_usuario`. - Lista outros componentes utilizados pelo cenário que fazem parte do software. Como, por exemplo, arquivos e bibliotecas.
Poscondição	Episódios	- Descreve, em ordem cronológica, as ações que são realizadas, para cada tipo de requisição que pode ser recebida pelo cenário. Estas ações incluem: (i) chamada aos cenários da camada de modelo, (ii) checagem das precondições dos cenários da camada de modelo e (iii) regras de negócio que devem ser aplicadas aos dados recebidos pelo cenário. A descrição deve permitir identificar o estado de um dado ao fim da execução de um episódio.
	Exceção	- Descreve o comportamento quando a precondição de um cenário de modelo não pode ser atendida. - Descreve o comportamento quando uma requisição desconhecida é recebida. - Descreve os comportamentos excepcionais associados com o controle do fluxo do sistema e suas regras de negócio. * A condição que os dados, manipulados durante a execução de comportamentos excepcionais, irá assumir ao fim de sua execução deve ser informada. * O tratamento de exceções não precisa, necessariamente, satisfazer o objetivo do cenário.

Requisitos	Restrição	- Descreve os atributos, associados com o cenário de controle, que podem restringir a qualidade com que o objetivo do cenário é alcançado. Estes atributos podem estar associados com: contexto, recursos ou episódios do cenário. Como, por exemplo, segurança quando estiver lidando com o número do cartão do usuário.

Tabela 9. Heurísticas para descrever cenários da camada de controle.

Na Tabela 9, é apresentado um exemplo de cenário de controle do software C&L. Este cenário, chamado "*Controle do grupo cenário*", é responsável por lidar com todas as requisições relacionadas aos dados de cenários. Assim com anteriormente, destacamos os símbolos do LAL referenciados sublinhando seus nomes. O título dos cenários referenciados, além de sublinhados, foram escritos em letras maiúsculas.

Título: Controle do grupo cenário
Objetivo: Recebe requisições relacionadas à consulta, alteração, remoção e cadastro de cenários. Encaminha as requisições ao serviço apropriado da camada de modelo.
Contexto:
Precondições:
- Grupo relacionado: cenário.
- Remoção ou consulta: o id do cenário é obrigatório.
- Alteração: o id do cenário e os dados alterados são obrigatórios.
- Cadastro: todos os dados do cenário são obrigatórios.

Localização: camada de controle.
Atores: software C&L
Recursos:
Dados: id - inteiro, título - texto, objetivo - texto, contexto - texto, atores - texto, recursos - texto, episódios - texto, exceção - texto.
Requisições: alterar (`alt_cen`), remover (`rmv_cen`), cadastrar (`cd_cen`) e consultar (`sel_cen`) cenário.
Utiliza: `modelo_cenario`

Episódios:

1. Se o software C&L recebe uma requisição para cadastrar cenário em um projeto, então os dados da requisição são validados. **Restrição:** a validação deve ser precisa e rápida.
   - Se os dados forem válidos, então INSERIR CENÁRIO NO PROJETO. Ao receber dados da inserção, EXIBIR RESULTADO INSERÇÃO.
   - Se os dados forem inválidos, então EXIBIR PÁGINA DE ERRO.
2. Se o software C&L recebe uma requisição para consultar um cenário em um projeto, então CONSULTAR CENÁRIO. Após receber os dados da camada de modelo, EXIBIR CENÁRIO.
3. Se o software C&L recebe uma requisição para remover cenário de um projeto, então REMOVER CENÁRIO. Após receber os dados da camada de modelo, EXIBIR RESULTADO REMOÇÃO.
4. Se o software C&L recebe uma requisição para alterar cenário em um projeto, então os dados da requisição são validados. **Restrição:** a validação deve ser precisa e rápida.
   - Se os dados forem válidos, então ALTERAR CENÁRIO. Ao receber dados da alteração, EXIBIR RESULTADO ALTERAÇÃO.
   - Se os dados forem inválidos, então EXIBIR PÁGINA DE ERRO.

Exceções:

- Se uma requisição desconhecida for recebida, EXIBIR PÁGINA DE ERRO.

Tabela 10. Exemplo de cenário de controle do software C&L.

Modelo

Cada cenário criado na camada de modelo descreve uma função do software. No objetivo deste cenário detalharemos a finalidade da função. As condições iniciais dos dados, exigidas para a correta execução da função, são definidas no contexto do cenário. Quando necessário, a localização geográfica e temporal exigidas, também pode ser detalhada no contexto.

Assim como os cenários de controle, os desta camada não interagem com o usuário do software. Desta forma, os atores do cenário se limitam ao próprio software ou partes dele. Nos recursos do cenário, listaremos os parâmetros da função e os seus respectivos tipos. Toda a lógica utilizada na função será descrita através de passos, ordenados cronologicamente, nos episódios do cenário. Mecanismos de desvio condicional e *loop* podem ser utilizados para

descrever com mais fidelidade à lógica necessária. Nos episódios também serão especificados os relacionamentos de subcenário, isto é, chamadas a outras funções. Os comportamentos excepcionais que podem ocorrer na função serão descritos através das exceções do cenário. Os atributos de qualidade, associados à função serão detalhados através de restrições. Na Tabela 11, as heurísticas para descrição dos cenários de modelo são organizadas e relacionadas com o trabalho de Staa (Staa, 2000). A descrição das funções deve conter todos os elementos da descrição apresentada. Todos são essenciais para o detalhamento adequado da camada de modelo do software.

Staa (Staa, 2000)	**Cenários (Leite et al., 1997)**	**Propósito**
Objetivo	Título	- Identifica o cenário de modelo. Deve ser único.
	Objetivo	- Descreve a finalidade da função que é descrita pelo cenário. Um resumo de como a finalidade é alcançada deve ser apresentado.
Precondição	Contexto	- Identifica a condição inicial dos dados para a execução do cenário. Como, por exemplo, um intervalo de valores específico. - Identifica, quando necessário, a localização geográfica e temporal, indispensável para a execução do cenário. Como, por exemplo, funções que executam de acordo com um horário específico. - Identifica o cenário inicial que o originou, a camada a que pertence e o arquivo onde está localizado.
Hipótese		- Identifica as especificações técnicas, requeridas para atender os critérios de qualidade estabelecidos para a funcionalidade. Como, por exemplo, velocidade mínima do processador.
Interface com usuário	Os cenários desta camada não interagem diretamente com o usuário.	
Acoplamento	Atores	- O único ator nestes cenários é o próprio

		software. Componentes do software podem ser utilizados para fornecer um detalhamento maior.
	Recursos	- Identifica os dados recebidos como parâmetro pela função descrita através do cenário. O tipo destes dados também deve ser informado. - Identifica os dados globais e arquivos (contendo dados) que são manipulados no corpo da função. - Identifica os componentes externos que são utilizados pela função. Como, por exemplo, bibliotecas e arquivos.
Poscondição	Episódios	- Detalham, através de passos em ordem cronológica, as ações executadas pelo software, utilizando os dados (recursos), com o propósito de alcançar o objetivo do cenário. Desvios condicionais e *loops* podem ser descritos através dos episódios, a fim de se obter uma descrição mais próxima do que será implementado. O estado que os dados irão assumir ao final da execução destes passos devem ficar claros.
	Exceção	- Descreve o comportamento quando uma precondição de um subcenário não é alcançada. - Descreve comportamentos excepcionais que podem ocorrer durante a execução da função. * O estado que os dados, manipulados durante os comportamentos excepcionais, irão assumir deve ser explicitado. * O tratamento de exceções não precisa, necessariamente, satisfazer o objetivo do cenário.
Requisitos	Restrição	- Descrevem os atributos de qualidade, associados com a função descrita pelo

		cenário, que podem restringir a qualidade com que o objetivo do cenário é alcançado. * Os atributos de qualidade não impedem o objetivo do cenário de ser alcançado. * Estes atributos podem estar associados com o contexto, recursos e episódios do cenário.

Tabela 11. Heurísticas para descrever cenários da camada de modelo.

Apresentamos, na Tabela 12, um exemplo de cenário de modelo do software C&L. Este cenário é responsável pela inserção dos dados, pertencentes a um cenário específico, no banco de dados. As referências aos símbolos do LAL foram sublinhadas. As referências a outros cenários, além de sublinhadas, foram escritas em letras maiúsculas. Desta forma, é possível identificar que o cenário descrito utiliza o cenário "*EXECUTAR CONSULTA DE INSERÇÃO NO BANCO DE DADOS* ", que faz parte de uma das bibliotecas do sistema.

Título: Inserir cenário no projeto

Objetivo: Cadastra um cenário no projeto informado, através da inserção de suas informações no banco de dados.

Contexto:

Precondições:

- Cenário inicial: CADASTRAR CENÁRIO NO PROJETO.
- O id do projeto recebido deve fazer referência a um projeto válido. Nenhum dado do cenário pode ser nulo.
- Localização: camada de modelo.

Atores: software C&L

Recursos:

- Parâmetros: id do projeto - inteiro, título - texto, objetivo - texto, contexto - texto, atores - texto, recursos - texto, episódios - texto, exceção - texto.
- Biblioteca para acesso ao banco de dados.

Episódios:

1. O software C&L monta consulta para inserção dos dados na tabela de cenários do banco de dados.
3. EXECUTAR CONSULTA DE INSERÇÃO NO BANCO DE DADOS.
4. O software C&L retorna o objeto com resultado da consulta ao CONTROLE DO GRUPO CENÁRIO.

Exceções:
- Se houver erro ao conectar com o banco de dados, escreve o código do erro e sua descrição no log do software C&L. Retorna ao CONTROLE DO GRUPO CENÁRIO mensagem informando que a operação não foi realizada.

Tabela 12. Exemplo de cenário de modelo do software C&L.

3.2.2.3. Verificar

O propósito desta atividade é procurar por possíveis problemas nos cenários de camada descritos. Com isto, esperamos resolver problemas que poderiam afetar as próximas atividades, garantindo, de certa forma, uma maior qualidade ao resultado final. A seguir, identificaremos os quatro pontos que precisam ser analisados. O comportamento a ser adotado, caso um dos problemas seja encontrado, também será descrito.

- **Descrição abstrata.** Neste caso devemos procurar por cenários com poucos episódios. Ou ainda, cenários com episódios curtos ou descritos de maneira muito superficial. Quando este tipo de situação é detectada, a atividade DESCREVER deve ser realizada novamente, com foco no maior detalhamento destes cenários. Apesar de requerer supervisão humana, partes desta tarefa, como encontrar cenários com poucos episódios, pode ter apoio automatizado.

- **Inconsistência de relacionamentos.** Existem dois tipos de inconsistência, relativas ao relacionamento entre cenários, que devem ser investigados. O primeiro, é que nenhum cenário deve estar isolado, isto é, sem nenhum relacionamento com outro cenário. Todo cenário deve ter, pelo menos, um relacionamento com outro cenário do software. Se um cenário for encontrado nesta situação, a atividade DESCREVER deve ser realizada novamente, com foco em encontrar os relacionamentos deste cenário. O segundo tipo de inconsistência ocorre quando os relacionamentos estabelecidos na atividade DIVIDIR, não são mantidos. Neste caso, a atividade DESCREVER deve ser executada novamente, de forma que a inconsistência seja resolvida. Ambas as inconsistências

citadas podem ser encontradas de forma automática, através da análise dos relacionamentos entre cenários.

- **Comportamento complexo.** A existência de episódios longos e confusos dentro de um cenário pode indicar a existência de comportamento complexo. Nestes casos, uma avaliação deve ser feita para determinar se a melhor opção não seria criar um novo cenário. Caso seja necessária a criação de um novo cenário, a atividade DIVIDIR deve ser executada novamente, de forma que estes cenário sejam criados. A correção deste problema melhora a descrição das situações do software. Pouca automatização pode ser utilizada na detecção deste problema, tornando-o um dos mais difíceis de ser encontrado.

- **Comportamento recorrente.** Como vimos, comportamentos repetidos em diversos cenários devem ser destacados em um novo cenário, a fim de promover o reuso e simplificar a descrição das situações. Porém, algum destes comportamentos pode ter passado despercebido. Assim, é aconselhável que seja feita uma nova análise para determinar se algum destes comportamentos não foi tratado adequadamente. Caso seja encontrada alguma ocorrência, a atividade DIVIDIR deve ser executada novamente, com foco na separação deste comportamento em um novo cenário, e criação dos relacionamentos necessários. Esta tarefa necessita de avaliação humana, portanto, pouco pode ser feito com uma ferramenta automática, tornando-a difícil de ser realizada.

3.2.3. Operacionalizar

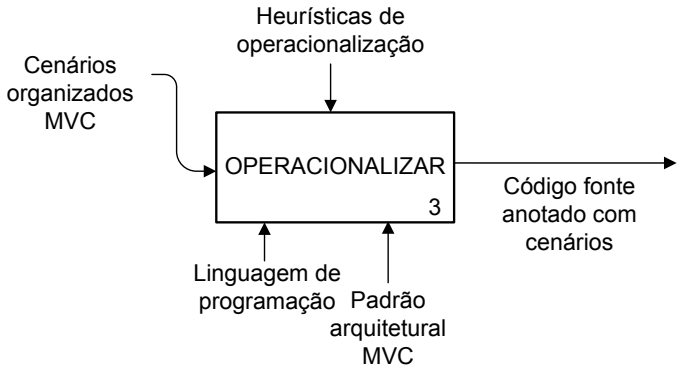

Figura 24. Atividade OPERACIONALIZAR.

Nesta última atividade, a meta principal é a escrita do código fonte do software. Para alcançar este objetivo, os cenários de camada, produzidos anteriormente, são refinados em novos cenários, os operacionais. A tarefa de refinamento é feita de modo que um rastro é mantido entre os cenários de camada e os operacionais. Tal rastro é criado a partir da inserção do nome do cenário de camada original, no contexto dos cenários operacionais criados a partir dele. Criando, desta forma, um relacionamento de precondição entre eles. Este rastro pode ser percorrido em ambos os sentidos, isto é, dos cenários de camada para os operacionais, e destes para os de camada.

Em outra etapa da atividade, os cenários operacionais são utilizados como um guia para escrita do código do software. Isto é feito de forma que, para cada episódio e exceção deste cenário, o código para operacionalizá-los é produzido. A fim de representar melhor as situações do código fonte, propomos uma alteração na representação dos cenários operacionais durante esta atividade. As exceções de um cenário são sempre representadas como o último elemento, ao fim de sua descrição. Porém, ao longo do desenvolvimento deste trabalho, percebemos que agrupando todas as exceções ao fim dos cenários, perdíamos o vínculo com o episódio que a lançou. Portanto, decidimos mudar a representação e incluir as exceções logo abaixo dos episódios que podem lançá-las.

No fim da atividade, temos o código integrado com os cenários operacionais, de modo que apenas um documento é produzido. Com isto, obtemos a rastreabilidade bidirecional entre o código (cenários operacionais) e a arquitetura (cenários de camada).

Dividimos a atividade em três passos: detalhar os cenários, codificar os cenários operacionais e verificar os cenários implementados. A Figura 25 apresenta o SADT decompondo a atividade OPERACIONALIZAR.

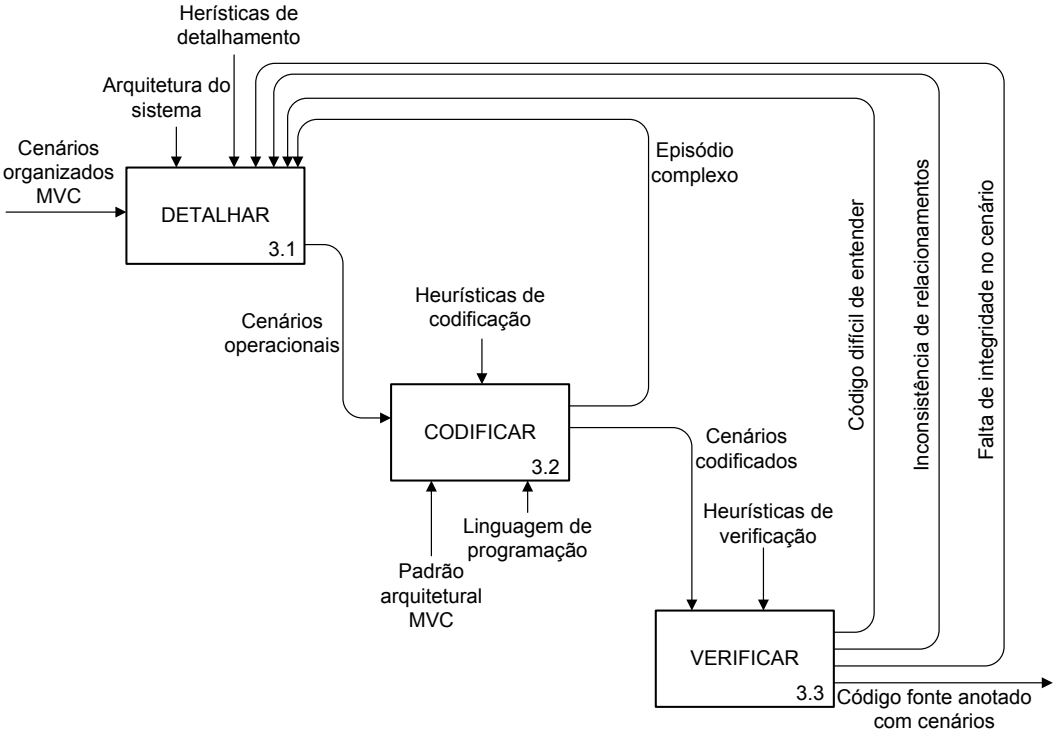

Figura 25. Atividade OPERACIONALIZAR detalhada através do SADT.

3.2.3.1. Detalhar

A fim de realizar o detalhamento, para cada cenário de camada pelo menos um cenário operacional deve ser criado. O detalhamento dos cenários de camada acontece, principalmente, nos episódios e exceção. Estes devem ser refinados, de forma que a descrição do cenário operacional resultante fique o mais próxima possível do código que será escrito. Como consequência, pode haver a necessidade de modificação destes elementos, para lidar com questões específicas da tecnologia envolvida, como fica claro adiante.

Outra consequência do detalhamento é a eliminação das restrições dos cenários. Como visto anteriormente, as restrições representam os RNFs associados ao cenário. Porém, a partir deste ponto do processo, todas estas restrições devem ser operacionalizadas através de um ou mais episódios.

As heurísticas para detalhamento dependem da camada a qual o cenário pertence. Nas seções subsequentes estas heurísticas serão apresentadas.

Visão

Os cenários de visão devem ser detalhados, de forma que cada componente da interface seja descrito, no cenário operacional, através de um episódio. Os relacionamentos de subcenários devem ser definidos através de um episódio a parte.

Como dito anteriormente, as tecnologias envolvidas na construção do software interferem diretamente no detalhamento dos cenários. Como nosso contexto é limitado a software Web, a camada de visão será, essencialmente, composta de páginas HTML, que são estáticas. O comportamento dinâmico deste tipo de interface é possível através do uso de linguagens de script, que manipulam seus componentes e informações. Dentre estas linguagens, a mais utilizada é o Java Script. Apesar de sua proximidade com o HTML, não consideramos esta linguagem como parte da camada de visão. Isto porque, cada vez mais, ela é utilizada para implementar comportamentos complexos, centrais ao software, com o intuito de se prover mais dinamismo a interação do usuário com interfaces Web. Um exemplo deste tipo de comportamento são as requisições assíncronas, através de XML, entre Java Script e servidor (AJAX) (Garret, 2005).

Portanto, neste trabalho, propomos a separação conceitual entre Java Script e visão. Para acomodar estas funções, que agregam dinamismo a interface, sugerimos a criação de uma subcamada, a de scripts. Os cenários operacionais desta camada atuam como um controle paralelo, interagindo com a camada de controle do software e atualizando a visão (interface).

Na Tabela 13 apresentamos as heurísticas para descrição dos cenários operacionais de script. Da mesma forma que nas demais camadas, situações complexas ou recorrentes, devem ser descritas através de um ou mais cenários operacionais, desde que se mantenham os relacionamentos apropriados entre eles.

Staa (Staa, 2000)	**Cenários (Leite et al., 1997)**	**Propósito**
Objetivo	Título	Identifica o cenário que está descrevendo o script. Deve ser único.
	Objetivo	Identifica o comportamento proporcionado pelo script e descreve, de forma breve, quais meios

		serão utilizados para proporcioná-lo.
Precondição	Contexto	- Identifica as condições iniciais dos recursos para execução do script. - Identifica, quando necessário, a localização geográfica e temporal, indispensáveis para execução do script. - Identifica a que cenário inicial está vinculado e o arquivo onde foi definido.
Hipótese		- Identifica as condições de ambiente indispensáveis para o correto funcionamento do script. Como, por exemplo, uma lista de navegadores compatíveis.
Interface com usuário	Os cenários desta camada não interagem diretamente com o usuário.	
Acoplamento	Atores	Apenas o sistema atua nestes cenários.
	Recursos	- Identifica os dados que são recebidos através de parâmetro pelo script. - Identifica as variáveis globais e arquivos (dados) que são manipulados no corpo do script. - Identifica componentes externos que são utilizados pelo script. Como, por exemplo, biblioteca JQuery. - Identifica os elementos do documento HTML (objeto *document*) que são utilizados pelo scritp. Como, por exemplo, o valor de um campo `input`.
Poscondição	Episódios	- Detalha o comportamento do script através de passos, que podem ser sequenciais ou condicionais. Estes passos devem contar com a utilização de um recurso. O estado de um recurso após a execução de um passo deve ficar claro. Se todos os passos forem executados corretamente, o objetivo do cenário deve ser alcançado.

	Exceção	- Descreve o comportamento caso uma precondição de um subcenário não seja atendida. - Descreve comportamentos excepcionais que podem ocorrer durante a execução de um script. * O estado que os dados, manipulados durante os comportamentos excepcionais, irão assumir deve ser explicitado. * O tratamento de exceções não precisa, necessariamente, satisfazer o objetivo do cenário.
Requisitos	Restrição	- Devem ser operacionalizadas através de um ou mais episódios.

Tabela 13. Heurísticas para descrever cenários operacionais de script.

Controle

Os cenários operacionais de controle podem ser utilizados para criar partes da interface que será exibida para o usuário (*tags* HTML). Ou ainda, quando a tecnologia AJAX é utilizada, para montar a estrutura dos objetos, normalmente XML ou JSON, que serão retornados a camada de visão contendo os dados necessários. Em ambos os casos, episódios devem identificar e descrever a criação das interfaces e objetos. Porém, de maneira geral, os episódios dos cenários operacionais de controle devem ser detalhados até que:

- Cada requisição, proveniente da camada de visão ou script, seja identificada através de um episódio;
- Chamadas a funções da camada de modelo sejam feitas através de um episódio;
- As regras de negócios sejam detalhadas por um ou mais episódios, de modo que cada episódio utilize apenas um recurso do cenário;

- Cada referência a componentes externos, como bibliotecas, deve ser feita através de um episódio a parte;

- O retorno para camada de visão é descrito através de um episódio a parte.

Se necessário, os outros elementos do cenário devem ser atualizados, a fim de acomodar as mudanças realizadas durante o detalhamento dos episódios.

Modelo

Na camada de modelo, é comum separar funções recorrentes em um cenário a parte. É possível, dependendo da linguagem, reescrever funções nativas ou criar novos métodos para objetos existentes. Isto é feito para facilitar o reuso e tornar o software mais fácil de entender e evoluir. Todas estas abordagens têm impacto nos cenários operacionais de modelo. Toda nova função, ou reescrita de uma existente, deve ser descrita através de um novo cenário operacional. A rede de relacionamentos entre os cenários, obtida na atividade anterior, deve ser evoluída para acomodar estas alterações. Entretanto, de maneira geral, o detalhamento dos cenários operacionais de modelo deve ser feito de forma que:

- Cada episódio descreva, em passos simples, as etapas realizadas, de forma que cada episódio envolva apenas um recurso do cenário em cada um.

- O uso de componentes externos, como bibliotecas, seja descrito por um episódio a parte;

- As chamadas a outros cenários da camada de modelo e DAL (acesso a dados) sejam feitas por episódios a parte;

- O retorno à camada de controle seja identificado e descrito por um episódio a parte.

Assim como nos cenários das outras camadas, se a modificação dos episódios tiver impacto nos outros elementos do cenário, estes devem ser atualizados para acomodar as mudanças realizadas.

3.2.3.2. Codificar

O processo de codificação consiste na tradução dos episódios e exceções, dos cenários operacionais, da linguagem natural para o código fonte apropriado. Esta tradução deve ser feita de forma que, quando o código produzido for executado, as exceções sejam tratadas ou o objetivo do cenário seja alcançado.

Devido à expressividade da linguagem natural, que é muito maior, mesmo após o detalhamento dos episódios, a tradução pode gerar um bloco grande e/ou complexo de código. Nestes casos, atividade DETALHAR deve ser executada novamente, para que o cenário operacional seja modificado e descreva melhor o código produzido.

Ao fim do processo de codificação dos episódios e exceções, o restante do cenário operacional deve ser verificado. A revisão deve considerar as modificações feitas nos episódios e exceções devido à implementação. Se, por exemplo, um novo recurso foi introduzido em um episódio devido à implementação, então ele deve ser incluído na lista de recursos do cenário. Nas seções subsequentes, apresentamos os detalhes específicos da atividade de codificação em cada camada.

Visão

Cada cenário operacional de visão é implementado através de uma nova interface. Desta forma, a tarefa de codificação consiste em, basicamente, incluir os componentes das interfaces descritos nos episódios do cenário. Porém, algumas modificações podem ser necessárias, como veremos a seguir. A Tabela 14 exibe um exemplo de cenário operacional de visão codificado do software C&L. Este cenário operacional descreve a exibição do formulário de cadastro de novos cenários. Como podemos observar, uma mudança em relação ao cenário de camada ocorreu no episódio 1. Uma subcamada de script foi criada para validar os dados do formulário, antes que ele fosse submetido para o controle. Desta forma, novos relacionamentos foram criados para contemplar a modificação. Outra observação importante de mudança, foi a criação de um novo

episódio (episódio 3), que descreve o espaço reservado a mensagens de erro sobre o campo "título". Por fim, é interessante ressaltar a operacionalização de uma restrição, que solicitava o destaque do botão para salvar o cenário. A implementação foi realizada no episódio 4, através de uma classe na folha de estilos.

<html>
@título: *Exibir formulário para cadastro de cenário operacionalizado.*
@objetivo: *Exibe um formulário para o usuário, contendo os campos e controles necessários para que ele informe os dados do cenário que será cadastrado no software.*
@contexto:
Precondições:
- cenário de camada relacionado: EXIBIR FORMULÁRIO PARA CADASTRO DE CENÁRIO.
- Usuário deve estar autenticado e ter permissão para adicionar cenário no projeto.
- Execução do cenário: EXIBIR PÁGINA INICIAL DO PROJETO.
Localização: camada de visão.
Arquivo: novo_cenario.html
@atores: software C&L, usuário administrador, gerente, colaborador.
@recursos:
- folha de estilos style.css e jquery-ui-1.8.13.custom.css, arquivo JavaScript cel_js.
- input título - texto, objetivo - texto, - contexto - texto, atores - texto, recurso - texto e episodios texto. Botões de salvar e confirmar.
<head>
<link rel="stylesheet" **type**="text/css" **href**="css/style.css" />
<link rel="stylesheet" **type**="text/css" **href**="css/smoothness/jquery-ui-1.8.13.custom.css"/>
<script src="js/cel_js.js" **type**="text/javascript"**></script>**
…
<!--
@episódio 1: *Se o usuário pressionar o botão para cadastrar o cenário "salvar cenário", então VALIDAR FORMULÁRIO DE CENÁRIOS.*
-->
$('# button_cen_cad_sav).button().click(function(){
modulo_scenario_js.validarDadosForm ("cadastro");
});
…
<!--
@episódio 2: *O Software C&L apresenta campo para usuário informar o título do*

cenário.
-->
<input id="inp_cen_cad_titulo" **name**="inp_scn_ cad _ titulo " **class**="input_style" type="text" value=""**>**
**
**
<!--
@episódio 3: *Se houver erro no campo nome, o Software C&L apresenta mensagem descrevendo o erro.*
-->
<span id="sp_error_scn_ cad _ titulo "_ **name**="sp_error_scn_ cad _ titulo " **class**="span_rg_error_style"**>**
…
<!--
@episódio 4: *Software C&L apresenta botão "Salvar Cenário" para usuário confirmar o cadastro do cenário. O botão é destacado através da classe* destaq_but_slv.
-->
<button type="button" **id**="button_cen_cad_sav" **name**=" button_cen_cad_sav " class="destaq_but_slv">Salvar Cenário**</button>**
…

Tabela 14. Exemplo de cenário operacional de visão codificado.

Controle

Um novo arquivo deve ser criado para cada cenário operacional de controle. Sua implementação será, normalmente, através de uma estrutura de seleção, onde cada tipo de requisição que o controle atende será representada. Os episódios do cenário serão codificados como opções da estrutura de seleção. As regras de negócio serão implementadas como passos, dentro do bloco de código de cada uma destas opções.

No exemplo de cenário de controle operacional codificado, apresentado na Tabela 15, apresentamos o controle responsável por todas as requisições do grupo cenário ("*Controle do grupo cenário")*. Entretanto, por questões de espaço, apenas o código que lida com a requisição para cadastrar novo cenário foi mantido. Neste código, podemos observar que relacionamento com o cenário operacional de modelo foi estabelecido no episódio 3. Porém, antes de realizar a chamada, foi necessário criar uma referência à camada de modelo antes, como descrito no episódio 2. Esta exigência surgiu devido à tecnologia utilizada, que neste caso foi a linguagem Lua (Ierusalimschy et al., 1996). Outra característica

interessante deste exemplo, é que ele responde uma requisição AJAX. Portanto, no final dos cenários, os episódios 4 e 5 detalham a criação do objeto XML que será retornado à camada de script. Finalmente, através do episódio 6, o retorno à camada de script destacado.

```
<%
@título: Controle do grupo cenário operacionalizado
@objetivo: Recebe requisições relacionadas a consulta, alteração, remoção e
cadastro de cenários. Encaminha as requisições ao serviço apropriado da camada de
modelo.
@contexto:
Precondições
- cenário de camada relacionado: CONTROLE DO GRUPO CENÁRIO
- Remoção ou consulta: o id do cenário é obrigatório.
- Alteração: o id do cenário e os dados alterados são obrigatórios.
- Cadastro: todos os dados do cenário são obrigatórios.
Localização: camada de controle.
Arquivo: controle_cenario.lua
@atores: software C&L
@recursos:
Dados: id - inteiro, título - texto, objetivo - texto, contexto - texto, atores - texto,
recursos - texto, episódios - texto, exceção - texto.
Requisições: alterar (alt_cen), remover (rmv_cen), cadastrar (cd_cen) e consultar
(sel_cen) cenário.
Utiliza: modelo_cenario
<!--
@episódio 1: Importar arquivo de configuração com os caminhos necessários.
-->
dofile("../configuracao/ configuracao.lua")
package.path = package.path..return_path_cel()
<!--
@ episódio 2: Criar referência a camada de modelo.
-->
local modelo_cenario = require("modelo.modelo_cenario")
?>
<html>
<head>
</head>
<body>
<?lua
```

```
<!--
@ episódio 3: Se o comando recebido foi para cadastrar cenário (cd_cen) então
INSERIR CENÁRIO NO PROJETO
-->
if (cgilua.POST.comando == "cd_cen") then
        local result_ins_cen, id_cenario, msg_erro =
        modelo_cenario.insere_cenario_projeto(dados_cenario);
<!--
@ episódio 4: Se ocorrer um erro, monta um XML de resposta informando que a
inserção no ocorreu e a mensagem de erro.
-->
        xml = xml..[[<resultado>]]
        if (not result_ins_cen) then
                xml = xml..[[< resultado _ ins >false</ resultado _ ins>]]
                xml = xml..[[<msg_erro >]].. msg_erro..[[</ msg_erro >]]
        else
<!--
@ episódio 5: Se a inserção foi concluída com sucesso, então monta XML de
resposta informando que o cenário foi incluído e seu id.
-->
                xml = xml..[[< resultado _ ins >true</ resultado _ ins >]]
                xml = xml..[[<id_cen>]]..id_ cenario..[[</id_ cen>]]
        end
<!--
@ episódio 6: Retorna o objeto XML com a resposta para o scrip solicitante.
-->
        return xml
end
```

Tabela 15. Exemplo de cenário operacional de controle codificado.

Modelo

Para cada cenário operacional de modelo, uma função deve ser criada. Se houver comportamentos recorrentes ou complexos dentro destas funções, elas podem ser decompostas em novas funções. Um cenário operacional deve ser criado para descrever cada uma das novas funções. Os novos cenários devem ser ligados aos anteriores através do relacionamento de subcenário. A Tabela 16 apresenta, como exemplo, um cenário de modelo do software C&L codificado. Como podemos observar, o cenário "*Inserir cenário no projeto operacionalizado*",

foi codificado através da função "*inserir_cenario*". Além disto, o episódio 3 é utilizado para indicar um relacionamento de subcenário com o cenário operacional "*Executar consulta de inserção no banco de dados*", que foi destacado por ser um comportamento recorrente, já que é responsável por executar todas as consultas de inserção no banco de dados. Por fim, o episódio 4 indica o retorno das informações à camada de controle.

```
<%
@título: Inserir cenário no projeto operacionalizado
@objetivo: Cadastra um cenário no projeto informado, através da inserção de suas
informações no banco de dados.
@contexto:
Precondições:
- cenário de camada relacionado: INSERIR CENÁRIO NO PROJETO.
- O id do projeto recebido deve fazer referência a um projeto válido. Nenhum dado do
cenário pode ser nulo.
- Localização: camada de modelo.
- Arquivo: modelo_cenaro.lua
@atores: software C&L
@recursos: id do projeto - inteiro, título - texto, objetivo - texto, contexto - texto, atores -
texto, recursos - texto, episódios - texto, exceção - texto, hora e data - objeto data.
- Biblioteca para execução de consultas no banco de dados.
function inserir_cenario(dados_cenario)
<!--
@episódio 1: O software C&L recupera a hora e data atual do sistema, no formato Ano-
dia-mês para indicar a hora em que o cenário foi incluído.
 -->
local data_inclusao = os.date("%Y-%m-%d %H:%M:%S")
<!--
@episódio 2: O software C&L monta consulta para inserção dos dados do cenário na
tabela de cenários do banco de dados.
-->
local consulta_inserir_cenario = "INSERT INTO cenario (TITULO, OBJETIVO,
CONTEXTO, ATORES, RECURSOS, EPISODIOS, EXCECAO, ID_PROJETO,
DATA_INCLUSAO, FLAG)"
.." VALUES (\""..dados_cenario ["titulo"].."\",\".. dados_cenario ["objetivo"].."\",\".."
dados_cenario ["contexto"].."\",\".. dados_cenario ["atores"].."\",\".. dados_cenario
["recursos"].."\",\".. dados_cenario ["episodios"].."\",\".. dados_cenario
["excecao"].."\",\".. dados_cenario ["id_projeto"].."\",\".. data_inclusao.."\", 1)"
```

```
<!--
@ episódio 3: EXECUTAR CONSULTA DE INSERÇÃO NO BANCO DE DADOS-->
local resultado_inserir_cen, id_novo_cen =
query_db.executar_consulta_insercao(consulta_inserir_cenario)
<!--
@ episódio 4: Retornar a camada de controle o resultado da inserção e o id do cenário
inserido.
-->
return resultado_inserir_cen, id_novo_cen
end
```

Tabela 16. Exemplo de cenário operacional de modelo codificado.

Para executar as consultas no banco de dados, é preciso adotar uma biblioteca específica para linguagem utilizada. Com o software C&L, utilizado como exemplo, foi desenvolvido em Lua, utilizamos a biblioteca LuaSQL (LuaSQL, 2013). Esta biblioteca permite a execução de consultas SQL em banco de dados MySQL, a partir de código desenvolvido em Lua. Neste caso, a biblioteca está desempenhando o papel do DAL (Data Access Library), indicado na arquitetura descrita na seção 2.3.

3.2.3.3. Verificar

Nesta atividade, examinamos os cenários operacionais codificados, a fim de assegurar que o resultado da codificação está correto, e mantém a rastreabilidade desejada com os demais cenários. A seguir, detalhamos os principais pontos que devem ser analisados e, caso algum problema seja encontrado, o tratamento a ser adotado.

- **Inconsistência nos relacionamentos.** Como vimos, durante a codificação novos relacionamentos entre cenários podem ser criados. Porém, em raras exceções, é necessário modificar os relacionamentos identificados nas atividades anteriores. Portanto, se algum destes relacionamentos não estiver presente, há um grande indício de inconsistência. Nestes casos, a atividade DETALHAR deve ser executava novamente, com o objetivo de resolver a inconsistência. Uma ferramenta que mapeie os

relacionamentos entre cenários pode facilmente encontrar este tipo de problema.

- **Integridade dos cenários.** Nos cenários operacionais de modelo e controle, todos os episódios são executados pelo mesmo ator, o software. Em cada episódio, o software utiliza um recurso para executar uma etapa que leva ao objetivo do cenário. Todos os recursos utilizados nos episódios devem ser descritos no componente "recursos" do cenário. Portanto, se um recurso for utilizado nos episódios e não aparecer nos recursos do cenário temos um problema de integridade. Para resolver o problema, basta executar novamente a atividade DETALHAR, e incluir os recursos que faltam no componente do cenário. Esta tarefa requer assistência de um humano, pois é muito difícil reconhecer um recurso na estrutura do episódio de forma automática.

- **Entendimento do código.** Com relação ao entendimento do código, podemos ter três problemas, que são: (i) grandes blocos de código sem episódios, (ii) trechos de código complexos descritos através de episódios curtos e (iii) longos trechos de código descritos através de episódios sucintos. Em qualquer um destes casos, a atividade DETALHAR deve ser realizada novamente, com o objetivo de criar descrições mais detalhadas das etapas executadas nos episódios do cenário operacional. Esta tarefa não pode ser automatizada, pois requer um julgamento humano para determinar se a descrição apresentada pode ser entendida facilmente.

Ao final desta atividade, após os cenários operacionais codificados passarem pela tarefa de verificação, teremos o software implementado. A evolução deste software deve ser feita com o apoio das atividades aqui apresentadas, de forma que se mantenha a atualidade dos cenários que o documentam, nos diferentes níveis de abstração.

3.3. Organizando o PDS+T para Reuso Orientado a Qualidade

Na seção 2.4 do capítulo anterior, apresentamos o Catálogo de Transparência de Software (CTS) e explicamos, de forma sucinta, o processo que levou a sua criação. Como visto anteriormente, este catálogo foi criado com o objetivo de permitir a organização de conhecimento, sobre operacionalizações associadas às qualidades que compõem o SIG de transparência (instanciado para o tópico software) e, consequentemente, o reuso destas operacionalizações. Através deste catálogo, podemos representar várias alternativas para operacionalizar uma determinada qualidade. Isto facilita o reuso orientado a qualidade (Leite et al., 2005), pois, por exemplo, se um desenvolvedor se deparar com uma destas qualidades em seu projeto, basta que procure no catálogo por uma operacionalização que atenda as exigências de seus envolvidos.

Uma das contribuições deste trabalho é a integração do PDS+T com o Catálogo de Transparência de Software (CTS), facilitando o seu reuso por outras pessoas. A seguir, apresentaremos como as atividades, propostas no processo PDS+T, foram integradas ao CTS. Como vimos, cada qualidade do CTS está decomposta em categorias, que, por sua vez, são utilizadas para organizar questões. A variabilidade do CTS se encontra justamente na forma como estas questões podem ser respondidas. Portanto, mostraremos como as operacionalizações elencadas nas atividades do processo PDS+T, podem ser utilizadas para responder as questões relacionadas a algumas qualidades presentes no CTS, que são dependência, detalhamento e rastreabilidade. O padrão Alternativa, proposto no CTS, foi utilizado a fim de permitir a representação das operacionalizações do PDS+T como alternativas para responder as questões relacionadas a estas qualidades.

3.3.1.
Atividade Agrupar

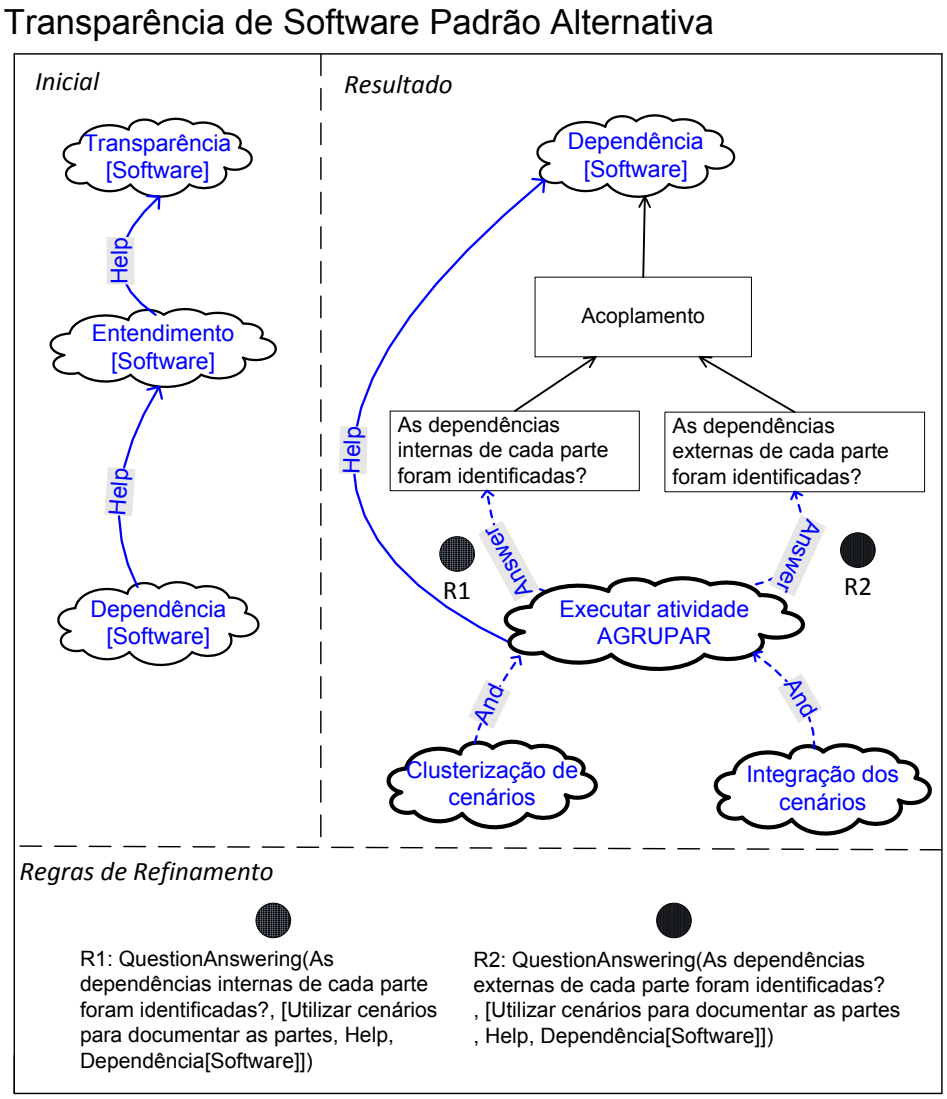

Figura 26. Alternativas para Dependência[Software].

De acordo com o Catálogo de Transparência de Software (CTS), uma das categorias da qualidade "*Dependência*", no contexto de software, é "*acoplamento"*. Esta categoria, por sua vez, possui duas questões associadas, que são *(Figura 26): "A dependência de cada parte interna foi identificada?*" e "*As dependências externas de cada parte foram identificadas?*". Considerando os cenários partes do software, é possível alegar que a arquitetura de alto-nível do sistema (modularização) torna explícita as dependências internas (intramódulo). Estas dependências são representadas através de relacionamentos entre cenários do mesmo grupo (módulo). Seguindo o mesmo raciocínio, é possível concluir que as dependências externas (intermódulo) são

representadas por relacionamentos entre cenários de diferentes grupos (módulos). Portanto, podemos afirmar que utilizar os cenários para documentar as partes do software, através da atividade AGRUPAR (Figura 12), é uma alternativa para responder as questões apresentadas anteriormente. Mais especificamente, as operacionalizações "*clusterização de cenários*" e "*integração de cenários*", presentes na AGRUPAR, são aquelas que nos permitem responder estas questões. Assim, contribuímos para o CTS, ajudando a qualidade de dependência aplicada ao tópico [Software]. Seguindo a rede de qualidades do catálogo, esta contribuição pode ser propagada até a qualidade detalhamento e, finalmente, até a transparência.

3.3.2. Atividades Agrupar e Organizar

Como vimos anteriormente, após a execução das atividades AGRUPAR e ORGANIZAR, definimos a arquitetura do software. Como foi demonstrado, estas atividades permitem a rastreabilidade entre os cenários de arquitetura e os de requisitos, isto é, rastreabilidade entre arquitetura e requisitos do software. No CTS, uma das qualidades que contribuem positivamente para transparência do software é a Rastreabilidade. Portanto, é possível adicionar estas duas atividades ao CTS, como alternativas para proporcionar rastreabilidade entre artefatos do software. Porém, a rastreabilidade aplicada ao tópico [Software] denota o rastro entre todos os artefatos desenvolvidos durante o processo de desenvolvimento do software. Enquanto as atividades identificadas acima, envolvem apenas requisitos e arquitetura. Desta forma, para representar a contribuição destas atividades no CTS, iremos decompor o tópico software em [Software.Requisitos-Arquitetura]. Assim, conseguiremos denotar exatamente a rastreabilidade compreendida pelas atividades AGRUPAR e ORGANIZAR.

Apresentamos na Figura 27 como esta alternativa é incorporada ao CTS. Neste caso, a pergunta "*Os rastros entre diferentes documentos da engenharia de software são mantidos*", que está relacionada à categoria "*Rastreabilidade em tempo de desenho*" é respondida pela operacionalização "*Derivar arquitetura dos requisitos*". Esta operacionalização foi criada para sumarizar as atividades do processo, de forma que a regra *QuestionAnswering* pudesse ser aplicada a apenas a uma operacionalização. Aplicar a regra *QuestionAnswering* a cada uma das operacionalizações (executar atividade AGRUPAR e executar atividade ORGANIZAR) implicaria que apenas uma das atividades seria

suficiente para responder a questão, o que não é verdade. A fim de explicar exatamente como a questão é respondida, a operacionalização "*Derivar arquitetura dos requisitos*" foi decomposta em outras duas, através de um relacionamento do tipo "and", o que implica que ambas devem ser executadas para responder a questão.

Transparência de Software Padrão Alternativa

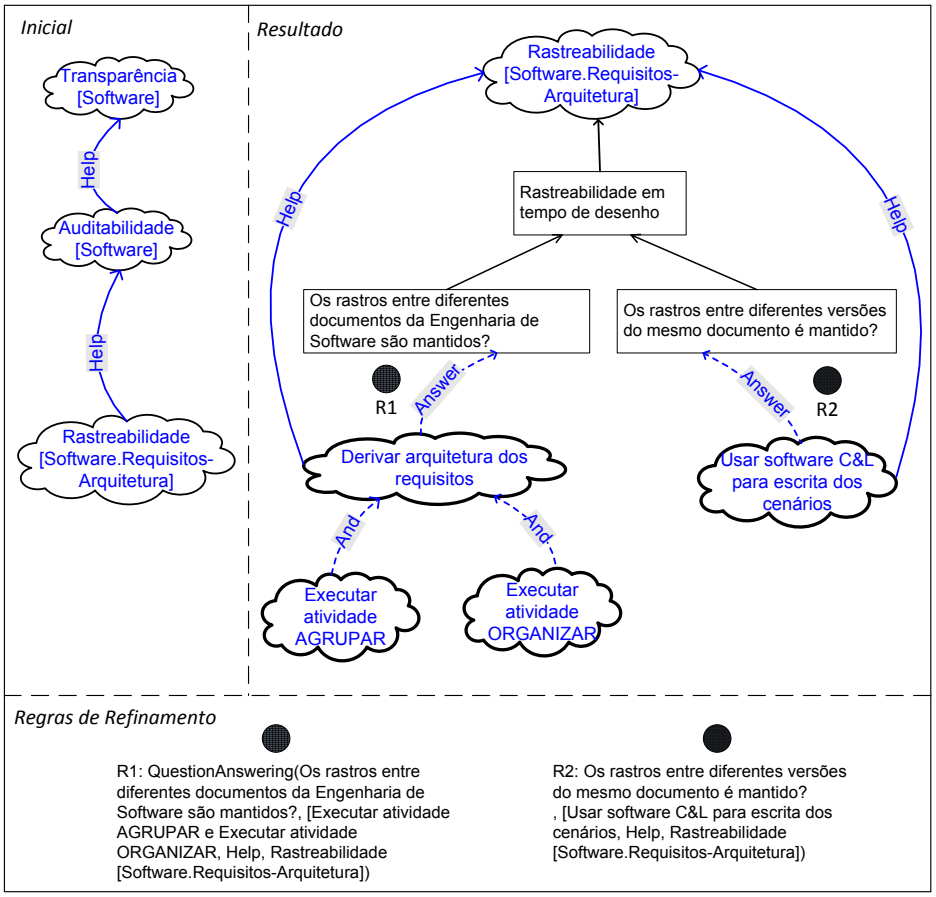

Figura 27. Alternativas para Rastreabilidade[Software.Requisitos-Arquitetura].

Da mesma forma, a questão "*Os rastros entre diferentes versões do mesmo documento é mantido?*", relacionada com a categoria "*Rastreabilidade em tempo de desenho*" é respondida pela operacionalização "*Usar software C&L para escrita dos cenários*" (Figura 27). Isto é verdade, pois o software C&L mantém um histórico de modificações feitas nos cenários em sua base de dados. Portanto, é possível resgatar qualquer versão anterior do cenário a qualquer momento.

É importante destacar que, nestes casos, apenas uma alternativa, que está dentre aquelas detalhadas neste trabalho, está sendo utilizada para responder

cada uma das questões. Porém, como se trata de um catálogo, é possível adicionar outras alternativas, como, por exemplo, "*Utilizar uma matriz de rastreabilidade*", que também contribuiriam para o CTS.

Respondendo a estas questões do catálogo através das atividades do processo PDS+T, estamos contribuindo com a rastreabilidade aplicada ao tópico [Software], para Auditabilidade e, consequentemente, contribuindo para a Transparência aplicada também ao tópico [Software].

3.3.3. Atividade Operacionalizar

Na Figura 28 apresentamos, através do padrão Alternativa, como a atividade OPERACIONALIZAR do PDS+T contribui para transparência do software. Assim como anteriormente, para manter o catálogo preciso, decompomos o tópico [Software] em [Software.Arquitetura-Código].

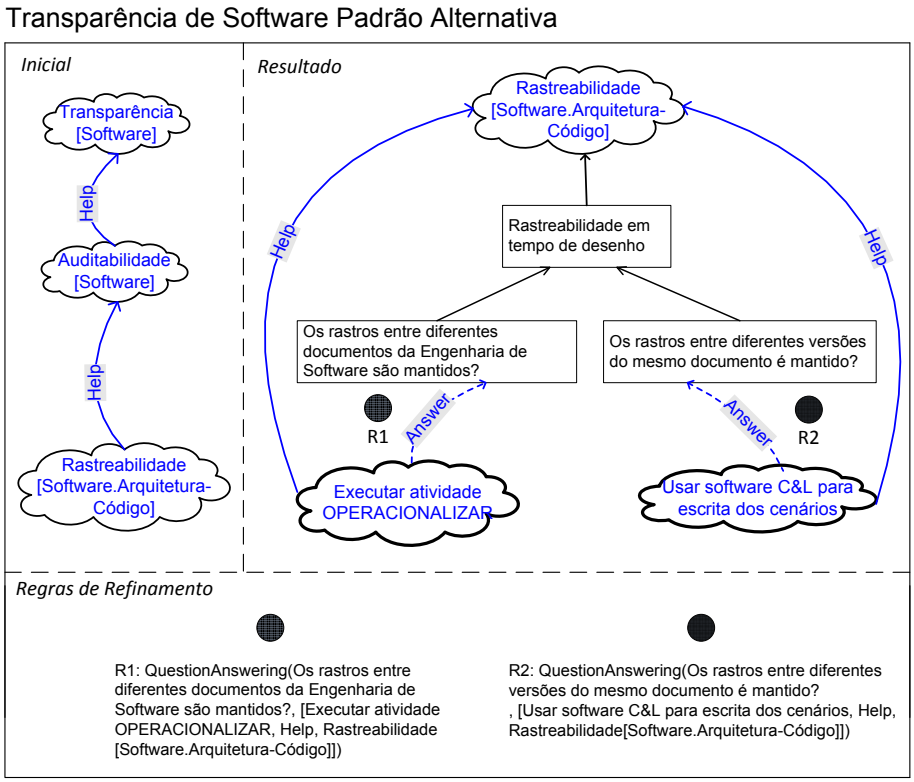

Figura 28. Alternativas para Rastreabilidade[Software.Arquitetura-Código].

A atividade OPERACIONALIZAR compreende o detalhamento dos cenários da arquitetura, tornando-os mais próximos ao código. Os cenários detalhados são implementados e permanecem integrados ao código, de forma

que é mantido um rastro entre estes cenários e os que compõem a arquitetura. Desta forma, podemos afirmar que a operacionalização "*Executar atividade OPERACIONALIZAR*" responde a questão "*Os rastros entre diferentes documentos da engenharia de software são mantidos*", relacionada à categoria "*Rastreabilidade em tempo de desenho*" especificamente para o tópico [Software.Arquitetura-Codigo].

Como dito anteriormente, o uso da ferramenta C&L responde a questão "*O rastro entre diferentes versões do mesmo documento são mantidos?*", pois permite que se tenha acesso a todas as versões dos cenários cadastrados nela.

Através da junção da rastreabilidade provida por esta atividade e pelas anteriores (Software.Requisitos-Arquitetura e Software.Arquitetura-Código), temos uma rastreabilidade desde os requisitos do software até o seu código. É importante ressaltar que esta rastreabilidade funciona em ambas as direções. Isto significa que é possível, partindo de um requisito, chegar ao código fonte que o implementa e também, partindo do código, identificar o requisito que deu origem a ele.

Assim, como feito anteriormente, as contribuições proporcionadas por esta atividade podem ser propagadas pelo CTS, até chegarmos à contribuição para a qualidade mais abstrata, que é a Transparência de Software.

3.3.4. PDS+T

Através da Figura 29, apresentamos o processo PDS+T como possível alternativa para responder questões relacionadas à qualidade Detalhamento. Neste caso, decompomos o processo PDS+T para que ficasse mais claro de que forma ele contribui para responder cada questão relacionada a esta qualidade. Para responder a questão "*O código está explicado?*", que está relacionada à categoria "*Explicar o software*" utilizamos a operacionalização "*Cenários integrados com o código*". Como foi explicado anteriormente, o produto final do processo PDS+T é o código do software integrado com cenários operacionais, de forma que seus episódios servem como uma descrição, em linguagem natural, do que está codificado. Portanto, acreditamos que esta característica pode ser considerada uma explicação do código do software.

Durante o processo PDS+T, os cenários iniciais são agrupados, integrados e decompostos em cenários de camadas, seguindo o padrão MVC. Considerando os cenários de camada componentes da arquitetura, e seus

relacionamentos as mensagens trocadas entre estes componentes, podemos afirmar que a arquitetura é explicada através de cenários. Isto é possível, pois cada componente é descrito através de um cenário, onde seu comportamento é detalhado, incluindo as mensagens trocadas com outros componentes (relacionamentos). Além disto, a integração entre estes diversos componentes é explicada através do cenário integrador. Portanto, incluímos a arquitetura baseada em cenários com uma alternativa para obter-se a explicação da arquitetura, como mostra a Figura 29. Ao descrever a contribuição através do padrão Alternativa, incluímos a operacionalização "*Arquitetura baseada em cenários*" como uma decomposição da operacionalização "Executar PDS+T. Esta operacionalização responde a questão "*A arquitetura está explicada?*", que esta relacionada à categoria "Explicar software", que, por sua vez, contribui positivamente para a qualidade Detalhamento aplicada ao tópico [Software].

Transparência de Software Padrão Alternativa

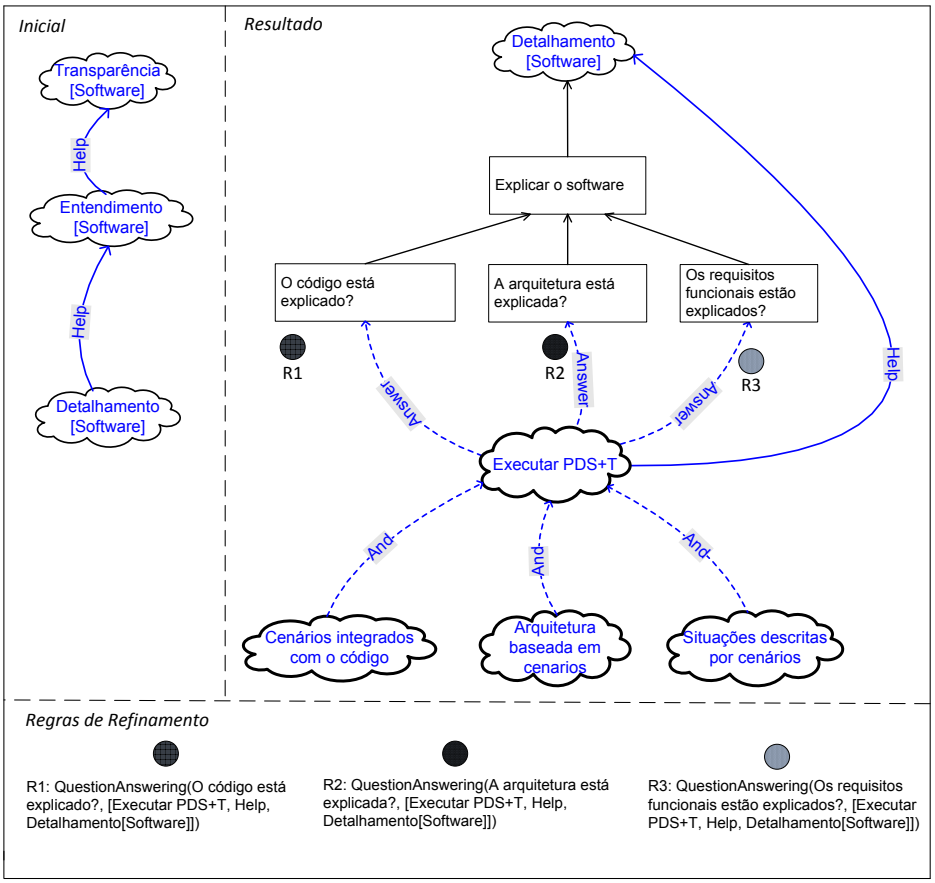

Figura 29. Alternativas para Detalhamento[Software].

Os cenários iniciais, que descrevem as situações do software, juntamente com o LAL, são o ponto de partida para o processo PDS+T. Esta representação foi escolhida, pois tem o foco na descrição do comportamento da situação, além de compreender o impacto dos RNF na situação. Dadas estas características, acreditamos que a utilização da linguagem de cenários é uma resposta sólida a questão "*Os requisitos funcionais são explicados*", que está associada à categoria "*Explicar os software*", contribuindo, desta forma, positivamente para o a qualidade Detalhamento aplicado ao tópico [Software]

3.4. Conclusão

Neste capítulo apresentamos e detalhamos as atividades do processo PDS+T. Com dito, estas atividades são iniciadas a partir do LAL e cenários de requisitos do UdI. Os cenários desempenham papel fundamental, permeando todas as atividades do processo, onde são refinados de forma a reduzir gradativamente seu nível de abstração. Ao final, temos os cenários operacionais integrados ao código, servindo como uma descrição em linguagem natural do que ele faz. A arquitetura do software também é descrita através dos cenários de camadas. Exemplos retirados do software C&L foram utilizados para ilustrar a execução das atividades do processo. Também descrevemos como as contribuições do processo PDS+T foram anexadas ao CTS, a fim de permitir o reuso orientado a qualidade. Como foi destacado, estas contribuições são propagadas pela estrutura hierárquica do CTS, de forma que podem ser consideradas como contribuições positivas para tornar o software mais transparente.