KARIN KOOGAN BREITMAN

EVOLUÇÃO DE CENÁRIOS

Tese apresentada ao Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, como parte dos requisitos para obtenção do grau de Doutor em Informática.

Orientador: Julio Cesar Sampaio do Prado Leite

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 12 de Maio de 2000.

Ao André pelo apoio, amor e muitos,
muitos meninos.

Agradecimentos

Ao meu orientador Julio Cesar Sampaio do Prado Leite pela orientação, por sua visão única e pelo incentivo durante todo o processo de elaboração desta tese.

To my friends:
Dan for pointing to the road less travelled by, and
Julio, for making all the difference.

Ao Rangel, por ter acreditado em mim.

Ao Departamento de Informática da PUC-Rio, seus professores, funcionários e alunos pelo auxílio em incontáveis ocasiões.

Ao CNPq e ao Laboratório de Engenharia de Software (LES – PUC-Rio) pelos apoios financeiro e institucional.

Resumo

Nos últimos anos observamos que um número crescente de pesquisadores na área de Engenharia de Software tem adotado a técnica de cenários. Esta, através da utilização de descrições de situações próprias ao mundo real, aproxima clientes e desenvolvedores facilitando sua comunicação. Por outro lado, o grau de informalidade da técnica, permite com que esta seja utilizada sem apoio metodológico adequado. Este aspecto, que vem sendo apontado em especial pela literatura da área de Engenharia de Requisitos, tem sido pouco explorado e dificulta a gerência, o controle e sistematização da utilização de cenários. Partindo de um enfoque empírico, aprofundamos nossos conhecimentos sobre a utilização de cenários durante o processo de desenvolvimento de software através da realização de dois extensos estudos de caso. Os resultados estão resumidos através um modelo para a evolução de cenários, que auxilia na compreensão da dinâmica associada a evolução e fornece uma estratégia incremental, baseada na aplicação de operações bem definidas sobre um ou mais cenários da base.

Neste trabalho propomos uma organização que permite sistematizar a evolução de cenários de modo a incorporar esta técnica entre outras práticas de Engenharia de Software. Levando em conta o conhecimento sobre cenários encapsulado no modelo de evolução e, incorporando aspectos de gerência, propomos o Framework de evolução de cenários de modo a racionalizar a utilização e gerência destes artefatos em um contexto geral de desenvolvimento de software. O Framework proposto é configurável e, através da instanciação de *hot spots,* permite que seus usuários talhem estratégias para a gerência da evolução de cenários adequadas a suas necessidades. Oferecemos apoio automatizado ao Framework proposto através da ferramenta SET.

Abstract

In recent years, a growing number of researchers have turned to scenarios to improve communication among the stakeholders of a computer-based system under construction. Scenarios are informal descriptions of situations in the system environment and help the elicitation and validation of information during all stages of system development. On the other hand, the informality of scenarios allow them to be used without any specific methodological support. This aspect, that has been pointed out specially by the Requirements community, has been given little attention and makes the sistematic use of scenarios in software practice more difficult.

To understand what is required for effective use of scenarios in system development, we have conducted a series of empirical case studies that shed light on the complexities of scenario evolution. As a result of these case studies, we have produced a model of scenario evolution that explains the dynamics of the evolutionary process while providing a multi-tier approach to scenario evolution, based on operations and relationships of the model.

This thesis proposes an organization that allows an engineering approach to scenario evolution.The organization is based on domain knowledge embodied in the scenario evolution model and takes into consideration other software management issues, such as configuration management and traceability. The organization implies a framework for scenario evolution that can be used in any open-ended, general software development environment. This framework can be configured, through the use of hot spots, to tailor specific user needs.We also offer a prototype tool, called SET (Scenario Evolution Tool), that provides automated support to the framework

Sumário

Capítulo 1 – Introdução

1.1 MOTIVAÇÃO

Nos últimos anos observamos que um número crescente de pesquisadores na área de Engenharia de Software tem abraçado a técnica de cenários. Esta, através da utilização de descrições de situações próprias ao mundo real, aproxima clientes e desenvolvedores facilitando sua comunicação. Existe um consenso na literatura da área em torno do fato de que cenários, enquanto artefatos resultantes do processo de desenvolvimento de software, são persistentes [Sutcliffe95, Hsia94, Leite97, Rosson95, Jacobson92, Potts94, Kyng95, McGraw97, Weidenhaupt98]. Como tal, é natural supor que os cenários de uma aplicação sofram modificações ao longo do desenvolvimento da mesma. Estas mudanças surgem como resultado do aumento da compreensão do software a ser construído [Yeh90] e podem se manifestar na forma de adição, subtração ou refinamento do conteúdo dos cenários. Qualquer um destes casos resulta no aumento do número de versões de cenários. A criação de novas versões implica em que uma grande quantidade de informações tem de ser manipulada, criando a demanda para um processo que seja capaz de gerenciar a evolução de cenários.

Na realidade, a evolução de cenários é um processo muito complexo que vai além da criação de novas versões, como tentamos mostrar na Figura 1.1. Nesta Figura ilustramos uma série de cinco diferentes configurações de cenários e enfatizamos duas operações, indicadas pelas setas. Definimos como configuração o conjunto composto das últimas versões dos cenários da base tomados em um um momento específico. Na primeira operação, ilustrada a esquerda da figura, mostramos que os cenários C2, C3 e C4, todos em sua primeira versão (indicado pelo índice V1) e que fazem parte da Configuração I foram unidos em um único cenário, o novo cenário C5, que aparece em sua primeira versão na Configuração II. Na segunda operação, representada pelo conjunto de setas a direita, temos o caso oposto, o cenário C5 em sua terceira versão (indicada pelo índice V3) na Configuração IV se desmembra em dois cenários na

próxima configuração, o cenário C5 na versão V4 e o novo cenário C6 em sua primeira versão, V1. Note que na primeira das operações todos os cenários iniciais foram eliminados dando espaço ao surgimento de um novo, enquanto que na segunda operação além do aparecimento do novo cenário, o cenário original também foi mantido, porém mudou de versão. Na prática, este tipo de operação onde novos cenários são criados, excluídos, incorporados ou divididos em outros cenários é muito comum, dificultando tanto o acompanhamento do processo evolutivo quanto a rastreabilidade da informação contida nos cenários.

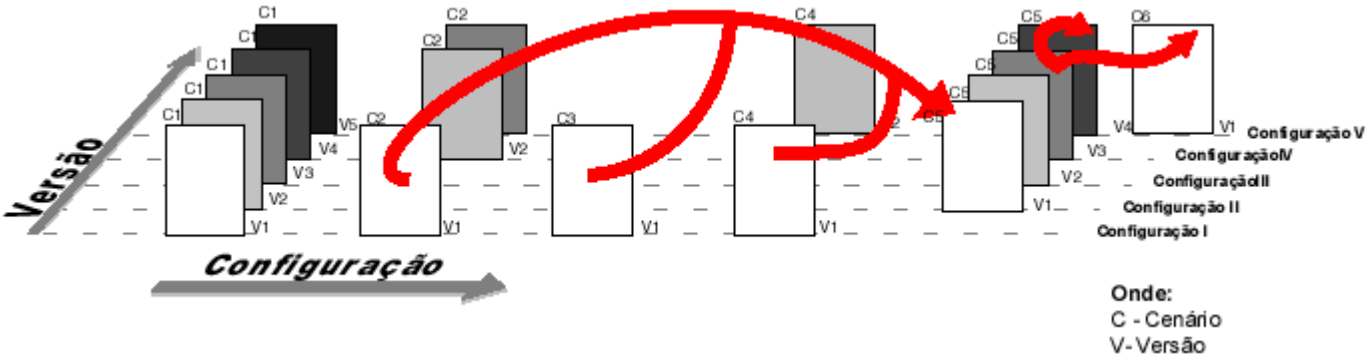


Figura 1.1 Um exemplo de evolução de cenários

Com o intuito de investigar o desenvolvimento de software baseado em cenários em maior detalhe, organizamos um estudo preliminar sobre o assunto. Utilizando um conjunto de heurísticas desenvolvidas e validadas no contexto da pesquisa em Engenharia de Requisitos de nosso grupo, [Leornadi 97, Hadad 97], orientamos sete grupos de alunos de graduação em um experimento de desenvolvimento de software cujo processo integrava as técnicas de cenários, elaboração de glossários da aplicação, projeto e implementação de sistemas orientados a objeto [Rumbaugh91, Booch99, Jacobson94, Whirfs-Brock90]. Um dos resultados deste experimento foi um embrião de uma taxonomia para a evolução de cenários, calcada em relacionamentos e operações, apresentada em [Breitman98].

A subsequente utilização da taxonomia na prática provou-se útil, porém limitada, apontando para a necessidade do aprofundamento destes resultados [Breitman98-b]. Paralelamente, outros autores indicaram problemas que usuários da técnica de cenários estavam enfrentando em relação a evolução dos mesmos: Weidenhaupt aponta para as dificuldades em manter consistência entre várias versões dos cenários,

e a falta de suporte para a gerência de mudanças entre os vários tipos e versões de cenários empregados atualmente, especialmente se estes encapsularem conhecimento em vários níveis de abstração [Weidenhaupt98]; Rolland aponta para deficiências que os vários modelos para a representação de cenários disponíveis na literatura apresentam em relação a definição do processo que deve ser empregado para a criação e gerência dos mesmos durante o desenvolvimento de software [Rolland98]; finalmente Jarke aponta que "*a organização e evolução de cenários devem ser vistas como grandes problemas, especialmente se:*

1. *múltiplos pontos de vista forem levados em conta (e.g., desenvolvedor, usuário e gerente em um único cenário) e,*
2. *o rastreamento de cenários através das várias fases do desenvolvimento (e.g., relacionamento entre cenários e protótipos e elaboração de cenários como forma de teste) for considerado*" [Jarke98]

Sob a influência da problemática descrita na literatura e partindo dos nossos resultados iniciais, decidimos investigar a fundo o processo de evolução de cenários, que culminou nos resultados que serão apresentados nesta tese. Descrevemos sucintamente o trabalho realizado na próxima subseção

1.2 VISÃO GERAL DA TESE

De modo a melhor compreender a evolução de cenários começamos por organizar um estudo de caso extenso, envolvendo um total de doze projetos, onde os objetivos principais eram a observação e a anotação de fenômenos relativos a evolução de cenários. Os projetos selecionados foram realizados ao longo dos dois últimos anos por terceiros, orientados por membros ou pessoas ligadas a da equipe de Engenharia de Requisitos da PUC-Rio. A autora, que organizou e analisou este estudo de caso, não teve envolvimento com os grupos.

A postura assumida ao investigar os mais de 800 episódios estudados foi de observador: anotamos e registramos os fenômenos relativos a evolução de cenários e os condensamos em uma Tabela que resume os resultados desta análise sob a forma

de uma taxonomia para a evolução de cenários. Esta taxonomia leva em conta tanto aspectos relativos a configurações de cenários, relacionamentos, quanto aspectos relativos ao versionamento dos mesmos, operações. A taxonomia é composta por oito relacionamentos que descrevem o modo em que os cenários estão interligados uns aos outros em uma mesma configuração, e.g. através de relações de pré condições ou relacionamentos de complementariedade, e um total de doze operações. As operações estão divididas em duas categorias, intra e inter cenários. A primeira diz respeitos a operações que envolvem apenas o conteúdo de um cenário, i.e., internas, enquanto que a segunda categoria se refere a operações que envolvem um ou mais cenários. Operações que resultam na adição ou exclusão de cenários a base pertencem a segunda categoria.

De modo a testar a robustez e aplicabilidade dos resultados do primeiro estudo de caso durante fases distintas do processo de desenvolvimento de software, elaboramos um segundo estudo de caso, desta vez de nossa autoria. A idéia era testar a taxonomia para a evolução de cenários em um sistema de software, desde sua elaboração até sua implementação. O sistema escolhido foi a simulação do controle do sistema de iluminação para o Departamento de Informática da Universidade de Kaiserslautern, proposto como estudo de caso durante um recente workshop em Engenharia de Requisitos realizado no Schloss Dagstuhl [Dagstuhl99]. Grupos de pesquisa em Engenharia de Software de diversas universidades se comprometeram em especificar este sistema, de modo a fornecer uma base para comparação entre as várias técnicas utilizadas. Os resultados desta e de outras simulações serão divulgados em uma edição especial do Journal of Universal Computer Science, editado on-line pela Springer-Verlag.

A realização do segundo estudo de caso nos permitiu refinar a taxonomia inicial e propor um modelo para a evolução de cenários. Além do modelo de evolução, a experiência adquirida no processo permitiu que elaborássemos uma primeira versão de um manual do usuário para a evolução de cenários sob a forma de um conjunto de heurísticas. Estas, através da detecção de relacionamentos entre cenários de uma configuração e da constatação de determinadas pré condições, sugerem a aplicação de determinadas estratégias de operações.

Ao longo da elaboração de ambos estudos de caso percebemos que o processo de evolução de cenários era grandemente influenciado por fatores estáticos, os relacionamento existentes entre cenários de uma mesma configuração. Desta forma, percebemos que seria impossível descrever o processo de evolução de cenários sem referenciar aspectos relativos ao produto cenário, i.e., a notação utilizada na captura e representação e os relacionamentos mantidos entre as instâncias dos cenários a cada configuração. O modelo de evolução de cenários proposto leva em conta esta constatação ao incorporar uma representação que evidencia tanto os relacionamentos existentes entre cenários, quanto as operações válidas a serem realizadas. Desta forma abrange tanto aspectos de produto quanto de processo relativos aos cenários. Os resultados dos estudos de caso realizados, apresentados através de uma taxonomia para evolução de cenários, um conjunto de heurísticas para a seleção e aplicação de operações e pelo modelo de evolução de cenários, são apresentados detalhadamente no capítulo quatro deste trabalho.

Percebemos, porém, que para realizar um processo eficaz de gerência de cenários, apenas a compreensão das particularidades associadas a evolução destes artefatos não seria suficiente. Aspectos mais gerais de gerência, tais como captura do *rationale* relacionado as operações, a armazenagem, gerência da configuração e apresentação da informação também teriam de ser levados em conta.

Estes aspectos são apresentados e discutidos em detalhe no capítulo cinco deste trabalho. Eles foram incorporados ao modelo de evolução e resultaram em um *Framework* para a evolução de cenários. O *Framework* proposto possibilita seus usuários instanciarem a estratégia de gerência de evolução mais adequada a suas necessidades. Desde um processo complexo, e.g., onde o *rationale* para todas as operações e capturado, várias formas de apresentação são disponibilizadas e o rastreamento entre os cenários e outros artefatos é explicitado, até processos mais enxutos, focados em operações específicas de interesse de clientes ou usuários, podem ser instanciados.

Finalmente apresentamos o protótipo da ferramenta SET (Scenario Evolution Tool) que oferece suporte automatizado para as atividades prescritas pelo *Framework* de evolução de cenários. A construção desta ferramenta se iniciou concomitantemente

com a elaboração do segundo estudo de caso, de modo a fornecer apoio automatizado a captura e armazenamento de cenários. A medida em que fomos avançando na elaboração do *Framework* proposto outras funcionalidades foram sendo adicionadas, tais como a captura do *rationale* ligado a operações. Finalmente foram implementados os mecanismos de instanciação dos projetos, que permitem que os usuários da ferramenta estabeleçam a estratégia para a gerência da evolução de cenários desejada.

A seguir resumimos as contribuições esperadas através da realização deste trabalho.

1.3 CONTRIBUIÇÕES ESPERADAS

Nesta seção apresentamos, sucintamente, as contribuições esperadas deste trabalho. Em primeiro lugar temos modelo de evolução de cenários que surgiu como resultados dos estudos de caso realizados. Este modelo explica a evolução de cenários ao mesmo tempo em que oferece uma estratégia, baseada no reconhecimento de relacionamentos e na aplicação de operações, que racionaliza este processo.

Juntamente com o modelo de evolução produzimos um conjunto de heurísticas para a aplicação de operações. Este pequeno compêndio relaciona a existência de determinados tipos de relacionamento entre cenários que figuram em uma mesma configuração e indica a aplicabilidade de operações sobre os próprios. Na prática serve como um guia *latu sensu* para a prática e escolha de operações

O conhecimento da evolução de cenários adquirido e representado pelo modelo de evolução, aliado a conceitos de Engenharia de Software, tais como gerência da configuração, visualização de grandes bases de dados e rastreabilidade permitiu a proposta de um *Framework* para evolução de cenários. Este permite a instanciação de um processo racional e organizado para a evolução de cenários talhado a necessidades dos clientes.

De modo a suportar o *Framework* de evolução de cenários apresentamos o protótipo da ferramenta SET. Este protótipo oferece apoio automatizado para a instanciação do *Framework* bem como suporte para o processo de gerência da evolução de cenários.

Finalmente disponibilizamos um exemplo extenso da evolução dos cenários de uma aplicação específica, o sistema para controle da iluminação. Este está publicado em forma de *site* e acreditamos que seja um exemplo didático que facilite a compreensão da complexidade envolvida no processo de evolução de cenários. Este *site* também evidencia que a técnica de cenários pode ser empregada durante diversas fases do desenvolvimento de software, desta forma auxiliando na disseminação da prática.

1.4 ORGANIZAÇÃO DA TESE

No capítulo 2 apresentamos conceitos básicos relativos a técnica de cenários, foco deste trabalho. Descrevemos algumas das particularidades do processo de software baseado em cenários e fazemos uma pequena revisão tanto das propostas para utilização e notação desta técnica, quanto para classificação de cenários presentes na literatura.

O capítulo 3 é focado em questões relativas a evolução de software. Apresentamos uma discussão sobre aspectos relativos a gerência da configuração e da captura do *rationale* de mudanças, com enfoque em questões relativas ao rastreamento destas informações. Apresentamos a baseline de requisitos, estrutura paralela ao desenvolvimento de software concebida para servir como apoio a atividades de gerência, entre outras as relativas a gerência de cenários, sobre as quais esta tese repousa.

No capítulo 4 apresentamos o modelo de evolução de cenários. Iniciamos por apresentar o primeiro estudo de caso conduzido e seus resultados na forma de uma taxonomia para evolução de cenários. A seguir apresentamos um segundo estudo de caso, realizado com o intuito de testar esta taxonomia. As conclusões que derivamos a partir da realização de ambos estão condensadas no modelo de evolução de cenários apresentado ao final do capítulo. Também introduzimos um conjunto de heurísticas para facilitar a utilização do modelo proposto por parte de seus usuários.

No capítulo 5 tecemos comentários sobre outros aspectos de gerência, além daqueles particulares a evolução de cenários, que devem ser levados em conta no

estabelecimento de uma estratégia eficaz para o processo evolutivo destes artefatos. O fruto desta discussão, adicionado ao modelo de evolução de cenários proposto no capítulo anterior, é traduzido através do *Framework* para a evolução de cenários proposto. Este *Framework* conta com apoio automatizado fornecido pelo protótipo da ferramenta SET (Scenario Evolution Tool) apresentada ao final do capítulo.

Finalmente, no capítulo 6 apresentamos nossas conclusões, juntamente com uma visão crítica das contribuições deste trabalho. Neste contexto, comparamos nossos resultados com outros da literatura da área. Concluímos a tese com um resumo do trabalho apresentado e sugerimos orientações para futuras pesquisas e melhoramentos.

No apêndice I apresentamos o material relativo aos cenários que compõem os doze projetos que formam o estudo de caso I, descrito no capítulo quatro.

1.5 RESUMO

Neste capítulo fizemos a introdução desta tese, cujo tema central é a evolução de cenários. Mostramos, de modo sucinto, as motivações que levaram a realização da mesma, uma visão geral do trabalho realizado e as contribuições esperadas. Finalizamos por apresentar a divisão em capítulos da tese.

No capítulo seguinte apresentaremos uma pequena revisão da utilização da técnica de cenários em diversas áreas do conhecimento, com o foco na ciência da computação.

Capítulo 2 - Desenvolvimento de Software Baseado em Cenários

Um grande passo no processo de definição dos requisitos de uma aplicação reside no reconhecimento dos atores, entidades e ações que descrevem o macrosistema do qual esta aplicação fará parte. Cenários vem surgindo como opção para a descrição de situações do mundo real que envolvem agentes interagindo dentro de um determinado contexto [Zorman95]. Esta interação é descrita através de ações enumeradas em um ou mais episódios. Cenários utilizam elementos conhecidos pelos clientes, facilitando tanto o processo de elicitação de requisitos quanto a validação dos mesmos [Carroll95].

Historicamente a técnica de cenários foi introduzida pela disciplina de planejamento militar e, subsequentemente adotada em várias outras áreas, tais como economia, gerência e planejamento [Becker83]. Nas últimas décadas sua utilização tem se alastrado por várias outras áreas, notadamente a de Interação Homem Máquina, onde cenários tem sido úteis na especificação de interfaces, Planejamento Estratégico, onde cenários são utilizados na exploração de alternativas futuras e na Engenharia de Sistemas onde são utilizados tanto na descrição de problemas como em suas soluções computacionais [Jarke98, Carroll95, Ringland98].

Nos últimos anos é crescente o interesse em cenários por parte da comunidade de Engenharia de Requisitos, em grande parte em resposta ao impacto causado pela introdução do enfoque de casos de uso, proposto por Jacobson no contexto do desenvolvimento de software orientado a objeto [Jacobson92]. Nas conferências importantes da área temos notado um aumento do número de publicações relativas ao emprego de cenários, como ilustrado pela Tabela 2.1. Foi realizado um workshop em torno do assunto em Daghstuhl, Alemanha coordenado pelo professor John M. Carroll, em 1999.

Conferências	
RE95 – 2nd IEEEInternational Symposium on Requirements Engineering	1 artigo
ICRE96 – 2nd International Conference on Requirements Engineering (IEEE e ACM)	1 artigo
RE97 – 3rd IEEEInternational Symposium on Requirements Engineering	3 artigos; 1 workshop
WER98 – Primeiro Workshop em Engenharia de Requisitos	3 artigos
ICRE98 – 3rd International Conference on Requirements Engineering (IEEE e ACM)	5 artigos
CEIRE98 – Conference on Industrial Requirements Engineering (BCS)	2 workshops
WER99 – Second International Workshop on Requirements Engineering	5 artigos
RE99 – 4th IEEEInternational Symposium on Requirements Engineering	7 artigos
ICRE00 – 4th International Conference on Requirements Engineering (IEEE e ACM)	4 artigos

Tabela 2.1 - Aumento do número de publicações no tema cenários nas conferêcias importantes da área de engenharia de requisitos

Edições especiais de periódicos também foram dedicados ao assunto. O Requirements Engineering Journal, editado pela Springer Verlag, dedicou as edições correspondentes ao Vol. 3 No.1 e No.3,4 (1998) ao assunto e também a Transactions on Software Engineering Vol. 24 No.12 (1998). Finalmente, o projeto europeu ESPRIT 21903 – CREWS foi dedicado a pesquisa e criação de ferramental de apoio ao desenvolvimento de software baseado em cenários.

Dedicaremos este capítulo ao estudo de cenários de modo geral, porém dando ênfase a sua aplicação na Engenharia de Requisitos. Na próxima seção descreveremos as várias áreas de aplicação onde podemos empregar a técnica de cenários. Nosso intuito é fornecer uma visão da amplitude de escopo de utilização da mesma. A seguir, mostramos algumas propostas para a classificação de cenários. Estas variam

grandemente entre si e são muitas as dimensões nas quais podemos categorizar o conteúdo e a utilização dos últimos. Finalmente, faremos uma breve revisão das propostas de utilização de cenários disponíveis na literatura.

2.1 ÁREAS DE APLICAÇÃO

Segundo Carroll, a propriedade que melhor define um cenário é o fato do mesmo projetar uma descrição concreta de uma atividade em que o cliente se engaja no momento em que está realizando uma tarefa específica. Esta descrição tem de ser suficientemente detalhada de modo que implicações sobre o desenho possam ser inferidas e discutidas [Carroll95].

Nesta perspectiva, o autor acredita que cenários podem cumprir vários papéis durante o processo de desenvolvimento de software. A seguir resumimos as áreas em que o autor acredita que a utilização de cenário pode ser benéfica:

- Elicitação de requisitos

  Captura e auxílio na compreensão do problema. Facilita o entendimento das relações entre elementos do macrosistema. Cenários também podem ser utilizados na identificação dos objetos do domínio.
- Comunicação entre clientes e desenvolvedores

  Os próprios clientes podem descrever cenários que ilustrem elementos de desenho importante para eles, problemas ou novas situações que desejam que o sistema implemente.
- Captura das justificativas do desenho do sistema (*design rationale*)

  Cenários podem ser utilizados como forma de registrar o processo de tomada de decisão. Neste enfoque não somente a solução para um problema é registrada, mas também outras propostas que foram rejeitadas e as razões que levaram os desenvolvedores a descartá-las.
- Projeções futuras

  Cenários podem ser utilizados como meio de prototipar o funcionamento do futuro sistema, especialmente do ponto de vista da interação com clientes e usuários.

- Desenho do Software

  Cenários podem ser usados na avaliação de alternativas de desenho.
- Implementação

  Cenários são úteis para ilustrar a interação entre os diversos subsistemas.
- Documentação e Treinamento

  Cenários podem ser utilizados para preencher o vazio que existe entre o artefato entregue, sistema, e as tarefas que usuários do último desejam cumprir através de sua utilização.
- Avaliação do sistema

  Através da utilização de cenários externos é possível verificar se os requisitos dos clientes foram atendidos.

Como podemos notar, Carroll acredita que cenários podem ser utilizados de modo a cumprir diversos papéis durante o desenvolvimento de software. Este ponto de vista é compartilhado por Filippidou que aponta a possibilidade de utilização de cenários nas seguintes áreas, segundo uma revisão da bibliográfica conduzida pela própria [Filippidou98]:

- elicitação de requisitos
- projeção de requisitos
- análise dos requisitos
- avaliação dos requisitos
- captura de *rationale*
- geração de interfaces

Como era de se esperar, de modo a abranger tantas atividades diferentes, ainda não existe um consenso em relação a que tipo de informação deve estar contida em um cenário [Weidenhaupt98]. Esta situação é exacerbada ao investigarmos as possíveis categorizações propostas para a classificação de cenários disponíveis na literatura. Na próxima seção apresentamos algumas destas propostas. Como veremos, os critérios utilizados por cada autor são muito diferentes, o que evidencia uma grande variedade de formatos entre possíveis representações e, a falta de um consenso em relação a uma definição universal para o termo cenário.

2.2 CLASSIFICAÇÃO DE CENÁRIOS

Nesta seção apresentaremos brevemente algumas das propostas de classificação de cenários existentes na literatura. Em primeiro lugar devemos tornar mais claro o espectro de possibilidades que estão disponíveis. Cenários podem ser desde simples descrições em linguagem natural até modelos mais complexos, contendo informação comportamental (ações, eventos e atividades) e até objetos (entidades, dados e atributos) [Rolland98].

Outros aspectos, tais como os objetivos de utilização e o período em que serão empregados, também devem ser levados em conta. Na realidade, existem várias pontos de vista pelos quais podemos analisar e, subseqüentemente categorizar a utilização destes artefatos durante o processo de desenvolvimento de software. Os esquemas para classificação de cenários que apresentaremos a seguir são multifacetados, evidenciando que não estamos tratando de um problema trivial . O ponto desta seção é expor a grande variedade semântica existente entre as diversas propostas para a representação de cenários e a dificuldade de se realizar um estudo comparativo entre as mesmas. Seguimos por apresentar a proposta de Antón e Potts para a classificação de cenários [Antón98].

2.2.1 FRAMEWORK PARA CLASSIFICAÇÃO DE CENÁRIOS PROPOSTO POR ANTÓN E POTTS

Segundo Antón e Potts, as proposta de cenários encontradas na literatura podem ser diferenciadas segundo apenas dois critérios. São eles: o que são e para que estão sendo utilizados [Antón98]. De modo a definir o que são os cenários, i.e., seu espaço representacional, os autores os classificam segundo seis critérios distintos. São eles: ênfase ontológica, estrutura de superfície, escopo, nível de detalhe, referência e modo. A Figura 2.1 ilustra o *Framework* proposto pelos autores.

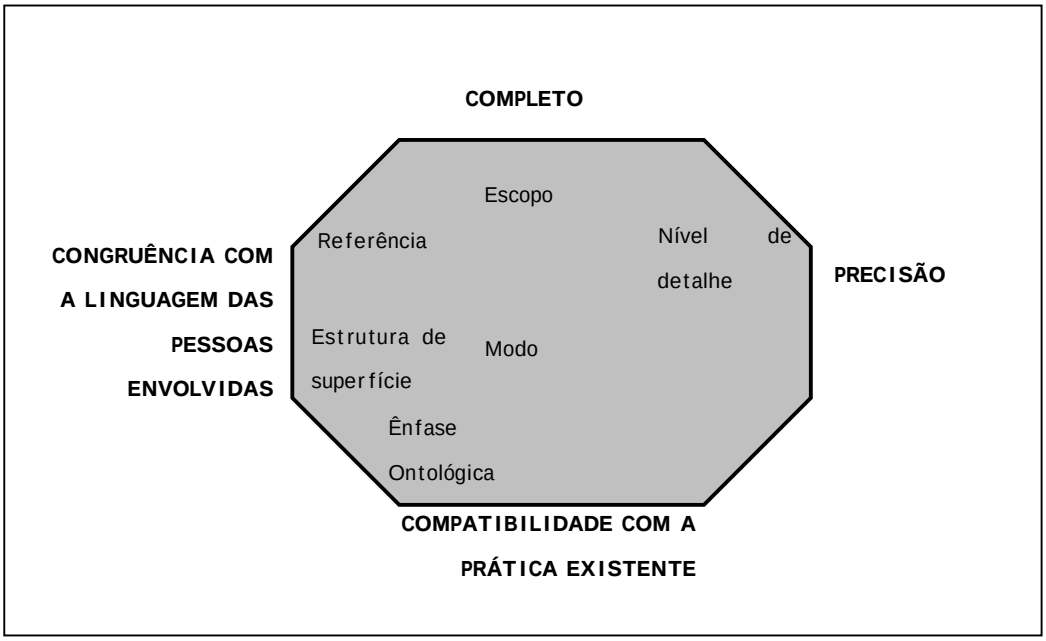


Figura 2.1Framework para a classificação de cenários proposto por Antón e Potts [Antón98]

A seguir detalhamos cada um dos critérios propostos. O primeiro, ênfase ontológica, se refere a aspectos como a temporalidade dos eventos tratados pelos cenários e o modo em que são caracterizados os agentes participantes. Segundo os autores, podemos distinguir entre agentes do mundo real, cujo comportamento é não determinístico, e aqueles que são projetados pelos designers (objetos que fazem parte do software, por exemplo) cujo comportamento podemos prever. O segundo critério, estrutura de superfície diz respeito ao modo em que o cenário em si se apresenta para os clientes, i.e., se é uma descrição em linguagem natural, utiliza gráficos, Tabelas, *storyboards* ou outras representações.

O critério de escopo define a duração dos comportamentos descritos no cenário. Na realidade, funciona como uma medida de autocontenção onde o cenário pode descrever uma transação completa ou apenas parte de um comportamento. O critério de nível de detalhamento se refere aos graus de complexidade em que são expressos os comportamentos descritos pelo cenário e refinamento dos agentes presentes. O critério de referência define o ponto de vista do autor dos cenários, e.g., os cenários são descritos a partir de uma perspectiva externa ou de uma das entidades do próprio sistema. Finalmente, o critério de modo define o tipo de comportamento que está sendo descrito. Eles podem ser do tipo indicativo, ou seja descrevem um fato do

mundo real, ou do tipo optativo, descrevem comportamento que é desejado ou requerido do sistema.

Além dos critérios descritos acima, Antón e Potts ainda incluem outros que avaliam a eficácia das representações de cenários. Em primeiro lugar, os cenários devem ser congruentes com a linguagem utilizada pelos clientes. Especificações de modo geral são de difícil compreensão para leigos. Cenários oferecem uma alternativa para a validação das mesmas porém somente serão eficazes se utilizarem uma linguagem que os clientes possam compreender. O segundo critério é o fato do cenário em questão ser completo. Segundo os autores, cenários são inerentemente incompletos pois apenas exemplificam comportamentos. Uma questão fundamental é se um conjunto de cenários é representativo da funcionalidade que se deseja implementar. O terceiro critério é precisão e significa a ausência de ambiguidade na semântica do cenário. Finalmente, devemos avaliar a utilização de cenários em relação a outras práticas concomitantes. Aplicação de cenários é apenas uma das técnicas que podem ser utilizadas durante o desenvolvimento de um sistema. Devemos verificar se as práticas utilizadas são realmente compatíveis, i.e., se os cenário estão sendo realmente eficazes ao suportar outras atividades do processo e vice versa. Estes critérios também estão ilustrados na Figura 2.1.

2.2.2 FRAMEWORK PARA CLASSIFICAÇÃO DE CENÁRIOS PROPOSTO POR ROLLAND ET AL

Uma classificação mais ampla foi proposta como um dos resultados do projeto europeu CREWS [Rolland98]. Neste *Framework* cenários são classificados de acordo com quatro visões ou aspectos: objetivo, ciclo de vida, conteúdo e formato. O *Framework* está ilustrado na Figura 2.2 a seguir.

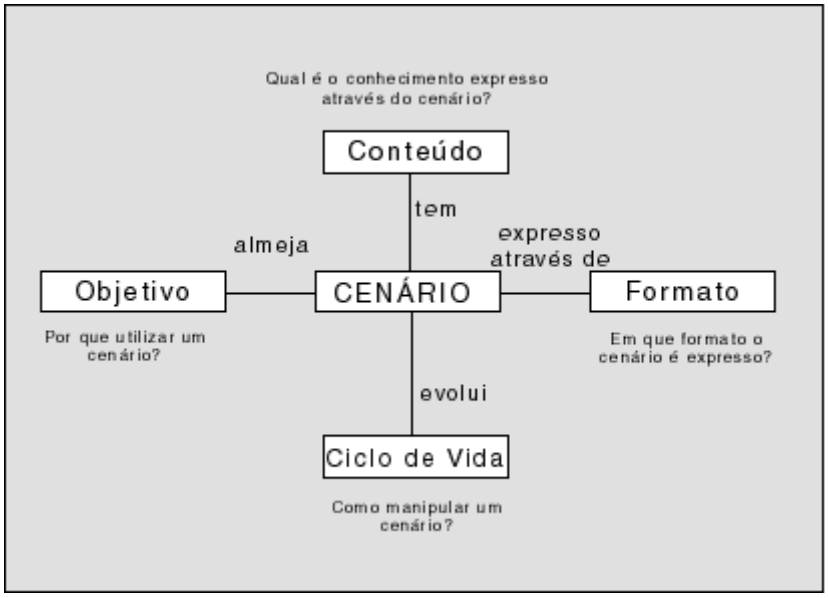


Figura 2.2 – Framework para classificação de cenário [Rolland98]

As visões de ciclo de vida e objetivo se referem a utilização de cenários durante o processo de Engenharia de Requisitos e são independentes de aspectos representacionais.

A visão de conteúdo possui as seguintes facetas: abstração, contexto, argumentação e cobertura. Por abstração os autores se referem a distinção entre tipos e instância. Um cenário que apresenta uma faceta de contexto estreito é aquele que possui apenas um agente que representa o sistema a ser construído, poucos ou nenhum agente externo e nenhuma interação entre os últimos, i.e., apenas representam interações com o sistema deixando de lado o macrosistema e sua dinâmica. Por contraste, um cenário com um contexto largo é aquele com um ou nenhum agente dos sistema e as interações ocorrem entre os agentes externos. A faceta de argumentação por sua vez descreve as razões que justificam o comportamento narrado pelo cenário. Finalmente a faceta de cobertura lida com aspectos de escopo, temporalidade e objetivo dos cenários.

A visão de conteúdo é divida em descrição e apresentação. Descrição corresponde ao tipo de informação contida no cenário, i.e., descrições em linguagem natural, estruturadas, utilização de gráficos, entre outras. A visão de apresentação, por sua vez,

corresponde ao ponto de vista do cenário, se toma partido de um dos agentes do macrosistema ou se é independente.

O *Framework* apresentado acima tenciona classificar modelos de cenários. A visão pregada pelos autores é orientada a produto, pois leva essencialmente em conta os aspectos relativos a objetivo, formato, conteúdo e ciclo de vida. Os autores justificam deixar de lado os aspectos relativos ao processo pela escassez de referências ao tópico na literatura disponível.

2.2.3 FRAMEWORK PARA CLASSIFICAÇÃO DE CENÁRIOS PROPOSTO POR FILIPPIDOU

De modo a organizar os diferentes enfoques para cenário encontrados na literatura Filippidou propõe a utilização de um *Framework* baseado em quatro perspectivas [Filippidou98]. O *Framework* classifica os cenários apenas em relação a seu conteúdo, diferentemente dos outros dois *Framework*s já apresentados, onde o conteúdo é apenas um de seus componentes.

A seguir apresentamos cada uma das perspectivas para a classificação previstas por Filippidou:

- perspectiva de processo – classifica os cenários quanto a sequência de ações ou eventos descritos.
- perspectiva de situação – classifica os cenários quanto as situações concretas que representam.
- perspectiva de escolha – classifica os cenários quanto a possibilidade de ramificações, sejam comportamento excepcional ou tratamento de diferentes alternativas.
- perspectiva de utilização – classifica os cenários quanto o ponto de vista do agente que vai utilizá-lo. Do ponto de vista dos clientes os cenários descrevem como estes utilizarão o sistema enquanto que do ponto de vista dos desenvolvedores esta perspectiva é responsável por mostrar como os sub produtos estão relacionados até o momento de entrega do sistema.

Na seção 2.2 mostramos uma grande variedade de áreas onde a técnica de cenários pode ser aplicada. Deste a captura de requisitos do sistema até auxílio para a tomada de decisões, profissionais tem utilizado cenários para apoiar suas tarefas. Dado a grande variedade de possíveis empregos para esta técnica, não conseguiu-se chegar a uma única definição para o termo cenário.

Atualmente existe tamanha variedade na representação, definição e aplicação de cenários que se justifica a proposta de *Framework*s para a classificação dos mesmos. Na seção 2.3 apresentamos três exemplos deste tipo de *Framework*. Potts e Antón ao discutir a variedade de representações disponíveis e possibilidade de emprego para a técnica de cenários apontam que todas partilham um único objetivo, que pode ser resumido como se segue:

> "*sistemas de software, que são complexos e difíceis de entender, podem ser compreendidos e melhor projetados se os comportamentos e os consequências de compromissos assumidos a um nível abstrato forem clarificados através de exemplos concretos. O modo de representar os cenários se torna, desta forma, crucial para esta compreensão* " [Antón98]

Os resultados de nossa experiência particular com a utilização de cenários no desenvolvimento de software fazem com que concordemos com os autores acima. Também acreditamos que a utilização de cenários auxilia a aproximação entre desenvolvedores e clientes trazendo benefícios na comunicação entre ambas as partes. Nos últimos anos o grupo de engenharia de requisitos da PUC-Rio tem obtidos resultados em pesquisas realizadas acerca da utilização de cenários durante o processo de desenvolvimento de software [Leite97, Breitman98, Breitman98-b, Hadad99, Breitman99, Breitman00, Breitman00-b]. Utilizamos a notação proposta em [Leite97], que utiliza linguagem natural semi-estruturada e propõe que cenários sejam aplicados durante todo o processo de desenvolvimento do software.

Na próxima seção faremos uma revisão comparativa de algumas das propostas encontradas na literatura semelhantes aquela que temos utilizado, [Leite97].

2.3 REPRESENTAÇÕES PARA CENÁRIOS

Nesta seção concentraremos nossos esforços em propostas para representação de cenários que utilizam linguagem natural semi-estruturada. Limitamos nosso escopo de modo a fazer uma comparação entre a notação que temos utilizado e outras semelhantes. Revisões extensas da utilização de cenários existem e estão disponíveis em [Filippidou98, Rolland98, Weidenhaupt98].

2.3.1 MÉTODO PARA ANÁLISE DE REQUISITOS BASEADO EM CENÁRIOS (SCRAM)

Neste enfoque os autores utilizam uma combinação das técnicas de prototipagem, entrevistas, cenários e *rationale* [Sutcliffe97, Sutcliffe98]. Embasados na hipótese de que a integração de técnicas fornece o melhor caminho para a engenharia de requisitos, propõem um método de integração em quatro estágios:

- entrevistas e técnicas para descobrimento de fatos – estas técnicas são utilizadas de modo a elicitar dados suficientes que permitam a construção de um protótipo, chamados demonstradores de conceito.
- construção de protótipos – neste estágios os autores pregam a utilização de ferramentas comerciais tais como o Macromedia Director® e MS Visual Basic®
- validação com clientes – neste estágio os protótipos ou demonstradores de conceitos são utilizados para validar os requisitos junto aos clientes. Este estágio é gravado através de vídeo ou áudio e é utilizado para análise futura.
- análise - neste estágio os autores propõe um *layout* específico para a análise dos requisitos. Nesta fase é conduzida uma reunião do tipo JAD (joint application design) onde existe um cronograma previamente definido e um mediador externo para conduzir a sessão.

Figura 2.3 Modelo ER do enfoque para representação de cenários proposto por [Sutcliffe98]

As sessões de análise utilizam cenários como meio de validação de alternativas de desenho. Os cenários descrevem como os clientes conduzem suas tarefas. Na Figura 2.3 mostramos o esquema da notação utilizada neste enfoque. Cada sessão se inicia como uma descrição verbal da situação que será demostrada pelo protótipo. Um desenvolvedor opera o protótipo enquanto que o segundo faz perguntas e anota as respostas dos clientes. Para cada episódio de um cenário várias opções de desenho são apresentadas.

Esta método foi utilizado em vários estudos de caso. Segundo os autores, a combinação de técnicas se provou útil na captura dos requisitos. O fato de utilizarem cenários na descrição de situações auxiliou em manter a atenção dos clientes. Por outro lado, o processo de desenvolvimento baseado em cenários se provou fraco na captura de requisitos não-funcionais.

2.3.2 CICLOS DE QUESTIONAMENTO (INQUIRY CYCLE)

Neste enfoque os autores propõem um modelo para a descrição e suporte de discussões sobre os requisitos do sistema que utiliza uma estratégia baseada em ciclos consecutivos de questionamento e refinamento destes requisitos [Potts94]. O método proposto é fortemente baseado em objetivos e os cenários são derivados a partir da identificação e decomposição destes. Uma vez identificados, os cenários farão parte de uma estratégia dinâmica que, segundo os autores permite o questionamento dos requisitos do sistema.

O ciclo se dá da seguinte forma, a documentação relativa aos requisitos que consiste dos cenários identificados e de uma lista de requisitos é discutida através de um processo de questionamento onde respostas e justificativas (*rationale*) para as questões colocadas são registradas. O processo de questionamento só termina quando uma decisão é tomada. As decisões resultantes deste processo podem implicar na mudança dos requisitos, o que por sua vez resulta na modificação da documentação e justifica um novo ciclo.

Cenários são utilizados na validação e esclarecimento dos requisitos. A comparação, refinamento e avaliação de cenários é realizada ao nível dos objetivos apenas. Os cenários podem ser documentados de modos diferentes, dependendo do nível de detalhe necessário. A forma mais simples é um caso de uso (use case), que consiste em uma breve descrição acrescida de um número identificador. Formas mais detalhadas são chamadas de *scripts* e são representadas através de Tabelas ou diagramas. Estas formas envolvem a identificação de agentes e de ações. A Figura 2.4 mostra um exemplo de cenário segundo este enfoque. Note que é composto apenas pelos componentes de agente e ação. Segundo Potts a narrativa, sequência de episódios, pode ser subdividida em 1 ou mais ações. Cada episódio possui um objetivo inicial e um resultado. Nenhuma descrição do macrosistema que contextualize os cenários é fornecida, porém.

Agente	*Ação*
Esther	Criar nova reunião
Esther	Determinar que Kenji é participante importante
Esther	Determinar que Annie estará apresentando
Esther	Determinar que Colin é participante comum
Esther	Digitar descrição da reunião
Esther	Determinar datas possíveis entre segunda e sexta da próxima semana
Esther	Determinar data limite como sexta feira da próxima semana
Programa	Determinar tempo de resposta até sexta-feira desta semana
Programa	Mandar mensagem para Colin pedindo para que liste suas restrições
Programa	Mandar mensagem para Colin pedindo para que liste suas restrições e requisitando local de sua preferência
Programa	Mandar mensagem para Colin pedindo para que liste suas restrições e equipamento necessário

Figura 2.4 – Exemplo de cenário segundo o enfoque proposto em [Potts94]

Segundo Potts os cenários, independente do seu nível de detalhe, são muito úteis na validação dos requisitos. Desta forma é possível identificar deficiências nas especificações e fazer as mudanças necessárias. Em seu papel de clarificar os requisitos, cenários auxiliam na compreensão geral do sistema, facilitando o processo de especificação. Este modelo, apesar de pregar a utilização de cenários, não aponta como e onde os mesmos devem ser elicitados.

2.3.3 CENÁRIOS COMO APOIO A VISUALIZAÇÃO DE REQUISITOS

A tese apresentada por Zorman discute a dificuldade na comunicação entre clientes e desenvolvedores, apontando ocorrências de mistura de terminologia e erros de interpretação [Zorman95]. Segundo a própria, especialistas de domínio e software necessitam colaborar com uma representação e vocabulários comuns para a elaboração de cenários. Nesta luz, cenários podem vir a ser utilizados eficientemente na comunicação entre pessoas com *backgrounds* diferentes. Linguagens mais formais devem ser utilizadas na comunicação com pessoas de uma mesma área. A Figura abaixo ilustra o fato de pessoas de uma mesma área tendem a se comunicar de modo

mais formal pois compartilham vocabulário (terminologia, jargão), representações e abstrações, enquanto que pessoas com diferentes *backgrounds* se utilizam de notações mais informais.

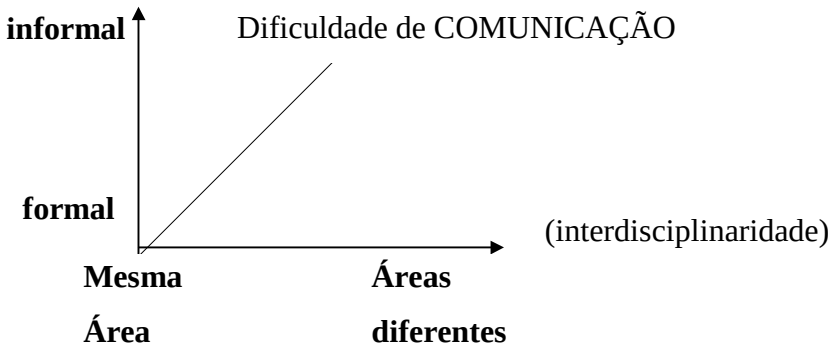


Figura 2.5 – Dificuldade de comunicação entre comunidades distintas apontada em [Zorman95]

O processo de desenvolvimento de software inerentemente envolve pessoas de várias áreas diferentes e com os mais diversificados *backgrounds.* De modo a dar conta da dificuldade de comunicação entre os indivíduos envolvidos na elaboração do software Zorman propõe a utilização de uma estratégia para a visualização (*envisaging)* dos requisitos do sistema. Este processo, que conta com o auxílio de cenários, consiste na transformação de noções informais do que se deseja de um sistema em uma descrição mais precisa do mesmo, traduzida pelo documento de especificação.

Cenários neste enfoque são definidos como descrições parciais do sistema e do comportamento do macrosistema. A autora apresenta uma ferramenta para a captura e representação de cenários, REBUS que se assemelha a representação utilizada por *storyboards*. Cada cenário é composto de um nome, categoria, descrição e uma representação gráfica. Com o auxílio da ferramenta é possível representar elementos tais como objetos e espaços.

Segundo Zorman, a maior contribuição de seu trabalho reside no aumento da qualidade da comunicação entre clientes e desenvolvedores através da utilização da técnica de cenários. A representação para cenários proposta é independente de

domínio de aplicação dos mesmos. Esta assertiva é justificada no fato de que a representação para cenários foi elaborada utilizando-se blocos de construções derivados de teorias linguísticas.

2.3.4 CASOS DE USO (USE CASES)

O conceito de casos de uso foi introduzido por Jacobson em 1994 como se segue:

> "*Casos de uso são maneiras de utilizar o sistema. O conjunto de casos de uso representam tudo que um usuário pode fazer com o sistema.*" [Jacobson94]

Os casos de uso são descritos ao nível de tipo enquanto o que os cenários são instâncias dos primeiros. A utilização deste enfoque prevê a construção de modelos de casos de uso, que reúnem agentes e classes. Os cenários somente existem quando o sistema está em operação, pois são as instâncias do modelo de casos de uso.

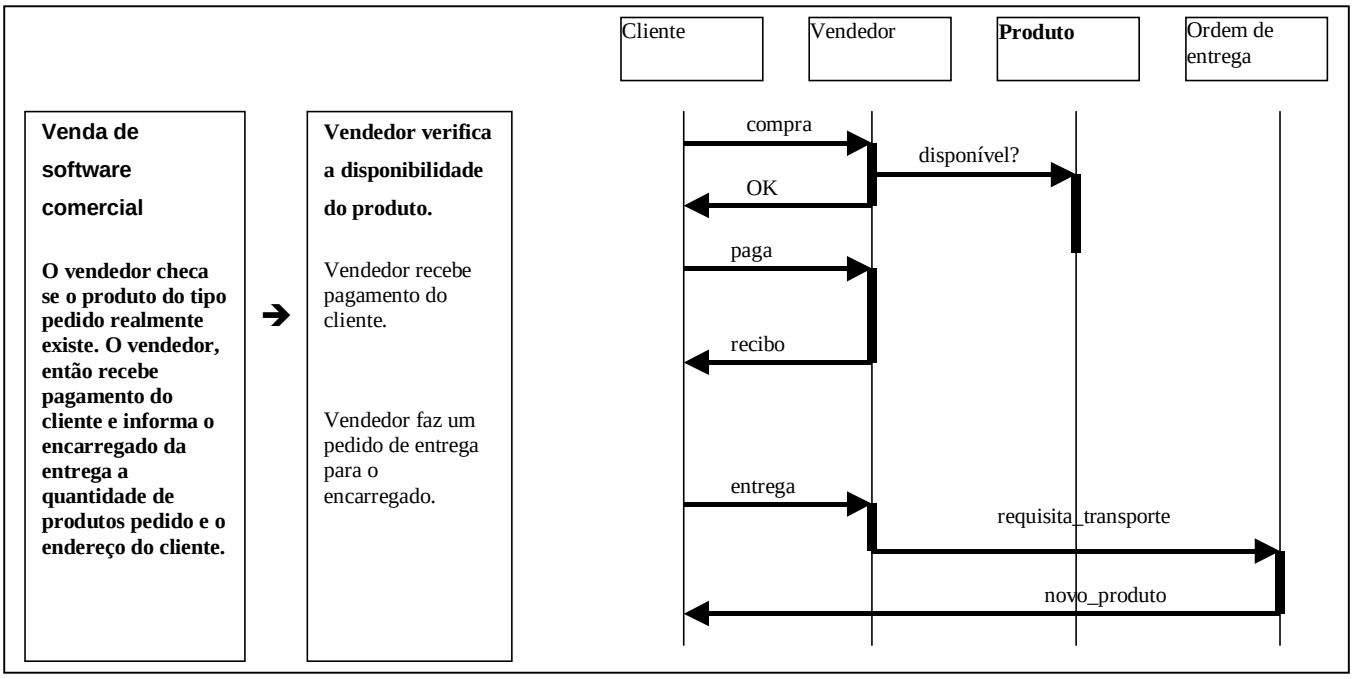


Figura 2.6 – Exemplo de cenários (esquerda) e casos de uso (direita)

Neste enfoque somente é representado a interação entre os clientes e o sistema, que é visto sob como uma caixa preta. Desta forma, aspectos internos do sistema e informações sobre o macrosistema não são capturadas. O modelo também não expressa diferentes pontos de vista nem argumentações sobre as decisões de desenho [Rolland98]. Os casos de uso são utilizados apenas durante a fase de elicitação de

requisitos e subsequentemente incorporados na documentação [Robertson99]. A Figura 2.6 mostra um exemplo de caso de uso e de possíveis cenários.

Note que a Figura 2.6 utiliza uma notação específica, proposta inicialmente por Jacobson e agora incorporada a linguagem UML (Unified Modelling Language) para descrição de sistemas orientados a objeto [Booch99, Texel97, Scheneider98]. Note também que o modelo é composto de representações para atores, classes e a comunicação entre os dois (arcos). Segundo Booch, existe um abismo entre os casos de uso e o desenho e implementação do modelo de objetos. Durante este abismo é que a identificação dos objetos e a solução de problemas como concorrência e consistência entre casos de uso serão levados em conta.

Na realidade, a representação para casos de uso é mais próxima da notação para cenários que temos utilizado, proposta em [Leite97], do que aquela que Jacobson e Booch se referem como cenários propriamente ditos. Observe na Figura 2.5 que os cenários são representados por descrições em linguagem natural enquanto que os casos de uso possuem uma estrutura. Neste trabalho quando fizermos uma comparação entre a notação de cenários utilizada por nós e aquelas proposta por Jacobson e Booch, estaremos na realidade nos referindo aos casos de usos.

2.3.5 UTILIZAÇÃO DE CENÁRIOS PARA ELICITAR OBJETIVOS

Rolland, Souveyet e Bem Achour pregam a utilização de um enfoque baseado em objetivos para a identificação dos requisitos do sistema [Rolland98]. Dentro deste contexto, propõem a utilização da técnica de cenários para auxiliar a elicitação destes objetivos. Segundo os autores um cenário pode ser definido como "*um comportamento possível limitado a um conjunto de interações propositais que são levadas a cabo por uma série de agentes*".

A notação para cenários proposta é composta de uma ou mais ações, onde a combinação destas descreve um caminho único que leva os agentes de um estado inicial a um estado final. Desta forma, a combinação de vários cenários é capaz de

descrever as interações entre um sistema complexo de agentes. A Figura 2. 4 mostra a estrutura proposta para a notação de cenários.

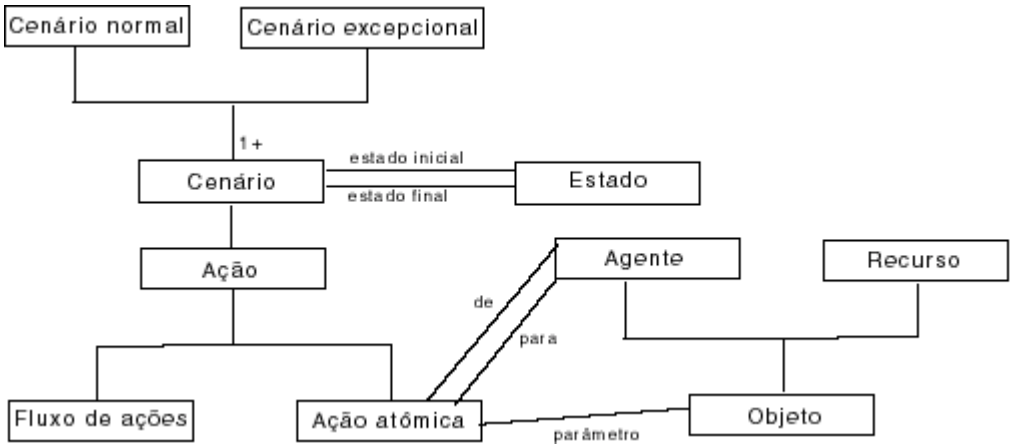


Figura 2.7 – Estrutura para notação de cenários proposta em [Rolland98]

Um cenário é caracterizado por seus estados inicial e final, onde o estado inicial representa as pré condições para a realização deste cenário. Os autores fazem uma distinção entre cenários ditos normais e excepcionais, onde a distinção reside no cumprimento de um objetivo associado (mas que não faz parte da estrutura) deste cenário. As ações podem ser de dois tipos, atômicas ou compostas de um fluxo. A última é a composição de uma série de ações atômicas.

Esta notação para cenários utiliza um formato textual semi-estruturado e linguagem natural.

2.3.6 CENÁRIOS DENTRO DO CONTEXTO DA BASELINE DE REQUISITOS

Leite define cenários como descrições evolutivas de situações próprias ao ambiente [Leite97, Leite97-b]. Mais abrangente do que a definição proposta na literatura de Interação Homem Máquina, este enfoque prega que cenários devem ser utilizados durante todo o processo de desenvolvimento do software. Compreende, além da interação entre sistema e clientes, a interação entre módulos do sistema. Nesta luz temos desde cenários iniciais que modelam o macrosistema onde o sistema será desenvolvido e instalado até cenários externos, i.e., que representam as interações dos

usuários com o sistema quando este estiver em funcionamento, passando por várias versões intermediárias ao longo do processo de desenvolvimento do software.

A notação proposta pelo autor utiliza linguagem natural semi-estruturada (ancorada no macrosistema), partindo da premissa que a utilização da linguagem da aplicação e não do software facilita o entendimento e a validação dos requisitos por parte dos clientes [Leite90].

Central a esta proposta são as seguintes premissas:

- Um cenário descreve situações do macrosistema e suas relações com o sistema a ser construído.
- Um cenário pode ser utilizado para descrever as interações entre os diversos objetos presentes no sistema.
- Um cenário evolui durante o processo de desenvolvimento do software.

O modelo de cenários faz parte de uma proposta maior, a baseline de requisitos [Leite97] que será apresentada em detalhes no capítulo 3 deste trabalho. A estrutura de um cenário está ilustrada na Figura 2.8 a seguir. Descreveremos cada um dos componentes do cenários separadamente.

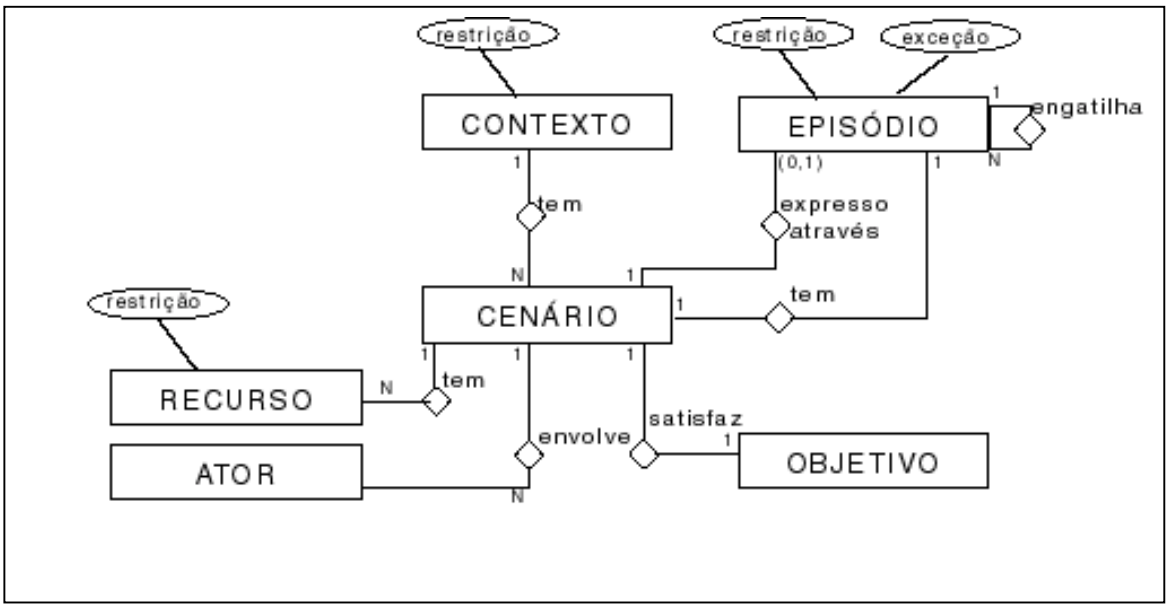


Figura 2.8 – Modelo ER da notação utilizada para representar cenários dentro do baseline de requisitos [Leite97]

- Título: identifica o cenário.
- Objetivo: estabelece a finalidade de um cenário. O cenário deve descrever o modo em que este objetivo deve ser alcançado.
- Contexto: Descreve o estado inicial de um cenário, suas pré-condições, o local (físico) e tempo.
- Recursos: identifica os objetos passivos com os quais lidam os atores.
- Atores: Pessoa ou estrutura organizacional que tem um papel no cenário.
- Episódios: Cada episódio representa uma ação realizada por um ator onde participam outros atores utilizando recursos disponíveis. Um episódio também pode se referir a outro cenário. Episódios podem conter restrições e exceções. Uma restrição é qualquer imposição que restrinja um episódio de um cenário. Uma exceção é o tratamento para uma situação excepcional ou de erro.

Na Figura 2.9 mostramos um exemplo de um cenário escrito utilizando-se a notação proposta por Leite. O cenário vem do exemplo do sistema de controle de luzes que utilizaremos ao longo deste trabalho. O exemplo completo se encontra disponível em http://stones.les.inf.puc-rio.br/Karin/exemplo/index.html

user occupies room	Goal: establish the procedure for occupied room Context: 4th floor of building 32 , motion detector in order, user entered room Resource: value T1 Default light scene for this room, Chosen light scene value Actors: user, Control system Episodes: 1. user enters room 2. user chooses light scene 3. IF room is reoccupied wihtin T1 minutes THEN activate last Chosen light scene 4.IF room is reoccupied after T1 minutes THEN activate Default light scene

Figura 2.9 – Exemplo de um cenário que utiliza a notação proposta por Leite [Leite97]

2.4 CLASSIFICAÇÃO PARA A NOTAÇÃO DE CENÁRIOS

Na seção 2.3 apresentamos o *Framework* de classificação de cenários proposto por Rolland et al [Rolland98]. Os autores classificam cenário de acordo com quatro visões distintas, conteúdo, formato, propósito e ciclo de vida. Na visão de formato existe uma distinção entre descrição e apresentação. Enquanto a primeira descreve as notações existentes a última dá conta da classificação dos modos de apresentar a informação contida nos cenários. Este trabalho chega a descrever os tipos de linguagem possíveis para descrever cenários, e.g., Tabelas, scripts, representações estruturas. As notações também são classificadas segundo seu nível de formalismo.

Em trabalhos anteriores notamos que a granularidade do conteúdo dos cenários desempenhava um papel fundamental na qualidade do processo de rastreamento da informação [Breitman98]. Distinções do tipo eventos, agentes e a separação dos objetos restantes nos permitiram conduzir experimentos de rastreamento muito mais ricos em detalhes e que, consequentemente, trouxeram resultados mais significativos. Na realidade o nível de detalhamento desejado é apenas uma especialização da classificação proposta por Rolland. Utilizando a última como ponto de partida fomos capazes de refinar a classificação da notação de cenários segundo o tipo de componente encontrado. Foram então criadas quatro novas categorias, intenção, agente, ambiente e ação. Como vimos na seção anterior, todas as notações apresentadas apresentam pelo menos um destes componentes. Na Tabela 2.2 a seguir resumimos as notações apresentadas segundo o critério proposto

	Intenção	**Agente**	**Ambiente**	**Ação**
[Jacobson94]		atores		use case
[Leite97]	objetivos	atores	recursos, contexto	Episódios, restrições, exceções

[Potts94]	objetivos tarefas	agentes		Ação
[Rolland98]		agentes	recursos objetos estados	Ação
[Sutcliffe98]	objetivos	agentes	estrutura, recursos, estados	Atividades, eventos
[Zorman95]		objetos	medidas, elementos espaciais	Elementos temporais, comportamento

Tabela 2.2 - Classificação dos enfoques apresentados na seção 2.4

Desta forma podemos generalizar uma classificação para a notação de cenários baseadas nos componentes dos próprios. No começo da última seção colocamos que, dado a grande diversidade existente para notações e emprego de cenários, reduziríamos o escopo de nossa apresentação a aquelas que utilizavam linguagem natural semi estruturada. Desta maneira, deixamos de fora representações formais como as propostas por Hsia [Hsia94] e Heymans e Dubois [Heymans98] e também as totalmente informais como as propostas por Kuutti [Kuutti95] e Rosson e Carroll [Rosson95].

Através da comparação dos enfoques para notação de cenários apresentados anteriormente notamos que poderíamos detalhar ainda mais sua classificação. Para tal estendemos o *Framework* proposto por Rolland de modo a acrescentar uma classificação que levasse em conta os componentes presentes na representação proposta para cenários em cada um dos enfoques. Esta classificação está representada na Figura 2.10, a seguir.

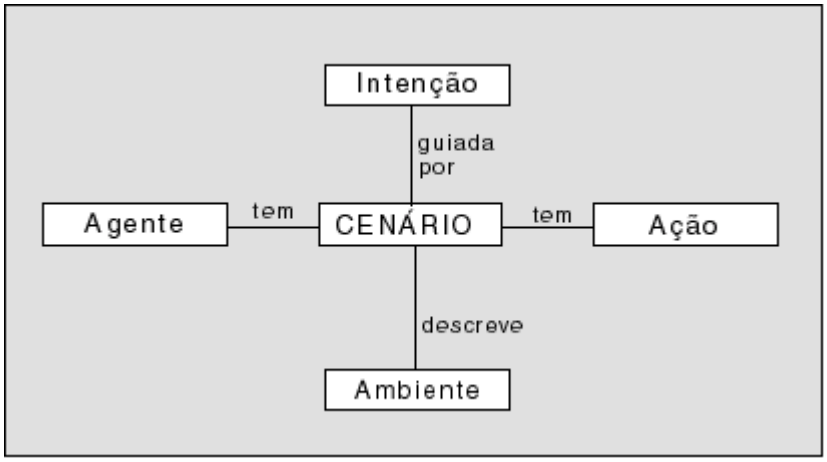


Figura 2.10 Classificação para notação de cenários segundo os componentes apresentados

2.5 RESUMO

Neste capítulo fizemos um apanhado geral das diversas áreas de aplicação e possíveis empregos para a técnica de cenários. Mostramos que não existe uma definição universal para o termo e que existem várias maneiras de classificar sua utilização. Apresentamos uma pequena revisão de notações para cenários, do tipo semi estruturada e que utilizam linguagem natural, de modo a poder fazer uma comparação com a notação proposta por Leite que utilizamos ao longo deste trabalho [Leite97].

No próximo capítulo discutiremos a utilização de cenários em um contexto mais amplo, onde levamos em conta a evolução de outros os artefatos associados ao processo de desenvolvimento de software. Neste contexto, apresentaremos a baseline de requisitos, uma estrutura paralela ao desenvolvimento, que tem como objetivo modelar e acompanhar a evolução dos requisitos externos ao software [Leite95].

Capítulo 3 - Evolução

3.1 INTRODUÇÃO

O desenvolvimento de sistemas de software é uma atividade complexa que envolve um grande número de recursos, coordenados de modo a atingir um mesmo objetivo. Este processo pode ser organizado de várias maneiras, a mais frequente é através da elaboração de vários modelos do sistema que serão sucessivamente refinados ao longo do desenvolvimento [Ghezzi91].

Através da introdução gradativa da complexidade, podemos gerenciar de forma mais eficaz o processo de produção de software [Jacobson94]. Nesta luz, os relacionamentos entre os diversos modelos, também chamados de artefatos, tem um papel muito importante, pois se torna fundamental poder mapear objetos entre as diversas representações. Esta preocupação é geral e está refletida tanto nos padrões para o desenvolvimento de software da IEEE [IEEE-Std830-1984] quanto do SEI (Software Engineering Institute) [SEI-CM1990].

É fato que os requisitos do sistema mudam ao longo do processo de desenvolvimento [Lehman80, Yeh84, Jackson95]. O primeiro impõe várias demandas sobre os desenvolvedores, uma das quais é que os requisitos do software tem de ser rastreados face a evolução do sistema [Ramill99]. A preocupação com aspectos relativos a rastreabilidade dos sistemas tem estado presente na literatura há bastante tempo e podem ser identificadas desde o Método para Engenharia de Requisitos (SREM) proposto por Alford [Alford77]. Muitas ferramentas comerciais e de pesquisa disponíveis atualmente fornecem apoio a rastreabilidade. Estas implementam um grande espectro de técnicas, que variam desde o uso de *templates* até a utilização de matrizes de rastreabilidade.

Gotel e Finkelstein apresentam um vasto estudo comparativo sobre o grau de implementação de mecanismos de rastreabilidades no ferramental disponível [Gotel93]. Este estudo, que envolveu mais de 100 diferentes produtos na época em que foi realizado, teve grande impacto na comunidade e trouxe a tona a importância

da questão de rastreabilidade do ponto de vista da Engenharia de Requisitos. Entre os resultados os autores avaliaram que nenhuma das ferramentas comerciais oferecia suporte adequado a rastreabilidade de requisitos e que a qualidade da última estava intimamente ligada a aderência da ferramenta aos métodos e técnicas que implementavam. O resultado mais significativo, porém, é de que os problemas relativos a rastreabilidade de sistemas podem ser divididos em duas grandes categorias. A primeira dá conta dos problemas de rastreabilidade de requisitos antes destes serem incluídos no documento de Especificação de Requisitos (ER), enquanto que a segunda categoria trata da rastreabilidade de requisitos que estão capturados por este mesmo documento. A primeira categoria recebeu a alcunha de pré-ER enquanto que a última de pós-ER. A Figura 3.1 ilustra a separação.

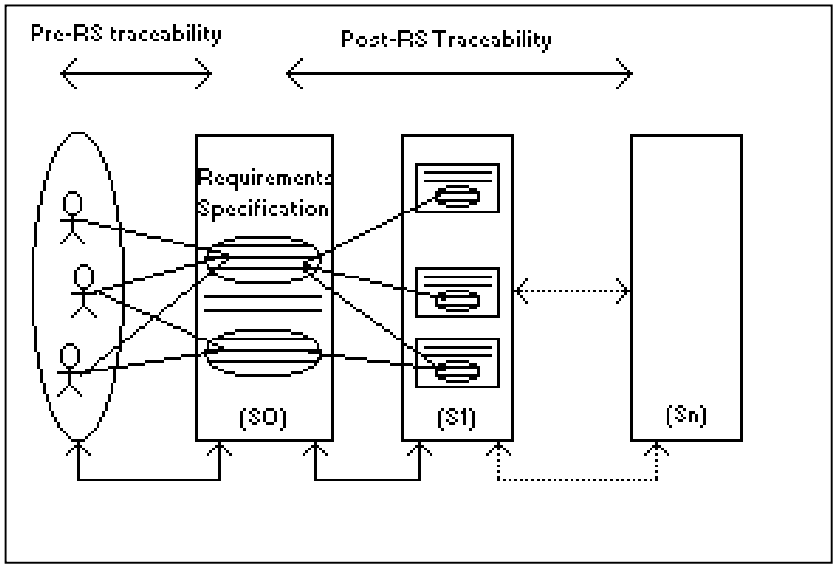


Figura 3.1 – Categorias para o rastreamento de requisitos [Gotel93]

Neste capítulo trataremos da evolução de sistemas dando ênfase a rastreabilidade da informação. Apesar de darmos um tratamento geral a questão, nos concentraremos em mostrar aspectos mais relevantes para a evolução de cenários. Na próxima seção fazemos uma breve exposição da rastreabilidade de sistemas, apresentando algumas das propostas mais novas que surgiram de modo a tratar o problema. A seguir, entramos em maior detalhe na baseline de requisitos proposta por Leite [Leite95], que será a base onde iremos calcar a proposta apresentada neste trabalho. A baseline de requisitos é definida como uma estrutura perene que incorpora artefatos de software.

Ela é desenvolvida durante a fase de requisitos e evolui ao longo da vida útil do software. Dentro do âmbito da baseline, a evolução de cenários é apenas um aspecto. A baseline é paralela ao eixo de desenvolvimento de software e trata da evolução de todos os artefatos produzidos durante este processo.

3.2 RASTREABILIDADE DE SISTEMAS

Rastreabilidade é definida como a habilidade de acompanhar a vida de um requisito tanto antes quanto depois de sua inclusão no documento de Especificação de Requisitos (ER) [Gotel93]. O momento antes da inserção do requisito na ER corresponde ao rastreamento do tipo pré-ER, como definido anteriormente. A possibilidade de realizá-lo depende da habilidade dos desenvolvedores de conectar o requisito as declarações que o originaram, através do processo de produção e refinamento dos requisitos, através do qual várias declarações (as vezes oriundas de mais de uma fonte) são eventualmente integradas para formar um único requisito[Ramesh98] Se torna óbvio que, durante este processo, a natureza das contribuições dos participantes tem um papel fundamental no apoio a rastreabilidade [Gotel95].

Gotel aponta que falhas na captura e registro da informação pré-ER são responsáveis pela queda na qualidade do rastreamento dos requisitos. De modo a minimizar alguns destes problemas é sugerido que alguns aspectos sejam levados em consideração, entre eles, aumentar a conscientização da equipe da importância da informação pré-ER, da obtenção e registro da mesma e disseminação de técnicas para a organização e manutenção da informação.

Em relação ao aspecto de obtenção e registro da informação do tipo pré-ER apontado por Gotel, Wood oferece uma alternativa baseada na utilização de técnicas hipermídia para a captura e registro de requisitos de sistema [Wood94]. O argumento que suporta este enfoque é que através da utilização de vídeo e áudio é possível armazenar requisitos do sistema em um formato muito próximo ao original. De modo geral, dados brutos passam por um processo de transformação que os tonam mais formais e permitem sua inclusão no documento de ER. Neste processo informações relevantes

podem ser perdidas e o rastreamento até a fonte que deu origem ao requisito é dificultado. Nesta luz, o autor propõe que a utilização de técnicas multimídia pode vir a reduzir este problema e ajudar a minimizar a volatilidade dos requisitos. Este ponto de vista é apoiado por Jirotka, que utilizou vídeo no auxílio a captura de requisitos para a elaboração de um sistema de controle financeiro [Jirotka95].

Kaidl também apoia a idéia da utilização de hipertexto como uma ponte entre as idéias informais dos clientes e as representação mais formais necessárias aos desenvolvedores. Este enfoque é suportado pela ferramenta RETH e oferece rastreamento através dos links do hipertexto [Kaidl93].

Na realidade estamos tratando de um problema muito complexo. Compreender e formalizar práticas humanas é uma tarefa muito difícil, pois envolve um grande volume de conhecimento tácito, i.e., implícito [Goguen94, Suchman87, Sommerville93]. Não há dúvida que o auxílio de novas técnicas, tais como utilização de áudio e vídeo podem vir a trazer contribuições de modo a minimizar o problema porém, um aspecto a ser levado em conta é o aumento dos custos e grande volume de espaço necessário a armazenar estes dados. Outro problema é que informações capturadas utilizando-se técnicas multimídia não estão diretamente disponíveis. Transcritos das sessões tem de ser realizados de modo a permitir acesso direto a informação, aumentando ainda mais o custo e o tempo gasto com a captura de informação neste formato.

Um outro enfoque para auxiliar no rastreamento das origens dos requisitos são as técnicas para captura de *rationale*. Conklin propôs um método para a captura de *rationale* baseado na técnica IBIS, o gIBIS, que funciona através de ciclos sucessivos de apresentação de assuntos, posicionamentos e argumentos até que se chegue a solução desejada. Todo o processo é registrado de modo que futuros leitores tenham acesso a discussão. [Conklin87]. Potts e Brun propõem um método para a captura e registro de deliberações de projeto através de uma representação esquemática em forma de rede. Estas redes contém nós de artefatos e deliberações que podem ser do tipo justificativas, assuntos ou alternativas. A vantagem desta representação é a separação entre artefatos e *rationale* que aumenta a compreensão de decisões tomadas durante o projeto [Potts88].

Carroll oferece apoio multimídia para a captura do histórico e *rationale* através da ferramenta Raison d'Etre [Carroll94]. O objetivo da última é fornecer um *Framework* para a captura e organização da história informal e do *rationale* utilizado pelos desenvolvedores durante sua colaboração. A ferramenta captura discussões utilizando vídeo e áudio mas mantém transcritos de todas seções de modo a permitir buscas por palavras chave. Potts, Takahashi e Antón propõem uma estrutura para descrever e oferecer suporte para discussões sobre os requisitos do sistema, o Inquiry Cycle. Esta estrutura discute a informação relativa aos requisitos, que pode ser capturada através de cenários informais, utilizando um ciclo de perguntas, respostas e justificativas. Ao final de cada ciclo mudanças podem ser incorporadas a documentação. Os autores desenvolveram um protótipo baseado em tecnologia de hipertexto para dar suporte ao Inquiry Cycle [Potts94].

Finalmente Gotel propôs a utilização de um mecanismo de estruturas de contribuição a serem utilizadas de modo a melhorar a comunicação e cooperação entre grupos. Estas estruturas contém informações sobre as soluções propostas, razões para aprovação ou rejeição das mesmas e pessoa (grupo) responsável pela contribuição. Desta forma acredita-se minimizar o esforço de mudança nos requisitos, pois são reduzidas as chances de se negligenciar considerações importantes, já que estas foram previamente registradas. Além disto, os autores pregam que o sistema também será melhor aceito pelos clientes, pois seu comportamento pode ser justificado através do *rationale* e de hipóteses de funcionamento registradas e mutuamente acordadas (clientes e desenvolvedores) [Gotel97].

Outro aspecto que deve ser levado em conta é o volume de informação envolvida no processo. O rastreamento de requisitos envolve uma grande quantidade de dados, tornando o apoio automatizado fundamental. Segundo Leite "sem o auxílio de automatização o rastreamento da informação é um mito" [Leite95].

Desta forma um aspecto crucial para a realização do rastreamento de requisitos é selecionar informação relevante de modo a armazenar somente dados significativos. O objetivo deste processo é manter o volume de dados em níveis tratáveis. Pinheiro propõe uma ferramenta que realiza o rastreamento da informação de modo

contextualizado [Pinheiro96]. O autor leva em conta que os dados que lidamos no processo de rastreamento são situados, i.e., são dependentes de detalhes de uma situação particular ou do contexto de onde surgiram. Desta forma o autor foi capaz de construir uma ferramenta que realiza rastreamento de modo seletivo.

Este aspecto também é tratado pelo ambiente PRO- ART (Process and Repository based Approach for Requirements Traceability) [Pohl96, Dömges98]. Neste enfoque os autores propõem um *Framework* tridimensional para a Engenharia de Requisitos que categoriza a informação segundo tipos básicos definidos pelos próprios. São eles medidas que vão de opaco a completo para medir a compreensão das especificações, medidas de formalidade para as representações utilizadas e medidas de concordância entre colaboradores de modo a verificar o grau de integração entre vários pontos de vista. O ambiente também prevê a existência de um repositório para o armazenamento da informação e uma ferramenta de apoio que permite a captura semi-automática de informações relativas ao rastreamento.

Utilizando um conceito de gerência de configuração, a baseline de requisitos proposta por Leite oferece suporte a rastreabilidade da informação paralelamente ao desenvolvimento do sistema [Leite95]. A baseline é perene no sentido em que é persistente durante todo o tempo de desenvolvimento do sistema, levando em conta a fase de manutenção. Desta forma ela dá conta da rastreabilidade tanto pré-ER quanto pós-ER dos requisitos.

O modelo de evolução de cenários que apresentaremos no capítulo seguinte repousa sobre esta baseline, portanto dedicaremos a próxima seção ao detalhamento da mesma.

3.3 BASELINE DE REQUISITOS

De modo a tratar a questão do rastreamento de requisitos Leite e Oliveira propõem uma baseline centrada no próprio cliente [Leite95]. Esta estrutura incorpora sentenças em linguagem natural sobre o sistema desejado. As sentenças são escritas segundo um padrão pré definido pelos autores e devem ser redigidas por engenheiros de software.

A idéia básica suportando a baseline é que a mesma é perene, i.e., é criada durante o estágio de requisitos e evolui conjuntamente com o sistema. É importante notar, porém, que a baseline é ortogonal ao processo de desenvolvimento, desta forma complemetamente independente do modelo escolhido para orientar o desenvolvimento em si. Ilustramos a baseline de requisitos na Figura 3.2, a seguir.

A baseline é dirigida aos requisitos externos, definidos como os requisitos impostos pelo macrosistema do qual o software será apenas uma parte. Requisitos externos podem ser comparados aos “requisitos reais” propostos por McMenamin e Palmer na Análise Essencial de Sistemas [McMenamin84]. Durante a etapa de elicitação os requisitos são capturados através da identificação de ações dos clientes. Esta estratégia é similar as ações previstas por Jackson em JSD [Jackson83].

Em sua primeira versão a baseline de requisitos previa três visões: configuração, modelo básico e hipertexto. Ela é calcada em uma versão estruturada do modelo de entidade e relacionamento e permite a representação de: clientes, ações, eventos externos, estímulos externos, saída, e estímulo interno. Também fazem parte dois atributos, restrição e diagnóstico. O atributo de restrição serve para delimitar o escopo do requisito enquanto que o de diagnóstico é uma observação que levanta algum problema em relação a uma das entidades representadas. O último é utilizado na validação com clientes.

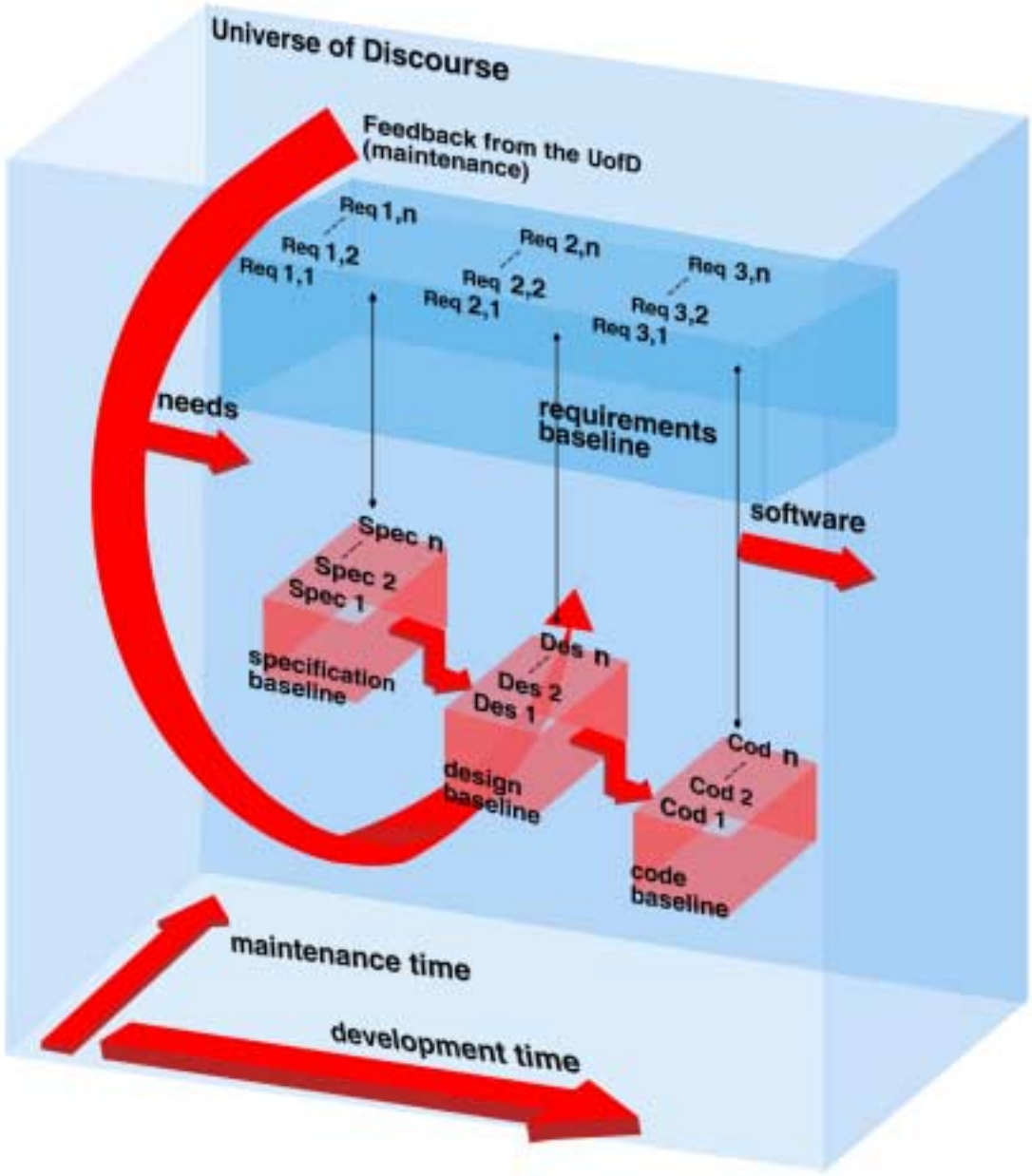


Figura 3.2 – Baseline de requisitos em sua versão atual [Leite97]

A visão de configuração é responsável por manter registro da evolução do sistema. Mantém registro do histórico de mudanças através de uma estrutura chamada etiqueta que captura as seguintes informações: pessoa responsável, razões para a mudança (inclui disparo, data do disparo e autorização) e tipo de mudança (entrada, modificação ou exclusão). Mostramos um exemplo de etiqueta utilizada pela visão de configuração na Figura 3.3. Esta visão é o ponto chave para a manutenção da rastreabilidade da informação.

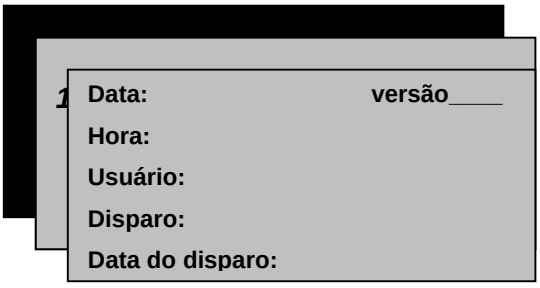


Figura 3.3. – Exemplo do mecanismo de etiqueta utilizado pela baseline de requisitos

A visão de hipertexto oferece suporte a todas as outras visões, em especial a do léxico ampliado da linguagem (LAL) pois os termos estão todos interrelacionados e, idealmente seriam implementados através de elos de hipertexto. [Leite93] O léxico é um mecanismo para a captura e registro da linguagem do problema. A idéia básica é começar a explorar os requisitos do sistema através da compreensão da linguagem da aplicação. O léxico vai além da captura do significado (denotação) dos termos, como a maioria dos glossários, pois também inclui a conotação dos mesmos. Desta forma podemos compreender o significado dos termos de modo independente e em relação a outros termos. A Figura 3.4 ilustra uma entrada do léxico do sistema para o controle de iluminação do Departamento de Informática da Universidade de Kairserslautern, exemplo que utilizaremos ao longo deste trabalho.

Lel entry:Rooms / Room

Notion:Part of a *hallway section* .
A *room* can be a *computer lab* , an *office* , a *hardware lab* , a *meeting room*, and or a peripheral *room* .

Behavioral responses:All *rooms* in a *hallway section* can be accessed via a *connected hallway section* .
For each *room* , the *chosen light scene* can be set by using the *room control panel* .
For each *room* , the *default light scene* can be set by using the *room control panel* .
For each *room* , the value of *T1* can be set by using the *room control panel* .

Figura 3.4 - Entrada do léxico

A grande contribuição da baseline de requisitos reside em fornecer um meio independente de realizar o rastreamento da informação. A baseline é ortogonal ao processo de desenvolvimento, de modo que não é influenciada pelos métodos, técnicas e ferramentas utilizadas durante o processo. Lembramos que Gotel e

Finkelstein apontaram como fator de redução da qualidade do processo de rastreamento a excessiva dependência das ferramentas em métodos e técnicas específicas [Gotel95]. A utilização de linguagem natural nas sentenças facilita a validação junto a clientes. Exemplos deste tipo de sentenças são expressões que descrevem alguma das entidades do modelo, i.e., clientes, ações, eventos externos, estímulos temporais e externos ou saídas. A sentença "*no final do mês o setor solicitante precisa de relatório de custos*" é um exemplo de estímulo temporal, enquanto que "*o setor financeiro fica no prédio central*" descreve um cliente, i.e., pode ser responsável por alguma ação.

A baseline foi subsequentemente estendida de modo a incorporar uma visão de cenários [Leite97]. A idéia era acrescentar a nova visão as outras restantes. Segundo os autores, através da adição desta visão uma série de aspectos que estavam obscuros puderam ser trazidos a tona. Entre eles se encontram estender a amplitude estática da baseline através da introdução explícita dos conceitos de objetivo, relacionamentos entre recursos e identificação de atores. De certa forma o último era tratado através da entidade cliente do modelo básico da baseline e pelos itens autorização e responsável das etiquetas do modelo de configuração. A introdução da noção de ator permitiu maior detalhamento das responsabilidades de cada colaborador.

Outra vantagem da introdução da visão de cenários são os episódios, que permitem a representação e gerenciamento de aspectos comportamentais. Na primeira versão da baseline estes aspectos eram apenas representados de modo sequencial através do modelo básico. Como a introdução dos cenários é possível descrever eventos paralelos, exceções e restrições. De modo geral, podemos afirmar que a introdução de cenários aumentou o poder expressivo da baseline. O modelo entidade relacionamento da notação utilizada pela visão de cenários da baseline é mostrada na Figura 2.5 do capítulo anterior.

A visão de modelo de cenário é a combinação de uma série de idéias presentes na literatura que foram estendidas de modo a atender as necessidades dos autores. A central é que cenários são inicialmente utilizados para descrever situações do ambiente e evoluem de modo a descrever o sistema a ser construído (enfoque de fora para dentro). A evolução dos cenários é um processo constante dentro da baseline,

que, lembramos é perene e segue paralela a evolução do sistema. Os cenários também se utilizam do léxico ampliado da linguagem, escritos em linguagem natural estruturada, para ancorar sua semântica.

Com a introdução da nova visão a já existente visão de hipertexto sofreu algumas modificações. Esta se transformou em um serviço, oferecido a todas as visões restantes. Desta forma o léxico ganhou vida própria, se tornando também uma visão na baseline, visão do modelo do léxico.

Acredita-se que a introdução da visão do modelo de cenários na baseline trouxe grandes benefícios do ponto de vista da validação junto a clientes e usuários. Na primeira versão da baseline o processo de validação era apoiado pela utilização de linguagem natural, supostamente de mais fácil compreensão por parte de clientes, porém a representação da informação através de sentenças ainda era um pouco obscura para leigos. Com a introdução dos cenários foi possível manter os benefícios da utilização de linguagem natural e aliá-los a uma representação mais palatável.

Uma vez pertencentes a baseline de requisitos os cenários também são perenes, ou seja acompanham o processo de desenvolvimento de software. Este é um processo complexo e merece maior detalhamento.

3.4 RESUMO

Neste capítulo apresentamos alguns aspectos da evolução de sistemas de software em geral, dando um tratamento especial a rastreabilidade da informação. Apresentamos um pequeno panorama de novas propostas encontradas na literatura da área que vem a dar conta deste problema. Neste contexto introduzimos a baseline de requisitos, uma estrutura paralela ao desenvolvimento de software projetada para dar suporte a evolução dos artefatos produzidos durante este processo.

Entendendo que cenários devam ser utilizados durante todas as fases do desenvolvimento de software, assumimos que a informação capturada pelos cenários está em constante evolução. Modificações nos cenários tem impactos bidirecionais,

e.g., uma mudança em um cenário de desenho vai impactar aqueles de teste e de implementação, da mesma forma que os cenários de requisitos. Desta forma, a rastreabilidade da informação contida nos cenários se torna fundamental.

De modo a aprofundar nosso conhecimento sobre a evolução de cenários realizamos dois extensos estudos de caso que apresentaremos no próximo capítulo. No primeiro, observamos a evolução de 12 projetos que utilizaram a técnica de cenários. Os resultados deste estudo de caso deram origem a uma taxonomia para a evolução de cenários que foi testada e refinada através de um segundo estudo de caso. A experiência adquirida através da realização de ambos estudos de caso está resumida e representada através de um modelo para evolução de cenários que será apresentado ao final do capítulo.

Capítulo 4 – Modelo de Evolução de Cenários

4.1 INTRODUÇÃO

Neste capítulo procuramos explicar a evolução de cenários durante o processo de desenvolvimento de software. Nos últimos anos o grupo de Engenharia de Requisitos da PUC-Rio tem sistematicamente utilizado cenários como parte de uma metodologia de desenvolvimento de software orientada a objetos. Esta metodologia consiste na evolução gradual das especificações do sistema a partir do Léxico Ampliado da Linguagem, passando por cenários e cartões CRC até chegar no modelo de objetos [Leite90, Wirfs-Brock95, Hadad97, Rivero98, Neto00]. Neste enfoque, utilizamos cenários em todas as fases do desenvolvimento de software, não apenas a durante a elicitação de requisitos. É claro que esta escolha implica na necessidade de um processo organizado para a gerência e manutenção destes cenários, que estarão evoluindo paralelamente ao desenvolvimento do software.

De modo a aprofundar nosso conhecimento sobre a evolução de cenários planejamos um estudo de caso que permitisse a observação imparcial deste processo. Para tal, selecionamos para análise um total de 12 projetos de desenvolvimento de software baseado em cenários, realizados durante os últimos dois anos no Brasil e Argentina. Mais de 200 cenários e 800 episódios foram observados. A Tabela 4.1 apresenta o detalhamento dos números deste estudo de caso. É importante notar que procuramos incorporar a maior variedade possível, tanto na complexidade do produto a ser desenvolvido quanto na atividade fim dos projetos. Desta forma utilizamos projetos contendo desde 3 a 35 cenários e cujos contextos variavam desde sistemas de controle acadêmicos até sistemas para consórcios de automóveis. Os projetos foram realizados por alunos dos cursos de informática, engenharia e profissionais liberais da área de sistemas orientados por membros pertencentes ou relacionados ao grupo de engenharia de requisitos da PUC-Rio. A análise dos projetos foi realizada sem contato direto com os grupos, baseada na documentação física e digital disponível apenas.

ESTUDO DE CASO I	Número total de cenários	Numero de episódios (total)	Número de relacionamentos entre cenários
Projeto I **Pós graduação do departamento** **Versão 1**	39	205	38
Projeto II **Sistema de Biblioteca – Versão 1**	20	98	19
Projeto III **Sistema de Biblioteca -Versão 1**	4	26	4
Projeto III **Sistema de Biblioteca -Versão 2**	9	74	6
Projeto III **Sistema de Biblioteca -Versão 3**	9	21	1
Projeto IV **Gráfica - Versão 1**	13	45	16
Projeto IV.II **Gráfica -Versão 2**	13	51	
Projeto V **Consórcio - Versão 1**	22	85	28
Projeto VI **Consórcio - Versão 1**	18	56	23
Projeto VII **Gerência acadêmica -Versão 1**	26	23	16
Projeto VII.II **Gerência acadêmica -Versão 2**	28	85	29
Projeto VII.III **Gerência acadêmica -Versão 3**	35	104	49
TOTAL	236	873	229

Tabela 4.1 – Detalhamento dos projetos envolvidos no Estudo de Caso I

O restante deste capítulo está dividido como se segue. Na próxima seção apresentaremos a metodologia de análise que empregamos nos doze projetos que constituem o Estudo de Caso I. Os resultados deste estudo de caso estão condensados na Tabela 4.1 sob a forma de uma taxonomia para a de evolução de cenários.

Na seção seguinte apresentamos o Estudo de Caso II, que elaboramos de modo a confirmar os resultados apresentados através da taxonomia de evolução e testá-los durante um ciclo de desenvolvimento de uma aplicação. É importante notar que nossa postura desta vez foi pró ativa, uma vez que éramos os próprios autores do estudo de

caso e nosso objetivo era testar a taxonomia de evolução. Além de servir como veículo para confirmar os resultados iniciais, o Estudo de Caso II também pode ser utilizado como exemplo extenso do processo evolutivo de um sistema, visto a partir da ótica de cenários. É com este espírito que decidimos por torná-lo público e colocá-lo disponível através da Internet[1].

No final do capítulo condensamos todos os resultados que foram derivados da experiência que adquirimos ao realizar ambos estudos de caso em um modelo de evolução de cenários.

4.2 METODOLOGIA DE ANÁLISE

Nesta seção apresentamos a metodologia de análise que empregamos no Estudo de Caso I. Esta se baseou na criação de uma representação interna para os cenários de modo a permitir um tratamento mais sistemático e automatizado da informação. Os cenários utilizados nos projetos foram inicialmente capturados utilizando-se linguagem natural e a notação proposta por Leite [Leite97]. Procedemos por indexar cada instância de componente de cenário e por atribuir a cada uma destas um número de identificação único. Desta forma foi possível referenciar componentes que aparecem em mais de um cenário de forma uniforme. No apêndice I apresentamos a listagem dos cenários dos projetos que compõe o Estudo de Caso I, bem como sua forma codificada. Na Figura 4.1 a seguir mostramos um exemplo de um cenário em sua forma literal e sua respectiva codificação. Este exemplo é derivado do projeto V – sistema para gerência de um consórcio de veículos.

Nome do cenário	objetivo	contexto	recursos	ator	Episódios	exceção	restrição
Entrega de bem sorteado	Outorgar bem a participante	Participante foi sorteado	Disponibilidade do veículo	Participante	1. Participante paga direito de outorgação		
		Participante ganhou licitação	Formulário da compania de seguros	Administradora	2.Participante paga impostos		

[1] O Estudo de Caso II se encontra disponível no endereço:

http:\\stones.les.inf.puc-rio.br/Karin/exemplo/index.html

			Formulário de pedido de veículo		3.Participante apresenta formulário da companhia de seguros		
					4.Participante paga complemento de quotas de integração mínima		
					5.Participante reembolsa despesas de frete		Participante foi notificado das despesas
					6.Participante apresenta formulário de pedido de veículo		
					7.ASSINAR CONTRATO DE RESGATE[2]		
					8..Participante apresenta comprovantes de pagamento		
					9.Veículo é retirado		
					10.Administradora coloca bem a disposição do participante		

Nome do cenário	número	objetivo	contexto	recursos	ator	Episódios	exceção	restrição
Entrega de bem sorteado	19	O8	CEN14	R15	A2	E13		
			CEN15	R19	A1	E15		
				R20		E34		
						E18		
						E11		RS6
						E34		
						CEN20		
						E44		
						E35		
						E36		

Figura 4.1 - exemplo de um cenário em formato literal e sua codificação

[2] Esta notação denota que o episódio na verdade se refere a um outro cenário, no caso o cenário Assinar contrato de resgate. Observe que na Tabela referente a codificação deste episódio, ele é referenciado como CEN20.

Uma vez codificados os cenários partimos para um processo de análise que chamamos de intra configuração, onde objetivamos determinar os relacionamentos existentes entre os cenários de uma mesma configuração. Neste processo, analisamos manualmente todos os cenários de modo a detectar relacionamentos e dependências entre eles.

Inicialmente utilizamos os relacionamentos identificados por Breitman em [Breitman98]. Separamos o conjunto de projetos em dois blocos distintos e identificamos relacionamentos entre os cenários do primeiro bloco. Separamos os resultados por tipo de relacionamento. Analisamos cada um deles de modo a coletar indicativos para o reconhecimento de cada relacionamento individualmente. Baseados nos componentes básicos dos cenários fizemos uma análise comparativa observando a presença, frequência e quantidade de componentes que cada par/conjunto de cenários que mantivessem um tipo de relacionamento apresentassem em comum. Como resultado deste processo elaboramos uma lista de heurísticas para a identificação de relacionamentos entre cenários, baseadas nos componentes dos mesmos. Em posse da lista de heurísticas, planejamos uma estratégia para teste e validação a partir da aplicação das mesmas sobre os projetos.

Primeiramente aplicamos as heurísticas sobre o primeiro bloco de projetos, os quais já tínhamos identificado relacionamentos entre os cenários utilizando a taxonomia proposta em [Breitman98], que utiliza os cenários em sua forma literal. Desta vez utilizamos a representação interna (codificada) pois esta facilitava a identificação de coincidências e o cálculo da frequência de aparecimento dos componentes dos cenários envolvidos.

Nosso objetivo era verificar se conseguiríamos detectar os mesmos relacionamentos detectados anteriormente. Para tal, utilizamos um misto de técnicas de casamento de padrão (*pattern matching*) e a análise manual. Na seção 4.3 apresentamos as heurísticas utilizadas e mostramos vários exemplos de relacionamentos detectados através da utilização das mesmas. Nesta fase, conseguimos detectar a maioria dos relacionamentos identificados na fase anterior. Os erros podem ser justificados pela necessidade de ajuste das heurísticas, por erros na codificação dos elementos, e.g., episódios *verificar senha* e *validar senha* com códigos distintos quando se referem a

mesma atividade e, finalmente, por falta de informação que permitisse definir o tipo de relacionamento em questão.

Analisamos o segundo bloco de projetos utilizando estas heurísticas de modo a refiná-las. Nos casos onde não havia informações suficientes para a determinação da existência de relacionamentos intervimos utilizando os cenários literais para a análise. Acreditamos que o cenário futuro de utilização das heurísticas será este. A maior parte do reconhecimento das operações poderá ser automatizada utilizando mecanismos do tipo *pattern matching,* porém sempre haverão casos onde a intervenção humana será necessária.

Uma vez identificada a totalidade dos relacionamentos para cada configuração, desenhamos um gráfico que ilustra a rede de interdependências entre os cenários. Um exemplo de um destes gráficos é mostrado na Figura 4.2, a seguir. Notamos que uma representação gráfica para a rede de relacionamentos entre os cenários de uma configuração facilitava a análise.

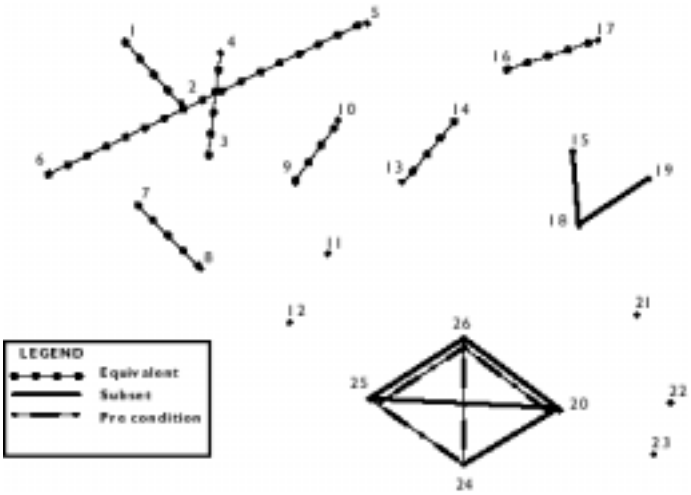


Figura 4.2 - Exemplo de gráfico de relacionamentos entre cenários de uma mesma configuração

Uma vez finalizada a análise intra configuração, passamos para o estudo do processo de evolução dos cenários ao longo do tempo. Nesta fase comparamos as mudanças ocorridas na passagem de uma configuração para outra consecutiva. Chamamos este

processo de análise inter configuração e o dividimos em três estágios, comparação entre os gráficos de relacionamentos, análise do versionamento de cenários, e análise dos relacionamentos. Detalharemos estes estágios a seguir.

No primeiro fazemos uma comparação entre os gráficos de relacionamento das duas configurações em questão e listamos os elementos que estão presentes em apenas uma delas. Analisamos em primeiro lugar os novos elementos. Anotamos os novos cenários bem como novos componentes que apareceram em virtude destes, e.g., episódios, atores e contexto que não existiam em cenários anteriores. Esta lista será importante para determinar se o cenário é realmente novo ou se é resultado de alguma operação. Se os componentes apresentados pelo novo cenário não forem novos, procuramos os cenários presentes na configuração anterior que possuíam estes componentes. Verificamos o que aconteceu com estes cenários nesta configuração e determinamos que se o novo cenário é fruto de uma operação do tipo *divisão*, por exemplo. Realizamos o procedimento inverso para os cenários que sumiram da base, i.e., investigamos os cenários da presente configuração que possam conter os elementos presentes no cenário que desapareceu. Desta forma determinamos se o desaparecimento foi fruto da eliminação da informação ou se resultado de uma operação do tipo *fusão*, por exemplo.

O segundo estágio é a análise do versionamento de cenários. De modo a descobrir se um mesmo cenário sofreu versionamento entre duas configurações analisamos se houve mudanças em seus componentes. Para cada cenário presente em ambas configurações fazemos uma pesquisa de modo a verificar se houve modificação, adição ou subtração de
componentes. No caso de adição de conteúdo, investigamos os relacionamentos que o cenário possuía na versão anterior de modo a verificar se este sofreu alguma operação que incorporasse outros cenário ao mesmo. No caso de subtração de informação tentamos localizar quais cenário da presente versão possuem os componentes que estão faltando (e não possuíam em versões anteriores). Desta forma podemos verificar se houve alguma operação de separação da informação.

O terceiro e último estágio da análise é a evolução dos relacionamentos. Comparamos os gráficos de modo a verificar a persistência destes durante o processo evolutivo.

Comparamos ambos esquemas e listamos os relacionamentos que sumiram e o aparecimento de novos relacionamentos. A seguir, verificamos individualmente cada um dos relacionamentos anotados de modo a refinar as operações detectadas através dos passos anteriores. Na prática, o melhor indicativo da aplicação de operações foi o aparecimento de relacionamentos de precedência, i.e., pré condição e possível precedência que serão detalhados na seção 4.3, a seguir. O aparecimento destes quase sempre indica a união de cenários e a necessidade da incorporação de algum indicativo de tempo. A investigação de mudanças em outros relacionamentos tais como equivalência e exceção trouxe pouco ganho de informação, na realidade.

Nas duas seções seguintes apresentaremos os frutos das análises intra configuração, sob a forma de relacionamentos, e inter configuração, sob a forma de operações.

4.3 RELACIONAMENTOS

Cenários descrevem situações em contextos do mundo real. Apesar de serem auto contidos por definição, no sentido de descrever uma série de episódios que tem por meta atingir um objetivo, não é razoável pensar em cenários como entidades independentes [Zorman95]. Cenários estão conectados a outros cenários em uma rede complexa e intricada de relacionamentos. Devemos pensar em um cenário como um nó de um hipertexto, é atômico no que se refere a informação que contém e está relacionados a outros cenários através de elos. Estes elos podem ser de vários tipos, e.g., podem representar precedência, equivalência e, até mesmo, comportamento excepcional. Os elos conectam porções de informação relacionada. Devemos enfatizar que os relacionamentos só tem sentido ao nível de instância, pois só podem ser determinados a partir do conhecimento do conteúdo de cada cenário.

Em trabalhos anteriores percebemos que existia uma ligação entre a existência de determinado relacionamento e a incidência de alguns dos componentes dos cenários [Breitman98]. Esta constatação foi reafirmada durante a realização do estudos de caso e pelos resultados de Alspaugh e outros baseado em nossos resultados iniciais [Alspaugh99]. A seguir detalhamos cada um dos relacionamentos e apresentamos heurísticas para sua detecção baseadas nos componentes dos cenários envolvidos.

4.3.1 COMPLEMENTO

Definimos que dois ou mais cenários estabelecem um relacionamento de complementaridade entre si quando, vistos em conjunto, respondem a um objetivo global. Nestes casos foi observado que os cenários envolvidos além de compartilhar o mesmo objetivo, apresentam coincidências de contextos e entre recursos. Estes cenários também devem apresentar um alto índice de coincidência entre atores e podem apresentar similaridade de episódios. Para ilustrar esta situação vamos tomar um exemplo do projeto I, sistema de gerência acadêmica. Neste projeto existem três cenários que compartilham o objetivo *cumprir requisito básico,* cenários fazer crédito, apresentar qualificação e apresentar tese. Analisados em conjunto, estes cenários provavelmente atendem a um objetivo geral *obter título de doutorado.* Estes cenários atendem o restante das exigências acima que caracterizam o relacionamento de complemento além de compartilhar o objetivo. Como ilustrado na Figura 4.3 os cenários compartilham recursos e contexto, apresentam coincidência de atores e pouca semelhança de episódios.

	Objetivo	contexto	recurso	ator	episódios	exceção	restrição
apresentar qualificação	O2	C32	R4	A16	E57		RT4
		C2	R6	A2	E93	EX8	
		C31		A3	E94		
				A7	E18		
					E95		
apresentar tese	O2	C3	R21	A1	E6		
		C30	R5	A3	E57		
		C2	R6	A7	E8	EX1	
		C1		A2	E9		
					E10		
fazer crédito	O2	C2	R4	A7	E57		
		C30	R15	A2	E58	EX8	
			R6	A8	E12		
				A3	E59		
					E60		
					E61		
					E56		
					E62	EX9	

Figura 4.3 – Exemplo de cenários que apresentam o relacionamento complemento (projeto I)

Na Tabela 4.2 a seguir resumimos as heurísticas para a detecção do relacionamento de complementariedade entre dois ou mais cenários.

Complemento
Cenários devem possuir objetivos idênticos.
Existe coincidência entre o contexto dos cenários envolvidos.
Existe coincidência de recursos entre os cenários. Esta coincidência pode ser total, todos os cenários apresentam o mesmo recurso ou parcial, i.e., notada entre pares dos cenários envolvidos.
Existe grande coincidência de atores entre os cenários envolvidos.
Existe notadamente pequena coincidência entre os episódios de cada um dos cenário envolvidos.

Tabela 4.2 – Heurísticas para a detecção do relacionamento de complementariedade

4.3.2 RELACIONAMENTO DE EQUIVALÊNCIA

Definimos o relacionamento de equivalência como indicativo de semelhança entre dois ou mais cenários. Este relacionamento foi observado em conjuntos de cenários que compartilham um mesmo objetivo ao mesmo tempo em que apresentam coincidência de contexto. Pouca coincidência de recursos foi observada. Estes cenários também devem apresentar alto índice de coincidência entre atores e episódios.

Durante a confecção dos estudos de casos percebemos que a análise era facilitada se procedêssemos por analisar os casos de equivalência e complemento concomitantemente. Em ambos os casos os cenários compartilham um mesmo objetivo, portanto é razoável supor que ou estes se complementem de modo a atingir um objetivo maior ou que sejam redundantes, i.e., tratem de situações semelhantes. Para ilustrar o relacionamento de equivalência vamos tomar um exemplo do projeto VII, sistema de gerência acadêmica. Neste projeto mostramos dois cenários que, a menos de pequenas modificações, tratam de uma mesma situação. Novamente,

mostramos os cenários em sua forma codificada, de modo a facilitar a comparação e visualização das diferenças. Os cenários em sua forma literal se encontram no apêndice I. Note que os episódios dos cenários são idênticos, variações são encontradas somente no contexto e atores envolvidos.

Nome	objetivo	contexto	recurso	ator	Episódio
matricular-se no programa de mestrado	O5	C9	R27	A14	E114
		C8		A3	E39
					E18
matricular-se no programa de doutorado	O5	C14	R27	A1	E114
		C8		A14	E39
		C4			E18

Figura 4.4 – Exemplo de dois cenários que apresentam o relacionamento de equivalência (projeto VII)

Na Tabela 4.3 a seguir resumimos as heurísticas para a detecção do relacionamento de Equivalência entre dois ou mais cenários.

Equivalência
Cenários devem possuir objetivos idênticos.
Existe coincidência entre o contexto dos cenários envolvidos.
Existe pequena coincidência de recursos entre os cenários
Existe grande coincidência de atores entre os cenários envolvidos.
Existe grande coincidência de episódios entre os cenário envolvidos.

Tabela 4.3 – Heurísticas para a detecção do relacionamento de Equivalência

4.3.3 RELACIONAMENTO DE CONTENÇÃO (SUBSET)

O relacionamento de contenção é observado entre um par de cenários que compartilham o mesmo contexto, ou pelo menos o contexto de um deles é totalmente contido no contexto do segundo. Estes cenários podem apresentar coincidência de recursos e tem de apresentar coincidência de atores. Pelo menos um dos episódios do

cenário contido deve estar presente no cenário contendor. Para ilustrar o relacionamento de contenção selecionamos um exemplo do projeto VII, sistema de gerência acadêmica. Neste projeto mostramos o cenário terminar programa de mestrado que mantém relacionamentos de contenção com os cenários apresentar dissertação e fazer disciplina (os relacionamentos são par a par e completamente independentes um do outro). O relacionamento é bastante explícito em ambos os casos, pois o contexto dos dois últimos cenários está contido no cenário pai. Também notamos que os cenários possuem atores em comum e que os cenários contidos aparecem na lista de episódios do cenário pai.

Nome	objetivo	Contexto	recurso	ator	Episódio
terminar programa de mestrado	O25	C2	R4	A1	E115
		C16	R48		CEN18
					CEN12
					E116
apresentar dissertação	O25	C2	R49	A1	
fazer disciplina	O26	C16	R4	A3	
			R31	A1	
			R10		

Figura 4.5 – Exemplo de cenários que apresentam o relacionamento de contenção (projeto VII)

Na Tabela 4.4 a seguir resumimos as heurísticas para a detecção do relacionamento de Contenção entre dois ou mais cenários.

Contenção
Cenários envolvidos podem apresentar objetivos diferentes.
O contexto de um cenário A contido em B deve ser igual ao de B, podendo ser acrescido de algumas restrições.
Pode existir coincidência de recursos entre os cenários envolvidos.
Existe grande coincidência de atores entre os cenários envolvidos.
Pelo menos um episódio do cenário contenedor deve fazer parte do cenário contido.

Tabela 4.4 – Heurísticas para a detecção do relacionamento de Contenção

4.3.4 RELACIONAMENTO DE PRÉ-CONDIÇÃO

Uma pré condição é um relacionamento um a um definido dentro do componente contexto de um cenário. Um cenário que age como pré condição para outro cenário tem de aparecer no contexto do último. Completando a caracterização do relacionamento notamos também a coincidência de pelo menos um ator e um episódio nestes casos.

A pré condição é um tipo especial de relacionamento no que permite incorporar uma dimensão temporal no gráfico de relacionamentos estáticos entre os cenários. Este relacionamento permite a definição de uma sequência para os cenários e a especificação de estágios que devem ser completos antes da execução de outros. A utilização de pré condições é muito importante mas não devemos abusá-la nos projetos. Durante a análise dos estudos de caso notamos que alguns autores utilizaram pré condições de modo excessivo resultando em altos graus de acoplamento em seus projetos. Para ilustrar o relacionamento de pré condição selecionamos um exemplo do projeto VI, sistema de consórcio de automóveis. Neste projeto mostramos o cenário sorteio como pré condição do cenário recusar veículo sorteado. Note que ambos cenários compartilham o ator A1 e que o último cenário aparece como pré requisito no contexto do primeiro.

Nome	número[3]	objetivo	contexto	recurso	ator	Episódios
Sorteio	17	O6	C7	R16	A1	E6
				R8	A6	E23
						E23
						E24
						CEN1
Recusar veículo sorteado	15	O7	C7	R10	A1	E6
			CEN17	R8	A2	CEN11

Figura 4.6 – Exemplo de cenários que apresentam o relacionamento de pré condição (projeto VI)

[3] Acrescentamos uma coluna atribuindo números aos cenários de modo a referenciá-los mais facilmente em sua versão codificada. A numeração não faz parte da notação para cenários proposta por Leite [Leite97]

Na Tabela 4.5 a seguir resumimos as heurísticas para a detecção do relacionamento de Pré-condição entre dois cenários.

Pré-condição
(cenário A mantém relacionamento de pré condição com cenário B)
Cenário A faz parte do contexto do cenário B
Existe coincidência de pelo menos um ator entre os cenários envolvidos.
Pode existir coincidência de episódios entre os cenários envolvidos.

Tabela 4.5 – Heurísticas para a detecção do relacionamento de Pré-condição

4.3.5 RELACIONAMENTO DE DETOUR

Definimos como detour um relacionamento entre dois cenários que se estabeleceu como consequência do desenrolar excepcional do primeiro. Durante o curso de um cenário, uma exceção pode ocorrer que resulte em que um de seus atores seja remetido para um cenário secundário. Neste cenário, o comportamento excepcional é resolvido, e se retoma o curso normal dos eventos do primeiro cenário, a partir do ponto onde se deu a exceção. A provisão para estes casos chamamos de detour. Para que este se caracterize é necessário que ponteiros para os cenários estejam explicitados na lista de episódios ou no contexto de ambos cenários. Para ilustrar o relacionamento de detour selecionamos um exemplo do projeto II, sistema para gerência e controle de bibliotecas. Neste projeto mostramos que a exceção presente no cenário Emprestar Livro tem como exceção tirar cartão da biblioteca e que este, por sua vez apresenta um episódio que remete o ator para o cenário inicial.

Nome	número[4]	Objetivo	contexto	recurso	ator	Episódio	exceção	restrição
Emprestar livro	5	O3	C3	R5	A5	E13	CEN8	
							CEN15	
			C5	R7	A6	E7		
					A8	E17	CEN14	
						E10		
						E15	EX3	RT1

[4] Novamente inserimos uma nova coluna de numeração para facilitar a referência aos cenários.

						E16		
tirar cartão de biblioteca	15	O6	C5	R15	A5	E24		
					A6	E33		
						E33		
						CEN5	CEN14	RT5

Figura 4.7– Exemplo de par de cenários que apresentam o relacionamento de detour (estudo de caso II)

Na Tabela 4.6 a seguir resumimos as heurísticas para a detecção do relacionamento de Detour entre dois cenários.

Detour
Um relacionamento de Detour entre os cenários A e B se estabelece quando: Cenário B é uma exceção de um dos episódios do cenário A, e Cenário A é um dos episódios do cenário B.

Tabela 4.6 – Heurísticas para a detecção do relacionamento de Detour

4.3.6 RELACIONAMENTO DE EXCEÇÃO

Na realidade o indicativo de exceção faz parte do conteúdo dos cenários, i.e., um dos componentes da notação de cenários utilizada para os estudos de caso é exceção de um ou mais episódios. Neste caso, apenas tornamos explícito este relacionamento de modo a facilitar a visualização da dinâmica de um conjunto de cenários. Este relacionamento é definido de forma bastante direta, basta que exista o indicativo no componente do cenário. Para ilustrar o relacionamento de exceção selecionamos um exemplo do projeto IV-versão 2, sistema para gerência de uma gráfica de impressos. Neste projeto mostramos que uma exceção para o cenário efetuar serviço é o cenário cancelar ordem de serviço. A Figura 4.8 mostra estes cenários.

Nome	número[5]	objetivo	contexto	recursos	ator	cenário	exceção
efetuar serviço	4	O2	C5	R2	A1	CEN 1	
			C12	R3	A2	CEN 3	
			C7	R4	A3	E1	

[5] Novamente inserimos uma coluna de numeração para facilitar a referência a cenários em sua forma codificada.

				R5		CEN8	
				R6		CEN 10	CEN 11
						CEN 12	
						CEN 13	
cancelar ordem de serviço	11	O6	CEN7	R4	A1	E15	
			CEN8	R5	A3	E16	
			C10			E17	
			C12			E22	

Figura 4.8 – Exemplo de par de cenários que apresentam o relacionamento de exceção (projeto IV)

Na Tabela 4.7 a seguir apresentamos a heurística para a detecção do relacionamento de Exceção entre dois cenários. Esta nada mais é do que a simples contatação da existência de uma exceção sob a forma de um segundo cenário. Explicitamos esta heurística de modo a tornar mais clara a distinção entre o relacionamento de Detour apresentado na última seção e o de Exceção. No primeiro caso, Detour, existe uma circularidade, ao se detectar comportamento excepcional, o ator é remetido a um outro cenário porém retoma ao inicial. No caso do relacionamento de Exceção não existe a volta ao cenário inicial, muito pelo contrário, indica que a ação tomou novos rumos.

Exceção
Existe um relacionamento de exceção entre os cenários A e B quando: Existe uma exceção em um dos episódios do cenário A, e esta exceção é o próprio cenário B.

Tabela 4.7 – Heurísticas para a detecção do relacionamento de Detour

4.3.7 RELACIONAMENTO DE INCLUSÃO

Este relacionamento foi proposto por Grady Booch e outros no contexto de *use cases* [Booch99]. Sua intenção é isolar um cenário que contém um procedimento comum a vários outros cenários e colocá-lo a disposição. Desta forma reduzimos redundância de informações e facilitamos a manutenção do cenário. O cenário que contém a informação, por outro lado, não tem significado se tomado independente de outros. A utilização deste artifício foi constatada em apenas um dos estudos de caso. Nossa

hipótese é que os autores tinham conhecimentos de modelagem orientada a objetos que foram estendidos e aplicados no design de cenários. Notamos porém que a introdução deste tipo de artifício torna o design artificial, como foi dito anteriormente o cenário introduzido não tem significado independente, dificultando o entendimento por parte dos clientes e se afastando do objetivo primeiro da utilização de cenários que é facilitar a comunicação. A relação custo benefício da utilização deste artifício deve ser considerada cuidadosamente pois se de um lado facilita o desenho, por outro prejudica o entendimento. Para ilustrar o relacionamento de exceção selecionamos um exemplo do projeto IV-versão 2, sistema para gerência de uma gráfica. Neste projeto mostramos um exemplo onde dois cenários compartilham um mesmo procedimento de entrada e seleção de função que, através da operação de inclusão pode ser isolado em um cenário separado. Desta forma se evita repetir o procedimento nos cenários que precisam fazer uso do mesmo, reduzindo a redundância de informação. Nas Figuras 4.9 e 4.10 a seguir mostramos um esquema representando o relacionamento entre os cenários e o código de cada um deles respectivamente.

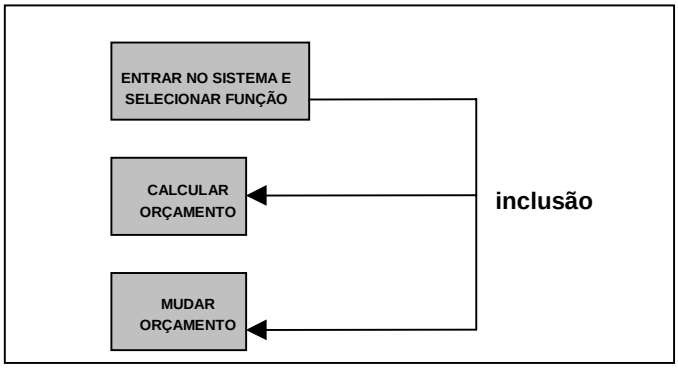


Figura 4.9 – Esquema exemplificando o relacionamento de inclusão entre cenários (projeto IV)

nome	número[6]	objetivo	Contexto	recurso	ator	episódio
entrar no sistema e selecionar função	6	O11	C15	R9	A2	E15
					A4	E16
						E21
						E17

[6] Novamente números aos cenários de modo a referenciá-los mais facilmente em sua versão codificada.

						E26
						E18
calcular orçamento	5	O3	C2	R2	A1	E6
			C13	R3	A2	CEN 6
			C7		A3	E31
						E5
mudar orçamento	9	O3	CEN7	R4	A1	E13
			C12	R3	A2	CEN 6
			C7		A3	E14
						E9
						CEN 7
						E11

Figura 4.10 – Exemplo de par de cenários que apresentam o relacionamento de inclusão (projeto IV)

Na Tabela 4.8 a seguir resumimos as heurísticas para a detecção do relacionamento de Inclusão entre dois ou mais cenários.

Inclusão
Pelo menos um episódio é compartilhado por todos os cenários envolvidos.

Tabela 4.8 – Heurísticas para a detecção do relacionamento de Inclusão

4.3.8 RELACIONAMENTO DE POSSÍVEL PRECEDÊNCIA

Definimos o relacionamento de possível precedência quando, a partir de um caso excepcional se estabelece uma situação onde é necessário fixar a ordem em que os cenários envolvidos devem seguir. A idéia por trás do relacionamento é de que existe a possibilidade de uma situação excepcional, que pode nunca vir a ocorrer porém, se acontecer, devemos tomar o cuidado de que aconteça em um tempo previsto. Na realidade este é um caso especial do relacionamento de Pré condição. O último define a necessidade incondicional da ocorrência de um ou mais cenários a priori da realização do cenário em questão. No caso do relacionamento de possível precedência os cenários anteriores não necessariamente devem ocorrer. Na verdade somente virão a ocorrer no caso de uma exceção no cenário original. Este relacionamento estabelece quais são os cenários que deverão ser realizados neste caso, e o momento em que estes devem ocorrer. Este relacionamento também se distingue do relacionamento de Detour porque não necessariamente remete o ator ao cenário original no fim do

tratamento da exceção, pelo contrário, nos casos observados, notou-se que os atores tomaram um curso de ações bastante distintos do que o previsto pelo cenário original onde o comportamento excepcional ocorreu. Da mesma forma que o relacionamento de Pré-condição é especial por introduzir a noção de temporalidade, o relacionamento de possível precedência embute a idéia de disponibilidade, i.e., de períodos de tempo ou situações onde um determinado cenário é válido. Para ilustrar o relacionamento de exceção selecionamos um exemplo do projeto VII- versão 3, sistema para gerência acadêmica. Neste projeto mostramos o cenário matricular em disciplina tendo como possível precedência o cenário oferecer nova disciplina. O relacionamento de possível precedência indica que é possível que novas disciplinas sejam oferecidas, porém se verdadeiro, deverá ocorrer antes do prazo de matrícula.. A Figura 4.11 ilustra a situação.

nome	número	objetivo	Contexto	recurso	atores	episódios
oferecer nova disciplina	27_3	O26	C7	R11	A3	E63
					A19	E64
					A1	CEN42_3
matricular em disciplina	42_3	O26	C7	R21	A1	E17
				R13	A2	E10
				R51		E18
						E19
						E23

Figura 4.11 – Exemplo de par de cenários que apresentam o relacionamento de possível precedência (projeto VII)

Na Tabela 4.9 a seguir apresentamos heurísticas que nos auxiliaram na detecção do relacionamento de Possível Precedência, entre dois ou mais cenários. Note que alguns dos conceitos presentes são dificilmente quantificados, tais como a coincidência entre episódios e contexto. Nestes casos, as heurísticas servem como apoio, porém o resultado final fica fortemente dependente do bom senso de quem as aplicou.

Possível Precedência
(cenário A mantém relacionamento de possível precedência com cenário B)
Pode existir coincidência entre os objetivos do cenários envolvidos.
Existe coincidência entre o contexto do cenários envolvidos.
Existe coincidência de pelo menos um dos atores.

Existe episódio em A que seja o cenário B. Cenário A não faz parte do contexto do cenário B.

Tabela 4.9 – Heurísticas para a detecção do relacionamento de Possível Precedência

4.4 OPERAÇÕES

O conjunto de operações é o núcleo do processo de evolução de cenários . Novas versões dos cenários são geradas a partir da aplicação de operações. Da mesma forma os relacionamentos, a partir da primeira versão da base de cenários, são modificados apenas como resultado indireto da aplicação de uma ou mais operações. Nesta seção apresentaremos as operações que identificamos em detalhe.

Identificamos dois tipos de operações, intra e inter cenários. As operações intra cenário levam em conta a manutenção do conteúdo de um único cenário. Através destas podemos modificar, retirar ou adicionar conteúdo a um cenário. As operações inter cenário, por sua vez, tem um escopo mais amplo. Elas lidam com grupos de cenários e tem influência sobre os relacionamentos entre os cenários resultantes. A seguir apresentamos os dois blocos de operações.

4.4.1 OPERAÇÕES INTRA CENÁRIO

São três as operações intra cenário, inclusão, modificação e retirada de cenário. A primeira operação diz respeito a criação e introdução de um novo cenário na base. De modo geral, é o resultado do processo de elicitação de informação e acontece com mais frequência durante as etapas iniciais do processo. Não obstante, novos cenários podem surgir em qualquer fase de desenvolvimento, pois lembramos que um sistema de software faz parte de um sistema maior e, desta forma, estará sempre sujeito a mudanças do ambiente onde está inserido.

A operação de modificação pode ser definida como um processo de refinamento da informarção codificada em um cenário. A medida em que se desenrola o desenvolvimento de um sistema, aumenta a compreensão do ambiente onde este será

inserido e de seus requisitos. Durante este processo, a informação capturada nas fases iniciais vai sendo refinada de modo a refletir mais precisamente as necessidades do sistema. Em consequência o conteúdo dos cenários associados sofre modificações. A observação empírica dos estudos de caso realizados mostrou que este é realmente um processo contínuo e permeia todas as fases de desenvolvimento.

Finalmente a operação de retirada consiste na exclusão de um cenário da base. Cenários podem ser excluídos por várias razões, as mais comuns são erros no processo de captura de informações. Neste caso estamos tratando de um problema comum a qualquer processo de elicitação de informação. Nem sempre é possível abarcar toda a informação relativa a um projeto de uma única vez e, erros são inevitáveis. Outro caso, mais raro, são circunstâncias externas ao software que determinam mudanças no meio ambiente e que tem impacto sobre os cenários.

4.4.2 OPERAÇÕES INTER CENÁRIOS

Nesta sub seção trataremos de operações que envolvem dois ou mais cenários. De modo a facilitar seu entendimento, dividimos o conjunto destas operações em três grandes blocos. O primeiro bloco é constituído das operações que reduzem a base através da união de dois ou mais cenários. Estas operações são as de fusão, encapsulamento, consolidação. Detalharemos cada uma destas operações na seção seguinte. O segundo bloco de operações é aquele que expande a base através da divisão e criação de novos cenários. Estas operações são as de divisão, múltipla-divisão, extensão e especialização. Novamente, estas operações serão detalhadas nas seções seguintes. O terceiro e último bloco corresponde as operações de que alteram em uma unidade o número de cenários na base. Neste enfoque são consideradas as operações de exclusão e adição de novo cenário. Este grupo é especial na medida em que os dois primeiros estão atrelados a ele, i.e. , toda operação de redução da base resulta na exclusão de um ou mais cenários, ao mesmo passo em que toda operação de expansão inclui um ou mais novos cenários na base. Visto deste ângulo somos induzidos a concluir que a adição destas operações ao conjunto seria redundante. Por outro lado, devemos lembrar que os requisitos estão em constante evolução. Durante o processo de desenvolvimento de software não é raro o aparecimento de novos requisitos, sejam consequência de fatores alheios ao software ou demandas dos

próprios clientes. Desta forma não só é possível mas também justificável o aparecimento de cenários completamente novos, i.e., que não resultam da divisão de cenários existentes, na base. Pela mesma regra entendemos a possibilidade da exclusão total (e não a realocação da informação, que resulta das operações de união) de cenários.

Esta seção está subdividida em três grupos, conforme visto anteriormente. Em cada uma das seções apresentaremos as respectivas operações e exemplos que ilustram o impacto da aplicação das mesmas sobre a base de cenários. Todos os exemplos apresentados são oriundos dos projetos que compõem o estudo de caso realizado.

4.4.2.1 OPERAÇÕES DE REDUÇÃO

Nesta seção apresentaremos as operações de *fusão*, *encapsulamento*, *consolidação*. O objetivo geral das duas primeiras operações é a união do conteúdo de dois ou mais cenários em um único cenário. A diversidade entre elas é explicada através dos relacionamentos prévios existentes entre os cenários envolvidos e na quantidade de informação (conteúdo) que será descartada ao final da operação. A operação de *consolidação* difere das demais por transformar um único cenário em outro com um grau de abstração maior. Como resultado desta operação temos um cenário com seu escopo aumentado ou uma generalização do procedimento tratado pelo cenário original. Esta operação evita a introdução de cenários equivalentes na base que no fundo tratam do mesmo procedimento, a salvo de pequenas modificações.

4.4.2.1.1 FUSÃO

A *fusão* visa a união de dois ou mais cenários que estão logicamente relacionados em um único cenário. Muitas vezes durante o desenvolvimento aparecem cenários que são muito breves, ou contém relativamente pouca informação para justificar sua presença independente. Esta operação objetiva juntar estes procedimentos em um único cenário tornando o conjunto mais expressivo.

A operação de *fusão* é caracterizada por condensar o conteúdo de dois ou mais cenários em um único, preservando parte da informação dos cenários originais. Os cenários iniciais devem ser complementares, o que significa que podem ter coincidência de conteúdo. Se houver indicação de sequência de ações, i.e., existência de relacionamentos do tipo pré condição e possível precedência entres os cenários envolvidos, estes relacionamentos devem ser levados em conta de modo que o cenário resultante reflita a sequência de episódios inicial . Especial atenção deve de ser dada aos relacionamentos que os cenários originais possuem com outros cenários não envolvidos na *fusão*. Após a operação estes relacionamentos devem ser reavaliados frente ao novo cenário, de modo a determinar a validade dos relacionamentos anteriores. Também devemos verificar os relacionamentos de equivalência, complemento e contenção, pois como o cenário resultante apresenta grandes diferenças estruturais se comparado um a um com os originais, é possível que estes relacionamentos não sejam mais válidos, uma vez que estão fortemente ligados a coincidência de componentes. Na Figura 4.12 mostramos um exemplo da operação *fusão* do projeto II – Sistema de biblioteca. Neste exemplo, bem como no restante dos exemplos desta seção, optamos por mostrar os cenários representados por caixas contendo o nome do cenário apenas. Escolhemos exemplos onde esta informação fosse significativa o suficiente para a compreensão do intento do cenário. Desta forma evitamos a sobrecarga de informação nas figuras e nos concentramos em ilustrar as operações e seus impactos. O texto integral dos cenários se encontra no apêndice I.

Note que os cenários cadastrar senha e apresentar identificação contém pouca informação e sua presença como cenários independentes não é justificada. Na realidade, no projeto o qual fazem parte, estes cenários se transformaram em episódios do cenário resultante.

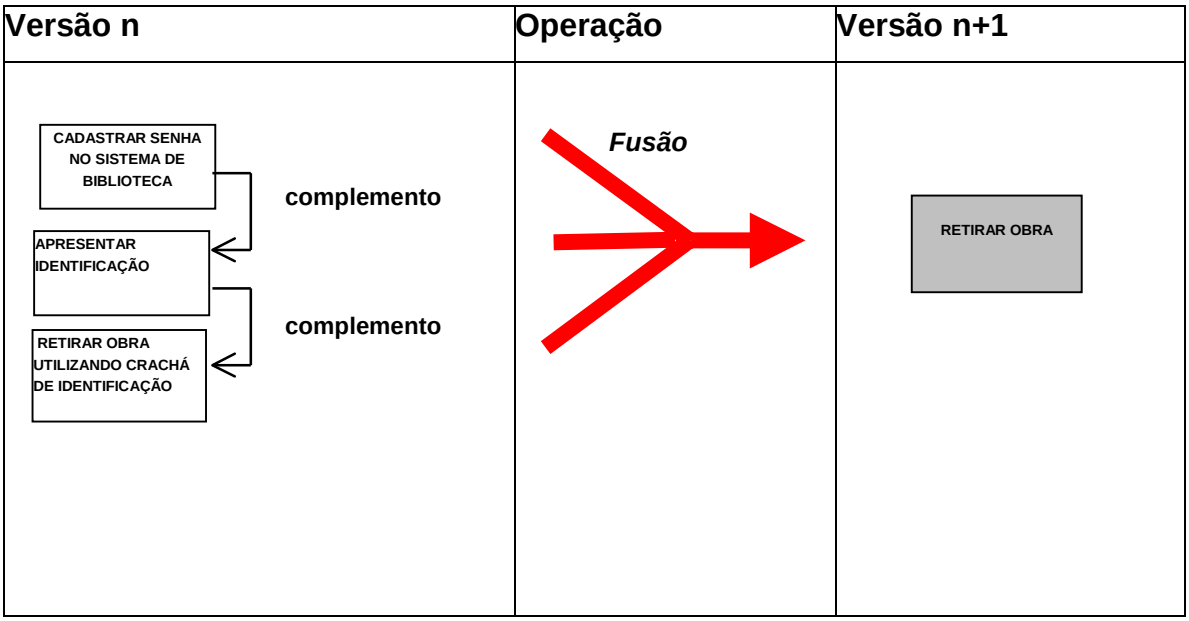


Figura 4.12 – Exemplo da operação de fusão *(projeto II)*

4.4.2.1.2 ENCAPSULAMENTO

O objetivo da operação de *encapsulamento* é a união de dois ou mais cenários que contém muita coincidência de conteúdo. A operação é bastante simples, qualquer par ou conjunto de cenários que mantém o relacionamento de equivalência é elegível para a operação. O cenário resultante, a princípio, deve herdar todos os relacionamentos que os cenários mantinham individualmente. Na realidade, percebemos que na maioria dos casos onde ocorreu o encapsulamento havia coincidência de relacionamentos além da de conteúdo entre os cenários. Esta é uma conjectura razoável, pois se os cenários apresentam grande coincidência de conteúdo, é justo que os mecanismos de detecção de relacionamentos apontarão para resultados similares. A única exceção é o relacionamento de pré condição. Uma vez que este é caracterizado mais fortemente pelo contexto dos cenários e não pela coincidência de seu conteúdo, pode ocorrer a situação onde um dos cenários iniciais possui uma pré condição enquanto que os outros não a compartilham. A validade desta operação para o cenário resultante do *encapsulamento* não pode ser automaticamente determinada pois varia caso a caso. A mesma regra é válida para o caso do relacionamento de possível precedência. Uma ferramenta que apoie a realização desta operação deve levar em conta este fato, e prover um mecanismo que permita que os autores dos cenários sejam consultados em caso de dúvida. Na Figura 4.13 mostramos um exemplo da

operação *encapsulamento* do projeto VII – Sistema de gerência acadêmica. Note que os dois cenários originais tratam, basicamente do mesmo procedimento. O cenário resultante fundiu os dois em um cenário único.

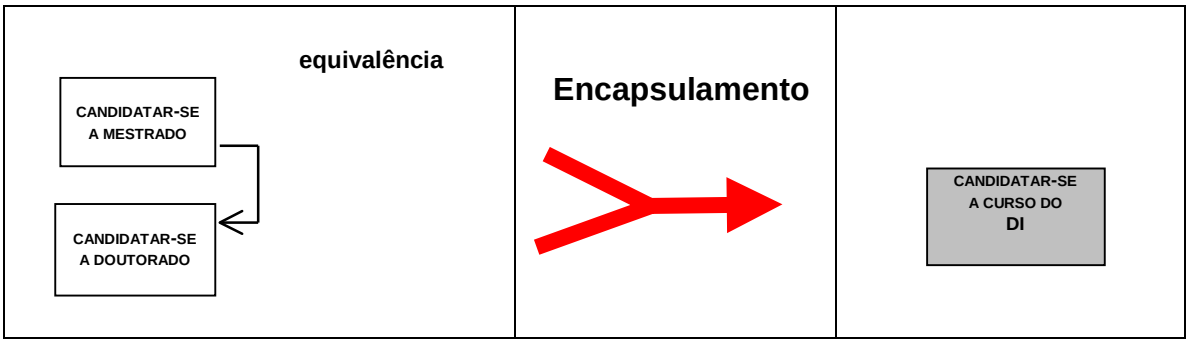


Figura 4.13 – Exemplo da operação de encapsulamento *(projeto VII)*

4.4.2.1.3 CONSOLIDAÇÃO

A operação de *consolidação* tem como objetivo aumentar o escopo de um único cenário. Devemos entendê-la como um procedimento de generalização. Durante o curso dos estudos de caso observamos que ao invés de criar cenários equivalentes, certos autores optaram por tornar um único cenário mais abrangente de modo a fazer com que este englobe todos os casos. Este é um procedimento bastante interessante, pois aumenta a capacidade expressiva da base sem introduzir redundância. Observamos porém, que este procedimento foi utilizado para a inclusão de uma variante apenas para o cenário original. Não sabemos qual seria o resultado se houvesse a necessidade de aumentar este número. Possivelmente teríamos que recorrer a criação de novos cenários, pois o original ficaria demasiado grande para abarcar todas as possibilidades.

A operação de *encapsulamento* é caracterizada pela generalização de dois ou mais cenários em um cenário que é capaz de lidar com variantes do processo inicial. O cenário resultante herda alguns dos relacionamentos que o cenário inicial possuía. A rigor, qualquer cenário pode estar envolvido em uma operação de *encapsulamento*. Na Figura 4.14 mostramos um exemplo da operação de *consolidação* do projeto VII – Sistema de Gerência acadêmica. Note que o cenário original tratava da inscrição de alunos de doutorado e foi generalizado de modo a tratar da inscrição de modo geral. Na seção anterior mostramos um exemplo onde haviam dois cenários equivalentes na

base que foram encapsulados em um único. Neste exemplo apenas contamos com o cenário inscrever-se em disciplina de doutorado na base (a contrapartida para alunos de mestrado ou graduação não existe nesta versão do estudo de caso). Os autores, ao invés de criar outros cenários equivalentes optaram por consolidar todos os procedimentos diretamente a partir do cenário existente. Na realidade, a idéia atrás das duas operações é a mesma, variações se devem a própria configuração da base de cenários, i.e., existência de cenários que possam ser encapsulados ou um único cenário que deve ser consolidado.

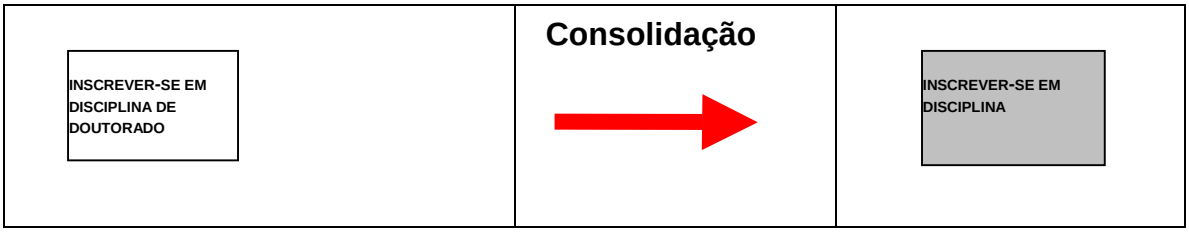


Figura 4.14 – Exemplo da operação de consolidação *(projeto VII)*

4.4.2.2 OPERAÇÕES DE EXPANSÃO

Nesta seção apresentaremos as operações de divisão, múltipla divisão, extensão e especialização. O objetivo geral de todas estas operações é a pulverização do conteúdo de um cenário em dois ou mais cenários. A diversidade entre elas é explicada através dos relacionamentos prévios existentes entre os cenários envolvidos e na redundância da informação (conteúdo) nos cenários resultantes.

4.4.2.2.1 DIVISÃO

Divisão é uma operação que visa a separação do conteúdo de um único cenário em dois ou mais cenário separados. Observamos durante o curso do estudo de caso que a necessidade da aplicação desta operação surgia a partir de duas situações distintas. A primeira é o refinamento do conteúdo dos cenários a medida em que se dá o desenvolvimento do software. Uma vez atingida maior compreensão sobre o problema, notamos a tendência da divisão de grandes blocos de informação em pedaços menores de modo a facilitar o manuseio dos mesmos. Esta é uma estratégia que tem sido utilizada nas ciências exatas a bastante tempo com sucesso [Polya45]. A

segunda razão é atender a certas decisões de projeto que ocasionalmente impõem uma divisão artificial na organização dos cenários na base. Esta estratégia, apesar de bastante utilizada durante os estudos de caso é pouco recomendável, pois introduz dificuldade na leitura dos cenários. Voltamos a lembrar que um dos aspectos positivos da utilização de cenários durante o desenvolvimento de software é promover uma notação de mais fácil compreensão por parte de clientes e usuários e, acreditamos, que a introdução deste tipo de artifício pode ter um efeito negativo neste processo.

A operação de *divisão* é caracterizada pela pulverização do conteúdo de um cenário em dois ou mais cenários distintos sem a preservação do cenário original. Os cenário resultantes da operação devem ser complementares, o que significa que podem ter coincidência de conteúdo. Se houver indicação de sequência de ações deve-se também estabelecer relacionamentos de pré condição entre os novos cenários. A rigor qualquer cenário pode sofrer a operação de *divisão,* desde de que sejam preservados os relacionamentos que este possuía anteriormente. Especial atenção tem de ser dada aos relacionamentos de complemento, contenção, possível precedência e pré condição pois como esta operação resulta em dois ou mais cenários é fundamental verificar para qual (ou quais) dos novos cenários os relacionamentos permanecem válidos. Na Figura 4.15 mostramos um exemplo da operação divisão do projeto II – Sistema de biblioteca.

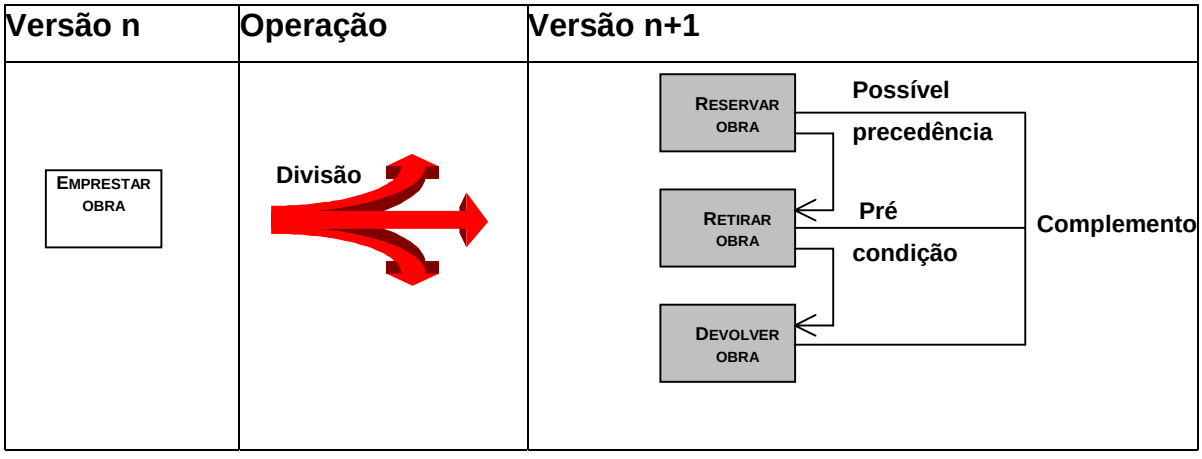


Figura 4.15 – Exemplo da operação de divisão (projeto II)

Note que o cenário original cedeu lugar para três outros cenários conectados através de um relacionamento de complemento. Estes cenários também apresentam os

relacionamentos de possível precedência e pré condição, indicando que havia algum indicativo de temporalidade no cenário original que foi levado em conta na aplicação da operação.

4.4.2.2.2 MÚLTIPLA DIVISÃO

A operação de múltipla divisão tem como objetivo básico isolar um tipo de comportamento comum partilhado por um certo número de cenários em um único cenário com o intuito de diminuir redundância. Durante o curso dos estudos de caso notamos que é bastante comum, especialmente quando tratamos de cenários de desenho, a presença de procedimentos básicos comuns a vários cenários. Estes procedimentos, de modo geral, dizem respeito as tarefas de manuseio de um sistema já existente, como por exemplo a ativação do próprio e a navegação entre telas. Notamos que, em alguns projetos que este comportamento foi isolado em um cenário distinto que por sua vez era referenciado por outros cenários. Esta solução é bastante interessante pois facilita a manutenção da base de cenários uma vez que reduz grandemente a redundância da informação. Do ponto de vista da compreensão dos cenários esta também é uma boa estratégia, na medida em que reduz e seleciona os episódios dos cenários. De modo geral os episódios correspondentes ao procedimento isolado são generalizados e dizem respeito atividades quase que clericais para a inicialização do sistema. Isolando estes procedimentos em um cenário distinto, os cenários que continham estes episódios ficarão reduzidos aos episódios que estão diretamente ligados ao seus objetivos básicos, facilitando sua compreensão.

A operação de *múltipla divisão* é caracterizada pelo isolamento de parte do conteúdo comum a vários cenários em um cenários distinto e pela preservação do restante do conteúdo dos cenários originais. Não é necessário que os cenário iniciais sejam conectados através de qualquer relacionamento. Individualmente cada um dos cenários deve manter o relacionamento de inclusão com o cenário resultante. Qualquer conjunto de dois ou mais cenários que partilhem de um mesmo procedimento, descrito através de um ou mais episódios pode recorrer a operação de *múltipla divisão* de modo a isolar este procedimento em um cenário separado. Deve-se porém ter cuidado para que os relacionamentos que os cenários envolvidos na operação possuíam anteriormente sejam preservados. Em especial devemos verificar

os relacionamentos de pré condição e possível precedência e verificar se estes pertencem aos cenário iniciais ou se estão relacionados ao novo cenário. O novo cenário somente poderá herdar um relacionamento se este for compartilhado por todos os cenários originais envolvidos na operação. Na Figura 4.16 mostramos um exemplo da operação de *múltipla divisão* do projeto III – Sistema de gerência para uma gráfica. Note que o procedimento de logar no sistema e selecionar opção foi isolado em um novo cenário incorporado nos cenário originais, preservados após a operação, através do relacionamento de *inclusão*.

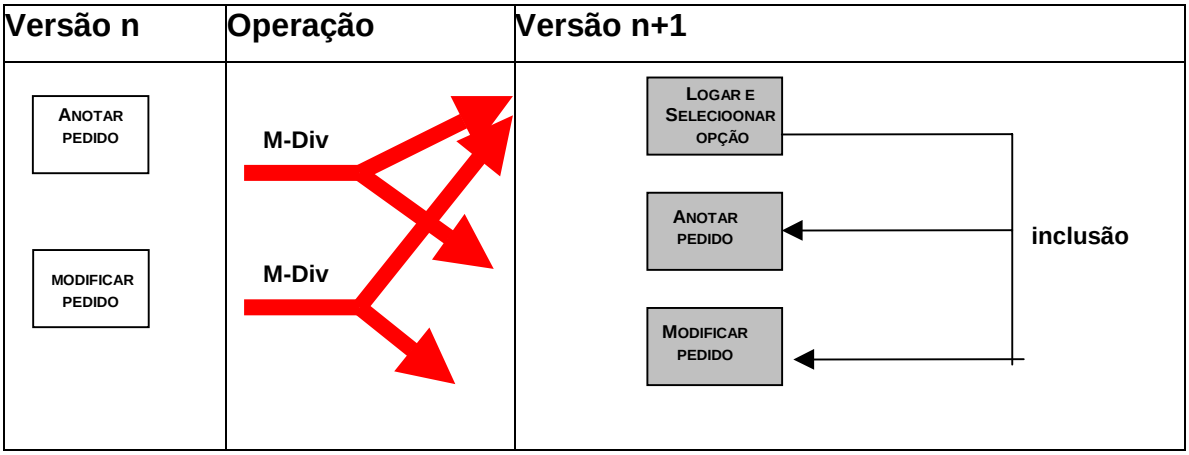


Figura 4.16 – Exemplo da operação de múltipla divisão (projeto III)

4.4.2.3 ESPECIALIZAÇÃO

A operação de *especialização* objetiva incorporar cursos alternativos para um único cenário genérico. O cenário original deve conter a abstração de um procedimento que pode tomar vários formatos dependendo de variações em seus componentes, e.g., atores ou contexto. A idéia da operação é tornar mais clara diversas possibilidades. Podemos traçar um paralelo com a definição de Booch para use cases e cenários. Use cases tratam de procedimentos gerais enquanto que cenários são as possíveis instanciações dos use cases [Booch99]. No nosso caso a notação utilizada não possibilita esta diferenciação e conjeturamos que talvez esta construção seja uma maneira encontrada pelos usuários de dar conta desta dificuldade. Como resultado da aplicação da operação de *especialização* temos no mesmo plano o cenário genérico original e suas instâncias. Acreditamos que esta operação tende a facilitar a compreensão pois reúne uma visão geral ao mesmo tempo em que apresenta

detalhamento sobre possíveis variações. Se por um lado esta operação facilita a entendimento, ela também aumenta a redundância na base, uma vez que as instâncias tem muitos componentes semelhantes. Da mesma forma encontramos algum grau de semelhança, ainda que menor do que no caso anterior, entre o cenário original e suas instâncias. Sob este ponto de vista a manutenção fica bastante prejudicada.

Novamente lembramos que estas observações são fruto de resultados de estudos de caso realizados por terceiros. Nosso papel durante a elaboração deste levantamento foi passivo e não nos cabe tecer hipóteses sobre as escolhas de cada um dos autores dos estudos de caso. Notamos porém, que a escolha das operações não é trivial e os impactos na base, tanto do ponto de vista de manutenção quanto da compreensão por parte dos clientes, tem de ser cuidadosamente levados em conta. Nosso objetivo é transpor estas observações para o próximo capítulo, onde teceremos comentários sobre a elaboração de uma estratégia de gerência da configuração de cenários que deve levar em conta estes aspectos.

Podemos caracterizar a operação de *especialização* pela instanciação de um cenário que contém um curso de ação genérico em novos cenários ao passo em que o cenário original é mantido. Os novos cenários serão conectados entre si através de relacionamentos de equivalência, resultado da grande coincidência de componentes que deve existir entre os mesmos. Da mesma forma o cenário original deverá se conectar com as instâncias através do relacionamento de contenção, uma vez que cada instância representa obrigatoriamente parte do cenário pai. Qualquer cenário pode sofrer a operação de *especialização* desde de que mantenha as restrições relatadas acima. Todos os relacionamentos que o cenário original possuía continuam a existir e são herdados pelas instâncias. Na Figura 4.17 mostramos um exemplo da operação de *especialização* do projeto I – Sistema de gerência acadêmica . Neste exemplo o procedimento de aceitar aluno foi especializado em aceitar aluno de doutorado e de mestrado. Note que o cenário original mantém o relacionamento de contenção com os dois novos cenário ao passo em que os últimos mantém o relacionamento de equivalência.

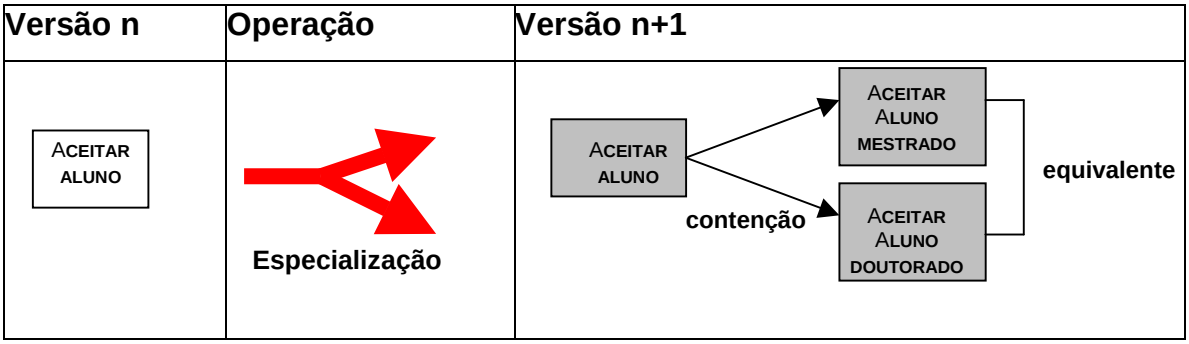


Figura 4.17 – Exemplo da operação de especialização *(projeto I)*

4.4.2.2.4 EXTENSÃO

A operação de *extensão* é um caso especial da operação de *especialização* descrita na seção anterior. Ao invés de criar uma hierarquia de abstração para os cenários, i.e., com um cenário genérico que guarda características gerais e suas instâncias, esta operação permite a inclusão de variantes para um cenário através da criação de um cenário equivalente. Neste caso estendemos a funcionalidade de um cenário através da mera criação de outro que compartilha uma grande quantidade de episódios do original. A menos da coincidência de componentes estes cenário não compartilham mais nada com o original e, desta forma, não podemos assegurar a validade de nenhum dos relacionamentos existentes com o cenário original. Em outras palavras, esta operação equivale a realização de uma operação de *especialização* de modo desordenado. Ao invés da criação de um cenário mais abstrato que captura a essência de um procedimento e organiza as possíveis instâncias, esta operação permite com que se crie variantes através da inserção de cenários semelhantes apenas. Do ponto de vista da redundância esta operação tem as mesmas desvantagens apresentadas pela operação de *especialização* sem a organização da última. Notamos, porém, que em certos casos esta operação foi utilizada com sucesso para explicitar exceções. Neste caso não estamos tratando de uma série de instâncias aleatoriamente criadas e sim de casos únicos onde não existe sentido na criação de um procedimento genérico. Este é o caso que tratamos no exemplo ilustrado pela figura 4.18. O cenário fazer disciplina já é o mais genérico possível não há sentido em criar uma instância fazer disciplina como aluno excepcional, já que este procedimento é raro e constitui uma exceção.

Caracterizamos a operação de *extensão* pela criação de um cenário que estende a funcionalidade de outro existente. Este cenário está conectado ao cenário original através de um relacionamento de equivalência e nada se pode afirmar em relação aos relacionamentos do cenário original em relação ao novo. Desta forma devemos verificar se algum destes são válidos para o cenário estendido. Em particular devemos verificar os relacionamentos de possível precedência e de pré condição em razão da questão temporal que estes relacionamentos incorporam. Qualquer cenário pode ter sua funcionalidade estendida. Na Figura 4.18 mostramos um exemplo da operação de *especialização* do projeto I – Sistema de gerência acadêmica .

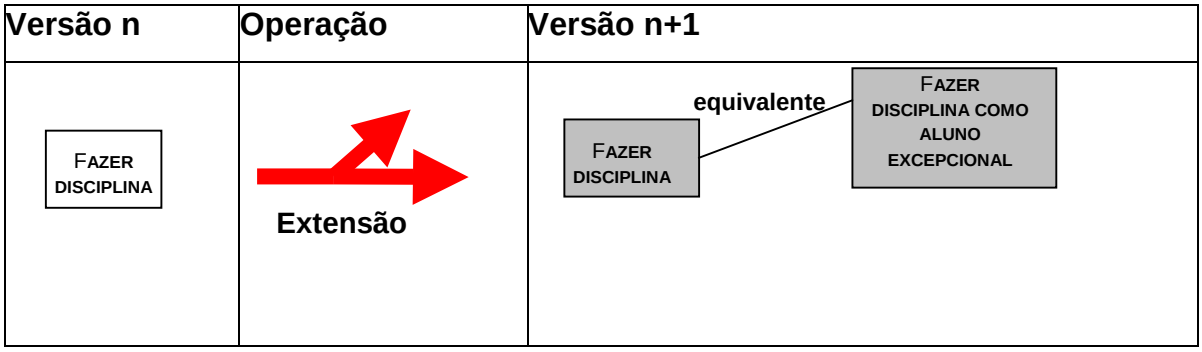


Figura 4.18 – Exemplo da operação de extensão *(projeto I)*

Neste exemplo o procedimento de aceitar aluno foi especializado em aceitar aluno de doutorado e de mestrado. Note que o cenário original mantém o relacionamento de contenção com os dois novos cenário ao passo em que os últimos mantém o relacionamento de equivalência.

4.4.2.3 OPERAÇÕES QUE MODIFICAM O NÚMERO DE CENÁRIOS DA BASE EM UMA UNIDADE

Nesta seção apresentaremos as operações de exclusão e de adição de novos cenários a base. Como mencionamos anteriormente a elicitação de requisitos para um sistema de software é um processo evolutivo onde não é rara a descoberta, modificação ou reformulação da informação. O processo de software baseado em cenários não é exceção e, portanto, devemos esperar que novos cenários apareçam e outros sejam eliminados da base como fato inerente ao próprio processo evolutivo do sistema. Nesta seção apresentaremos e exemplificamos ambas operações.

4.4.2.3.1 EXCLUSÃO

A operação de *exclusão* visa a retirada de informação que se tornou obsoleta na base de cenários. As razões para a retirada podem vir do aumento do entendimento do problema, erro na modelagem dos cenários ou até mesmo como resultado de fatores externos ao processo de desenvolvimento de software. Durante o curso dos estudos de caso percebemos que esta operação ocorre pouco frequentemente e, até onde pudemos inferir, como resultado do refinamento do conhecimento do problema.

Definimos a operação de *exclusão* como a simples retirada de um cenário da base. Todo cenário a rigor pode ser retirado se observados algumas restrições de integridade. Devemos observar todos os relacionamentos que o cenário a ser retirado mantém com o restante da base de modo a verificar qual o impacto desta operação. Em especial se o cenário manter relacionamentos de precedência, i.e., servir de pré condição ou possível precedência para outros cenários. Se este for o caso não devemos realizar a operação pois a retirada do cenário invalida o acesso a estes cenários.

4.4.2.3.2 ADIÇÃO DE NOVO CENÁRIO

A operação de *adição* visa a introdução de nova informação na base. De modo geral esta operação acontece como resultado da descoberta de novos requisitos para o sistema ou de mudanças no Universo de Discurso. Notamos que, ao contrário da operação de *exclusão*, esta operação aconteceu com frequência durante os estudos de caso, sendo rara a versão que não contava com pelo menos um novo cenário.

Definimos a operação de *adição* como a inserção de cenários inteiramente novos na base. Definimos como novos os cenários que não resultam de nenhuma operação de expansão da base, como as descritas na seção *4.4.2*.2, mas do aumento do conhecimento do problema. Quando um cenário é inserido na base devemos verificar se este não mantém nenhum relacionamento com outro cenário já existente. Para tal devemos acionar as heurísticas de detecção de relacionamentos e reavaliar a base como um todo. Na realidade, apesar de trazer novas informações, os cenário contém

alguma coincidência de conteúdo com aqueles pré existentes. Afinal cenário não existem no vácuo, é somente razoável imaginar que novos procedimentos estejam de alguma forma ligados com os antigos.

4.5 TAXONOMIA PARA A EVOLUÇÃO DE CENÁRIOS

Nas duas seções anteriores descrevemos detalhadamente os componentes estáticos e dinâmicos que regem o processo evolutivo de cenários. Os relacionamentos e operações, respectivamente, são os elementos básicos que compõem um modelo genérico da evolução de cenários. Nas duas últimas seções apresentamos e exemplificamos cada um dos relacionamentos e operações detectados através do estudo de caso realizado. A partir destes elementos fomos capazes de montar uma taxonomia para a evolução de cenários que está ilustrada na Tabela 4.10 a seguir.

Relacionamentos	**Operações**	
Aspectos estáticos (intra configuração)	**Aspectos dinâmicos (inter configuração)**	
	intra cenário	**Inter cenários**
Complemento	Inclusão	Fusão
Equivalência	Modificação	Encapsulamento
Contenção	Retirada	Consolidação
Pré – condição		Divisão
Detour		Múltipla divisão
Exceção		Especialização
Inclusão		Extensão
Possível precedência		Exclusão
		Adição

Tabela 4.10 - Taxonomia para a evolução de cenários

De modo a confirmar os resultados apresentados na Tabela 4.10 acima realizamos um segundo estudo de caso. O objetivo principal deste estudo, conforme mencionado anteriormente, era testar a robustez do modelo através do processo de

desenvolvimento de software. Apresentamos o segundo estudo de caso realizado sua repercussão sobre a taxonomia para a evolução de cenários na próxima seção.

4.6 ESTUDO DE CASO II

Nesta seção apresentamos o segundo estudo de caso que realizamos de modo a confirmar os resultados iniciais obtidos e resumidos pela taxonomia para evolução de cenários. O sistema escolhido para ser implementado foi proposto inicialmente como estudo de caso em um workshop de engenharia de requisitos realizado no Schloss Daghstuhl, Alemanha em 1999 [Dagstuhl99]. Os anais derivados a partir deste encontro estão no processo de se tornar uma edição especial do Journal of Universal Computer Science editado pela Springer Verlag. O sistema proposto faz a simulação do controle da iluminação do Departamento de Informática da Universidade de Kaiserslautern. A completa descrição do problema pode ser encontrada em http://rn.informatik.uni-kl.de/~recs/problem.

Da maneira em que foi conduzido este estudo de caso foram elaboradas um total de 8 configurações ao longo do desenvolvimento do sistema. A Tabela 4.11 detalha o segundo estudo de caso realizado.

.

Sistema de controle de iluminação				
Nome	**Descrição**	**Cenários**	**Relacionamentos entre cenários**	**Operações aplicadas**
SpecI	Primeira configuração dos cenários da base. Derivada a partir do Léxico Ampliado da Linguagem desta aplicação através de heurísticas próprias.	23	21	23
SpecII	Segunda configuração do conjunto de cenários. Resultado da remoção de redundâncias e reorganização.	17	16	9

SpecIII	Terceira configuração do conjunto de cenários. Resultado do aumento da compreensão da descrição do problema e do refinamento das informações.	10	14	5
DesignI	Primeira configuração dos cenários de desenho. Adaptados da configuração anterior e atualizados de modo a refletir a modelagem dos cartões CRC.	10	14	10
DesignII	Segunda configuração dos cenários de desenho. Refletem o modelo orientado a objetos (OO) derivado a partir dos cartões CRC.	11	15	7
DesignIII	Última configuração dos cenários de desenho. Reflete o corte no modelo OO e representa as interações entre o sistema a ser construído e entidades externas.	10	11	7
ImpI	Primeira configuração dos cenários de implementação. Reflete os ajustes realizados no modelo OO e incorpora as classes de interface que tiveram de ser adicionadas.	7	4	5
ImpII	Esta configuração descreve os cenários do ponto de vista externo (do cliente). Representa a interação dos usuários do sistema com sua implementação.	7	5	7
Totais:		*96* [7] / 78	100	73

Tabela 4.11 - Dados do segundo estudo de caso

Através da Tabela 4.11 temos uma visão geral do segundo estudo de caso conduzido. Observamos que o número inicial de cenários tende a diminuir e se estabilizar. O

[7] No caso do número total de cenários gostaríamos de enfatizar que uma versão de um cenários pode participar de mais de uma configuração. Desta forma, o número absoluto encontrado a partir da soma do número de cenários de cada configuração traz redundâncias. Se subtrairmos as últimas chegamos a um total de 78 cenários distintos.

primeiro grande degrau pode ser explicado pelo fato do primeiro conjunto de cenários ter sido derivado do léxico ampliado da linguagem para esta aplicação através da aplicação direta das heurísticas descritas em [Leonardi97]. O resultado da aplicação destas heurísticas foi um grande número de cenários contendo informações redundantes. A seguir um processo de refinamento de informação teve lugar a medida em que foi se obtendo uma maior compreensão da descrição do problema. A partir da configuração SpecIII o número de cenários se tornou constante, evidenciando que um equilíbrio foi atingido. Na realidade o que ocorreu é que grandes mudanças, do tipo exclusão e criação de novos cenários deixaram lugar para operações de refinamento e atualização da informação, tais como as operações intra cenário de modificação e inclusão. Resumindo, as operações inter cenários, que trazem maiores impactos a base, pois resultam na adição e exclusão de cenários, deixam lugar para operações de impacto local, i.e., operações intra cenário. Na Figura 4.19 mostramos a distribuição das operações classificadas por tipo ao longo do desenvolvimento do sistema.

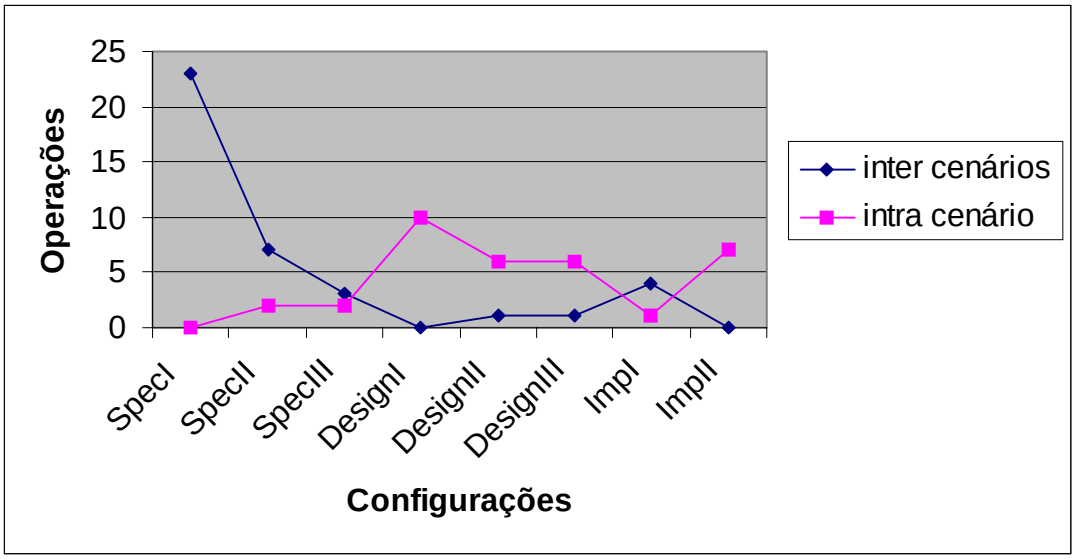


Figura 4.19 Distribuição das operações nas configurações por tipo

O número de operações aplicadas também tende a permanecer estável durante o desenvolvimento, a menos da primeira configuração. Neste caso, estamos computando apenas a operação de adição de novo cenário, uma operação para cada novo cenário adicionado a base. Aparentemente o número de operações decresce bruscamente da primeira para a segunda configuração. Na realidade, o que acontece é

que na primeira configuração a operação de adição é realizada de modo um para um, enquanto que nas demais configurações as operações podem envolver mais de um cenário, e.g. uma das nove operações que é realizada na configuração SpecII, como veremos a seguir, envolve 4 cenários na realização de uma fusão. Desta forma, se computarmos o número total de cenários envolvidos em operações por configuração teremos um número estável. Este fato ficará mais evidente a seguir, quando detalharemos a passagem da configuração SpecI para SpecII de modo a ilustrar a evolução dos cenários. No restante desta seção utilizaremos o segundo estudo de caso como veículo para ilustrar a evolução de cenários segundo vários aspectos. Na próxima sub seção mostraremos a evolução entre duas configurações consecutivas. Partindo de uma configuração e dos relacionamentos existentes entre seus cenários mostraremos todas as operações realizadas e seus resultados na configuração seguinte. Tomando como base um único cenário, na sub seção seguinte mostramos a evolução deste ao longo do processo de desenvolvimento do software. Mostraremos todas as versões pelas quais este cenário passou, bem como o novo cenário que foi efetivamente implementado, que resultou da fusão do cenário em questão com outro cenário da base. Finalmente mostraremos a evolução dos relacionamentos deste cenário em relação a outros artefatos da base, e.g., termos do léxico, cartões CRC e objetos pertencentes ao modelo de objetos do sistema.

4.6.1 EVOLUÇÃO ENTRE CONFIGURAÇÕES CONSECUTIVAS

Nesta seção mostraremos detalhadamente o processo de evolução de uma configuração para outra consecutiva. Vamos verificar o que acontece com cada um dos cenários participantes e tentar traçar um paralelo com os relacionamentos que os cenários envolvidos nas operações possuíam anteriormente. Escolhemos como exemplo a transição da configuração SpecI para SpecII pois esta conta com ambos os tipos de operação, i.e., intra e inter cenários. Conforme apontado pela Tabela 4.11 a configuração SpecI é resultado da aplicação de heurísticas para extração de cenários a partir do léxico ampliado da linguagem para o sistema de controle de iluminação. A transição para a configuração SpecII é responsável pela redução da redundância, correção e reorganização da informação contida nestes cenários.

Nesta sub seção evitaremos mostrar o conteúdo dos cenários envolvidos pois o objetivo é mostrar um apanhado geral do processo. Neste caso, o excesso de detalhes prejudicaria o entendimento global deste exemplo. No entanto, disponibilizamos os cenários na íntegra para consulta no *site* do exemplo do sistema de Iluminação[8].

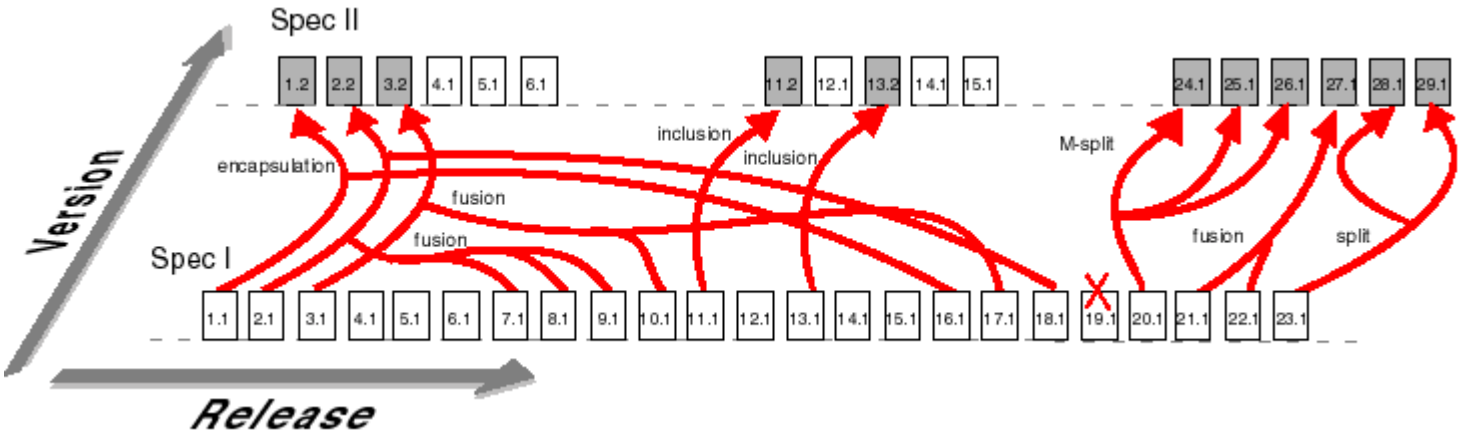


Figura 4.20 - Operações aplicadas sobre a configuração SpecI

Na figura acima os cenários pertencentes a configuração SpecII ilustrados com fundo branco representam aqueles que não sofreram modificações entre ambas configurações. Como mencionado anteriormente, os cenários não tem de ser necessariamente modificados entre uma configuração e outra, podendo permanecer constantes. Desta forma, uma mesma versão de um cenário pode pertencer a mais de uma configuração. Este é o caso dos cenários 4.1 5.1 6.1 12.1 14.1 e 15.1[9] que pertencem tanto a configuração SpecI quanto a SpecII.

Das nove operações ilustradas na Figura 4.20 duas são do tipo intra cenário e as sete restantes do tipo inter cenários. A Tabela 4.12 a seguir faz um resumo das operações.

[8] Disponível em http://stones.les.inf.puc-rio.br/Karin/exemplo/index.html

[9] A lista com os títulos dos cenários numerados está disponível como parte da Figura 4.21 a seguir.

Nome	Número	Cenários envolvidos	Número de cenários envolvidos	Cenários resultantes	Número de cenários resultantes	Tipo
Encapsulamento	I	1.1 e 16.1	2	1.2	1	Inter cenários
Fusão	II	2.1, 7.1, 8.1, 9.1 e 18.1	5	2.2	1	Inter cenários
Fusão	III	3.1, 10.1 e 17.1	3	3.2	1	Inter cenários
Inclusão	IV	11.1	1	11.2	1	Intra cenário
Inclusão	V	13.1	1	13.2	1	Intra cenário
Exclusão	VI	19.1	1	-	0	Inter cenários
Múltipla Divisão	VII	20.1	1	24.1, 25.1 e 26.1	3	Inter cenários
Fusão	VIII	21.1 e 22.1	2	27.1	1	Inter cenários
Divisão	IX	23.1	1	28.1 e 29.1	2	Inter cenários

Tabela 4.12 - Relação das operações realizadas sobre os cenários da configuração SpecI

É importante notar a variedade de operações disponíveis na Tabela 4.12; temos desde operações que não alteram o número de cenários da base (intra cenário) passando por operações que contraem este número e que resultam no aumento do montante total de cenários. Vários cenários desapareceram na transição das configurações como resultado da aplicação das operações de contração, i.e., neste caso as operações de número I, II, III, VI e VIII resultaram no desaparecimento dos cenários de número 7.1, 8.1, 9.1, 10.1, 16.1, 17.1, 18.1, 19.1, 20.1, 21.1, 22.1 e 23.1. Na realidade, apenas os cenários sob estes números desapareceram, as informações contidas nos mesmos foram realocadas ou para cenários já existentes, e.g., no caso da operações I onde o cenário 16.1 foi incorporado na segunda versão do cenário 1 (1.2) ou para novos cenários, e.g., operação VII onde o cenário 20.1 desapareceu para dar lugar aos novos cenários 24.1, 25.1 e 26.1. De modo semelhante a última, outras operações também resultaram na criação de novos cenários que não existiam na configuração inicial. É o caso das operações de fusão VIII que apesar de resultar no desaparecimento dos cenários 21.1 e 22.1 deu margem ao aparecimento do novo cenário 27.1 e da operação

IX, que resultou no aparecimento dos novos cenários 28.1 e 29.1. No caso da última, o desaparecimento do cenário inicial da operação de Divisão foi uma decisão de desenho.

Mostramos a seguir o gráfico de relacionamentos existentes entre os cenários da configuração SpecI. Todos os relacionamentos foram detectados através das heurísticas propostas na seção 4.3 deste capítulo. Tentaremos traçar um paralelo entre a detecção dos relacionamentos e do envolvimento de cenários nas operações aplicadas. Segundo a Tabela 4.12 as operações que envolvem dois ou mais cenário são as de número I, II, III e VIII. Passamos a analisar cada uma destas em detalhes.

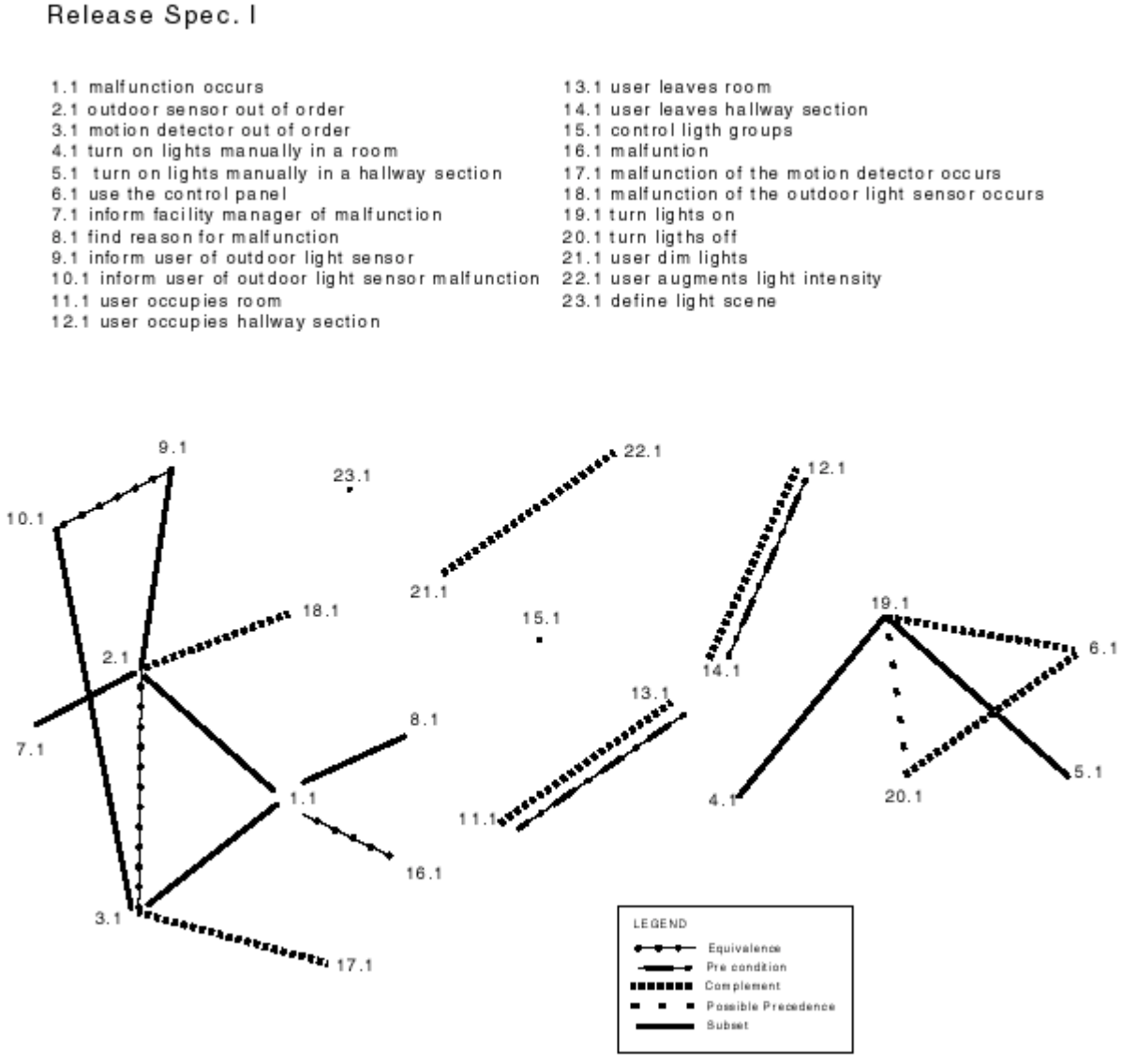
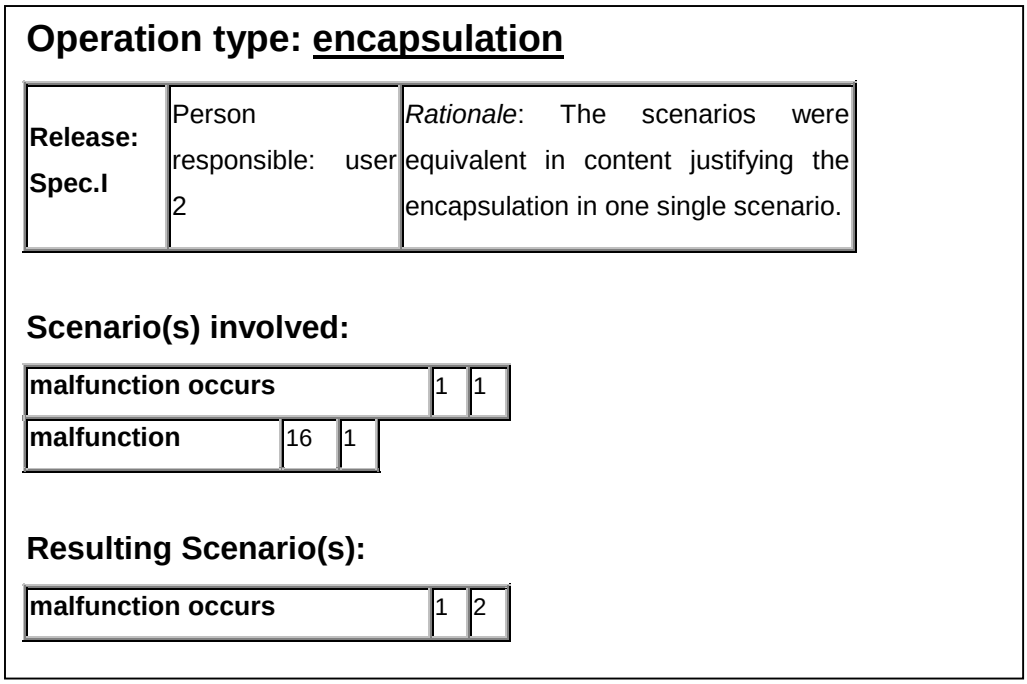


Figura 4.21 - Relacionamentos entre os cenários da configuração SpecI

A operação de número I, encapsulamento, envolve dois cenários equivalentes. Estes relacionamento, segundo a seção 4.3.2 foi observado entre cenários que compartilham um mesmo objetivo ao mesmo tempo que situados em um mesmo contexto. Também foi notado coincidência de atores e de alguns episódios. No caso destes cenários em particular a operação de encapsulamento foi realizada talvez com o intuito de diminuir a redundância da base. Segundo as justificativas para a operação, mostradas na Figura 4.22 a seguir, o fato dos cenários possuírem conteúdo equivalente já é suficiente para a realização da operação.



Operation type: encapsulation

Release: Spec.I	Person responsible: user 2	*Rationale*: The scenarios were equivalent in content justifying the encapsulation in one single scenario.

Scenario(s) involved:

malfunction occurs	1	1
malfunction	16	1

Resulting Scenario(s):

malfunction occurs	1	2

Figura 4.22 Justificativas para a operação de encapsulamento entre os cenários 1.1 e16.1

A operação de número III, por sua vez, funde os cenários complementares, 17.1 e 3.1 e o subcenário 10.1 do último. Segundo a seção *4.4.2*.1.1 o objetivo da operação é unir cenários que apresentam procedimentos logicamente conectados. Este é o caso, pois estamos unindo cenários que tratam da ocorrência de um defeito, constatação do mesmo e da necessidade de advertir o responsável do ocorrido, (cenários 17.1, 3.1 e 10.1 respectivamente). O caso da operação VIII é similar, pois estamos fundindo dois cenários que apresentam situações complementares, diminuir e aumentar a quantidade de luz em um cômodo. Estes cenários serão reunidos sob um novo cenário que se chamará “variar intensidade da luz em um cômodo” na configuração SpecII. Na

Figura 4.23 mostramos os relacionamentos existentes entre os cenários da configuração SpecII, resultantes da evolução da configuração SpecI que estamos estudando.

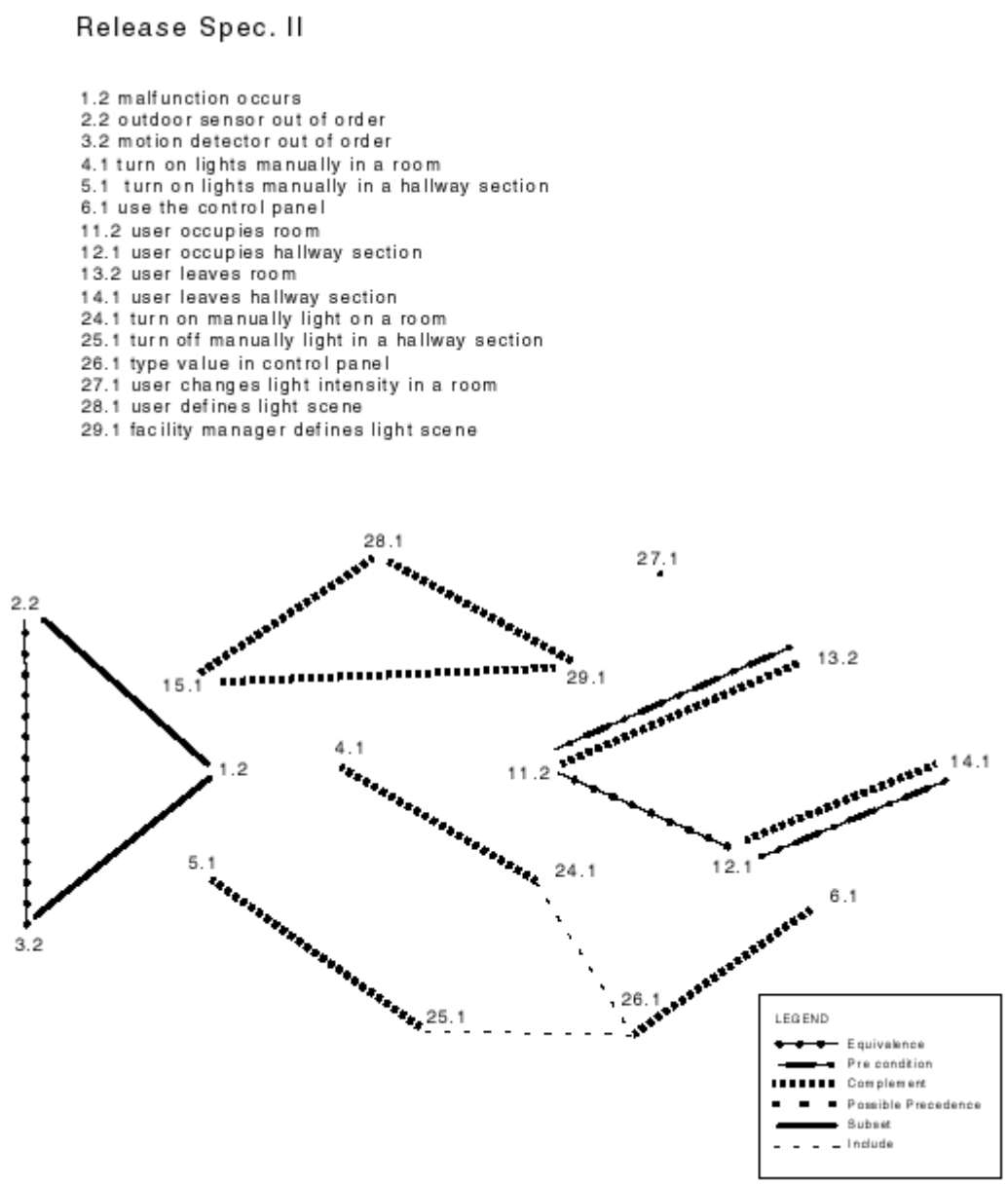


Figura 4.23- Relacionamentos entre os cenários da configuração SpecII

Note que as novas versões dos cenários mantiveram relacionamentos originais, e.g., o cenário 2.2 é equivalente ao 3.2 e subcenário de 1.2. Como resultado de operações que expandem a base, tais como a divisão rotulada com o número IX e a múltipla divisão rotulada VII na Tabela 4.12 temos a criação de novos relacionamentos. No

caso da primeira, que resultou na criação dos cenários 28.1 e 29.1 observamos que estes mantém um relacionamento de complementariedade entre si. No caso da última os novos cenários 24.1 e 25.1 ambos mantém o relacionamento de inclusão com o cenário 26.1 porém nenhum relacionamento entre si. Finalmente temos a remoção de relacionamentos como consequência do processo evolutivo. Este é o caso dos cenários 21.1 e 22.1 da configuração SpecI que mantinham um relacionamento de complementariedade entre si e, que ao serem fundidos forçaram o desaparecimento deste relacionamento na configuração seguinte. Note que o cenário resultante, 27.1 não mantém relacionamento com nenhum outro cenário desta configuração. Se observarmos a Tabela 4.11, que contém os dados relativos a este segundo estudo de caso, podemos observar que existe um equilíbrio no número de relacionamentos de modo geral, apontando para a possível existência de mecanismos intrínsecos de auto compensação.

Finalmente, a operação de número II envolve cinco cenários que estão relacionados de formas bastante diversas. Note que os cenários 9.1 e 10.1 são equivalentes, os cenários 2.1 e 18.1 complementares e os cenários 9.1 mantém um relacionamento de contenção como o cenário 2.1. Finalmente o cenário 8.1 que também faz parte da operação não está diretamente ligado a nenhum dos outros cenários envolvidos diretamente. Na realidade ele é um subconjunto do cenário 1.1 que, por sua vez, é subconjunto do cenário 2.1 que faz parte da fusão. Em casos como este fica evidente a complexidade associada ao processo de evolução de cenários. Em situações menos elaboradas, como a descrita no parágrafo anterior ou nas seções *4.4.2*.1.1 e 4.4.1.2 relativas as operações de fusão e encapsulamento respectivamente, mostramos exemplos menos complexos onde tanto a indicação quanto a justificativa para a aplicação das operações fica clara. Nestes casos uma vez detectado um relacionamento de complementariedade entre cenários a fusão é indicada enquanto que se o relacionamento for de equivalência a operação de encapsulamento é a recomendada. O simples fato da existência dos relacionamentos, que assegura uma quantidade de coincidência de conteúdo entre os cenários envolvidos, já serve como justificativa para a aplicação das operações. Face a este exemplo, estas prescrições se tornam quase que ingênuas, pois nenhuma poderia empregada satisfatoriamente neste caso. Fica claro a dificuldade de prescrever uma estratégia global para a aplicação de operações. Por outro lado, situações como esta não são comuns. Na maioria dos casos

observamos que a aplicação de regras simples, que registramos ao longo da confecção de ambos estudos de caso, serviam como guia para a escolha de operações. Neste espírito apresentamos um pequeno compêndio, na forma de heurísticas para aplicação de operações sobre cenários, para auxiliar na escolha e aplicação de operações. Levamos em conta pré-condições e os impactos que a operação terá em relacionamentos anteriores e no cenário original. A Tabela 4.13 resume estas heurísticas organizadas por operação.

Nome da Operação	Impacto nos relacionamentos anteriores de:	Preserva cenário original	Pré condições	Cardinalidade
Intra Cenários				
Inclusão	Nenhum	Sim	nenhuma	1 → 1
Modificação	Nenhum	Sim	nenhuma	1 → 1
Retirada	Nenhum	Sim	nenhuma	1 → 1
Inter Cenários				
Fusão	Complemento Equivalência Contenção	Não	Cenários originais tem de ser complementares Se houver relacionamentos de precedência transportar para cenário resultante	N → 1
Encapsulamento	Pré-condição Possível precedência	Não	Cenários originais tem de ser equivalentes Se houver relacionamentos de precedência verificar a validade para o cenário resultante	N → 1
Consolidação	herdados pelo cenário resultante	Não	nenhuma	1 → 1
Divisão	Complemento Contenção Possível precedência Pré condição	Não	Cenários tem de ser complementares (se necessário) estabelecer precedência	1 → N
Múltipla Divisão	Possível precedência Pré condição	Sim	Cenários mantém relacionamento de inclusão com novo cenário individualmente	N → N + 1
Especialização	nenhum	Sim	Cenário original mantém relacionamento de contenção com novos cenários	1 → N

Nome da Operação	Impacto nos relacionamentos anteriores de:	Preserva cenário original	Pré condições	Cardinalidade
			Novos cenários mantém relacionamento de equivalência entre si	
Extensão	Possível precedência Pré condição	Sim	Verificar relacionamentos do cenário original para determinar validade em relação ao novo cenário	1 ➔ N
Exclusão	todos	Não	Não deve ser realizado se o cenário original possuir relacionamentos do tipo pré condição ou possível precedência	1 ➔ 0
Adição	nenhum	Não se aplica	a base tem de ser reavaliada de modo a determinar relacionamentos do novo cenário com os anteriores	0 ➔ 1

Tabela 4.13 Heurísticas para aplicação de operações

Na próxima sub seção vamos explorar um outro aspecto da evolução. Ao invés de tomarmos uma configuração como ponto de partida vamos mostrar a evolução de um cenário ao longo de todas as configurações. Tomaremos um exemplo em particular e mostraremos todo o processo de versionamento e operações sofridas pelo cenário.

4.6.2 EVOLUÇÃO DE UM CENÁRIO EM PARTICULAR

Nesta sub seção deslocaremos o eixo de pesquisa e vamos mostrar a evolução do ponto de vista de um cenário específico. Levaremos em conta todas as configurações previstas e o processo de versionamento que este cenário sofreu ao longo das mesmas. O cenário escolhido foi *user occupies room* (usuário ocupa cômodo) pois sofre um processo de versionamento ao longo de algumas configurações para depois ser envolvido em uma fusão com outro cenário. A Figura 4.24 ilustra a evolução deste cenário ao longo do desenvolvimento. Note que algumas versões do cenário aparecem em mais de uma configuração, e.g., a versão 3 aparece nas configurações DesignI e DesignII. Mostraremos a seguir detalhes da evolução do mesmo.

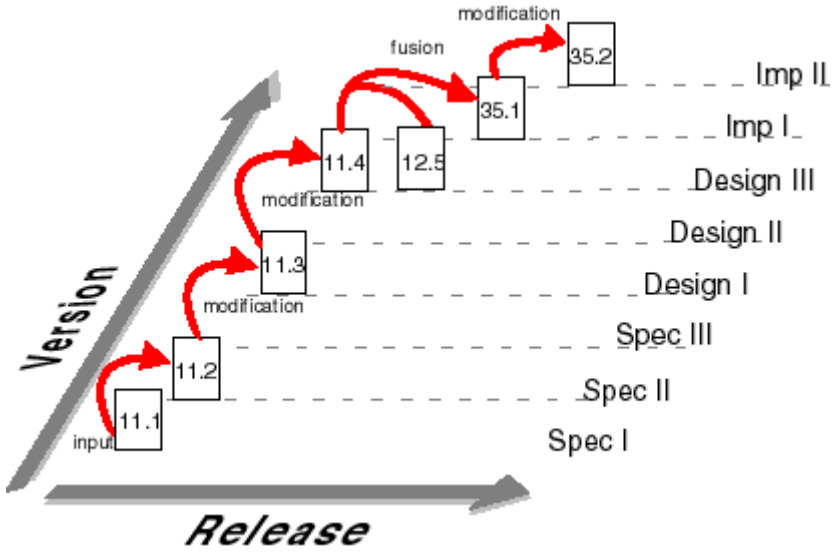


Figura 4.24 - Evolução do cenário 11 (user occupies room) ao longo de todas as configurações

O histórico deste cenário, bem como dos restantes, está disponibilizado através da implementação do *site* do exemplo utilizando-se o protótipo da ferramenta SET. A Figura 4.^^ a seguir mostra o resultado da consulta de histórico para este cenário em particular. Note que além das informações de criação do cenário, a ferramenta também mostra o *rationale* para a criação do cenário, se este estiver disponível.

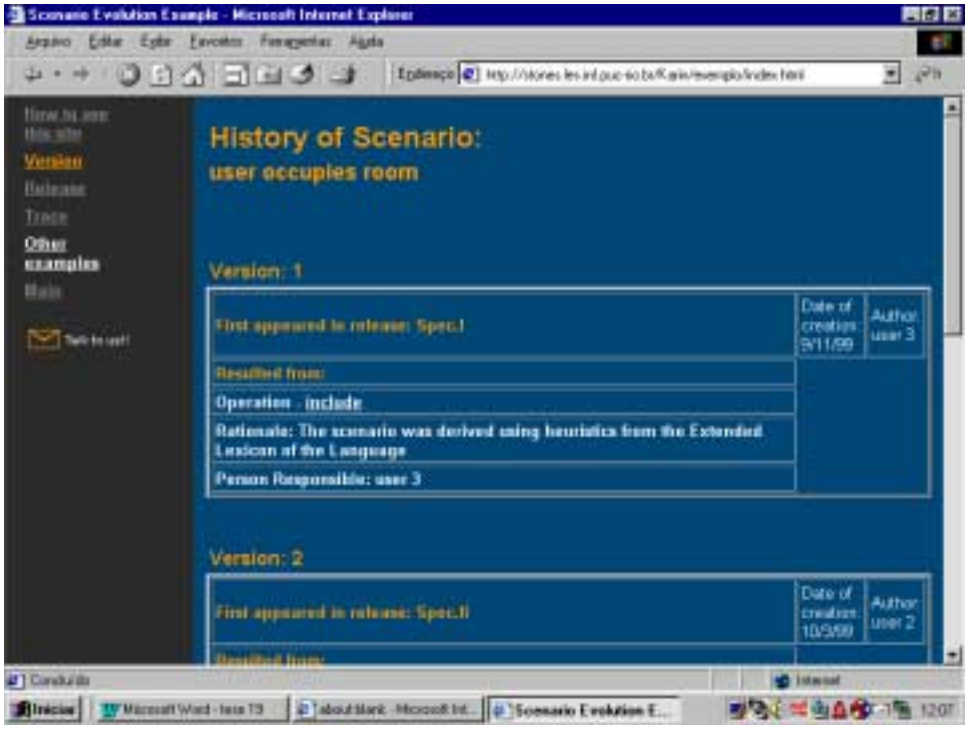


Figura 4.25- histórico do cenário user occupies room

Nas Figura 4.26 e 4.27 a seguir, mostramos o conteúdo das duas primeiras versões deste cenário. Estas figuras também foram extraídas do *site* que disponibiliza este estudo de caso. Note que são disponibilizadas informações relativas tanto a criação da versão mas também quanto as configurações em que a versão específica do cenário participa. Este é o caso da segunda versão do cenário que participa de duas configurações. As outras versões deste cenário, bem como as do cenário *person occupies place* resultante da fusão com o cenário user *occupies hallway section* que ocorreu durante a configuração Design III, vide Figura 4.20 , estão disponíveis no *site*.

Scenario: user occupies room

Version 1

First appeared in release: Spec.I	Date of creation: 9/11/99	Author: user 3
Appears in Release(s):	**Spec.I**	

Resulted from:

Operation: include	*Rationale*: The scenario was derived using heuristics from the Extended Lexicon of the Language	Person Responsible: user 3

Contents:

user occupies room	Goal: establish the procedure for occupied room Context: 4th floor of building 32 , motion detector in order, user entered room Resource: value T1 Default light scene for this room, Chosen light scene value Actors: user, Control system Episodes: 1. user enters room 2. user chooses light scene 3. IF room is reoccupied wihtin T1 minutes THEN activate last Chosen light scene 4.IF room is reoccupied after T1 minutes THEN activate Default light scene

Figura 4.26 - informações relativas a primeira versão do cenário user occupies room obtidas no site do estudo de caso.

Scenario: user occupies room
Version 2

First appeared in release: Spec.II	Date of creation: 10/3/99	Author: user 2
Appears in Release(s):	**Spec.II**	**Spec.III**

Resulted from:

Operation: input	*Rationale*: We noticed some of the information in the scenario was incorrect. We made proper corrections	Person Responsible: user 4

Contents:

user occupies room	Goal: establish the procedure for occupied room Context: 4th floor of building 32 , motion detector in order, user entered room Resource: value T1, outdoor light sensor last correct management value, Default light scene for this room, Chosen light scene value Actors: user, Control system Episodes: 1. user enters room 2. System keeps last correct measurement of outdoor light sensor 3. user chooses light scene 4. IF room is reoccupied wihtin T1 minutes THEN activate last Chosen light scene 5. IF room is reoccupied after **T2** minutes THEN activate Default light scene

Figura 4.27 - informações relativas a segunda versão do cenário user occupies room obtidas no site do estudo de caso.

Se compararmos ambos cenários notamos que realmente não só o conteúdo da segunda versão foi acrescido de informações mas também foi corrigido. Note que na primeira versão os episódios 3 e 4 trazem uma contradição pois sugerem que depois de decorrido um tempo igual ao valor de T1 tanto o parâmetro *Default* quanto o *Chosen light scenes* seriam ativados. A segunda versão corrige este erro e estabelece valores diferentes para a ativação dos diferentes parâmetros. As expressões que aparecem sublinhadas correspondem a elos para termos do Léxico Ampliado da Linguagem (LAL). Na próxima seção mostraremos a terceira e última dimensão da evolução dos cenários, o rastreamento com outros artefatos presentes na baseline de requisitos.

4.6.3 EVOLUÇÃO DE CENÁRIOS EM RELAÇÃO A OUTROS ARTEFATOS

Ao longo do desenvolvimento de software são produzidos uma série de artefatos que tem como objetivo documentar o processo e registrar os produtos gerados a cada um de seus estágios [Pressman92]. No capítulo 3 introduzimos a baseline de requisitos, uma estrutura concebida para capturar e gerenciar a evolução destes artefatos. Neste contexto a evolução de cenários é apenas uma faceta. Vários outros processos paralelos se desdobram ao longo do desenvolvimento de software. Entre outros, temos a evolução do Léxico Ampliado da Linguagem que tem de ser atualizado de modo a refletir mudanças no desenvolvimento do software.

No contexto da evolução é fundamental que possamos rastrear a informação contida em um cenário não só em relação aos restantes da base, mas também em relação a outros artefatos de software produzidos. No restante desta seção nos dedicamos a rastreabilidade dos cenários. A Figura 4.28 a seguir mostra a faceta *trace* da evolução de cenários e a distinção que é feita entre a rastreabilidade entre os próprios cenários e outros artefatos de software.

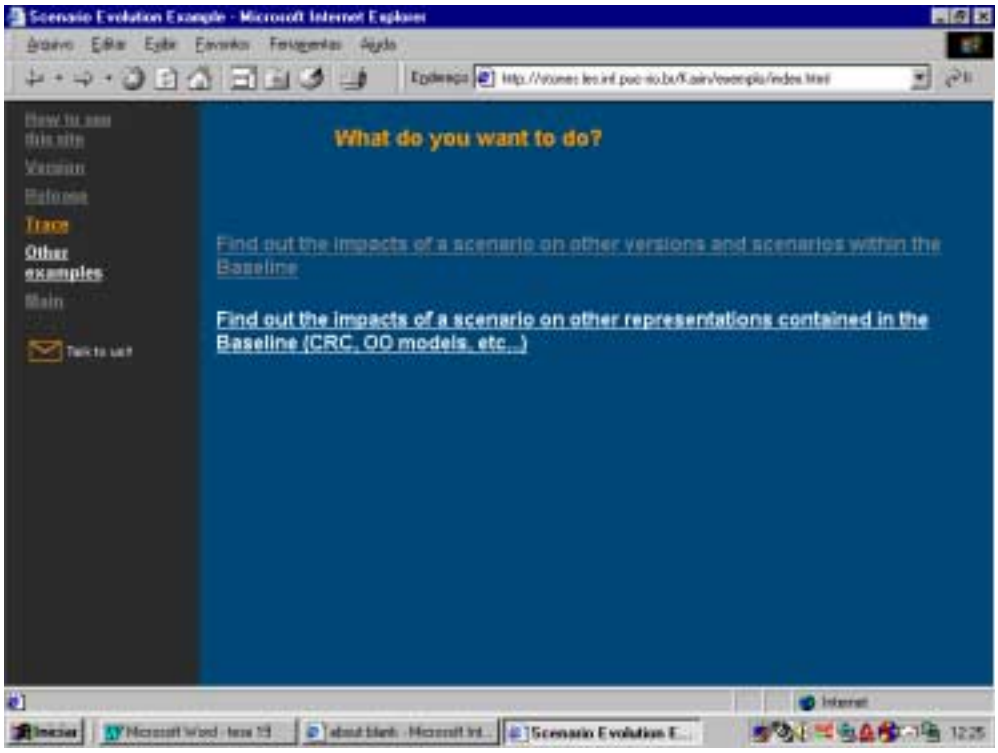


Figura 4.28 - Rastreabilidade em relação a cenários da base e outros artefatos de software

A seguir mostramos um exemplo mais pontual, a rastreabilidade da terceira versão do cenário *user occupies room*, que aparece como resultado da operação de modificação durante a configuração DesignI. Enfatizamos ligação deste cenários com outros artefatos de software, neste caso com cartões CRC [Whirfs-Brock90].

Scenario: user occupies room
Version 3

First appeared in release: Design.I	Date of creation: 11/23/99

Can be traced to:

Artefact: CRC cards
Detail: The functionality of this scenario is distributed in the following cards: control panel, user and room

Scenario Contents:
Goal : establish the procedure for occupied room Context: 4th floor of building 32 , motion detector in order, user entered room Resource: value T1, outdoor light sensor last correct management value, Default light scene for this room, Chosen light scene value, Actors: user, Central Control, outdoor light sensor , Control panel, room Episodes: 1. user enters room 2. In case of malfunction , outdoor light sensor sends last correct measurement to room 3. user chooses light scene using the Control panel Exception: user input is not reasonable 4. IF room is reoccupied wihtin T1 minutes THEN activate last Chosen light scene stored in room 5. IF room is reoccupied after T2 minutes THEN activate Default light scene stored in room

Figura 4.29 - Exemplo de um cenário e sua rastreabilidade em relação a outros artefatos

Repare que este cenário está ligado ao artefato CRC cards, em particular aos cartões *control panel*, *user* e *room*. Mostramos estes cartões na Figura 4.30 a seguir. O episódio 2 do cenário, enviar último valor correto para o cômodo no caso de problemas com o sensor está coberto pelo cartão *room*. A colaboração entre os cartões *control panel* e *user* é responsável pelo episódio e os dois últimos episódios são cobertos pelo par de cartões *user* e *room*.

User	
Defines chosen light scene	room
Enters or leaves room or hallway section	control panel
set the value T1 of room	hallway section
set the default light scene of room	motion detector
set each ceiling light group of room	outdoor light sensor

Room	
keep the outdoor light sensor measurements	facility manager
store value T1	user
IF room is reoccupied wihtin T1 minutes THEN activate last chosen light scene	control panel
IF room is reoccupied after T1 minutes THEN activate default light scene	
store chosen light scene	
store moment when person left the room	

Control Panel	
IF user input is not reasonable THEN system issues a warning	user facility manager

Figura 4.30 - Cartões CRC diretamente relacionados a versão 3 do cenário 11

Note na Figura 4.29 que alguns dos termos que fazem parte do conteúdo do cenário estão sublinhados. Esta é indicação que o termo ou expressão faz parte do Léxico Ampliado da Linguagem(LAL). Na Figura 4.31 que apresentamos a seguir, mostramos a entrada do LAL equivalente ao termo *control panel* que aparece tanto como ator quanto no episódio 3 da terceira versão do cenário 11 ilustrado na Figura 4.29.

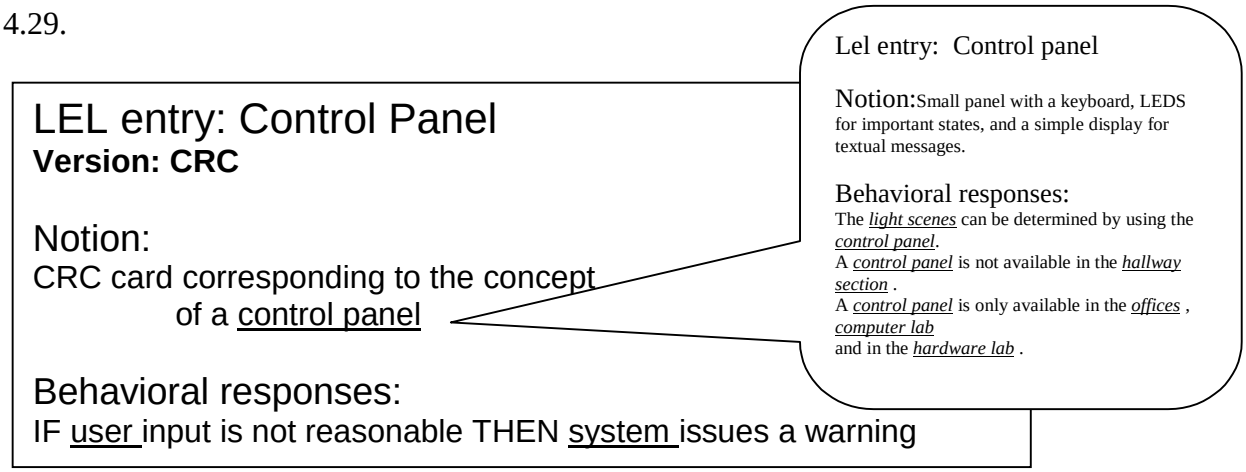


Figura 4.31 -Entrada do LAL correspondente ao termo control panel que aparece no cenário 11.3

Como mencionado anteriormente o Léxico está em constante evolução da mesma forma que a base de cenários. Desta forma a definição dos termos também sofre modificações ao longo do processo de desenvolvimento. De modo a ilustrar este fato mostramos na Figura 4.32 abaixo a evolução das entradas para o termo *room* na versão 1, 3 e depois da fusão com o cenário user *occupies hallway section* que resultou na primeira versão do cenário *person occupies place* na configuração ImpI, vide Figura 4.24. Note que a evolução dos termos vai desde a representação do espaço físico e real de cômodo para a implementação (em Java, no caso) da classe que representa este conceito no sistema.

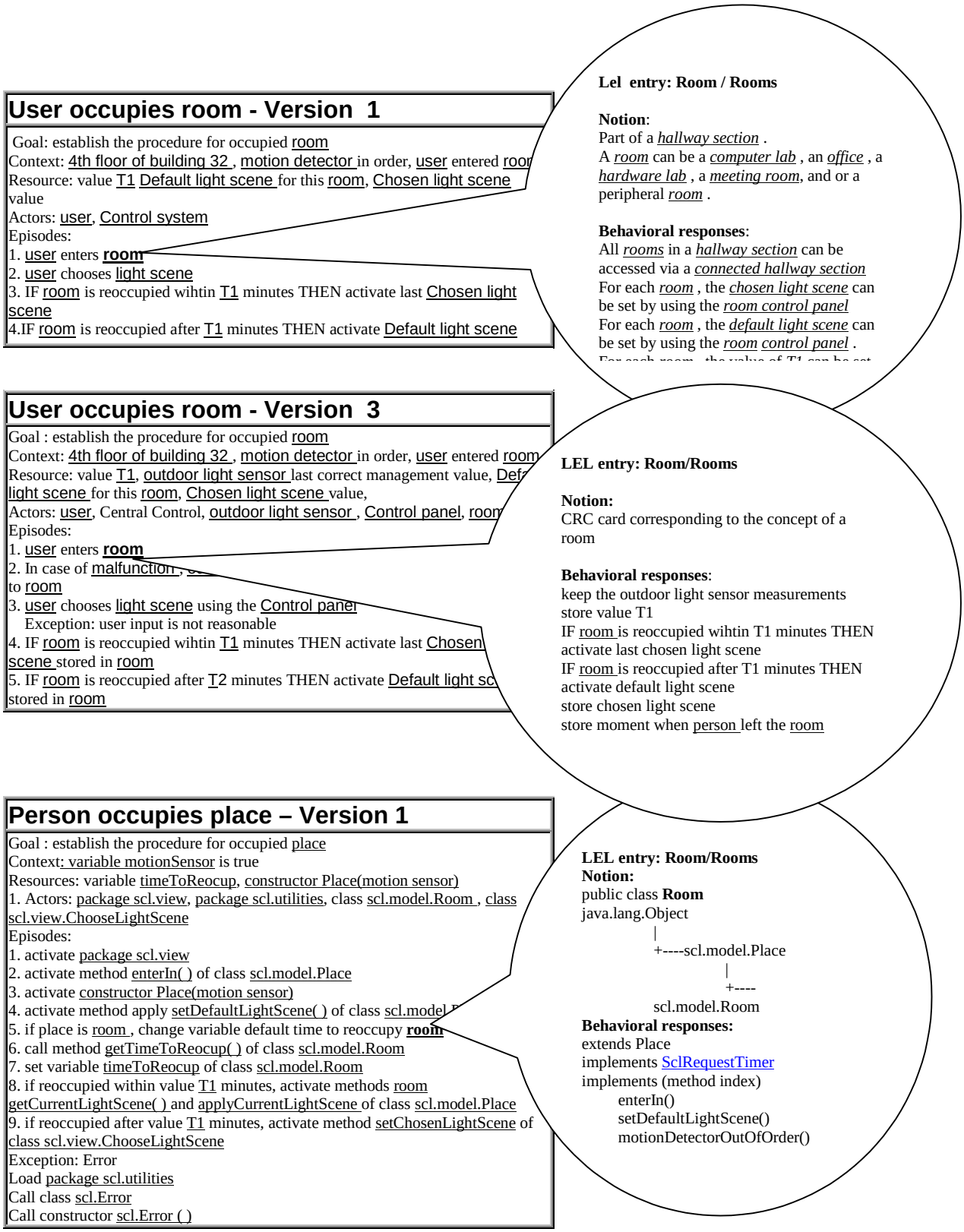


Figura 4.32 Correspondência entre a evolução dos cenários e dos termos do LAL

A estratégia utilizada para a construção do sistema foi do tipo de "dentro para fora" (*outside - inside*), o que significa que a partir da descrição do macrosistema fomos refinando a especificação até atingirmos a implementação satisfatória para o software. Neste contexto existe um refinamento dos conceitos, evidenciado na Figura 4.32

apresentada anteriormente. Mostramos que o conceito geral de cômodo foi sendo elaborado desde local físico até uma abstração que representa o conceito para o sistema. A figura mostra apenas parte desta evolução. Como visto na Figura 4.24 o cenário em questão possui 4 versões sob este nome e outras duas a partir da fusão com o cenário *user occupies hallway section* na configuração DesignIII. O *site* do estudo de caso disponibiliza todas as versões do cenário bem como os elos para os termos do léxico e outros artefatos.

Nesta seção apresentamos o segundo estudo de caso que utilizamos como forma de validação e teste para a taxonomia para evolução de cenários apresentada na seção 4.5. A partir da experiência que agregamos ao realizar este segundo estudo de caso, fomos capazes de refinar esta taxonomia e incluir aspectos que vão além do processo de evolução de cenários, representado pelo conjunto de operações. Entendemos que aspectos estáticos, relacionados aos cenários enquanto produtos e representados pelos relacionamentos existentes entre os cenários de cada configuração estão intimamente ligados ao processo de evolução dos últimos e, portanto, não podem estar dissociados.

Resumimos estes resultados no modelo de evolução de cenários que apresentaremos na próxima seção.

4.7 MODELO DE EVOLUÇÃO DE CENÁRIOS

Na seção anterior tentamos mostrar, a partir de um segundo estudo de caso, a complexidade do processo evolutivo de cenários. Nossa proposta, como ficou claro, é de que cenários devem ser utilizados no apoio a todas as fases do desenvolvimento de software. De modo a apoiar a utilização de cenários neste enfoque foi necessário compreender a fundo como se dá o processo evolutivo dos mesmos. A partir das observações resultantes do segundo estudo de caso apresentado na seção anterior, fomos capazes de não só refinar a taxonomia para evolução de cenários, mas também incorporar ao informações que auxiliem na compreensão da mecânica do processo evolutivo como um todo.

Apresentamos nossos resultados sob a forma de um modelo de evolução de cenários na Figura 4.33. Note que o modelo é dividido em três partições, processo, produto e instância. Ao nível de processo estão as operações que irão transformar a base de cenários. No nível de produto encontramos os esquemas para a representação de conhecimento dos cenários, como por exemplo as notações propostas por [Sutcliffe95, Rosson95, Jacobson92, Potts94, Leite95, Kyng95]. É a partir destas representações, i.e., na realidade a partir do detalhamento de informação contido na representação escolhida, que são determinadas as instâncias dos cenários e seus relacionamentos.

Resumimos nosso entendimento do processo evolutivo de cenários como se segue: ao nível de processo a evolução de cenários é explicada através de um conjunto de operações que surtem efeito sobre um único ou um grupo de cenários, e que como resultado modificam a configuração anterior. Estas operações são aplicadas sobre cenários que foram codificados utilizando-se uma única notação[10] determinada ao nível de produto. Através das instâncias destes cenários, que contém a informação particular a cada caso, vamos inferir a rede de relacionamentos e dependências existente entre os cenários de uma mesma configuração. Estes relacionamentos, por sua vez, determinam a escolha das operações que poderão ser aplicadas sobre a base e resultarão em uma nova configuração.

É importante voltar a frisar que a riqueza de tipos de relacionamentos está intimamente relacionada com o nível de detalhamento existente na notações utilizada para capturar os cenários. Como vimos no capítulo 2 deste trabalho, existe uma gama de variações para a descrição de cenários utilizando-se linguagem natural. Na Figura 2.7 mostramos uma abstração dos tipos de componentes usualmente presentes nas notações existentes na literatura. É razoável supor que notações que utilizam um maior número de componentes, codificam mais informações e, portanto permitem a inferência de um número maior de relacionamentos.

A partir da rede de relacionamentos e dependências entre cenários de uma mesma versão que poderemos definir a pertinência ou validade da aplicação de uma

[10] Uma das hipóteses básicas na realização dos estudos de caso foi a utilização de uma notação homogênea por projeto.

operação. É, por exemplo, através da constatação da existência de um relacionamento de pré condição entre os cenários A e B, A é pré condição de B, que decidimos que A não deve ser retirado da base com o risco de impossibilitar B.

Na realidade devemos compreender o processo de evolução de cenários como uma reação em cadeia entre os elementos presentes no modelo de evolução. Neste enfoque temos que os relacionamentos existentes ao nível de instância, que por sua vez são determinados a partir da estrutura de componentes da notação escolhida ao nível de produto, acabam por determinar a aplicabilidade das operações de cenários ao nível processual. A evolução de cenários é, portanto um processo dinâmico, calcado na configuração dos relacionamentos entre instâncias de cenários, que, por sua vez, são limitados pela notação utilizada ao nível de produto.

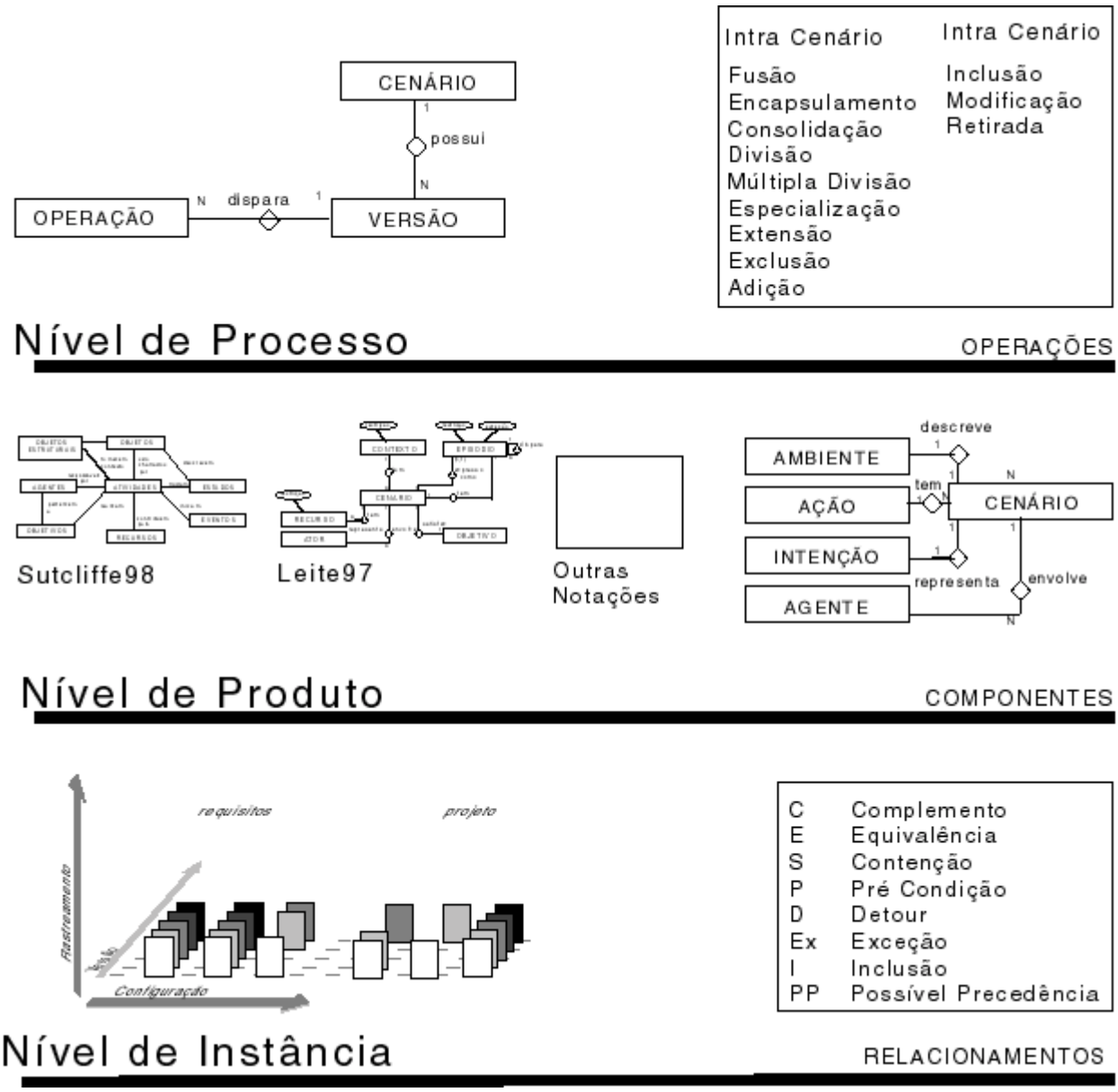


Figura 4.33 - Modelo de Evolução de Cenários

4.8 RESUMO

Neste capítulo mostramos os estudos de casos que utilizamos para refinar nosso conhecimento da evolução de cenários. Apresentamos os resultados de modo condensado no Modelo de Evolução de Cenários. Neste modelo, a informação está organizada em três níveis, instância, processo e produto. Estes correspondem, respectivamente aos relacionamentos, as operações e a notação utilizada para a representação dos cenários. O modelo serve para explicar o processo de evolução ao mesmo tempo em que provê os operadores básicos para coordenar este processo.

No próximo capítulo, baseado no modelo de evolução de cenários apresentado e levando em consideração aspectos gerais de Engenharia de Software, propomos um *Framework* para a gerência da configuração de cenários.

Capítulo 5 – *Framework* Proposto

Resultados da nossa prática têm apontado uma crescente necessidade de ferramental para apoio à criação e evolução de especificações baseadas em cenários. Existem na literatura várias propostas de ferramentas que apoiam a criação dos mesmos [Zorman95, Maiden98, Antonelli98, BenAchour99]. Todas ferramentas oferecem, em variados graus, apoio as atividades de entrada de dados, criação de conexões (ou elos) entre cenários e facilitam a verificação da consistência do conjunto. Estas atividades são fundamentais na criação da primeira versão do conjunto de cenários de uma aplicação, porém, à medida em que esta evolui, outras necessidades emergem. Entendemos como evolução o processo de transformação sofrido por uma especificação ao longo da vida útil da aplicação que esta corresponde. Desta forma, uma especificação evolui durante as fases de desenvolvimento e manutenção da aplicação. Nenhuma das ferramentas citadas acima oferece apoio satisfatório à evolução das especificações baseadas em cenários.

Nos capítulos anteriores, em especial na seção 4.6 através de exemplos do segundo estudo de caso, mostramos que a evolução é um processo complexo e que sua gerência está longe de ser trivial. Talvez a maior dificuldade resida no fato do processo evolutivo ser intermitente, fazendo com que sempre haja mudanças na base que podem ter impacto tanto em estágios futuros quanto em outros já encerrados.

O modelo para evolução de cenários, apresentado no capítulo anterior, auxilia na coordenação da evolução uma vez que estabelece uma estratégia definida para a evolução dos cenários. Segundo o modelo, um cenário sofre versionamento como resultado da aplicação de operações. Uma vez que o desenvolvedor opte por uma operação, está na verdade escolhendo um curso de ações que tem pré condições e impactos bem definidos. Neste enfoque, a sequência de operações utilizada durante o processo evolutivo tem uma semântica associada que auxilia na própria descrição deste processo.

Infelizmente, para realizar uma gerência eficaz do processo somente as informações intrínsecas as operações não fornecem dados suficientes. Idealmente deveríamos

manter outros registros, relativos a fatores externos que influenciam o processo propriamente dito. Devemos registrar informações tais como local, data, hora e o autor de mudanças bem como razões ou justificativas para as mesmas. Devemos manter o conjunto de cenários sempre atualizado ao mesmo tempo em que devemos ser capazes de recuperar versões anteriores. Em resumo, o processo evolutivo deve levar em conta outros aspectos além daqueles diretamente relacionados a evolução dos cenários propriamente ditos.

Baseados no conhecimento representado através do modelo de evolução apresentado no capítulo anterior e na discussão dos requisitos para o suporte automatizado à evolução de cenários que apresentaremos a seguir, iremos propor um *Framework* para o apoio a gerência da evolução de cenários. Uma implementação para o *Framework* será apresentada através do protótipo da ferramenta SET (Scenario Evolution Tool) que introduziremos ao final deste capítulo.

O restante deste capítulo está dividido como segue. Apresentamos uma breve discussão dos requisitos para o suporte a evolução de cenários. Logo depois introduzimos o *Framework* para a evolução de cenários. Nesta seção detalhamos os componentes do *Framework* e a estratégia para sua configuração. A seguir apresentamos a ferramenta SET que oferece apoio automatizado ao *Framework*. Neste capítulo também utilizaremos o exemplo do sistema para a simulação do controle da iluminação do Departamento de Informática da Universidade de Kaiserslautern para ilustrar tanto o processo de instanciação do *Framework*, quanto a aplicação da ferramenta SET.

5.1 REQUISITOS PARA O SUPORTE AUTOMATIZADO A EVOLUÇÃO DE CENÁRIOS

O desenvolvimento de um sistema implica na compreensão do macrosistema em que este será futuramente inserido. Para tal é necessário modelar uma série de aspectos que levem em conta os atores, entidades e ações envolvidas. O macrosistema como um todo não é uma entidade estática, se modifica ao longo do desenvolvimento. A documentação do sistema deve ser capaz de refletir estas mudanças. Em capítulos

anteriores discutimos aspectos evolutivos dos cenários de uma aplicação. Vários aspectos, tais como a manutenção da consistência entre diferentes configurações, estão envolvidos tornando mais complexa a rastreabilidade da informação. Nesta seção apresentaremos uma discussão sobre a possibilidade de fornecer apoio automatizado à evolução de cenários.

Nossa experiência no desenvolvimento de sistemas utilizando cenários apontou diversas áreas que poderiam se beneficiar de apoio automatizado. Uma característica básica do enfoque de cenários é a geração de um grande volume de informação, o que dificulta a realização de consultas específicas e a manutenção da consistência entre cenários ao longo do processo evolutivo [Rolland98]. Outro problema é a visualização da dinâmica entre cenários. A partir de um único cenário várias derivações são possíveis, levando-se em conta o contexto em que este se encontra e suas possíveis exceções. Por último, a própria estrutura dos cenários sugere uma leitura não-linear, recheada de referências a outros documentos, o que dificulta sua apresentação [Tufte97]. Estas observações estão resumidas na Tabela 5.1, a seguir.

Versão	**Configuração**	**Rastreamento**
▪ Manutenção e pesquisas na base de dados ▪ Mecanismos de visão ▪ Listagens e relatórios	▪ Manutenção da consistência entre versões ▪ Manutenção da consistência entre documentos ▪ Registro de tomada de decisões	▪ Permitir leitura não-sequencial de documentos ▪ Permitir a incorporação de outros documentos ▪ Mapas

Tabela 5.1 – Requisitos para a apoio automatizado à gerência da evolução de cenários [Breitman98]

Nas três próximas subseções detalhamos estes requisitos.

5.1.1 VERSÃO

Cenários basicamente descrevem situações do cotidiano dos clientes que, com frequência, podem ter mais de uma consequência. Deste contexto surge uma questão

de ordem prática, qual seria a maneira mais adequada de se organizar um conjunto de cenários de modo a facilitar sua validação? Devemos levar em conta que cada cenário pode admitir várias exceções que por sua vez conduzem a outros cenários. Este fato impede que os cenários sejam organizados sequencialmente. Outra dificuldade é garantir que todas as possibilidades foram exploradas. Conforme colocado anteriormente cada cenário pode se subdividir em uma série de ramificações conduzindo a novos cenários e assim sucessivamente. Chamamos este fato de efeito cascata. O efeito cascata dificulta grandemente a leitura dos cenários pois leva a uma explosão de possibilidades em curto espaço de tempo.

Uma forma de combater o efeito cascata na leitura de cenários seria adotar mecanismos de visão parcial similares aos utilizados por bancos de dados. Para adotar este enfoque é necessário contemplar o conjunto cenários (e documentação associada) como elementos de um grande repositório de dados. Este repositório tipicamente conteria os cenários da base e os artefatos relacionados ao mesmo. Como mostramos na seção 4.6.3 deste trabalho os cenários estão relacionados a uma série de outros objetos produzidos durante o desenvolvimento de software. Entradas do léxico ampliado da linguagem, o documento de especificação de requisitos e modelos de objeto são apenas alguns exemplos.

Desta forma cada visão da base de dados seria um corte longitudinal que englobaria um subconjunto destes elementos. Um exemplo seria verificar todas as versões que um único cenário sofreu durante sua vida útil. Na Figura 5.1, a seguir ilustramos um exemplo de um corte deste tipo. Este exemplo faz parte do segundo estudo de caso conduzido e foi implementado utilizando-se a ferramenta SET, que será apresentada no final deste capítulo. Note que além da listagem de todas as versões que o cenário *malfunction occurs* sofreu ao longo de sua vida útil, a ferramenta também disponibiliza a informação relativa a configuração em que cada uma das versões apareceu pela primeira vez, mais uma vez lembrando que uma versão pode aparecer em mais de uma configuração.

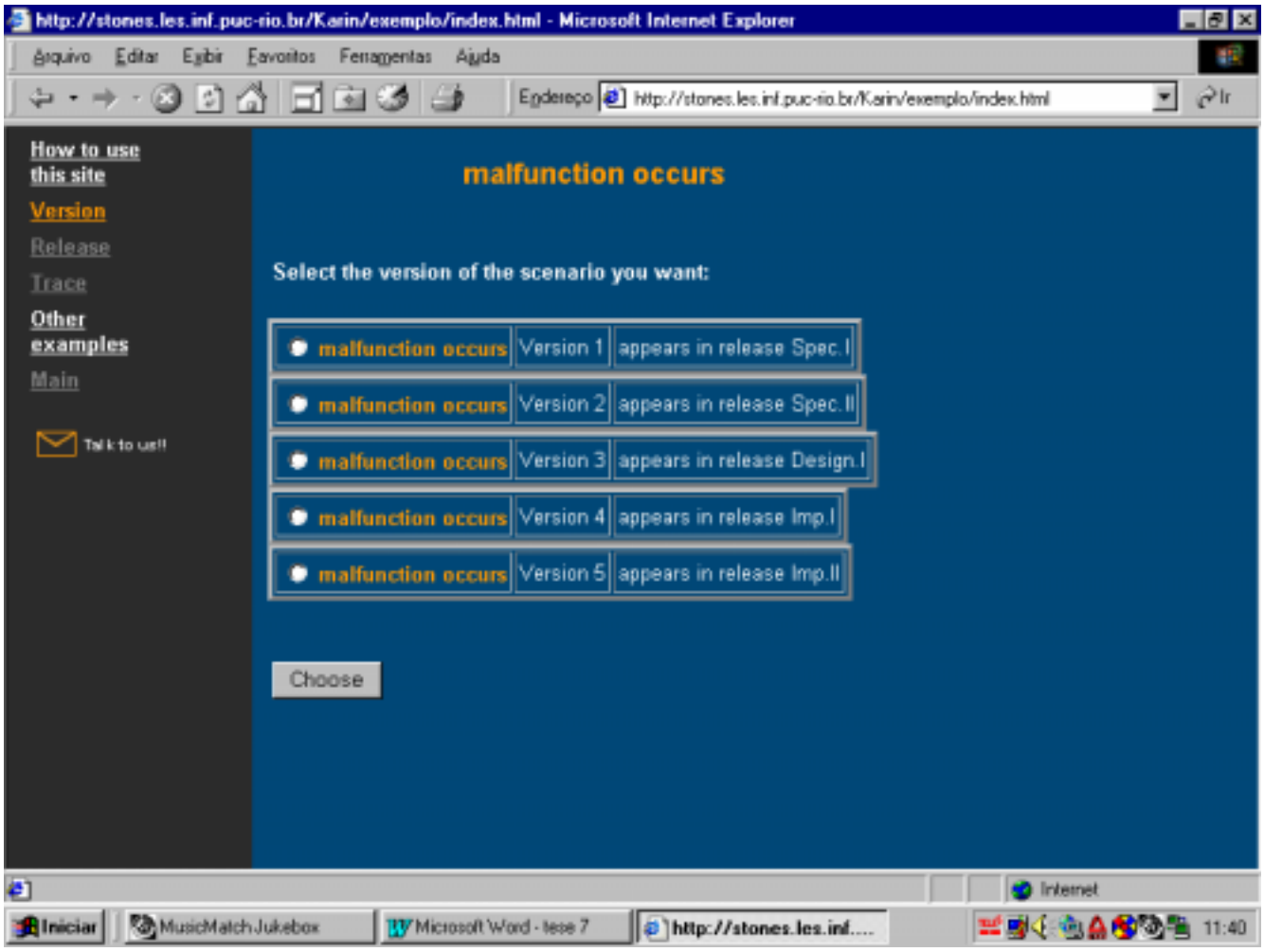


Figura 5.1 – Todas as versões para o cenário malfunction occurs

Uma outra possibilidade de visualização, é realizar a investigação da rastreabilidade da informação de um cenários em particular. Uma vez escolhido o cenário e a versão que se deseja investigar, podemos observar as conexões que este mantém com outros artefatos de software. Tipicamente teremos elos com entradas para o léxico ampliado da linguagem e, dependendo da fase de desenvolvimento em que o cenários foi criado, elos para artefatos correspondentes ao estágio. Mostramos um exemplo deste tipo de corte na base de cenários na Figura 5.2 a seguir. Esta figura mostra a versão 4 do cenário *malfunction occurs* e sua ligação com o modelo de objetos do sistema. Note que a ferramenta SET disponibiliza informações relativas a criação da versão do cenário e um pequeno detalhamento acerca dos elementos do modelo de objetos em que o cenário teria maior ligação. A rastreabilidade com termos do léxico é

implementada diretamente através de elos de hipertexto que são representados por termos sublinhados no conteúdo do cenário.

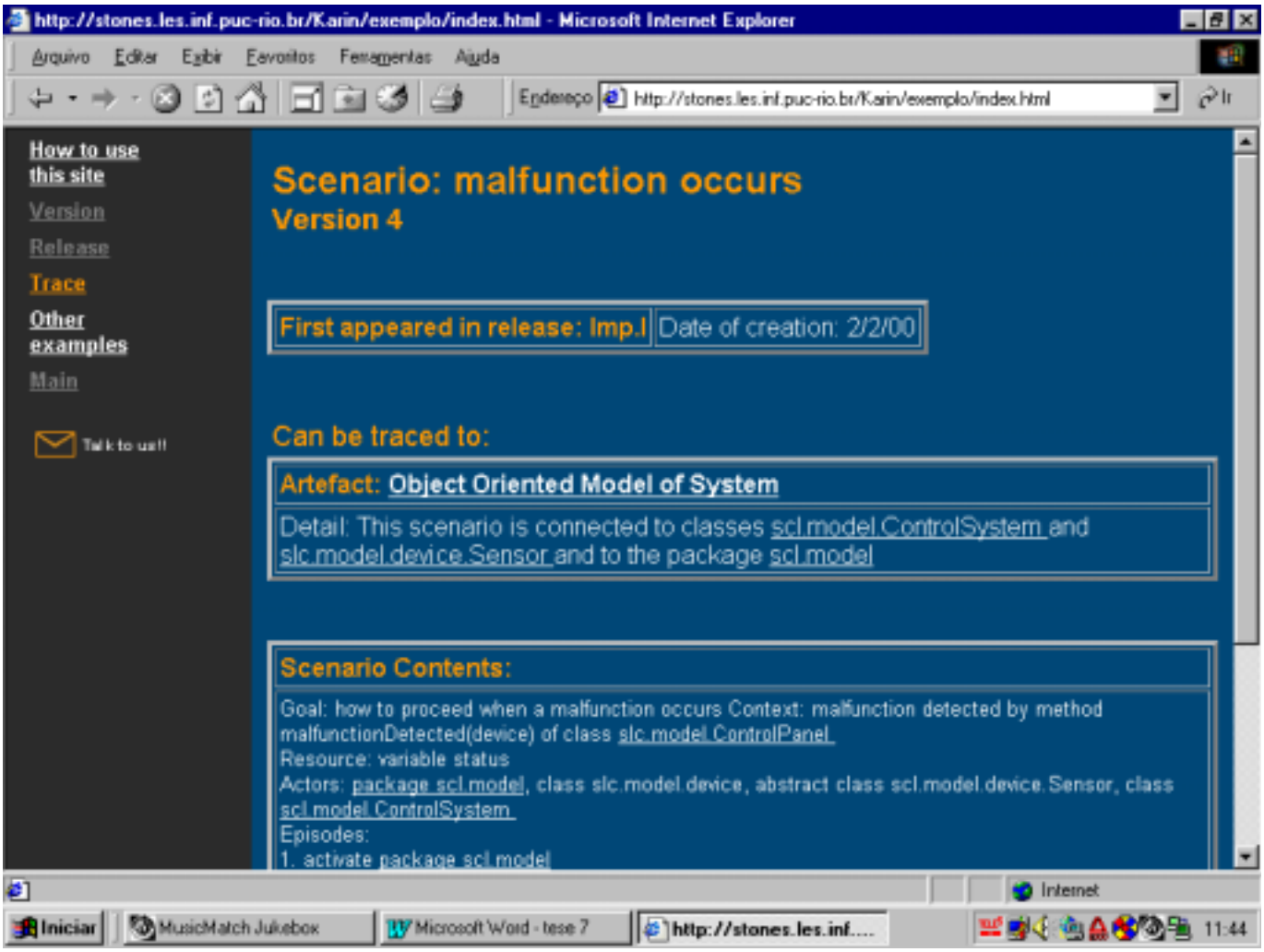


Figura 5.2 – Versão 4 do cenário malfunction occurs e seus relacionamentos com outros artefatos de software (representados através de elos de hipertexto)

Outras possibilidades de visualização seria verificar a rastreabilidade de um cenário em particular em relação a outros da base. Neste enfoque teríamos todos os cenários que estariam diretamente conectados com o cenário em questão, ou através de relacionamentos, no caso de cenários pertencentes a mesma configuração, ou através de operações, neste caso envolvendo outros cenários ao longo da vida útil do cenário. Ilustramos o último caso na Figura 5.3 a seguir.

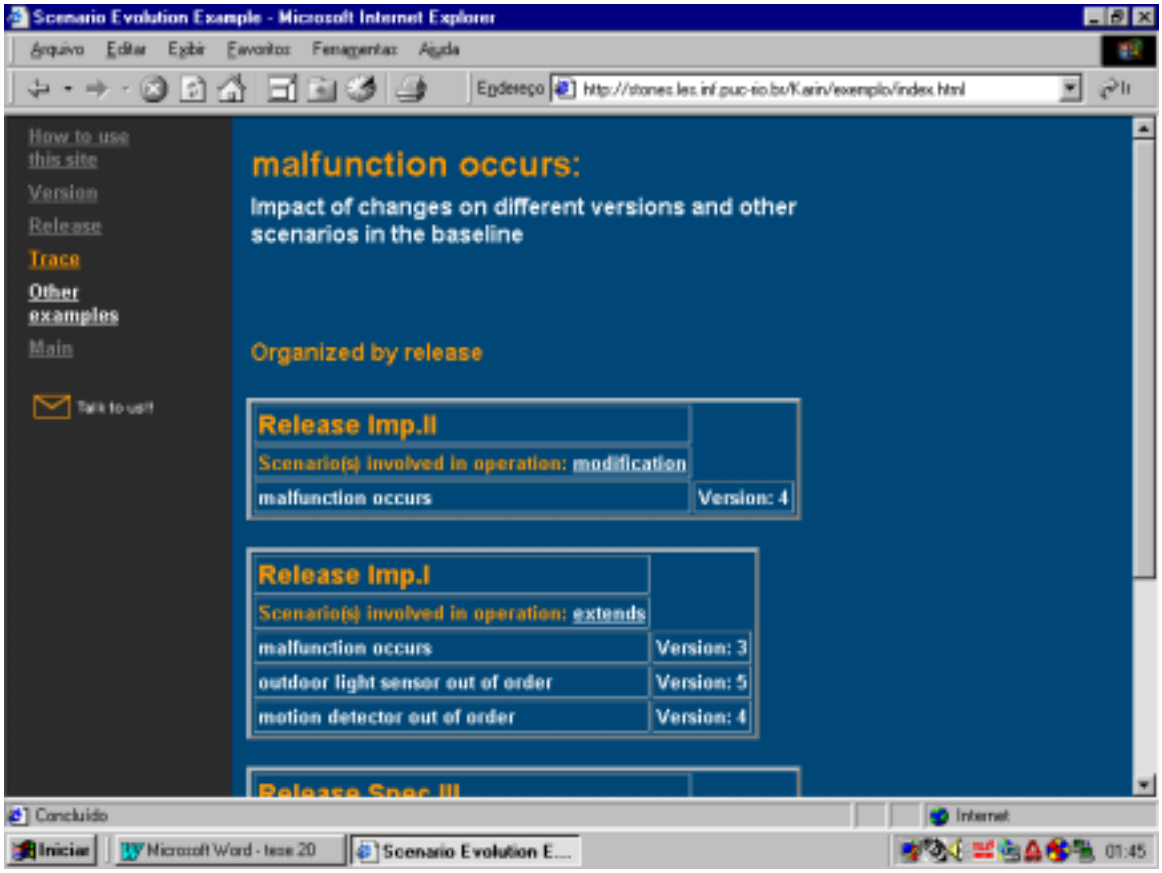


Figura 5.3 – Rastreamento do cenário malfunction occurs em relação a outros cenário da base

Note que a figura ilustra todas as operações em que o cenário esteve envolvido e, se for o caso, os outros cenários que também fizeram parte das mesmas. Se computarmos o número de cenários que estiveram, em alguma de suas versões, envolvidos com o cenário *malfunction occurs* em questão, teremos um número razoável de conexões. Se adicionarmos a este número os elos com entradas dos léxico e outros artefatos de software que foram sendo produzidos paralelamente ao processo de versionamento deste cenário, teremos um número com ordem de grandeza em torno de dezenas de conexões. Nesta luz, um processo de gerência de configuração eficaz se torna fundamental. Na próxima subseção nos concentraremos neste tópico.

5.1.2 CONFIGURAÇÃO

Acreditamos que, do ponto de vista do apoio à evolução natural dos cenários, mecanismos de gerência de configuração serão aqueles que trarão maiores benefícios. Chamamos de evolução natural o refinamento da informação básica dos cenários

elicitada ao início do processo. Este se dá como resultado do aumento da compreensão do macrosistema e suas implicações. Este tipo de controle pode ser executado primeiramente através de mecanismos clássicos de controle de versão, que fornecem um histórico dos cenários e um *log* das modificações que estes sofreram ao longo do tempo.

Controle de versão é definido como o processo de coordenar e gerenciar o desenvolvimento de objetos em evolução [Hicks98]. No desenvolvimento de software baseado em cenários, assim como em outras áreas, o processo evolutivo é caracterizado por sucessivos refinamentos dos objetos envolvidos. Especificações, modelos de dados e até mesmo código estão em constante refinamento durante todo o desenvolvimento. Em várias ocasiões é de interesse manter versões intermediárias destes objetos, seja para consulta ou para retomar pontos chave. O volume destas versões pode se tornar muito grande, gerando necessidade para apoio automatizado a este processo [Tichy82].

No caso do desenvolvimento baseado em cenários esta máxima é verdadeira. No capítulo anterior mostramos os resultados de estudos de caso onde acompanhamos a evolução de cenários de várias aplicações. A partir destes resultados, concluímos que durante o desenvolvimento de um sistema, cenários sofrem grandes modificações que podem ser traduzidas por um conjunto de operações, como evidenciado pelo modelo de evolução de cenários apresentado no capítulo anterior. A partir deste estudo observamos que a evolução de cenários está longe de ser um processo de refinamento e que, portanto, poderia ser gerenciado através do controle de versões apenas. Na realidade, o processo evolutivo de cenários tem um caráter transformacional. Raros são os cenários que permanecem inalterados durante a evolução do processo; união com outros cenários, separação, consolidação e eliminação são algumas das possibilidades, além do processo de refinamento da informação mencionado acima. Na Figura 5.4 a seguir tentamos mostrar a complexidade desta evolução.

Na figura mostramos dois eixos, o de versão e o de configuração. Estes eixos são completamente independentes, i.e., os cenários não precisam mudar de versão a cada vez que uma nova configuração é lançada. Note que o cenário C3 na versão V1 permanece inalterado durante o curso das cinco configurações ilustradas pelo

exemplo. O processo de versionamento de cenários, como visto anteriormente, é condicionado a aplicação de operações apenas. Como resultado destas obtemos, não somente novas versões de um cenário mas também cenários inteiramente novos, e.g., o cenário C5V4 evolui para nova versão C5V5 e dá origem ao novo cenário C6V1. Também é possível que cenários contribuam no processo de versionamento de outros, como no caso dos cenários C2V1, C4V1 e C1V4, resultando nos cenários C1V5 e C4V2. Na realidade, como resultado da aplicação de operações novas versões de cenários são criadas, destruídas ou incorporadas em outros cenários da base.

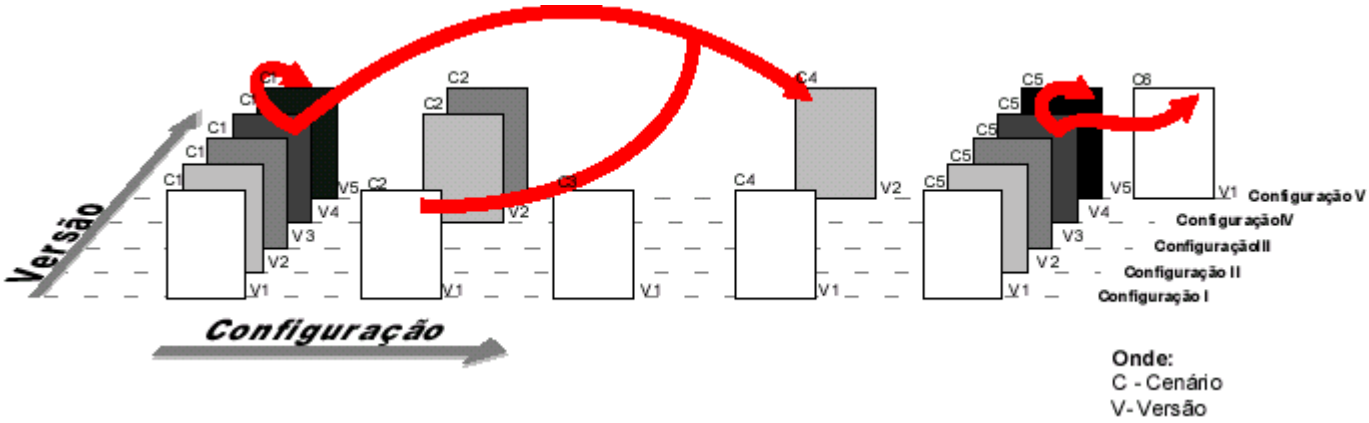


Figura 5.4 - Exemplo de um gráfico de evolução de cenários [Breitman00]

Do ponto de vista de gerência de configuração podemos mapear o processo de versionamento de objetos de software segundo três possíveis gráficos, conforme ilustrado na Figura 5.5 [Conradi98]. São eles gráficos sequenciais, do tipo árvore e acíclicos. No caso mais restritivo as versões são organizadas como uma série de revisões sucessivas. No caso dos gráficos em árvore, apenas as versões localizadas nas folhas podem ser utilizadas para gerar novas versões. Finalmente no caso de gráficos acíclicos, uma nova versão pode possuir muitos predecessores, retomando conceitos estabelecidos em versões anteriores. Em se tratando do processo de versionamento de cenários caímos na classificação mais complexa, i.e, gráficos acíclicos.

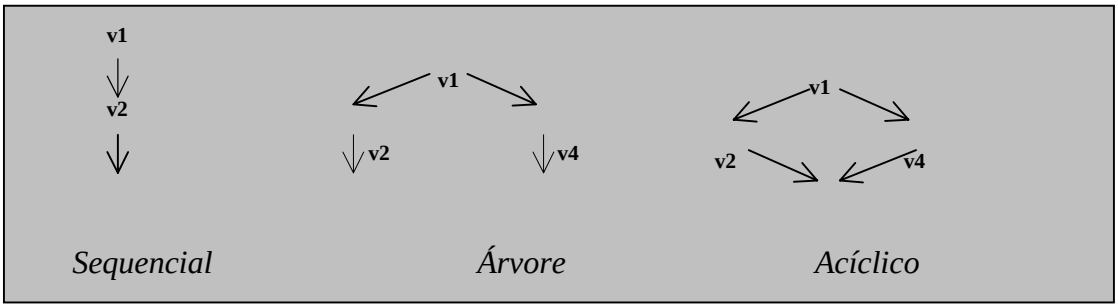


Figura 5.5 - Classificação de gráficos de versão segundo [Conradi98]

Como ilustrado na Figura 5.6 o processo de versionamento pode se basear indiscriminadamente em um ou mais cenários de versões anteriores. Tomemos, por exemplo os processos ilustrados pelas duas setas da figura. Na seta da esquerda temos um processo casado onde através de um único cenário (C1V4) derivamos uma nova versão do próprio (C1V5) e, conjugado com os cenários C2V1 e C4V1 temos a criação do cenário C4V2. Na seta da direita temos uma situação mais simples, o cenário C5V4 gera a versão C5V5 e uma versão para o novo cenário C6V1. Ilustramos os gráficos acíclicos para este exemplo na Figura 5.6.

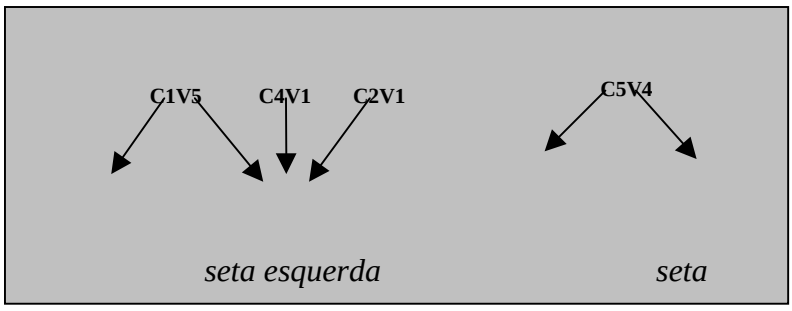


*Figura 5.6 - Gráficos de versão acíclicos para o exemplo ilustrado na Figura 5.**

De modo a rastrear a informação codificada através dos cenários é fundamental manter um registro destas mudanças, de modo em que possamos recriar os passos que levaram a determinada versão. Desta forma é necessário manter-se um histórico do conteúdo de cada cenário a cada versão de modo a permitir a reconstrução de

versões anteriores. A possibilidade de recuperação de versões anteriores é inclusive utilizada como indicativo de maturidade do processo de desenvolvimento de software segundo o Capacitação e Maturidade de Software (CMM) [Paulk93]. É claro que para viabilizar este processo de recuperação teremos que lidar com grandes volumes de informação, tornando crucial o apoio automatizado.

5.1.2.1 RATIONALE

Em uma abordagem clássica da gerência da configuração, tais como as prescritas por [Tichy82, Bersoff80], deveríamos ter para cada versão um registro especial contendo maiores informações sobre o próprio processo de versionamento. Tipicamente informações do tipo data da criação, autor e comentários fazem parte da maioria dos sistemas para a gerência da configuração [Mikkelsen97]. O grupo de Engenharia de Requisitos da PUC- Rio adotou o *Requirements Baseline* apresentado no capítulo 2 como o modelo base para a gerência do processo de desenvolvimento de software. O Baseline através de um sistema de etiquetas (labels) prescreve os mecanismos básicos para este controle. A Figura 5.7 a seguir ilustra uma das etiquetas utilizada. Note que além de data, hora e autor também mantemos registro dos eventos que dispararam a mudança e que para cada versão de cada cenário é necessariamente gerada uma etiqueta.

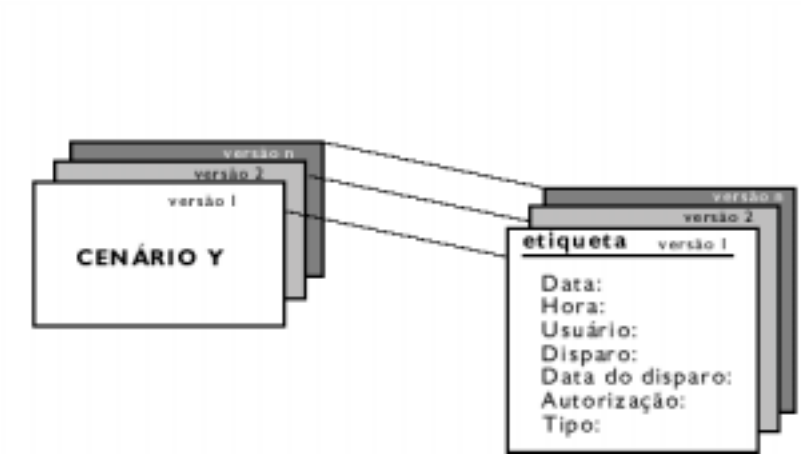


Figura 5.7 - Exemplo das etiquetas utilizadas pela Requirements Baseline e sua relação com cenários

Em nossa experiência no desenvolvimento de software baseado em cenários notamos que mecanismos clássicos de gerência da configuração, tais como os gráficos de versão e etiquetas abordados acima são fundamentais mas não suficientes para garantir a rastreabilidade dos sistemas. Idealmente devemos também manter um registro das decisões tomadas durante o processo de desenvolvimento.

Acreditamos que um mecanismo válido seria o emprego de um sistema para o armazenamento do *rationale* que antecede a realização das operações. Se pudéssemos armazenar as razões que ditaram a realização de uma operação, e.g., união de dois cenários, talvez fosse possível determinar padrões de modo a automatizar parte do processo. Outra vantagem da captura do *rationale* de decisões é fornecer apoio a manutenção do sistema. Acredita-se que vários problemas inseridos em de um código que sofreu manutenção são resultado direto de mudanças nos requisitos do sistema realizadas inadvertidamente. É possível que em presença documentação de melhor qualidade este problema seja minimizado. Em uma avaliação empírica de documentação de design *rationale* Karsenty concluiu que esta é bastante útil e que designers a utilizam extensivamente para auxiliar sua compreensão do problema [Karsenty96].

Na seção 3.2 deste trabalho apresentamos uma série de propostas para a captura de *rationale*. Apresentamos desde técnicas baseadas na utilização de hipertexto [Kaidl93, Wood94] até propostas que se baseiam na utilização de objetivos [Potts94], passando por propostas com estruturas específicas para a captura e registro das informações[Carroll95, Conklin88, Gotel95]. Apesar de não ter sido encontrada nenhuma referência específica ao emprego destas proposta para a captura de design *rationale* no contexto da utilização de cenários, acreditamos que não teríamos dificuldades na adaptação das mesmas aos nossos propósitos.

5.1.3 RASTREAMENTO

Finalmente, o último requisito para uma ferramenta de apoio a gerência da evolução de cenários é oferecer suporte a rastreabilidade da informação envolvida. [Domges98, Ramesh95]. Definimos como rastreabilidade a capacidade de identificação das fontes

de informação que deram origem aos requisitos do software [Gotel93]. Estas podem ser usuários, clientes, outros sistemas, documentação e quaisquer outro tipo de informação diretamente relacionada aos requisitos. A tarefa de manter a rastreabilidade é bastante complexa e envolve grande esforço por parte dos desenvolvedores.

Até agora discutimos a evolução de cenários somente através da perspectiva dos próprios, i.e., o processo de versionamento e o lançamento de novas configurações de cenários. Sabemos porém que o processo de desenvolvimento de software, independente da metodologia que o suporte, pode ser caracterizado pela elaboração de um grande volume de artefatos que descrevem seus diversos estágios. Entre outros estão modelos de objetos, cartões CRC, glossários em geral, modelos essenciais e modelos do tipo entidade relacionamento. Cada um destes artefatos reflete o desenho que existe por trás de cada um dos estágios do desenvolvimento do sistema. A medida que o processo se desenrola, novos cenários são gerados e estes devem refletir as decisões que estão capturadas através destes artefatos. De modo geral, as conexões entre cenários e outros artefatos resultantes do processo de desenvolvimento existem de modo reflexivo, ou seja, os cenário refletem as mudanças registradas por estes artefatos. De modo a permitir o rastreamento da informação é fundamental tornar estas conexões explícitas. Desta forma passamos a ter um entendimento mais amplo da evolução do sistema.

5.2 DIMENSÕES DA GERÊNCIA DO PROCESSO EVOLUÇÃO DE CENÁRIOS

Levando em conta a discussão anterior sobre os requisitos para apoio a gerência da evolução de cenários, adicionamos uma nova perspectiva ao nosso entendimento inicial. Este levava em conta apenas a evolução segundo os eixos de versão e de configuração. Ficou claro que devemos entender o processo de evolução em um contexto mais amplo, levando em conta outros artefatos de software produzidos paralelamente aos cenários da aplicação. Desta forma incluímos uma terceira dimensão a gerência do processo, a de rastreamento.

Desta forma entendemos que o processo de evolução de cenários se dá em três dimensões distintas, são elas: versão, configuração e rastreamento. A primeira, versão, trata das mudanças que ocorreram em um cenário em particular ao longo do tempo, i.e, mudanças em seu conteúdo, operações em que suas várias versões estiveram envolvidas e os impactos das mesmas em outros cenários. Em resumo tratamos dos históricos de cada um dos cenários. Devemos levar em conta as razões que levaram a aplicação de cada operação e suas consequências. Desta forma mecanismos de controle tais como as etiquetas e captura de *rationale* discutidos anteriormente se tornam fundamentais.

A segunda dimensão em que podemos observar a evolução de cenários é ao longo do processo de desenvolvimento de software. Como sabemos, ao longo do deste processo várias atividades são conduzidas de modo a atingir o produto final. Entre outras estão a elicitação de requisitos, modelagem, testes e análise, e implementação. Durante o desenvolvimento, uma série de conjuntos de cenários que refletem estas atividades são produzidos. Naturalmente alguns cenários tiveram de sofrer mudanças de modo a torná-los compatíveis com o estágio que representam.

Finalmente, a última perspectiva na evolução de cenário está ligada ao relacionamento dos primeiros a outros artefatos. Note que as outras dimensões tratam apenas do relacionamento entre cenários. É sabido que durante o processo de desenvolvimento de software são produzidos uma série de artefatos que auxiliam desenvolvedores em suas tarefas. Modelos orientados a objeto, entidade relacionamento e cartões CRC são apenas alguns exemplos. Nesta perspectiva estudamos os relacionamentos dos cenários com estes artefatos. Chamamos esta dimensão de rastreamento. Entenda-se que o rastreamento da informação acontece em todas as três dimensões. Chamamos esta em particular de rastreamento por economia, na realidade estamos designando o rastreamento dos cenários em relação a outros artefatos presentes na baseline. A Figura 5.8 ilustra as três dimensões.

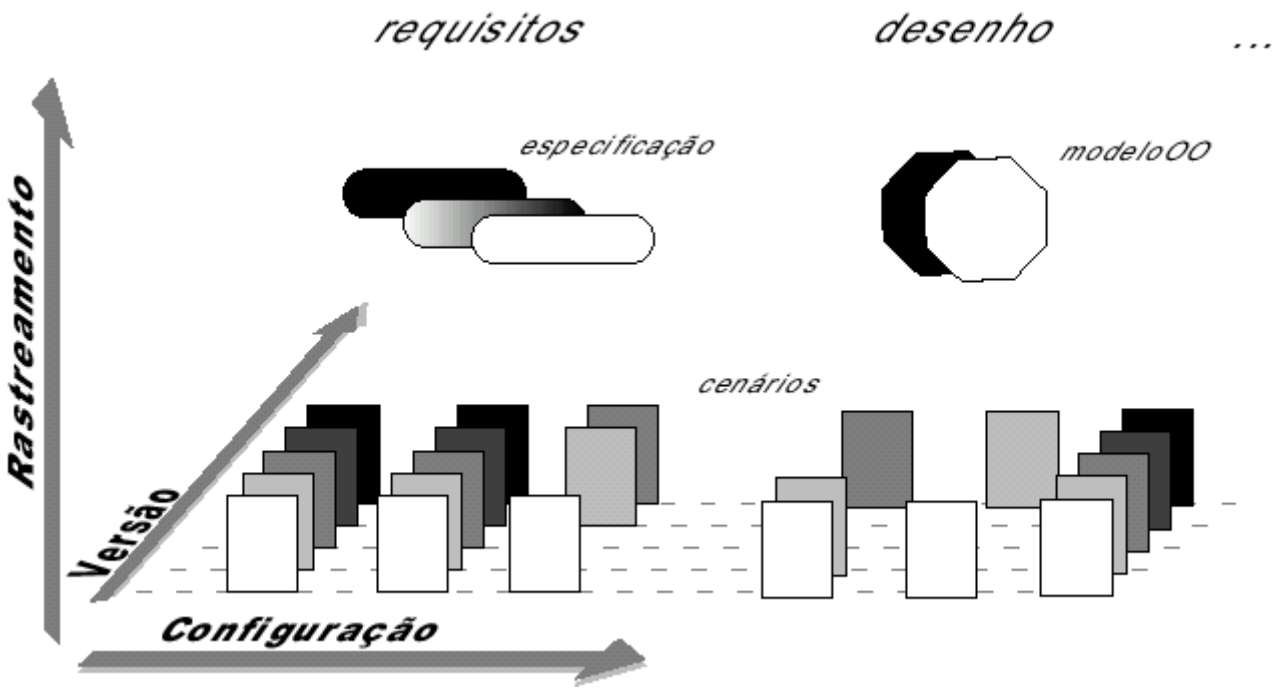


Figura 5.8 - Dimensões de gerência do processo de evolução de cenários

Baseados na discussão apresentada nesta seção e no modelo de evolução apresentado no capítulo anterior, iremos propor um *Framework* para a gerência da evolução de cenários. Este *Framework* será apresentado na próxima seção.

5.3 FRAMEWORK PROPOSTO

Baseados no conhecimento adquirido através da elaboração dos estudos de caso, que resultaram no modelo de evolução de cenário apresentado no capítulo anterior e nos requisitos para o suporte a gerência deste processo propomos um *Framework* para a gerência do processo de evolução de cenários. O *Framework* proposto pode ser customizado de modo a servir a classe de representações de cenários que utilizam linguagem semi- estruturada, como classificado no capítulo 2 deste trabalho. Nesta seção apresentamos o *Framework* utilizando a notação UML estendida proposta por Fontoura [Fontoura99]. Dividimos o restante desta seção como se segue. Nas próximas sub seções detalhamos o *Framework* através de seu próprio processo de instanciação. Desta forma seremos capazes de enfatizar os aspectos customizáveis e apontar outros comuns a todas implementações do mesmo. Na sub seção seguinte

apresentamos uma possível instância para o *Framework* através do protótipo da ferramenta SET (Scenario Evolution Tool).

5.3.1 NOTAÇÃO UTILIZADA

O *Framework* que apresentamos a seguir foi elaborado utilizando-se a notação UML [Booch99] estendida por Fontoura para acomodar uma representação que pudesse descrever o design de *Framework*s [Fontoura99]. Os *hot spots* ou pontos de flexibilidade são indicados através da presença de chaves. Segundo a notação existem três tipos de *hot spots*, os métodos de variação denotados por *{variable}* que permitem modificar a implementação de um método na instanciação do *Framework*. As classes de extensão, denotadas por *{extensible}* que permitem a adição de novos métodos a uma classe na instanciação do *Framework* e, finalmente os *hot spot* de interface que permitem a modificação dos tipos do sistema uma vez que permitem a criação de novas classes na instância do *Framework*. O último é implementado através de estereótipos UML do tipo *Instance Class* e é denotado pela restrição *{incomplete}* . Desta forma o autor permite flexibilidade a vários níveis, i.e., utilizando esta notação é possível modificar a implementação de um método, criar novos métodos ou, até mesmo, novas classes nas instâncias do *Framework*. Além da classificação acima, os *hot spots* podem ser do tipo {*static*} ou {*dynamic*}. O último define *hot spot*s que precisam de reconfiguração em tempo de execução enquanto que {*static*} representa aqueles que não precisam ser reconfigurados.

De modo a limitar o comportamento de *hot spot*s a notação prevê a utilização de notas que podem ser anexadas a classes. Nestas notas são explicitadas as limitações que se deseja impor a cada *hot spot*, como por exemplo, a gama de valores que pode ser retornada por um método. A seguir apresentamos o *Framework* na Figura 5.9. Note que os pontos de flexibilização estão indicados através das indicações de *hot spot* que aparecem entre chaves. Utilizamos todos os três tipos de *hot spot* no *Framework* para a evolução de cenários. Na seção seguinte detalhamos cada um dos componentes do *Framework.*

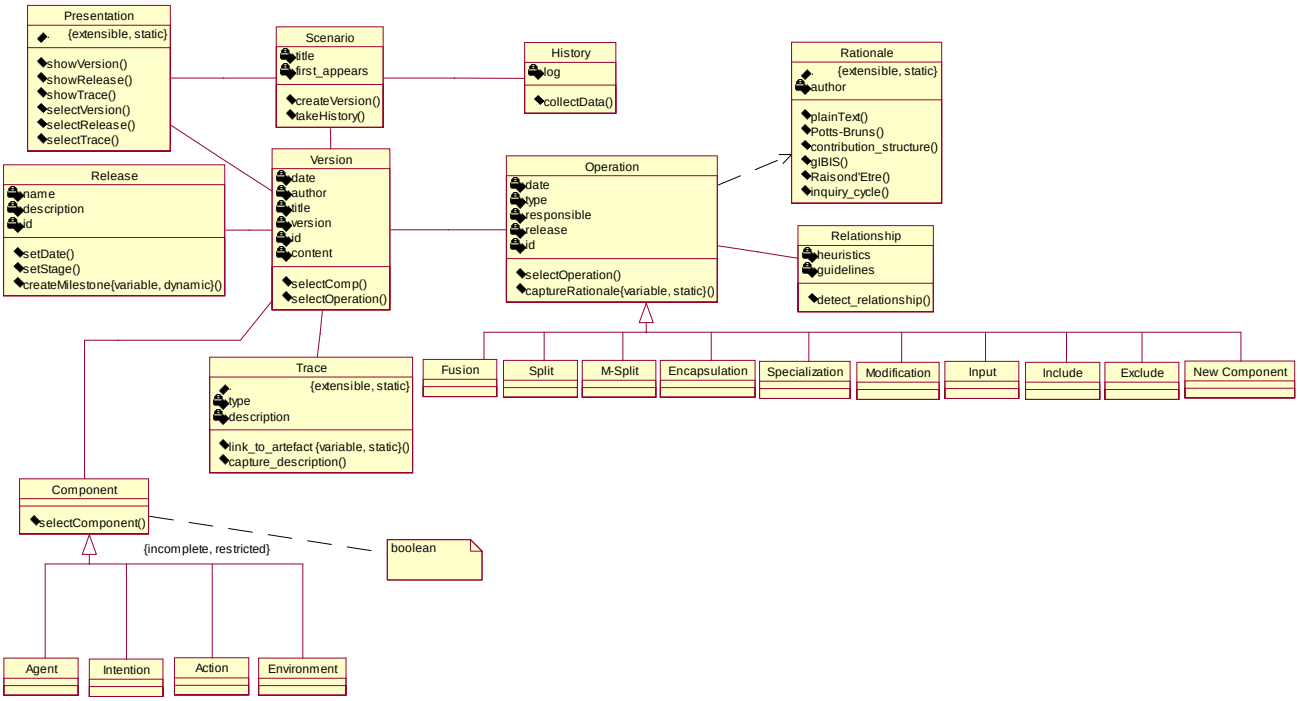


Figura 5.9 – Framework de evolução de cenários.

5.3.2 FRAMEWORK DE EVOLUÇÃO DE CENÁRIOS

Antes de descrevermos o *Framework* propriamente dito esclareceremos alguns conceitos presentes no modelo.

Primeiramente gostaríamos de fazer a distinção entre os conceitos de cenário e versão do cenário. O conceito de cenário representa uma abstração enquanto que o de versão do cenário a representação física de uma instância do primeiro. No momento em que registramos o conteúdo do primeiro cenário da base esta já é a primeira versão deste

cenário. Desta forma, o que geralmente referenciamos como cenário é, na verdade, o par cenário/versão.

Também é necessário clarificar as diferenças existentes entre versão e configuração . Configuração é um conjunto de cenários tomado a um determinado momento no tempo. Cada configuração utiliza a versão mais atualizada de um cenário . Uma versão de um cenário, por sua vez, pode participar de mais de uma configuração , pois os processos de versionamento cenário e de obtenção de configurações são completamente independentes.

O processo de versionamento de cenários, como detalhado no capítulo anterior é baseado em operações que podem ou não ser aplicadas durante o intervalo existente entre uma configuração e outra. Se um cenário não sofreu nenhuma modificação que determinasse sua mudança de versão, esta mesma versão do cenário fará parte de quantas configurações forem lançadas neste intervalo. Os critérios para o lançamento de uma configuração, como veremos detalhado pelo *Framework*, são determinados pelo usuário através da instanciação do *Framework* e podem ser baseados em datas específicas, estágios de desenvolvimento (final da fase de especificação, por exemplo) ou através de critérios específicos a serem customizados pelo próprio usuário.

Passaremos agora a descrever cada um dos componentes do *Framework* de evolução de cenários por classe:

5.3.2.1 SCENARIO

Como mencionado acima a classe cenário é uma abstração. Desta forma cabe a ela apenas os atributos *title* (título) e *first_appears*, momento em que o cenário aparece pela primeira vez. Lembramos que o processo de desenvolvimento de software é também um aprendizado e, não raro, novos cenário surgem como resultado do aprofundamento do conhecimento do problema a ser resolvido. Esta classe possui os métodos createVersion() que permite a instanciação da classe abstrata e o método takeHistory() que coleta dados para a elaboração dos históricos dos cenários. Note

que esta é uma classe fixa, não possui nenhum *hot spot*, de maneira que todas as instâncias do *Framework* possuirão a classe cenário.

5.3.2.2 VERSION

A classe versão representa a instanciação da classe abstrata cenário. Também é uma classe fixa e contém os seguintes atributos: *title* e *first_appears*(herdados do cenário abstrato), *date, author, version, id e content.* O método selectOperation( ) permite a seleção de operações que, lembramos, são as responsáveis pelo mecanismo de versionamento, i.e., uma nova versão de um cenário somente pode ser criada como resultado de uma operação. O método selectComp( ) por sua vez, permite que o usuário determine os componentes sob o qual deseja manter registro do processo de versionamento.

5.3.2.3 COMPONENT

Esta classe é a generalização dos componentes pertinentes aos cenários. No capítulo 3 onde fazemos um revisão da literatura de cenários estendemos a classificação proposta por Rolland [Rolland98] de modo a integrar os quatro componentes ilustrados no *Framework*, a saber, agente, intenção, ação e ambiente. Esta classificação leva em conta os componentes presentes nas representações para cenários estudadas ate então. Acreditamos que, futuramente, outros tipos de componentes possam surgir. Desta forma fizemos uma provisão para esta eventualidade no *Framework* através da restrição *{incomplete}* adicionada a classe Component. Este é um *hot spot* de interface, ou seja, permite que o novas classes representando outros tipos de componente sejam incorporadas a instâncias do *Framework* através da instanciação da classe New Component. Note que além de *incomplete* a classe também é *restricted.* Esta restrição limita o comportamento da nova classe impedindo que novos métodos sejam acrescentados as suas instâncias, ou seja, qualquer novo componente instanciado através do estereótipo *New Class* terá apenas o método selectComp( ).

A classe Component possui apenas um método, selectComp( ) que determina a seleção dos componentes os quais faremos o controle de versão. Relacionado a este

método existe uma nota, que representa uma restrição ao valor retornado por este método. No caso a restrição indica que apenas valores booleanos (falso ou verdadeiro) podem ser retornados pelo método selectComp ( ).

5.3.2.4 TRACE

Esta classe é responsável pelo rastreamento dos cenários com outros artefatos resultantes do processo de desenvolvimento de software mantidos pela baseline de requisitos. Contém dois atributos, *type* e *description*, que capturam o tipo e uma descrição dos objetos que têm maior ligação com o cenário. Quando tratamos de rastreamento uma questão sempre presente é a granularidade em que trataremos a informação [Gotel95, Pohl96]. No caso de artefatos como por exemplo um modelo de objetos podemos considerar unidades de informação o próprio modelo, os objetos em separado ou detalharmos ainda mais e tratar a informação em termos de atributos e métodos de cada classe em separado. Esta decisão, como discutimos anteriormente, fica a cargo dos usuários responsáveis pela processo de rastreamento. É neste contexto que encaixamos o atributo *description*. Ele vem de modo a permitir um expediente onde o usuário possa acrescentar mais detalhamento as conexões que o cenário tenha com um artefato.

A classe Trace possui dois métodos porém é uma classe de extensão, ou seja, é um *hot spot* que permite a criação de novos métodos em sua instanciação. Possibilidades são a criação de um novo método que conectasse os cenários a artefatos automaticamente ou um método que fizesse um registro de data, hora e autor responsável pelo rastreamento. Os métodos previstos pela classe são link_to_artefact( ) que é um método de variação, i.e., permite que o usuário mude sua implementação durante o processo de instanciação e o método capture_description( ) que permite o acréscimo de informações sobre os objetos envolvidos.

5.3.2.5 RELEASE

Esta classe possui como atributos o nome da configuração (*release)*, um identificador e uma descrição. A última funciona no mesmo espírito do atributo *description* da classe Trace, ou seja, permite que o usuário acrescente informações sobre uma configuração em particular. Os métodos desta classe permitem que o usuário determine os critérios que serão utilizados para a criação de novas configurações. Os métodos setDate( ) e setStage( ) permitem a definição de critérios gerais tais como uma data do ano ou início/término de um dos estágios do desenvolvimento de software. É importante lembrar que o *Framework* não está vinculado a nenhum processo em particular. Desta forma os estágios serão aqueles correspondentes ao processo de desenvolvimento adotado pelos usuários.

O método createMilestone( ) que é um método de variação, i.e., permite que o usuário mude sua implementação durante o processo de instanciação, permite o estabelecimento de novos critérios. Estes critérios para a geração de novas configurações podem estar vinculados a qualquer processo externo e cabe ao usuário definir sua implementação. Note que o método é do tipo *dynamic*, ou seja, permite que seja reconfigurado em tempo de execução (pode estar vinculado a algum parâmetro específico, como um flag externo enviado por outro sistema, por exemplo).

5.3.2.6 OPERATION

Como vimos no capítulo anterior o processo de versionamento de cenários é vinculado a aplicação de operações. Desta forma esta classe concatena não somente informações sobre as operações em si mas também sobre sua aplicação e resultado. Seus atributos são *date*, *type*, *responsible, release* e *id*. É uma classe fixa e os métodos que implementa são selectOperation( ) e o método de variação capture*Rationale*( ). O último permite que o usuário mude sua implementação durante o processo de instanciação. Este fato fica óbvio uma vez que levamos em conta os vários métodos para a captura de *rationale* disponíveis na classe *Rationale*.

5.3.2.7 RATIONALE

Esta classe que somente possui o atributo *author* responsável pelo preenchimento do *rationale* é uma classe do tipo extensão, i.e. , além dos métodos disponíveis para a captura de *rationale* permite que durante sua instanciação o usuário crie um novo método. As propostas para a captura de *rationale* implementadas pelos métodos desta classe estão descritos em detalhe na seção 5.1 deste capítulo. Os métodos são plain_Text( ), Potts-Bruns( ), contribution_structure( ), gIBIS( ), Raisond'Etre( ) e inquiry_cycle( ). O primeiro não implementa nenhum método específico porém permite que o usuário entre suas *rationale* livremente.

5.3.2.8 HISTORY

Esta classe fixa serve de repositório para os dados que vão permitir a elaboração do histórico de cada um dos cenário da base. Contém apenas um atributo, *log*, responsável pela captura da data, hora e dia da criação de novas versões e o método collectData( ) que mostra a evolução de um cenário ao longo de sua vida útil.

5.3.2.9 PRESENTATION

Esta classe permite a implementação das três dimensões da evolução de cenário discutida na seção anterior. É uma classe de extensão, de modo que permite que instâncias do *Framework* implementem novos métodos além dos previstos. Possui seis métodos que podem ser agrupados em dois tipos, os de seleção e os de demonstração. Os métodos de seleção permitem que o usuário selecione os subconjuntos de informação que deseja pesquisar para cada uma das dimensões. São eles selectVersion( ), selectTrace( ) e select Release( ). Todos estes métodos são também métodos de variação, permitindo que sua implementação seja modificada no momento de instanciação do *Framework* (imagine que o usuário deseja limitar o número de critérios que podem ser utilizados na seleção). Os três métodos restantes dão conta da demonstrar a evolução dos cenários (ou subconjuntos previamente

selecionados) nas dimensões correspondentes. São eles showVersion( ), showTrace( ) e showRelease( ).

Um último comentário é acerca da entidade etiqueta responsável por registrar a criação de novos elementos na baseline de requisitos. Como ilustrado na seção 5.1.2.1 deste capítulo as etiquetas guardam informações que auxiliam na rastreabilidade de informações. A totalidade de informações capturadas pela etiqueta, data, hora, usuário, trigger, data trigger, autorização e tipo são capturadas pelas seguintes classes fixas do *Framework*: Version, Operation e *Rationale*. Desta forma optamos por omitir o objeto Tag (Etiqueta) pois o mesmo somente traria redundância ao *Framework* sem acréscimo de informação relevante.

5.3.2.10 RELATIONSHIP

Na seção 4.6.1 discutimos a causalidade existente entre a constatação de relacionamentos e a aplicabilidade de operações. Esta classe estática desempenha dois papéis no *Framework*. O primeiro é fornecer as heurísticas que permitem a detecção da existência de relacionamentos entre dois ou mais cenários de uma mesma configuração. O método detect_relationship() oferece um apoio semi automático para a detecção de relacionamentos entre cenários, desde de que estes estejam no formato codificado, vide seção 4.1 deste trabalho.

O segundo papel desta classe é oferecer um pequeno guia que relaciona a existência de relacionamentos e a aplicabilidade de operações, nos moldes discutidos na seção 4.6.1.

5.3.3 PROCESSO DE INSTANCIAÇÃO DO FRAMEWORK

O *Framework* prescreve um processo de instanciação que permite seus usuários decidir a política de versionamento dos cenários de sua aplicação através da definição do conjunto de operações legais. A dinâmica das versões é definida em três etapas, em termos da relevância dos componentes no processo evolutivo, em termos da natureza das informações extras, *rationale*, necessárias a compreensão do processo e,

finalmente, através da definição das operação legais que farão parte do núcleo do *Framework*.

O primeiro passo é a definição dos componentes dos cenários importantes do ponto de vista da evolução. As operações surtirão efeito somente sobre os componentes presentes na instância do *Framework*. Desta maneira, o usuário pode deixar de fora os componentes cujas modificações não considera relevantes de modo a gerar uma nova versão dos cenários. Segundo Hicks um aspecto importante da estrutura de versionamento é a decisão de quais objetos devem ser versionados [Hicks98]. Conradi e Westfechtel também fazem uma distinção entre objetos que são versionáveis daqueles que se mantém durante o desenvolvimento de software de modo a simplificar os modelos de gerência. [Conradi98]

Imagine um cenário entitulado “apresentar tese”, por exemplo. Este cenário contém um componente recursos que lista todos os objetos e contexto necessários para sua realização, tais como retro-projetor, disponibilidade de sala para apresentação e microfonia adequada. Imagine que a organização do evento decidiu disponibilizar data shows para os apresentadores. Neste caso, o ítem data show deve ser incluído na lista de recursos do cenário. Do ponto de vista semântico, esta modificação é bastante importante, significa que os apresentadores possuem uma escolha de equipamento que pode ser utilizado para a apresentação de seus trabalhos. Do ponto de vista da gerência da configuração do cenários, todavia, esta modificação é pouco significativa e não justifica a criação de uma nova versão para o cenário. Neste caso deveríamos simplesmente adicionar o ítem a lista de recursos e armazenar o cenário na baseline. Nenhuma nova versão teria de ser criada nem o usuário teria de perder tempo com burocracia a respeito do *rationale* e detalhes da modificação. Devemos lembrar que um dos aspectos mais importantes para a realização de um processo de gerência de configuração eficaz é evitar sobrecarregar usuários com demandas desnecessárias.

Vale a pena ressaltar que a estratégia configurada pelo usuário age a nível de processo. Os cenários são mantidos na íntegra a nível de produto pela base de dados. O fato de um componente estar ausente ao nível processual significa apenas que modificações no artefato serão absorvidas apenas a nível físico, i.e, os cenários da

baseline vão refletir esta modificação, porém o usuário não perceberá nenhum registro da operação nem será requisitado para descrever justificava para a mesma. Este fato corresponde a distinção que Hicks faz entre os serviços de controle de versão oferecidos ao usuário e as operações intrínsecas de controle de versão [Hicks98]. Em vários sistemas que realizam gerência da configuração existe uma correspondência direta entre conceitos e sua representação física (arquivos). Desta forma a evolução dos objetos, como percebida pelo usuário, corresponde diretamente a evolução deste objeto no nível físico [Tichy82]. No nosso caso existe uma distinção, a percepção do usuário do sistema será a abstração correspondente a instanciação que o mesmo fizer do *Framework* que em certos casos, como o acima, será diferente do que acontece ao nível físico (Baseline).

Enquanto os componentes dos cenários são instanciáveis, a sua estrutura se mantém fixa. A classe abstrata scenario e sua implementação, classe version, estarão obviamente presentes em todas as instâncias do *Framework.* A unidade de trabalho do processo evolutivo é a versão de um cenário. O conceito representado pela classe abstrata scenario somente tem significado do ponto de vista de gerência de configuração através de sua forma instanciada, a versão do cenário. Através das instâncias podemos obter os relacionamentos mantidos com outros cenários, vide o modelo de evolução de cenários, e consequentemente determinar as operações que poderão ser aplicadas sobre esta versão do cenário.

Durante o processo de instanciação é necessário definir qual informação adicional será necessária. Informações mais detalhadas sobre o processo e as razões da aplicação das operações são capturadas pela instanciação da classe *rationale*. Note que esta classe é diretamente conectada ao conjunto de operações. O usuário pode definir a estratégia para a captura do *rationale* de maneira genérica ou atrelar a mesma a operações específicas. Esta é uma classe extensível pois o usuário poderá implementar métodos de captura de *rationale* diferente daqueles disponibilizados pelo *Framework*, e.g., gIBIS[Conklin87], método de estruturas de contribuição [Gotel95].

O núcleo do processo evolutivo se encontra na aplicação das operações. Novas versões de cenários são criadas unicamente como resultado da aplicação de uma ou mais operações. Instanciar o conjunto legal de operações é, portanto, a parte mais

crítica deste processo. O usuário tem liberdade de escolher as operações que vai permitir serem aplicadas sobre os cenários ao longo de seu desenvolvimento. Estas irão modelar o formato e organização de sua baseline de cenários no futuro.

Finalmente, o processo de instanciação é concluído quando o usuário define o escopo da utilização de cenários durante o processo de desenvolvimento de software. Em nosso entendimento a utilização dos cenários é benéfica durante todos os estágios de desenvolvimento, porém o usuário pode definir que sua utilização será restrita a determinados estágios apenas. O *Framework* proposto é independente do modelo de processo de desenvolvimento adotado. Através da instanciação da classe Release o usuário pode determinar os momentos em que deseja extrair configurações da base. O processo de versionamento é constante e independente porém a política de criação de configurações, tal como proposta por Hicks, é determinada pelo usuário. Desta forma, novas configurações (releases) somente serão geradas através de requisições do usuário. A classe prevê métodos onde o usuários pode estabelecer datas específicas ou estágios de desenvolvimento. O método *setMilestone()* permite que o usuário especifique outros parâmetros para a geração de novas configurações.

O *Framework* proposto foi implementado pela ferramenta SET que apresentamos na próxima seção. Mostraremos também um exemplo de instanciação do mesmo utilizando a ferramenta. O exemplo escolhido é, novamente, o sistema para a simulação do controle da iluminação do Departamento de Informática da Universidade de Kaiserslautern, na Alemanha.

5.4 - PROTÓTIPO PARA A FERRAMENTA SET

5.4.1 ARQUITETURA

Construímos um protótipo de ferramenta que implementa o *Framework* de evolução de cenários, apresentado na seção anterior, segundo uma arquitetura em três camadas como ilustrado pela figura 5.10 a seguir.

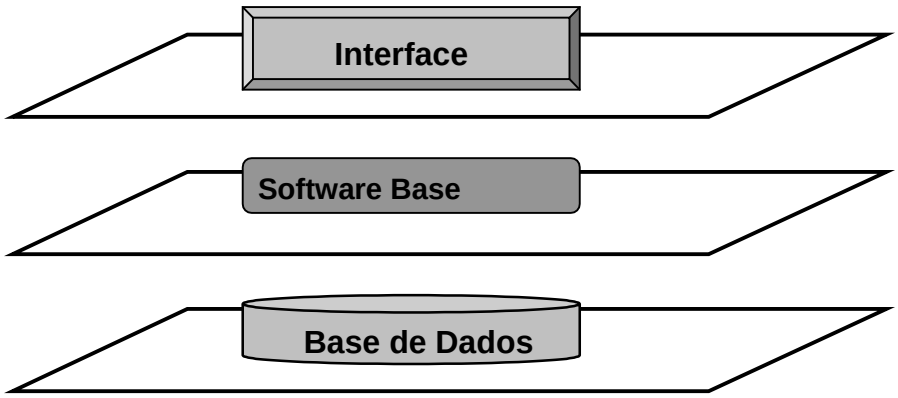


Figura 5.10 - Arquitetura da ferramenta SET

Na camada inferior temos a base de dados, responsável pelo armazenamento do conteúdo dos cenários bem como informações relativas a evolução propriamente dita, tais como *rationale*, log de operações e informações relativas as configurações. Além destas também são armazenados dados sobre o próprio processo de criação destes objetos, data, hora e autor além de conexões com outros artefatos da baseline, de modo a permitir a rastreabilidade das informações.

Na camada de base temos o software responsável pelo processamento das informações do modelo. Por um lado temos todo a crítica e processamento das informações fornecidas pelo usuário para um formato aceito pelo banco de dados e pelo outro o tratamento de respostas a consultas ao banco de modo a adequá-las aos requisitos de apresentação da ferramenta. Além destas funções, esta camada implementa a navegação que permitirá o rastreamento das informações. Na camada

superior temos o software de interface com os usuários. Através desta camada todas as interações com o sistema serão realizadas. Para os usuários esta é a única camada que têm acesso, as duas restantes são transparentes.

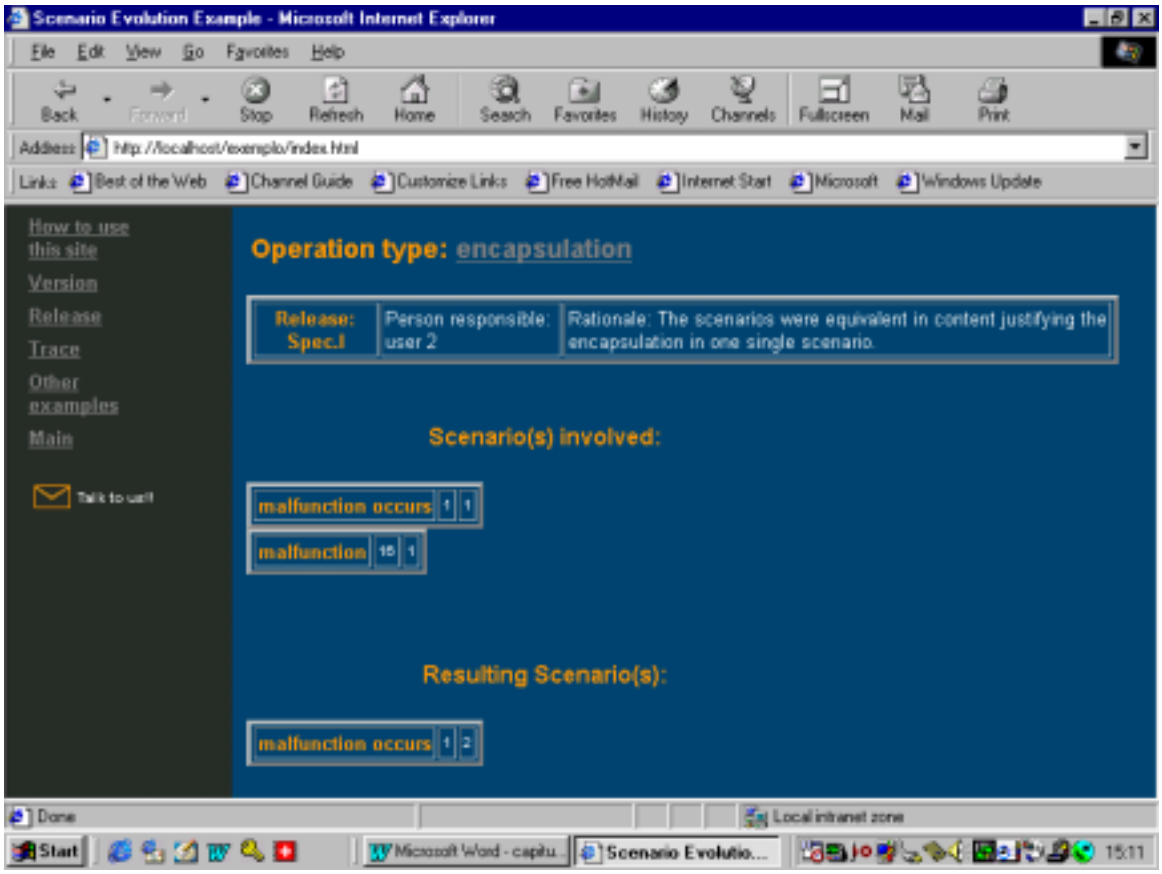
O protótipo da ferramenta foi implementado utilizando-se o banco de dados relacional MS Access© sobre uma plataforma WindowsNT©. A escolha deste banco é justificada pela sua disponibilidade nesta plataforma, grande base instalada e portabilidade do banco se comparado a bancos relacionais de maior porte, tais como SQL Server e Oracle. Testes de acesso remoto via Internet foram conduzidos, tanto no Brasil quanto no exterior, e concluímos que a performance do banco era adequada.

A camada de software base foi implementada utilizando-se a tecnologia de Active Server Pages (ASP©), versão 2.0 que fornece mecanismos adequados tanto para o acesso a bancos de dados via drivers ODBC quanto facilidade para criação de páginas dinâmicas. Escolhemos esta linguagem entre similares, tais como Perl© e Lua©, pois acreditamos que por ser do mesmo fabricante do banco de dados e do sistema operacional incorreríamos em um número menor de problemas de compatibilidade. A hipótese se provou verdadeira. A figura 5.11 exemplifica parte de um código em ASP. Note que parte do código é um comando SQL que envia uma consulta ao banco de dados segundo o parâmetro fornecido pelo usuário (&nome_do_release&) ao mesmo tempo em que retorna o resultado desta consulta utilizando a sintaxe HTML (representado pela linha que inicia com o comando de escrita **Response.Write**.

```
(...)

'SELECIONANDO OS DADOS DOS CENARIOS
        'Criando o Connection
        Set MinhaConexao = Server.CreateObject("ADODB.Connection")
        MinhaConexao.open "exemplo"

        'Criando o RecordSet
        Set MeuRecordSet = Server.CreateObject("ADODB.RecordSet")
        'Construindo o select do Cenario
        value = Request.Form("release")
        sql = "SELECT DISTINCT operation_in.scenario, operation_in.version, operation.rel_name,
operation.op_id, operation.type, cenario.title FROM cenario INNER JOIN (cen_release INNER JOIN (operation
INNER JOIN operation_in ON operation.op_id = operation_in.op_id) ON (cen_release.scenario =
operation_in.scenario) AND (cen_release.version = operation_in.version)) ON cenario.id = cen_release.id WHERE
cen_release.rel_name='"& nome_do_release &"' ORDER BY operation.rel_name, operation_in.scenario "

        MeuRecordSet.Open sql, MinhaConexao, 3,2
        MeuRecordSet.MoveFirst
        if MeuRecordSet.EOF then
                'Nao existe cenario cadastrado na base de dados
                Response.Write "There are no scenarios in the database"
        else
                MeuRecordSet.MoveFirst
```

Figura 5.11 - código em ASP

A interface com usuários, é implementada através de *browsers*. Escolhemos este paradigma por várias razões, entre elas a facilidade de uso por parte de usuários, a grande base instalada e possibilidade de disponibilizar o protótipo para a comunidade de requisitos. Utilizando uma interface do tipo HTML diminuímos grandemente a curva de aprendizado da ferramenta, uma vez que a grande parte dos usuários possui noções básicas de navegação através de *browsers*. Do ponto de vista de desenvolvimento também tivemos um ganho, uma vez que este tipo de interface oferece blocos prontos para a confecção das páginas. A figura 5.12 mostra o resultado de uma consulta utilizando-se a ferramenta. Neste caso mostramos os detalhes de uma operação específica, realizada durante o release Spec.I do exemplo, mostramos a pessoa responsável pela operação, o *rationale* de sua aplicação e os cenários envolvidos.



5.12 - Exemplo de interface da ferramenta SET

5.4.2 ESTRUTURA DA FERRAMENTA

Iniciamos esta seção por mostrar um exemplo ilustrativo do funcionamento da ferramenta SET para, a seguir, definir mais precisamente sua estrutura interna e relacionamento com o *Framework* proposto apresentado na seção anterior. O exemplo de interação está ilustrado na Figura 5.13, a seguir.

Figura 5.13 Esquema da ferramenta SET ilustrando uma interação

Neste exemplo fazemos uma consulta de modo a determinar a rastreabilidade de um cenário, em uma versão específica, em relação a outros artefatos da base. O processo, visto da esquerda para direita começa com a primeira interação via a janela do browser. Por razão de economia vamos assumir que o usuário já escolheu a opção *trace* do menu principal e a seguir a opção de rastreabilidade de um cenário em relação a outros objetos de software (ao invés da rastreabilidade em relação ao outros cenários da base, como ilustrado na Figura 5.3 deste capítulo). Desta forma o usuário é confrontado com uma tela contendo todos os cenário existentes na base para escolher aquele que deseja pesquisar. Esta tela é a primeira da esquerda para direita.

Uma vez que o usuário faça uma escolha válida de cenário, esta é processada pelo código ASP que monta uma querie onde requisita as versões existentes do cenário. Caso contrário, i.e., se o usuário não escolheu um cenário, ele recebe uma mensagem de erro do sistema. A querie é então enviada ao repositório de cenário, que manda uma a resposta para a página em ASP. Esta, por sua vez, rescreve o resultado da querie, que foi enviado em formato Tabela pelo banco de dados, no formato HTML escolhido como interface para a ferramenta. O resultado desta operação está mostrado na segunda tela de browser ilustrada na Figura 5.13.

Desta vez o usuário tem a lista das versões existentes para o cenário que escolheu na interação anterior. Ele faz sua escolha, e se esta for válida é enviada para outra página em ASP. Caso contrário o usuário recebe mensagem de erro. De modo a montar o rastreamento do par cenário/versão escolhido é necessário montar consultas a várias bases. A base de cenário retornará o conteúdo da versão do cenários em questão. Os elos para os termos do léxico que se encontram no conteúdo deste par cenário/versão são providos pela ligação existente entre os repositórios de cenário e de artefatos. Da mesma forma são recuperados os ponteiros para outros artefatos de software. Finalmente o repositório de histórico retorna dados relativos a criação desta versão do cenário, configurações a que pertence e operações em que está envolvido. Estas informações serão processadas pelo código ASP, que adicionadas as informações de entrada do usuário, serão transformadas para o formato HTML que desejamos para a saída.

O resultado da consulta está mostrado na tela a direita da Figura 5.13. Note que ela possui uma série de elos, que se acionados, serão tratados por uma terceira página ASP. Estes elos são entradas do Léxico Ampliado da Linguagem ou ponteiros para outros artefatos. Os artefatos estão armazenados através do repositório de dados, de modo a facilitar sua manutenção. Desta forma as páginas HTML que vão exibí-los, tem de ser montadas dinamicamente através de código ASP. Esta possibilidade é representada pela seta de resultados que aponta para fora da Figura 5.13. Na realidade, a maioria das consultas que podem ser realizadas utilizando-se o protótipo tem como resposta páginas com elos dinâmicos, que levam a outros artefatos ou cenários da base em um processo contínuo.

Acreditamos que, através deste exemplo foi possível dar uma idéia da dinâmica de utilização da ferramenta. A partir deste entendimento detalharemos a estrutura da ferramenta, tendo em vista os componentes do *Framework* proposto na Figura 5.9.

Dividimos a estrutura da ferramenta em duas partes, uma fixa, que implementa os objetos não configuráveis presentes no *Framework* e uma segunda parte configurável, onde o usuário pode entrar com os parâmetros que vão determinar sua instância em particular. A parte fixa é implementada ao nível de dados, onde encontramos os repositórios que vão manter as informações e ao nível de software base, onde realizaremos a crítica dos dados, gerência da configuração e pesquisas na base. Os objetos do *Framework* ilustrado na Figura 5.9 que fazem parte da parte fixa da ferramenta são: scenario, version, operation, release, relationship, history e component. Estes se encontram distribuídos nos repositórios localizados na camada inferior da ferramenta

Os métodos previstos por todos os objetos fixos, estão implementados em ASP ao nível do sofware base e utilizam como parâmetros informações que se encontram nos repositórios de dados.

Os objetos configuráveis por sua vez são parametrizados no momento de instanciação do *Framework* pelos seus usuários. Estes, ao contrário dos objetos fixos, utilizam-se de parâmetros fornecidos pelos usuários da ferramenta ao longo do processo de instanciação. Além dos métodos das classes, implementados no nível base, também

encontramos o software responsável pela instanciação do *Framework*, i.e., configuração do banco de dados, software e da estratégia de navegação da própria instância. Os objetos configuráveis são, a saber, presentation, trace e *rationale*. O objeto presentation, responsável pela apresentação dos dados é o único que tem sua implementação distribuída em todos os níveis. Apesar da estratégia de navegação da ferramenta estar fortemente implementada através do software localizado na camada base, através da criação dinâmica de páginas, também encontramos elementos navegacionais nas outras duas camadas. Na camada de banco de dados estão implementados ponteiros para outros artefatos de software. Estes estão distribuídos no próprio conteúdo dos cenários e em uma Tabela especial, chamada trace, localizada no repositório de cenários, que mantém registros de rastreamento entre os cenários e outros objetos. Na camada de interface temos todos os mecanismos de navegação próprios aos browsers, tais como botões de ida e volta e histórico das páginas visitadas.

O objeto *rationale*, do modo em que está apresentado pelo *Framework*, é diretamente relacionado as operações realizadas. Desta forma é armazenado dentro do repositório de histórico de operações, que também mantém informações relativas aos objetos fixos.

Os métodos previstos por estes objetos, tais como o software responsável pelo estabelecimento dos parâmetros para cada instância está localizado no nível base, conforme mencionado anteriormente. Na próxima seção mostramos um outro exemplo de uma sessão de utilização da ferramenta onde alguns destes parâmetros são instanciados.

5.4.3 INSTANCIAÇÃO DA FERRAMENTA

A ferramenta SET permite que seus usuários instanciem o *Framework* de evolução de cenários através de uma sequência interativa de páginas no momento da criação da base de cenários. O usuário define os parâmetros que deseja em sua instância segundo o processo descrito na seção 5.2.3 deste capítulo. Na figura 5.14 mostramos uma das páginas onde o usuário pode escolher se deseja capturar o *rationale* das operações e, no caso afirmativo, definir os parâmetros para tal.

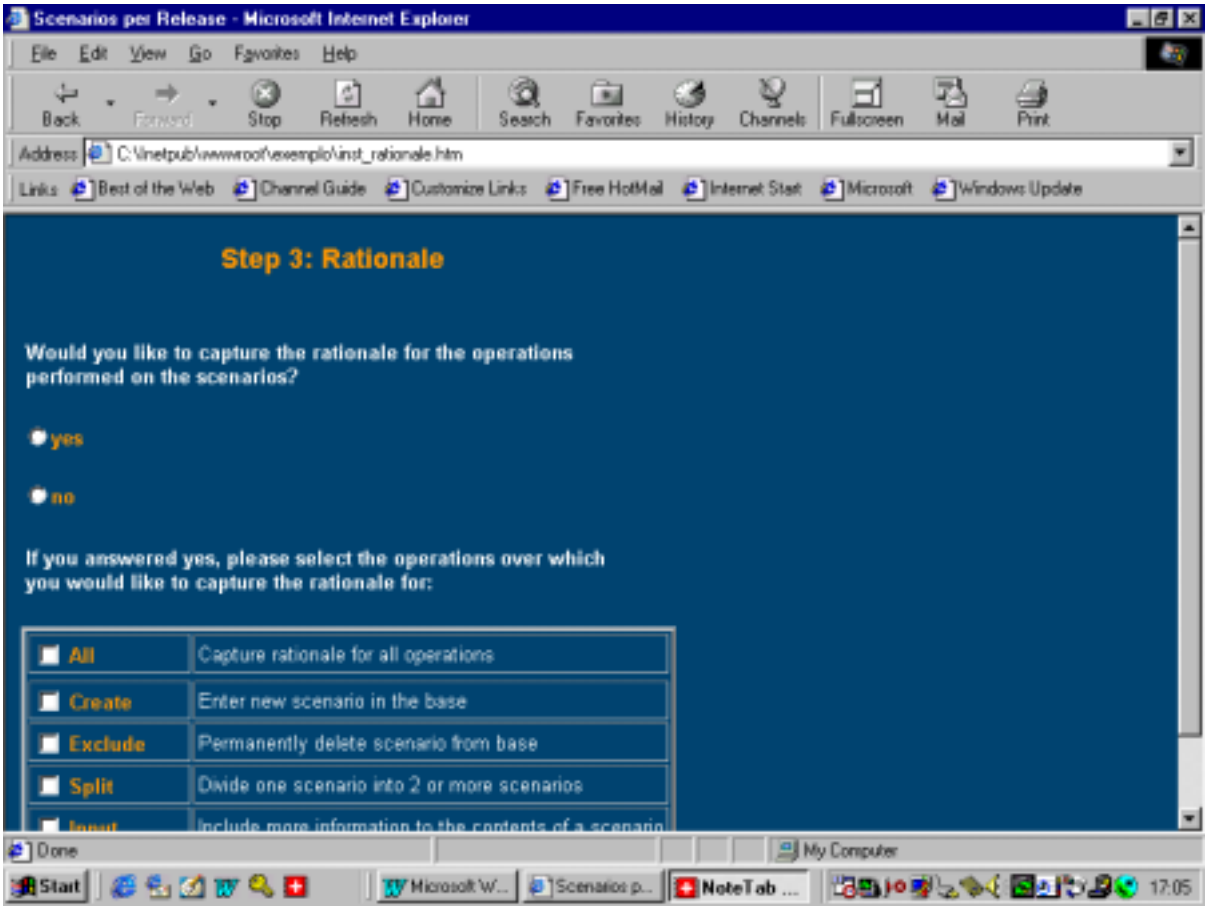


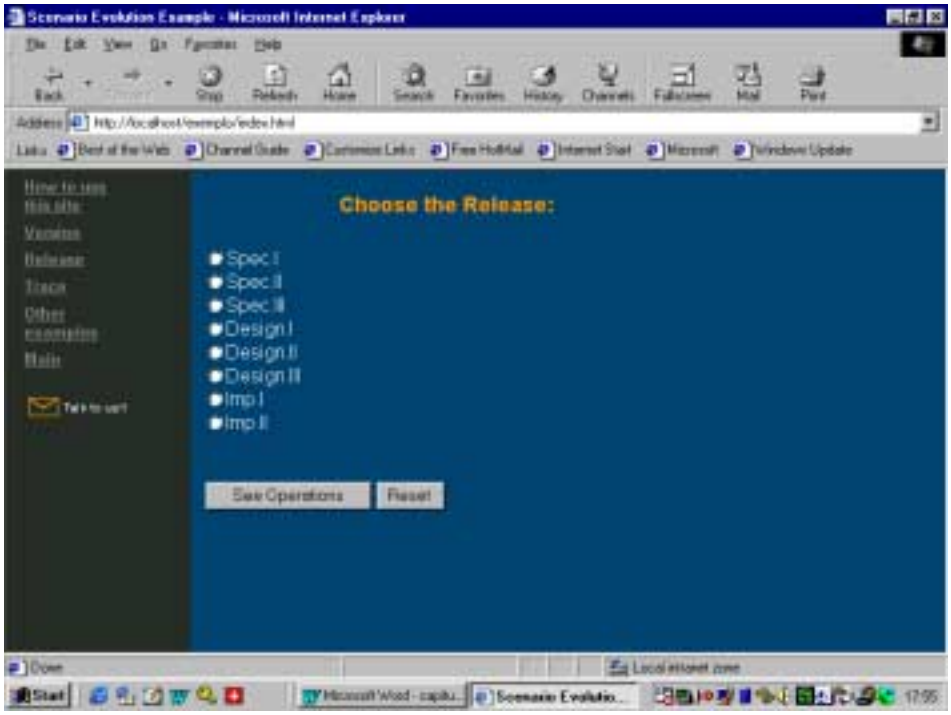
Figura 5.14 - Exemplo de um dos passos de instanciação do Framework de evolução de cenários utilizando-se a ferramenta SET.

Uma vez finalizado o processo de instanciação do *Framework* podemos dar início ao povoamento da base de cenários. Caso o usuário tenha escolhido capturar o *rationale* para todas as operações ele terá de entrar sua justificativa para a entrada dos novos cenários uma vez que estes resultam de uma operação de inclusão. Caso contrário, apenas entrará com o conteúdo dos cenários.

Toda modificação nos cenários da base do momento da instanciação em diante será feito através da aplicação de operações. O usuário primeiramente escolhe a operação e os cenários que estarão envolvidos. Em um segundo passo realiza as modificações relativas a operação escolhida, e.g, em uma operação de divisão escolhe o nome para os novos cenários, realiza a divisão do conteúdo do cenário original e redige a informação que deseja acrescentar, no caso de uma operação de modificação ele

retifica o conteúdo do cenário em questão, no caso de uma operação de encapsulamento escolhe os componentes dos cenários envolvidos que farão parte do cenário resultante. Estas modificações serão realizadas através de páginas contendo formulários HTML a menos da operação de exclusão onde o usuário somente tem que confirmar sua realização (e redigir o *rationale*, se for o caso).

Uma vez instanciada e povoada com no mínimo um cenário, a ferramenta oferece apoio a evolução de cenários segundo o modelo prescrito pelo *Framework*. Permite a aplicação de operações sobre os cenários cadastrados, consultas a base e visualização do processo evolutivo segundo três dimensões, versões, releases e rastreabilidade como descrito na seção 5.2.1 deste trabalho. Na Figura 5.15 a e b mostramos um exemplo onde o usuário pode selecionar um release específico e verificar todas as operações que foram realizadas sobre os cenários pertencentes ao primeiro.



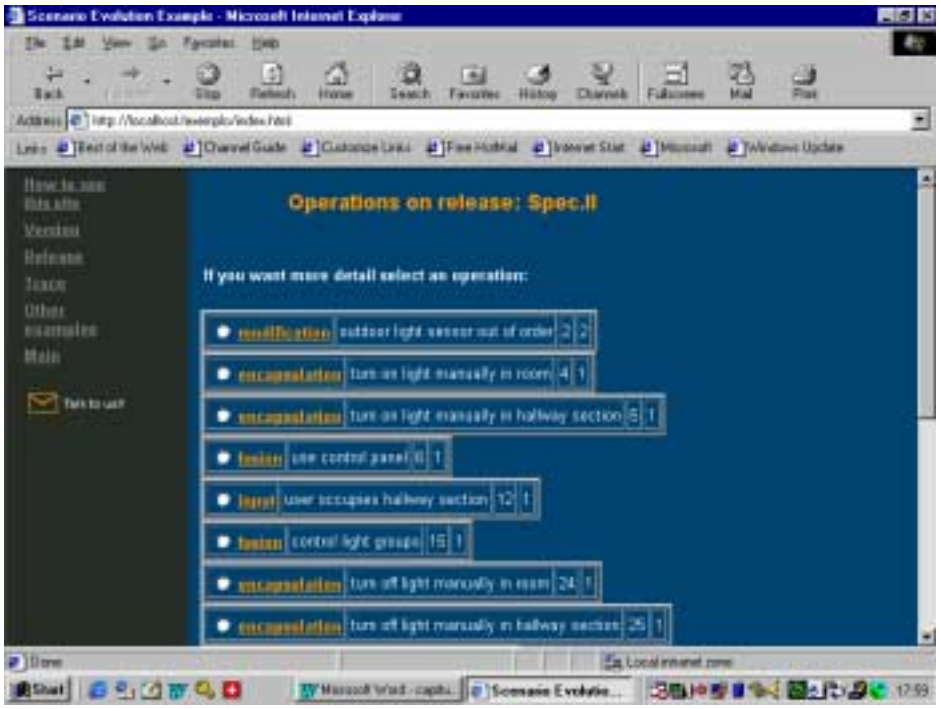


Figura 5.15 a e b – Exemplo de seleção de um release e todas as operações associadas

A ferramenta SET foi utilizada para implementar o exemplo da simulação do controle do sistema de iluminação da Universidade Kaiserslautern. Este exemplo foi proposto para o workshop de Engenharia de Requisitos realizado em 1998 no castelo de

Daghstuhl e subsequentemente se transformou em edição especial do Journal of Universal Computer Science, editado eletronicamente pela Springer Verlag. A implementação do exemplo faz parte do artigo [Breitman00].

5.5 RESUMO

Neste capítulo apresentamos o *Framework* para a evolução de cenários, que fornece subsídios para auxiliar desenvolvedores no processo de desenvolvimento de software baseado em cenários. Baseado na aplicação de em um conjunto de operações sobre cenários, o processo que suporta o *Framework* auxilia na organização e racionalização da evolução dos próprios. O *Framework* proposto leva em conta a existência de outros artefatos que serão produzidos durante o desenvolvimento de software e provê para o rastreamento entre estes objetos. Também foi apresentado o protótipo de uma ferramenta que implementa este *Framework*. A ferramenta SET permite com que usuários, através de uma sessão de configuração, criem instâncias do *Framework* que sirvam seus propósitos. Na próxima seção apresentamos nossas conclusões e apontamos para trabalhos futuros.

Capítulo 6– Conclusão

6.1 CONTRIBUIÇÕES

Neste trabalho mostramos que dificuldades relacionadas a gerência do processo de desenvolvimento de software baseado em cenários tem sido sistematicamente apontadas na literatura de sistemas, em particular pela comunidade de Requisitos [Rolland98, Jarke98, Weidenhaupt98, vanLamsweerde98]. A falta de uma metodologia que suporte a utilização e evolução de cenários, tem dificultado a integração desta técnica na prática da Engenharia de Sistemas.

De modo a minimizar as dificuldades apontadas acima e, buscando oferecer suporte automatizado a evolução de cenários, partimos para a realização deste trabalho. De início nos concentramos no aprofundamento do conhecimento acerca da complexidade envolvida no processo de evolução de cenários. Para tal organizamos estudos empíricos detalhados, cujos resultados estão resumidos na forma de um modelo de evolução de cenários, que oferece uma visão explicativa do domínio da evolução destes artefatos. Baseado em um conjunto de operações e relacionamentos, este modelo explica as variáveis envolvidas na evolução de cenários fazendo uma distinção entre aspectos de processo e de produto. Limitando os usuários a um conjunto fechado de operações, impomos uma certa disciplina no processo de evolução, facilitando sua compreensão. De modo a auxiliar usuários também fornecemos um pequeno guia, na forma de heurísticas, para a seleção e aplicação de operações.

A maior contribuição deste trabalho reside, porém, em propor uma organização que permite sistematizar a evolução de cenários segundo os preceitos da Engenharia, i.e., de forma segura e que possa ser repetida [Pressaman92, Ghezzi91, Humphrey95]. Levando em conta o conhecimento sobre cenários encapsulado no modelo de evolução e, incorporando aspectos de gerência de software, propomos o Framework de evolução de cenários de modo a sistematizar a utilização e gerência destes artefatos em um contexto geral de desenvolvimento de software. O Framework proposto é

configurável, e através da instanciação de *hot spots*, permite que seus usuários talhem estratégias para a gerência da evolução de cenários adequadas a suas necessidades.

Oferecemos apoio automatizado para o Framework proposto através da ferramenta SET. Implementada sobre uma plataforma WEB, a ferramenta conta com dispositivos gráficos que facilitam a visualização dos aspectos relacionados a evolução dos cenários. Lembramos que uma das grandes vantagens apontadas no contexto da utilização da técnica de cenários está em facilitar a validação de informação junto a clientes [Sutcliffe98, Maiden98, Rolland98].

6.2 TRABALHOS RELACIONADOS

Aspaugh et al propõem uma estratégia para gerência de cenários baseada na utilização de glossários, controle de episódios e nos relacionamentos [Aspaugh99]. Os relacionamentos identificados pelo autores levam em conta somente a semelhança existente entre cenários, que é detectada a partir da comparação entre componentes de modo similar ao que utilizamos na análise do Estudo de Caso I. Nosso enfoque difere do proposto pelos autores, pois fornecemos uma taxonomia muito mais rica para a classificação dos relacionamentos entre os cenários.

A questão da rastreabilidade de cenários face a outros artefatos de software é tratada pelo ambiente PRIME-CREWS [Haumer98, Haumer99]. Em sua versão atual, este ambiente faz o relacionamento entre cenários de utilização dos sistemas e modelos de objetivos através de relacionamentos de dependência tipados, e.g., elos do tipo positivo, negativo, atinge objetivo e falha. Segundo os autores, o enfoque implementado para o ambiente pode ser estendido de modo a incluir outros modelos conceituais, tais como modelos entidade relacionamento, modelos de objeto e modelos de negócios. O *Framework* proposto neste trabalho permite incorporar ponteiros para outros documentos de software, facilitando o rastreamento da informação entre os artefatos da baseline de requisitos. Apesar de não tipar os relacionamentos entre os artefatos, permite que o usuário indique os componentes dos mesmos que possuem maior ligação com o cenário em questão, através de um campo de comentários.

Em termos de ferramental para apoio ao desenvolvimento de software baseado em cenários temos a ferramenta l'Ecritoire que captura requisitos do sistema utilizando um misto de cenários e objetivos [Rolland99]. Para ser utilizada durante a fase de requisitos, esta ferramenta faz o mapeamento bidirecional entre o conjunto de cenários e o modelo de objetivos, porém a rastreabilidade com outros artefatos de software não é tratada. O ambiente CREWS-EVE trata de aplicações onde cenários do tipo textual não são suficientes para a captura dos requisitos [Haumer99]. Desta forma o ambiente utiliza o que chama de cenários multimídia, onde as situações do mundo real são capturadas através da utilização de vídeo e animação. Esta ferramenta também foi concebida para ser utilizada durante a fase de requisitos do sistema apenas, ao contrário da ferramenta SET proposta que fornece apoio a utilização de cenários durante todas as fases do desenvolvimento de software.

Sutcliffe et al propõem a ferramenta CREWS-SAVRE que automatiza a utilização de cenários para a elicitação e validação de cenários [Sutcliffe98, Sutcliffe98-b, Maiden98]. Baseada na utilização de usos de caso, esta ferramenta fornece apoio para a geração semi-automática de cenários a partir dos primeiros. Apesar da utilização desta ferramenta ser pontual, apenas durante o estágio de requisitos, ela apresenta um diferencial sobre outras pois permite a identificação de inconsistências e incompletezas nos requisitos a partir de uma análise de padrões de eventos em cenários. Os autores enfatizam a importância de oferecer suporte a visualização do conjunto de cenários, pois estes serão utilizados em uma estratégia para a validação dos requisitos do sistema. Parte fundamental do *Framework* proposto é apresentação dos cenários da base. Entendendo que cenários facilitam a comunicação entre clientes e desenvolvedores, é de suma importância que todas as informações necessárias sejam disponibilizadas. Como discutido nos capítulos anteriores desta tese, são várias as dimensões nas quais podemos observar a evolução de cenários. Alguns exemplos são a evolução de cenários em particular, de um conjunto de cenários, configuração, ou o relacionamento de um cenários com outros artefatos de software.

Finalmente, Ben Achour e outros propõem um conjunto de heurísticas para guiar o processo de autoria de casos de uso [BenAchour99]. Segundo os autores, é muito comum que a primeira versão dos casos de uso capturados contenham erros e,

portanto, tenham que ser corrigidos. Neste intuito fornecem linhas guias para estruturar a linguagem utilizadas na descrição dos casos de uso. Os resultados do estudo conduzido apontam que a utilização de linguagem natural estruturada induz a um número menor de erros. Este argumento fortalece a notação que temos utilizado na descrição de cenários que, apesar de utilizar linguagem natural semi estruturada na descrição dos cenários. [Leite97].

6.3 TRABALHOS FUTUROS

Neste trabalho realizamos um estudo aprofundado da evolução de cenários e apresentamos um *Framework* para a gerência da evolução destes artefatos que resume nossos resultados. Atualmente o *Framework* proposto é suportado pelo protótipo da ferramenta SET, que permite sua instanciação em aplicativos para o apoio automatizado a gerência de cenários. A validação do *Framework* foi realizada através da automatização do segundo estudo de caso e da construção da protótipo da ferramenta SET. Acreditamos que a realização de um processo de validação junto a usuários pode resultar em possíveis refinamentos no *Framework* e na própria ferramenta, em especial no que tange aspectos de usabilidade da mesma.

A versão atual do protótipo implementa apenas alguns dos métodos instanciáveis previstos pelo *Framework*. Planejamos estender a funcionalidade do protótipo através da adição de novos métodos que ofereçam um leque maior de possibilidades aos usuários do *Framework*. Entre as possibilidades estão a incorporação de novos métodos para a captura de *rationale* e a integração ferramentas responsáveis pela confecção de outros artefatos de software.

Finalmente, as heurísticas para a aplicação de operações necessitam refinamento. Apesar de concordarmos com Sutcliffe, em que “algumas heurísticas são melhores do que nenhuma” [Sutcliffe98-b], apontamos para o fato que estas foram derivadas da experiência na evolução de cenários e, acreditamos, que a futura utilização do *Framework* vai fornecer os dados suficientes para o enriquecimento destas heurísticas.

[Alford77] – Alford, H.W. – A Requirements Engineering Methodology for Real Time Processing Requirements - IEEE Transactions on Software Engineering – Vol 3 No. 1 – January 1977, pp.60-69.

[Alspaugh99]- Alspaugh, T.A.; Antón, A.; Barnes, T.; Mott, B. - An integrated scenario management strategy - Proceedings of the 4th. IEEE Symposium on Requirements Engineering (RE99), Limerick, Ireland 1999 - pp. 142-149.

[Antón98] – Antón, A.; Potts, C. – A Representational Framework for Scenarios of System Use - Requirements Engineering Journal – Springer Verlag - Vol. 3 No. 3 & 4, 1998 - pp. 219-241.

[Antonelli98] - Antonelli, L. – ReCase – trabalho final de graduação – UNLP – Faculdade de Ciências Exatas, Departamento de Informática – Argentina, 1998.

[Becker83] – Becker, H.A. – The role of gaming and simulation in scenario project. Operational Gaming: an international approach. International Institute for Applied Systems Analysis, Luxemburg, Austria – 1983, pp.187-202.

[BenAchour99] – Ben Achour, C.; Rolland, C.; Maiden, N.A.; Souveyet, C. – Guiding Use Case Authoring: results from an empirical study - Proceedings of the 4th. IEEE Symposium on Requirements Engineering (RE99), Limerick, Ireland 1999 - pp. 36 - 43.

[Bersoff80] – Bersoff, E. et al. – *Software Configuration Management* – Prentice Hall, 1980.

[Booch99] - Booch, G. ; Rumbaugh, J.; Jacobson, I. - *The Unified Modeling Language user guide* - Addison Wesley - 1999.

[Breitman98] – Breitman, K.K.; Leite, J.C.S.P. – A framework for scenario evolution – in Proceedings of the Third International Conference on Requirements Engineering (ICRE) - Colorado Springs, USA– 1998. – pp. 214-221.

[Breitman98-b] – Breitman, K. K.; Leite, J.C.S.P. - Suporte Automatizado à Gerência da Evolução de Cenários – I Workshop de Engenharia de Requisitos (WER98) – Maringá, PR – 1998, pp. 49-56.

[Breitman99] – Breitman, K.; Leite, J.C.S.P.; Processo de Software Baseado em Cenários - II (Ibero-American) Workshop on Requirements Engineering - Buenos Aires, September, pp. 95-105 –1999.

[Breitman00] – Breitman, K.K.; Leite, J.C.S.P. **–** Scenario Evolution: A Closer View on Relationships – in Proceedings of the Fourth International Conference on Requirements Engineering (ICRE'00) – pp. 102-111.

[Breitman00-b] - Breitman, K.K.; Leite, J.C.S.P. – Scenario Based Software Process in Proc.), 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000) - IEEE Computer Society Press, 2000. to appear.

[Carroll94] – Carroll, J.; Alpert, S.; Karat, J.; Van Deusen, M.; Rosson, M. – Raison d'etre: capturing design history and *rationale* in multimedia narratives. Proceedings of Human Factors in Computing Systems (CHI94) – ACM Press - Boston, USA, 1994. pp. 192-197

[Carroll95] – Carroll, J.M. – *Scenario Based Design: Envisioning Work and Technology in System Development* – John Wiley and Sons, 1995

[Conklin87] – Conklin, J.; Begeman, L. – gIBIS: a Hypertext Tool for Team Design Deliberation – Proceedings of Hypertext'87 – ACM Computer Society Press, November, 1987. pp 247 - 251

[Conklin88] – Conklin, J. Begemann, M. – gIBIS: a hypertext tool for exploratory policy discussion. ACMTOOIS– 1988. pp. 303-331

[Conradi98] – Conradi, R.; Westfechtel, B.; - Version Models for Software Configuration Management – ACM Computing Surveys – Vol. 30 No. 2, June 1998. pp.232-282

[Daghstuhl99]- Requirements Capture, Documentation and Validation – Dagstuhl-Seminar Report 242 – 13.06.99 – 18.06.99 (99241) - Schloss Dagstuhl, 1999

[Dömges98] – Dömges, R.; Pohl, K. – Adapting traceability environments to project specific needs - IEEE Software – vol.41 No.12, December 1998 – pp. 54-63

[Fillippidou98] – Fillipidou, D. – Designing with Scenarios: a critical view on current research and practice – in Requirements Engineering Journal – edited by Springer Verlag - Vol.3 No.1 – pp. 1-22, 1998

[Fontoura99] – Fontoura, M.F.M.C. – Uma abordagem sistemática para o desenvolvimento de frameworks – Tese de doutorado – Departamento de Informática – PUC-Rio, 23 de Julho de 1999.

[Ghezzi91]- Ghezzi, C.; Jazayeri, M.; Mandrioli, D. – *Fundamentals of Software Engineering* – Prentice Hall International Editions – 1991.

[Goguen94] - Goguen, Joseph - Requirements Engineering as the reconciliation of social and technical issues - in *Requirements Engineering: Social and Technical Issues* edited by Joseph Goguen and Marina Jirotka - Academic Press 1994. pp.165-200.

[Gotel93] – Gotel, O. and Finkelstein, A. – An analysis of the Requirements Traceability Problem - Imperial College Department of Computing Technical Report TR-93-41 – 1993.

[Gotel95] – Gotel, O. and Finkelstein, A. – Contribution Structures – in the *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)* – York, March 27 to 29 – IEEE Computer Society Press, 1995, pp. 100-107.

[Gotel97] – Gotel, O. and Finkelstein, A. – Extended Requirements Traceability: Results from an Industrial Case Study– in the Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'95) – Annapolis, January 6-10 – IEEE Computer Society Press - 1997, pp. 169-179.

[Hadad97] - Hadad, G., Kaplan, G., Oliveros, A., Leite, J.C.S.P - Construción de Escenarios a partir del Léxico Extendido del Lenguaje – SoST'97 (Simposio de Teconologia de Software) 26 Jornadas Argentinas de Informática e Investigacion Operativa, Agosto,1997, 14 pp.

[Hadad99] – Hadad, G.; Doorn, J.H.; Kaplan, G.N.; Leite, J.C.S.P. – Enfoque middle-out en la construcción e integración de escenarios - II (Ibero-American) Workshop on Requirements Engineering - Buenos Aires, September, 1999. pp. 79-94.

[Haumer98] - Haumer, P.; Pohl, K.; Weidenhaupt, K. – Requirements Elicitation and Validation with Real World Scenes – Transactions on Software Engineering – Vol. 24, No.12, December- 1998. Pp.1036-1055.

[Haumer99] - Haumer, P.; Heymans, P.; Jarke, M.; Pohl, K.– Bridging the Gap Between Past and Future in RE: A Scenario Based Approach – Proceedings of the 4th. IEEE Symposium on Requirements Engineering (RE99), Limerick, Ireland 1999 - pp. 66-73.

[Heymans98] – Heymans, P.; Dubois, E. – Scenario Based Techniques for supporting the elaboration and the validation of formal requirements – Requirements Engineering Journal – Vol. 3 No. 3&4 – 1998, pp. 202, 218.

[Hicks98] – Hicks, D.; Legget, J.; Nurnberg, P.; Schanse, J. – A Hypermedia Version Control Framework – ACM Trasnsactions on Information Systems, Vol. 16, No. 2, April 1998. pp 127-160

[Hsia94] – Hsia, P.et al – Formal Approach to Scenario Analysis – IEEE Software, vol. 11 no. 2 – 1994. pp.33-41.

[Humphrey95] – Humphrey, W. – *A discipline for Software Engineering* – SEI Series in Software Engineering – Addison Wesley, 1995.

[IEEE-Std830-1984]– The Institute of Electrical and Electronic Engineers - *IEEE Guide to Software Requirements Specifications* – ANSI/IEEE Std 830 -1984.

[Jackson83] – Jackson, M. – *Systems Development* – Englewood Cliffs, N.J.; Prentice Hall, 1993.

[Jackson95] – Jackson, M. – *Software Requirements Specification: a lexicon of practice, principles and prejudices* – Addison Wesley, 1995.

[Jacobson92] - Jacobson, I. et al - *Object Oriented Software Engineering: A use case driven approach* - Addison Wesley/ACM Press, Reading MA, 1992.

[Jacobson94] – Jacobson, I. – *Object Oriented Software Engineering: a use case driven approach* – Addison Wesley, 1994.

[Jarke98] -Jarke, M.; Tung Bui, X.; Carroll, J.M. – Scenario Management: na interdisciplinary approach - in Requirements Engineering Journal – edited by Springer Verlag - Vol.3 No.3 & 4 – pp. 155-173, 1998.

[Jirotka95] – Jirotka, M. et al. – Ethnography by Video for Requirements Capture – mini tutorial presented at the in the Second IEEE International Symposium on Requirements Engineering (RE'95) – York, March 27 to 29 - 1995. pp. 190-193.

[Kaidl93] – Kaidl, H. – The Missing Link in Requirements Engineering – Software Engineering Notes – ACM SIGSOFT - Vol. 18 - April 1993, pp.30-39.

[Karsenty96] – Karsenty, L. – An Empirical Evaluation of Design *Rationale* Documents - Proceedings of the Conference on Human Factors in Computing Systems – CHI'96 – Vancouver, Canada, 1996. pp150 – 156.

[Kuutti95] – Kuutti, K. – Work Processes: Scenarios as a Preliminary Vocabulary – in Scenario Based Design: Envisioning Work and Technology in System Development – John Wiley and Sons, 1995. pp.19-36.

[Kyng95] - Kyng, M. – Creating contexts for design - in Scenario-based design: envisioning work and technology in system development – pp. 85-107 - John Wiley and Sons, New York – 1995. pp. 85-108.

[Lehman80] – Lehman, M. – Programs, Life Cycles and Laws of Software Evolution – Proceedings of the IEEE 68(9), 1980. pp. 1060-1076.

[Leite90] - Leite, J.C.S.P.; Franco, A. P. – O uso de hipertexto na elicitação de linguagens de da aplicação – em Anais do 4° Simpósio Brasileiro de Engenharia de Software– editado pela Sociedade Brasileira de Computação– 1990. pp.124-133

[Leite95]- Leite, J.C.S.P. and Oliveira, A.P. – A Client Oriented Requirements Baseline - in the *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)* –York, March 27 to 29 – IEEE Computer Society Press, 1995. pp.108-115.

[Leite97] – Leite, J.C.S.P. et al. – Enhancing a Requirements Baseline with Scenarios - – Requirements Engineering Journal Vol. 2 No. 4 – Springer Verlag - December, 1998. pp. 184-198.

[Leite97-b] – Leite, J.C.S.P. et al. – Enhancing a Requirements Baseline with Scenarios in the Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97) –Annapolis, USA – IEEE Computer Society Press, 1997. pp.44-53.

[Leonardi 97] Leonardi, Maiorana, V.; Balaguer, F. – Una Estrategia de Analisis Orientada a Objetos Baseada em Escenarios – Anais da II Jornadas de Ingernieria de Software JIS97. Donostia, San Sebastian, Espanha, 1997.

[Maiden98] - Maiden, N.A.M.; Minocha, S.; Manning, K.; Ryan, M. – CREWS-SAVRE: Systematic Scenario Generation and Use- Proceedings of the International Conference on Requirements Engineering, IEEE Computer Society Press, 1998. pp.148-155.

[McGraw97] – McGraw, K.; Harbison, K. – *User Centered Requirements: the Scenario Based Engineering Process* – Laurence Erlbaum Associates – 1997.

[McMenamin84] – McMenamin, S.; Palmer, J. – *Essential Systems Analysis*, Yourdon Press, 1984

[Mikkelsen97] – Mikkelsen, T.; Pherigo, S. – *Practical Software Configuration Management* – Hewlett Packard Professional Books – Prentice Hall, 1997.

[Neto00] – Neto, J.M.S. – Integrando Requisitos Não Funcionais a Modelagem OO – dissertação de Mestrado – Departamento de Informática, PUC-Rio, Março – 2000. 206pp.

[Paulk93] -Paulk, M.; Weber, C.; Garcia, S.;Chrissis, M.; Bush, M. - Maturity Model, Version 1.1 - Software Engineering Institute - Carnegie Mellon University - CMU/SEI-93-TR-25 -ESC-TR-93-178, 1993.

[Pinheiro96] - Pinheiro, F.; Goguen, J. - An Object-Oriented Tool for Tracing Requirements - IEEE Software – Vol. 13 No.2, 1996 - pp 52-64.

[Pohl96] – Pohl, K. – PRO-ART: Enabling Requirements Pre-Traceability – in *Proc*eedings of the Second International Conference on Requirements Engineering (ICRE), IEEE Computer Society Press – Colorado Springs, April 14 to 18 - 1996, pp.76-85.

[Polya45] – Polya, G. – *How to solve it: a new aspect of mathematical method* – Princeton University Press – Princeton, NJ, 1945.

[Potts88] – Potts, C. and Bruns, G. – Recording Reasons for Design Decisions – in Proceedings of the 10th International Conference on Software Engineering – IEEE Computer Society Press – April, 1988, pp.418-426.

[Potts94] - Potts, C. , Takahashi, K., Antón, A. - Inquiry-Based Requirements Analysis - IEEE Software, March 1994, pp. 21,32.

[Pressman92] – Pressman, R. – *Software Engineering: a Practitioner's Approach* – McGraw Hill, 1992.

[Ramesh95] - Ramesh, B.; Stubbs, C.; Edwards, M. - Implementing Requirements Traceability: a Case Study - Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE95) - IEEE Press, - York, England, 1995. pp. 89-95.

[Ramesh98]- Ramesh, B. - Factors influencing requirements traceability practice – IEEE Software – vol.41 No.12, December 1998 – pp. 37-44.

[Ramil99] - Ramil, J.F.; Lehman, M.M. - Modelling process dynamics in software evolution process - some issues - submitted to the workshop on software change and evolution (SCE'99).

[Ringland98] – Ringland, G. – *Scenario Planning: Managing for the future* – John Wiley and Sons – 1998.

[Rivero98] – Rivero; Doorn; Fresno; Mauco; Ridao – Derivación de Objetos Utilizando LEL y Escenarios en un Caso Real - I Workshop de Engenharia de Requisitos (WER98) – Maringá, PR – 1998, pp. 89-98.

[Robertson99] – Roberstson, S.; Robertson, J.; - *Mastering the Requirements Process* – ACM Press and Addison Wesley – 1999.

[Rolland98] – Rolland, C.; Achour, B.; Cauvet, C.; Ralyté, J.; Sutcliffe, A.; Maiden, N.; Jarke, M.; Haumer, P.; Pohl, K.; Dubois, E.; Heymans, P. - A proposal for a scenario classification framework – Journal of Requirements Engineering – vol. (3)– Springer Verlag, 1998. pp. 23-47.

[Rolland99] – Rolland, C.; Grosz, K.; Kla, R. – Experience with Goal-Scenario Coupling in Requirements Engineering – Proceedings of the 4th International Symposium on Requirements Engineering- IEEE Computer Society Press - Limerick, Ireland - 1999 pp.74-83.

[Rosson95] – Rosson, M.B.; Carroll, J. – Narrowing the specification implementation gap in *Scenario-based design: envisioning work and technology in system development*, John Wiley and Sons, New York – 1995. pp. 247-278.

[Rumbaugh91] – Rumbaugh, J.; Blaha, J.; Premerlani, W.; Eddy, F.; Lorensen, W. – *Object Oriented Modeling and Design* – Prentice Hall, Englewood Cliffs – 1991.

[Schneider98] – Schneider, G.; Winters, J. – *Applying Use Cases: a Practical Guide*- Addison Wesley – 1998.

[SEI-CMI1990] – Carnegie Mellon University - Software Specifications: A framework- SEI Curriculum Module SEI-CM-11-2.1, January 1990.

[Sommerville93] - Sommerville, I., Rodden, T., Sawyer, P., Bentley, R. and Twidale, M.. Integrating ethnography into the Requirements Engineering process, in Proceedings of the First IEEE international Symposium on Requirements Engineering, San Diego, Ca, IEEE Computer Society Press - 1994, pp 165-173.

[Suchman87] – Suchman, L. – *Plans and Situated Actions: the problem of human machine communication* – Cambridge University Press – 1987.

[Sutcliffe95] - Sutcliffe, A. - Requirements *Rationale*s: Integrating Approaches to requirement analysis - Proceedings of the Symposium on Designing interactive systems: processes, practices, methods and techniques - ACM Press - Ann Arbor, USA, 1995. pp.33 – 42.

[Sutcliffe97] – Sutcliffe, A. – A technique combination approach to requirements engineering - Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97)– IEEE Computer Society Press – Annapolis, 1997. pp.65-74.

[Sutcliffe98] – Sutcliffe, A. – Scenario-based requirements analysis - Journal of Requirements Engineering – vol. (3)– Springer Verlag, 1998. pp. 48-65.

[Sutcliffe98-b] – Sutcliffe, A.; Maiden, N. – Supporting Scenario Based Requirements Engineering – IEEE Transactions on Software Engineering – Vol. 24 No. 12, December, 1998. pp.1072-1088.

[Texel97] – Texel, P.; Willians, C.B. – *Use Cases Combined with Booch, OMT and UML* – Prentice Hall – 1997.

[Tichy82] – Tichy, W.F. – Design, Implementation and Evaluation of a Revision Control System – Proc. of the 6th. International Conference on Software Engineering, IEEE, Tokio, 1982 – 58-67.

[Tufte97] – Tufte, E. – Visual Explanations: Images and quantities, evidence and narrative – Graphics Press– Cheshire, Connecticut – 1997.

[van Lamsweerde98] – van Lamsweerde, A.; Darimont, R.; Letier, E. – Managing conflicts in goal driven requirements engineering – IEEE Transactions of Software Engineering – vol.24 number 11, November 1998 – pp. 908-926.

[Weidenhaupt98] – Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P. – Scenario Usage in system development: current practice – IEEE Software Vol. 15 No.2 – March, 1998. pp.34-45.

[Wirfs – Brock90] - Wirfs – Brock, R., Wilkenson, B. and Wierner L. – *Designing Object Oriented Software* - Englewood Cliffs, NJ, Prentice Hall, 1990.

[Wood94] - Wood, D.P., Christel, M.G. and Stevens, S.M., A Multimedia Approach to Requirements Capture and Modeling, in Proceedings of the First International Conference on Requirements Engineering IEEE Computer Society Press – Colorado Springs, April 18 to 22 - 1994, pp.53-58.

[Yeh84] - Yeh, R.; Zave, P.;Conn, A.; Cole, G. - Software Requirements: New Directions and Perspectives - *Handbook of software engineering* – 1984, pp.519-543.

[Yeh90] - Yeh, R.; Ng. P. - Software Requirements - a management perspective - System and Software Requirements Engineering - Dorfman, M. and Thayer, R. eds. - Institute of Electrical and Electronics Engineers, Inc., 1990. pp. 450-461.

[Zorman95] – Zorman, L. – Requirements Envisaging through utilizing scenarios - REBUS – Ph.D. Dissertation, University of Southern California – 1995.

Apêndice I

Nesta seção disponibilizamos os cenários relativos aos projetos que fizeram parte do Estudo de Caso I.