

**Luiz Marcio Cysneiros**

# **Requisitos Não Funcionais: Da Elicitação ao Modelo Conceitual**

Tese apresentada ao Departamento de  
Informática da PUC/RJ como parte dos  
requisitos para a obtenção do título de Doutor  
em Ciências da Computação.

Orientador: Julio Cesar Sampaio do Prado Leite

**DEPARTAMENTO DE INFORMÁTICA**

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

# Índice

Índice .....	2
Índice de Figuras .....	4
1 - Introdução .....	9
1.1 - Motivação .....	9
1.2 - Visão Geral da Solução Proposta .....	12
1.3 - Contribuições .....	14
2 - Conceitos Básicos .....	16
2.1 - Engenharia de Requisitos .....	16
2.2 - Requisitos Não Funcionais .....	18
2.3 - Requisitos Funcionais versus Requisitos Não Funcionais .....	22
2.4 - Classificação de RNFs .....	23
2.5 - O Léxico Ampliado da Linguagem .....	27
2.6 - Cenários .....	31
2.6.1 - Construção de cenários a partir do LAL do UdI .....	34
2.7 - Grafo de RNFs .....	37
2.8 - Unified Modeling Language .....	41
2.8.1 - Visão Estática .....	43
2.8.2 - Visão de Interação .....	46
2.8.3 - Object Constraint Language (OCL) .....	48
3 - Lidando com RNFs: da Elicitação ao Modelo Conceitual .....	51
4 - Estendendo o Léxico ampliado da Linguagem .....	58
5 - Desenvolvendo a Visão Não Funcional .....	69
5.1 - Introduzir RNFs no LAL .....	70
5.2 - Gerar Grafos de RNFs .....	72
5.2.1 - Alterações no Grafo de RNF .....	72
5.2.2 - Construção do Grafo de RNFs .....	78
5.3 - Identificar interdependências e Resolver Conflitos .....	83
5.3.1 Comparar Grafos de um Mesmo Tipo .....	85
5.3.2 - Avaliar Grafos de Tipos Possivelmente Conflitantes .....	86
5.3.3 - Comparação Por Pares de RNFs .....	87
5.3.4 - Resolução de Conflitos .....	90
6 - Estendendo os modelos da visão funcional .....	92
6.1 - Extensão aos Cenários .....	92
6.2 - Extensão ao Diagrama de Classes .....	94
6.3 - Extensão aos Diagramas de Sequência e Colaboração .....	97
7 - Integrando as Visões Funcionais e Não Funcionais .....	100
7.1 - Integrando RNFs aos Cenários .....	100
7.2 - Integrando RNFs ao Diagrama de Classes .....	105
7.3 - Integrando RNFs aos Diagramas de Sequência e Colaboração .....	111
8 - Validação da Estratégia .....	116
8.1 Sistema de Iluminação I .....	122
8.1.1 - Desenvolvendo a Visão Não Funcional .....	123
8.1.2 - Integrando as Visões Funcional e Não Funcional .....	131
8.2 - Sistema de Iluminação II .....	144
8.2.1 - Integrando as Visões Funcional e Não Funcional .....	145
8.3 - Sistema de Automação de Laboratórios de Análises Clínicas .....	155
8.3.1 - Desenvolvendo a Visão Não Funcional .....	156
8.3.2 - Integrando as Visões Funcional e Não Funcional .....	160
8.4 - Consolidando os Estudos de Caso .....	169
9 - Conclusão .....	172
9.1 - Contribuições .....	177
9.2 - Trabalhos Futuros .....	181
10 - Bibliografia .....	183
Apêndice A - Estudo de Caso I .....	191
Grafos de RNF .....	191
Classes .....	198
Apêndice B - Estudo de Caso II .....	202
Classes .....	202

Diagramas de Colaboração .....	206
Diagramas de Sequência.....	207
Apêndice C – Estudo de Caso III .....	208
Grafos de RNF.....	208
Classes .....	215
Diagramas de Sequência e Colaboração Antes dos RNFs.....	221
Diagramas de Sequência e Colaboração Após dos RNFs.....	223

## Índice de Figuras

Figura 2.1 - Classificação de RNFs proposta por Mamani [Mamani99].....	24
Figura 2.2 - Classificação de RNFs segundo Sommerville [Sommerville 92].....	24
Figura 2.3 - Árvore de Características de Qualidade de Software (Boehm 76) .....	25
Figura 2.4 – Uma Taxonomia para RNFs .....	26
Figura 2.5 - Exemplo do LAL de um Sistema de Controle de Luzes.....	28
Figura 2.6– Processo de Construção do LAL.....	30
Figura 2.7 - Heurísticas de representação de entradas de LAL (extraídas de [Hadam 97]) .....	30
Figura 2.8 – Diagrama Entidade-Relacionamento descrevendo cenários .....	32
Figura 2.9 - Esquema para descrição de cenários (Extraída de [Leite 97]).....	33
Figura 2.10– Exemplo da Linguagem de Descrição de Cenários.....	34
Figura 2.11– Exemplo de Decomposição do Grafo de RNF .....	38
Figura 2.12– Grafo de RNFs Decomposto Até Suas Operacionalizações.....	39
Figura 2.13– Exemplo de Grafo com Interdependências .....	40
Figura 2.14 – Grafo de RNFs com interdependências analisadas .....	41
Figura 2.15 – Exemplo de um Diagrama de Classes em UML .....	45
Figura 2.16– Exemplo de um Diagrama de Seqüência .....	47
Figura 2.17 –Diagrama de Colaborações .....	48
Figura 3.1 – SADT da estratégia proposta .....	52
Figura 3.2 – Detalhamento da Estratégia Proposta.....	53
Figura 4.1 – Incluindo um RNF Primário em um Símbolo .....	<b>Erro! Indicador não definido.</b>
Figura 4.2 – RNFs Primários contidos na Base de Conhecimento.....	61
Figura 4.3 – Influências entre RNFs Contidas na Base de Conhecimento.....	62
Figura 4.4 – Entrada do LAL com RNFs Primários Assinalados.....	63
Figura 4.5 – Selecionando o RNF que Originou uma Noção ou Impacto .....	64
Figura 4.6 – Exemplo de Entrada do Léxico com Identificação de RNF de Origem.....	65
Figura 4.7 – Exemplo da Navegação Entre RNFs e suas Conseqüências .....	67
Figura 5.1 – Processo de Elicitação de RNFs.....	69
Figura 5.2 – Exemplo de Grafo de RNFs com Identificador de Origem do RNF .....	74
Figura 5.3 – Exemplo de Operacionalizações Estáticas e Dinâmicas .....	76
Figura 5.4 – Raiz de um Grafo de RNF Elicitada com o Uso da Ferramenta OORN.....	80
Figura 5.5 – Exemplo da Construção do Grafo de RNFs.....	81
Figura 5.6 – Analisando as Conseqüências de um RNF.....	83
Figura 5.7 – Interdependência Entre Grafos de Mesmo Tipo .....	85
Figura 5.8 – Exemplo de RNFs Tipicamente Conflitantes.....	86
Figura 5.9 – Grafo Antes de Comparação por Pares .....	88
Figura 5.10 – Segundo Grafo para Comparar.....	89
Figura 5.11 – Grafo Resultante de Novas Decisões de Desenho .....	90
Figura 6.1 – Exemplo de Cenário com a Representação do RNF de Origem .....	93
Figura 6.2 – Exemplo de um Classe Estereotipada para Representar um RNF.....	94
Figura 6.3 – Exemplo da Representação de Atributos advindos de RNFs.....	95
Figura 6.4– Exemplo de Condições de Exceção Representadas em Comentários .....	97
Figura 6.5 – Exemplo de Extensão ao Diagrama de Seqüência para expressar RNFs .....	98
Figura 6.6– Exemplo da Extensão ao Diagrama de Colaboração para Representar RNFs .....	99
Figura 7.1– Integração de RNFs à Cenários.....	101
Figura 7.2 – Exemplo de Uso de Grafos de RNFs para Avaliar Cenários .....	102
Figura 7.3 – Cenário Elicitado na Visão Funcional.....	103
Figura 7.4 – Grafo de RNF com Símbolo Presente no Cenário em Avaliação .....	103
Figura 7.5 - Exemplo da Aplicação da Heurística de Integração de RNF a Cenários.....	104
Figura 7.6 – Integração de RNFs ao Diagrama de Classes .....	105
Figura 7.7 – Classe Sala Extraída da Visão Funcional.....	107
Figura 7.8 – Grafo de RNF Sendo Integrado à Classe Sala .....	108
Figura 7.9 – Classe Lugar, Superclasse de Sala .....	109
Figura 7.10 – Alterações na Classe Lugar para Satisfazer um RNF .....	110

Figura 7.11 – Alterações na Classe Painel de Controle para Satisfazer o mesmo RNF .....	110
Figura 7.12 – Processo de Integração de RNFs aos Diagramas de Seqüência e Colaboração .....	112
Figura 7.13 – Diagrama de Colaboração antes da Integração dos RNFs .....	113
Figura 7.14 – Diagrama de Colaboração Exibindo as Alterações Advindas de RNFs.....	113
Figura 7.15 – Diagrama de Seqüência.....	114
Figura 7.16 – Diagrama de Seqüência Exibindo as Alterações Advindas de RNFs .....	115
Figura 8.1 – Grafo do RNF Usabilidade Aplicada a Controle de Grupo de Luz.....	125
Figura 8.2 – Início da Construção do Grafo para o RNF Seguro, Aplicado à Sala.....	126
Figura 8.3 – Grafo Resultante .....	127
Figura 8.4 – Versão Final do Grafo do RNF Seguro Aplicado à Sala.....	128
Figura 8.5 – Grafo com Interdependência Negativa Identificada.....	129
Figura 8.6 – Grafos Após Resolução de Conflitos .....	129
Figura 8.7 – Conflitos Identificados entre dois RNFs .....	130
Figura 8.8 – Grafo Representando Nova Solução de Desenho .....	131
Figura 8.9 – Analisando um Cenário Contra Grafos de RNF.....	132
Figura 8.10 – Cenário Atualizado para Satisfazer a RNF .....	133
Figura 8.11 – RNFs Sendo Integrado a um cenário .....	134
Figura 8.12 – Cenário Resultante .....	134
Figura 8.13 – Diagrama de Classes Definido por Breitman [Breitman 00] .....	136
Figura 8.14 – Diagrama de Classes Após a Integração .....	136
Figura 8.15 – Um dos Grafos de RNF Usado Quando Avaliando a Classe Sistema de Controle .....	137
Figura 8.16 – Classe Sistema de Controle Antes dos RNFs.....	138
Figura 8.17 – Classe Grupo de Luzes do Teto Resultante da Satisfação do RNF <i>Seguro</i> Aplicado ao Símbolo Sistema de Controle .....	138
Figura 8.18 – Grafo de RNF Aplicável à Classe Grupo de Luzes do Teto .....	139
Figura 8.19 – Resultado Final da Classe Grupo de Luzes do Teto.....	140
Figura 8.20 – Classe Sistema de Controle com RNFs.....	141
Figura 8.21 – Grafo de RNF para Usabilidade Aplicada á Painel de Controle .....	141
Figura 8.22 – Detalhamento da Classe Linha de Status .....	142
Figura 8.23 – Grafo do RNF Seguro Aplicado a Sistema de Controle.....	143
Figura 8.24 – Classe Painel de Controle e suas Subclasses.....	143
Figura 8.25 – Classe Painel de Controle Após Integração com RNFs .....	144
Figura 8.26 – Diagrama de Classes Antes dos RNFs .....	146
Figura 8.27 – Diagrama de Classes com RNFs .....	146
Figura 8.28 – Conjunto de Grafos de RNF Usados para a Classe Painel de Controle .....	147
Figura 8.29 – Detalhamento da Classe Painel de Controle .....	148
Figura 8.30 – Detalhamento da Classe Painel de Controle Administrador .....	149
Figura 8.31 – Detalhamento da Nova Classe Painel de Controle de Sala .....	149
Figura 8.32 – RNFs Aplicados a Corredor .....	150
Figura 8.33 – RNFs Aplicados a Corredor .....	150
Figura 8.34 – Classe Corredor Antes dos RNFs.....	152
Figura 8.35 – Classe Corredor Após RNFs .....	152
Figura 8.36 – Classe Lugar que é Superclasse de Corredor .....	153
Figura 8.37 – Princípio da Composição de um Grafo de RNF para Laudo.....	157
Figura 8.38 – Refinamento do Grafo da Figura 8.37.....	158
Figura 8.39 – Grafo Resultante Depois de validação com Cliente.....	159
Figura 8.40 – Diagrama de Classes Original do Sistema de Laboratórios .....	160
Figura 8.41 – Parte do Diagrama de Classes Após a Aplicação da Estratégia .....	161
Figura 8.42– Grafo de RNF Integrado à Classe Resultados a Inspeccionar .....	161
Figura 8.43 – Classe Resultado a Inspeccionar Antes da Integração .....	162
Figura 8.44 – Classe Histórico de Inspeção Criada após a Integração das Visões.....	162
Figura 8.45 – Classe Central Médica após Integração com RNFs .....	163
Figura 8.46 – Classe LIS Depois da Integração com RNFs .....	163

Figura 8.47 – Classe Fila de Impressão Resultante da Integração de RNFs .....	164
Figura 8.48 – Diagrama de Seqüência para a Entrada de Resultados só com os Requisitos Funcionais .....	165
Figura 8.49 – Classe Exames Admitidos com RNFs.....	166
Figura 8.50 – Diagrama de Seqüência para Entrada de Resultados com RNFs .....	167
Figura 8.51- Diagrama de Colaboração Original para Inspeccionar Resultado .....	168
Figura 8.52 – Diagrama de Colaboração para Inspeccionar Resultados com RNFs.....	168

## **Resumo**

O desenvolvimento de sistemas de informações complexos necessita que utilize-se modelos conceituais que lidem com aspectos que vão além de entidades e atividades. Em particular, pesquisas recentes apontam que esses modelos conceituais necessitam modelar metas de forma a capturar intenções que fundamentam situações complexas que ocorrem em uma empresa. Uma classe em particular dessas metas é chamada de requisitos não funcionais (RNF) que necessitam ser capturados e analisados desde os primeiros momentos do desenvolvimento do software. Erros provocados por não se lidar convenientemente, ou simplesmente não lidar com RNFs são apontados como estando entre os mais caros e difíceis de corrigir. Apesar disso poucas propostas apresentam maneiras sistemáticas de se lidar com os RNFs. Essa tese procura abordar dois aspectos de como se lidar com RNFs que ainda não foram convenientemente abordados na literatura: como elicitar RNFs e como integrá-los aos modelos conceituais. Para isso, propomos uma estratégia que trata da elicitação dos RNFs ainda no início do processo de desenvolvimento de software e de como integrar os RNFs elicitados aos modelos conceituais. A tese foi validada através de três estudos de casos que apontam, como esperávamos, que o uso desta estratégia pode levar a ganhos na qualidade do modelo conceitual final bem como a um processo de produção de software mais produtivo.

## **Abstract**

The development of complex information systems calls for conceptual models that model aspects beyond entities and activities. In particular, recent research has pointed out that conceptual models need to model goals, in order to capture the intentions which underlie complex situations within an organizational context. One particular class of goals is named non-functional requirements(NFR) which need to be captured and analyzed from the very early phases of the software development process. Errors caused by not conveniently dealing, not dealing at all with NFR are among the most difficult and expensive to correct. In spite of that, a few proposals are met on the literature to systematically deal with NFRs. This thesis broaches two aspects on dealing with NFR that were not covered in the literature: how to elicit NFR and how to integrate them to the conceptual models. To do that, we propose a strategy that deals with NFR during the early phases of software development and integrate these NFR to conceptual models. The thesis was validated by three case studies. The results of these case studies suggest, that the use of the proposed strategy can lead to a final conceptual model with better quality as well as to a more productive software development process.



# 1 - Introdução

## 1.1 - Motivação

A crescente complexidade dos sistemas de software e o aumento da exigência de qualidade por parte dos clientes vêm impulsionando o mercado a cada dia produzir mais softwares que atendam, não somente as funcionalidades exigidas, mas também a aspectos não funcionais exigidos pelos clientes tais como: custo, confiabilidade, segurança, manutenibilidade, portabilidade, performance, rastreabilidade de informações entre outros. Estes aspectos não funcionais devem ser tratados como requisitos não funcionais (RNF) do software. Devem ainda ser tratados desde o início do seu desenvolvimento [Chung 95] [Chung 00] [Cysneiros 97] estendendo este tratamento por todo o ciclo de vida do software.

Requisitos não funcionais vêm sendo citados em vários processos de desenvolvimento de software, dentre outras formas como restrições e condições de contorno, porém sempre de maneira quando muito secundária e altamente informal do ponto de vista da elicitação de requisitos. Este tipo de tratamento faz com que, quando tratados, estes requisitos sejam freqüentemente contraditórios, difíceis de serem considerados durante o desenvolvimento de software e difíceis de serem validados. O fato destes requisitos terem sido mal elicitados ou não elicitados tem causado uma série de histórias de insucessos, incluindo a desativação de sistemas pouco após terem sido implantados [Finkelstein 96][Lindstrom 93]. Estudos apontam estes requisitos como estando entre os mais caros e difíceis de corrigir [Brooks 87] [Davis 93] [Cysneiros 99].

Por outro lado, requisitos não funcionais têm recebido muito pouca atenção na literatura e são muito menos compreendidos do que os outros fatores menos críticos

ao desenvolvimento de software [Chung 00]. A maior parte dos trabalhos que abordam RNFs, o faz orientado ao produto, ou seja, se situa no campo da quantificação do grau de conformidade de um software para com os RNFs a que ele deveria satisfazer [Keller 90] [Boehm 76] [Boehm 78] [Basili 91] [Fentom 97] [Musa 87] [Lyu 96].

Poucos trabalhos propõem um tratamento explícito para RNFs sob a ótica do processo de desenvolvimento de software. Os trabalhos que seguem esta ótica propõem o desenvolvimento de técnicas para justificar decisões sobre a inclusão ou exclusão de requisitos que se refletirão no desenho do software durante o desenvolvimento deste. Ao contrário da abordagem orientada a produto, preocupada em medir quanto um software satisfaz a RNFs, este tipo de abordagem procura racionalizar o processo de desenvolvimento de software em termos de RNFs. Frequentemente, quando adicionamos um RNF a uma especificação de requisitos, forçamos tomadas de decisões que poderão afetar positiva ou negativamente outros RNFs, efeito esse visualizado como interdependências entre RNFs. Estas interdependências podem ser positivas ou negativas, ou seja, um RNF pode influenciar positivamente outro RNF contribuindo assim para sua satisfação, bem como pode influenciar negativamente, ou seja, contribuir para que um destes RNFs não seja satisfeito ou satisfeito apenas parcialmente. Dentre os trabalhos existentes, destacamos os seguintes:

- Boehm [Boehm 96] monta uma base de conhecimento com RNFs priorizados pela visão do cliente, mas que entretanto só trata RNFs primários, ficando assim num nível de abstração ainda muito alto para identificação de conflitos entre RNFs ou com requisitos funcionais.

- Kirner [Kirner 96] descreve as propriedades de 6 RNFs no domínio de sistemas de tempo real, performance, confiabilidade, seguro, segurança, manutenibilidade e usabilidade. O trabalho de Kirner descreve como medir estes RNFs e propõe algumas heurísticas de como tratar com estes requisitos durante o desenvolvimento deste tipo de software.
- O Framework proposto por Chung [Chung 93] [Chung 95] [Chung 00] é a abordagem mais completa até o momento e procura tratar de todos os tipos de requisitos não funcionais desde as primeiras etapas do processo de desenvolvimento de software. Apesar de ser um framework bastante completo e eficaz para lidar com requisitos não funcionais durante a elicitação de requisitos, este framework não favorece a integração destes requisitos nas etapas seguintes do desenvolvimento de software, ficando, assim, uma lacuna aberta para que conflitos entre requisitos funcionais e não funcionais passem despercebidos durante o projeto do software.
- Neto [Neto 00] propõe uma estratégia apoiada pela ferramenta OORNF para o desenvolvimento de software, baseada no léxico ampliado da linguagem estendido para tratar RNFs. A partir do léxico, são gerados os cenários, cartões CRC e um modelo conceitual orientado a objetos. Apesar desta proposta ter apresentado bons resultados nos estudos de casos controlados a que foi submetida, por tratar de RNFs apenas no Léxico Ampliado da Linguagem [Leite 93], ela não favorece o tratamento de interdependências entre RNFs. Outro problema é que o tratamento de RNFs até o desenho do software só é efetivo se utilizarmos a estratégia de derivação de cenários proposta por Hadad [Hadad 97] e a estratégia de derivação de cartões CRCs e diagramas de classes proposta por Leonardi [Leonardi 97].

Da análise da literatura existente é possível observar que o uso dos RNFs durante o processo de desenvolvimento de software é abordado apenas de forma parcial. Pode-se ainda observar, que existe uma lacuna no que tange à existência de propostas que apresentem uma estratégia para o uso de RNFs, não apenas nas primeiras etapas do processo de desenvolvimento de software, mas também na integração dos RNFs elicitados com os modelos conceituais gerados, e na conseqüente avaliação de impactos resultantes no desenho do software.

Trabalhos apresentados por Cysneiros [Cysneiros 99, Cysneiros 01] enfatizam premissas apresentadas por Chung [Chung 00] de que a falta de integração dos RNFs aos requisitos funcionais, e por conseqüência sua integração aos modelos conceituais, pode resultar em tempos maiores para se finalizar um software, assim como maiores custos com manutenção.

## **1.2 - Visão Geral da Solução Proposta**

Apresentaremos nesta tese um processo para se lidar com requisitos não funcionais desde as etapas iniciais do processo de desenvolvimento de software. A integração dos RNFs, elicitados durante as fases iniciais do desenvolvimento do software, com os modelos conceituais gerados a partir dos requisitos funcionais deverá permitir obtermos um modelo conceitual que enfoque os requisitos funcionais e não funcionais ao mesmo tempo, obtendo-se assim modelos mais completos e com os impactos da satisfação dos requisitos não funcionais já analisados, resultando em uma melhoria da qualidade final do produto sob a ótica, tanto do cliente quanto do desenvolvedor, bem como em um menor tempo de entrega de produtos e menores custos de manutenção.

Para atingir este objetivo propomos uma estratégia que se baseia no uso de um léxico dos termos utilizados no Universo de Informações (UdI) <sup>1</sup> como âncora da construção dos modelos funcionais e não funcionais.

A construção deste léxico, nomeado Léxico Ampliado da Linguagem (LAL) [Leite 93], é norteadada pela inclusão de símbolos neste léxico utilizando-se de um vocabulário restrito e com referências circulares a outros símbolos do léxico. O uso do LAL permite-nos conhecer a linguagem do Domínio sem necessariamente entender o problema sendo elicitado.

Como forma de sustentar a natureza evolutiva e de negociações da elicitação dos RNFs, propomos estender o Léxico Ampliado da Linguagem definido por Leite [Leite 93]. A ferramenta OORNF [Neto 00] será atualizada para espelhar as novas características que são necessárias ao LAL para auxiliar o engenheiro de software a lidar com o aspecto evolutivo dos RNFs. O aspecto de tratamento de interdependências entre RNFs terá atenção especial visto que estas interdependências podem ser fontes de importantes decisões de desenho que implicam em grande montante de retrabalho se não forem prévia e convenientemente identificadas.

Em seguida iremos mostrar como representar os RNFs que foram elicitados e estão representados no léxico, utilizando-se uma adaptação do grafo de RNFs proposto por Chung [Chung 00]. As adaptações aqui propostas tem por objetivo, primariamente, propiciar algum nível de rastreabilidade dos RNFs de forma a auxiliar no aspecto evolutivo destes. Esta rastreabilidade se preocupa com a fonte de conhecimento que expôs a necessidade do RNF. Um segundo objetivo destas adaptações é o de facilitar a integração dos RNFs aos modelos que expressam a visão funcional do Domínio

---

<sup>1</sup> “Universo de Informações é o contexto geral no qual o software deverá ser desenvolvido e operado. O UdI inclui todas as fontes de informação e todas as pessoas relacionadas ao software. Essas pessoas são também conhecidas como os atores desse universo. O UdI é a realidade circunstanciada pelo conjunto de objetos definidos pelos que demandam o software” [Leite 93].

Mostraremos ainda como organizar os grafos de forma a podermos realizar uma análise sistemática das interdependências intra-grafos.

Uma vez de posse dos modelos funcionais e não funcionais proporemos uma estratégia para integrar ambos os modelos de forma a obtermos um modelo conceitual consolidado que reflita ambas as visões.

Finalmente, como essa estratégia precisa ser apoiada por um método a ser utilizado na especificação de software, iremos apresentar uma extensão a UML [Fowler 97] [Jacobson 99] [Rumbaugh 99] em seus diagramas de classe, seqüência e colaboração. Essa extensão pretende representar nesses diagramas as conseqüências das operacionalizações dos RNFs encontrados na visão não funcional, bem como estabelecer uma ligação para os modelos dessa visão que possibilite a rastreabilidade desses RNFs. Essa rastreabilidade é de grande auxílio no tratamento do aspecto evolutivo do software, uma vez que podemos nos reportar às fontes que originaram a inclusão nos modelos conceituais de uma determinada classe, operação, atributo ou mesmo mensagem oriunda da operacionalização de um RNF.

### **1.3 - Organização**

Esta tese é composta por 9 capítulos, onde o capítulo dois irá abordar conceitos gerais da engenharia de requisitos, incluindo um detalhamento dos conceitos e problemas relacionados a RNFs, bem como uma visão geral do grafo de RNFs e da UML.

O capítulo três irá mostrar uma visão geral da estratégia proposta para lidar com os RNFs da elicitação à finalização do modelo conceitual

O capítulo quatro abordará as extensões propostas ao LAL, bem como seus efeitos na ferramenta OORNF. Mostrará ainda como utilizar a base de conhecimento sobre RNFs no auxílio a elicitação dos mesmos.

O capítulo cinco mostrará as extensões propostas para o grafo de RNF, como obtê-los a partir dos RNFs representados no léxico e uma sistematização de processo para identificação de interdependências.

No capítulo seis abordaremos as necessárias extensões aos modelos conceituais utilizados, neste caso, o modelo orientado a objetos sob a notação UML.

O capítulo sete irá detalhar a estratégia proposta nesta tese para a integração dos RNFs aos modelos conceituais funcionais, obtendo assim modelo conceituais que espelhem as necessidades funcionais e não funcionais.

No capítulo oito apresentaremos os estudos de caso realizados.

Na conclusão apresentaremos as principais contribuições advindas da tese e discutiremos possíveis trabalhos futuros.

## 2 - Conceitos Básicos

### 2.1 - Engenharia de Requisitos

A engenharia de requisitos procura sistematizar o processo de definição de requisitos. Essa sistematização é necessária porque a complexidade dos sistemas exige que se preste mais atenção ao correto entendimento do problema antes do *comprometimento de uma solução*. [Leite 94]. Uma boa definição para requisitos pode ser encontrada em [Leite 94] e é mostrada a seguir.

**“Requisitos:** *Condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo objetivo.*“

Pesquisas sobre a aquisição (elicitação) de requisitos são valorosas por duas razões: primeiramente, da perspectiva da engenharia de software, a elicitação de requisitos é talvez a mais crucial parte do processo de desenvolvimento de software. Estudos [Boehm 84] indicam que, quando só detectados depois do software implementado, erros em requisitos de software são até 20 vezes mais caros de se corrigir que qualquer outro tipo de erro. Além disso, elicitação de requisitos não é atualmente muito bem apoiada por ferramentas de software.

Para a engenharia de requisitos é fundamental que o engenheiro de software delimite os contornos do macrosistema em que a definição de software ocorrerá, ou seja, delimitar o Universo de Informações do sistema (UdI).

É importante ressaltar que o UdI sempre existe. O UdI independe do modelo que estamos utilizando. Mesmo que o macrosistema não esteja bem definido, sempre podemos, e devemos, estabelecer os limites de nossa atuação. Quanto mais bem delineado um UdI, maiores são as chances de um software bem definido.



A elicitação de requisitos é um processo que está constantemente em evolução, ou seja, toda vez que o UdI muda, o modelo resultante do processo de elicitação de requisitos deve mudar junto. Esta realidade é muitas vezes desconsiderada através do que se procurou determinar como um congelamento dos requisitos. Se por um lado a constante evolução do UdI pode gerar custos altos no desenvolvimento, uma vez que temos de estar atualizando nossos requisitos, por outro lado o congelamento dos requisitos em um determinado momento não deve produzir efeitos contrários, aumentando consideravelmente o custo do produto final em decorrência de alterações que serão demandadas para a aceitação do software. Este é um tema ainda pouco explorado e maiores pesquisas deverão ser realizadas neste campo para que possamos conhecer o ponto ideal de equilíbrio.

Em muitas organizações, os requisitos são escritos como parágrafos em linguagem natural e suplementados por diagramas [Kotonya98][Sommerville98]. A linguagem natural é a única notação que é compreendida por todos os leitores potenciais (ex. clientes, usuários, engenheiros de software, engenheiros de requisitos) dos requisitos. Não obstante, alguns pesquisadores não compartilham desta visão tendo feito pesadas críticas a esta utilização e ressaltando a ambigüidade como um dos problemas inerentes aos requisitos escritos em linguagem natural [Meyer85]. Para nós, os eventuais problemas que advêm do emprego da linguagem natural são fundamentalmente resultantes da má utilização desta linguagem, e não necessariamente problemas inerentes à linguagem natural. Sommerville e Sawyer apresentem cinco orientações para se escrever requisitos em linguagem natural [Sommerville98]. Toro apresenta padrões lingüísticos e padrões de requisitos para representar requisitos em linguagem natural [Toro99]. Leite apresenta uma estratégia de representação de requisitos na qual utiliza listas de requisitos em linguagem

natural, suplementadas por léxicos ampliados da linguagem do UdI [Leite92]. Estas propostas podem minimizar os problemas identificados pelos críticos da linguagem natural.

É interessante ressaltar que diversos pesquisadores que apontavam a inerência da ambigüidade da linguagem natural, recentemente mudaram de opinião. Kotonya [Kotonya 95] afirma que alguns problemas devem ser tratados por todo método da engenharia de requisitos. Entre estes problemas, encontra-se a inerência de ambigüidade da linguagem natural que poderia ocasionar más interpretações . Posteriormente, Kotonya indicou que requisitos devem ser escritos em linguagem natural e suplementados por diagramas [Kotonya98].

Uma vez que dificilmente teremos à disposição recursos que permitam satisfazer todos os requisitos dos usuários [Berry98], requisitos podem ser classificados como desejáveis ou obrigatórios, utilizando-se aqui de um enfoque voltado para a necessidade de priorizar requisitos. Os requisitos podem ser ainda classificados como estáveis, mudam mais lentamente, ou voláteis, mudam mais rapidamente. Esta classificação auxilia a atividade de gerência de requisitos, uma vez que possibilita antecipar mudanças prováveis de requisitos. Por fim, uma outra classificação que tem tido bastante aceitação na comunidade acadêmica [Yeh84] [Roman85] [Brackett90] [Mylopoulos 92] [Sommerville98] [Kotonya98] [Mamani99] [Cysneiros99], divide os requisitos em requisitos funcionais e requisitos não funcionais.

## **2.2 - Requisitos Não Funcionais**

A complexidade de um software é determinada em parte por sua funcionalidade, ou seja, o que o sistema faz, e em parte por requisitos gerais que fazem parte do desenvolvimento do software como custo, performance,

confiabilidade, manutenibilidade, portabilidade, custos operacionais entre outros [Chung 00]. Estes requisitos podem ser chamados de requisitos não funcionais. Esta denominação foi utilizada por Roman [Roman 85], sendo os RNFs também conhecidos como atributos de qualidade [Boehm 78] [Keller 90], restrições [Roman 85], objetivos [Mostow 85] entre outros. Recentemente, o termo requisitos não funcionais vem se firmando como nomenclatura corrente no meio acadêmico, sendo esta a nomenclatura que iremos utilizar.

Os RNFs desempenham um papel crítico durante o desenvolvimento de sistemas, e erros devido a não elicitação ou a elicitação incorreta destes estão entre os mais caros e difíceis de corrigir, uma vez que um sistema tenha sido implementado [Brooks 87] [Davis 93].

Embora sem formalmente definir RNFs, alguns trabalhos propõem classificações destes. Em um relatório do *Rome Air Development Center* [Bowen 85], RNFs são classificados da seguinte maneira:

- orientados ao consumidor (ou critérios de qualidade de software), aqui se referindo a requisitos observáveis pelo cliente como eficiência, correção, amigabilidade e outros; e,
- atributos orientados tecnicamente (ou critérios de qualidade) associados mais a requisitos ligados ao sistema, como tratamento de erros e anomalias, completeza, processamento veloz em pontos críticos de tempo real e outros.

Duas diferentes abordagens são utilizadas em tratamento sistemático de RNFs. Elas podem ser classificadas como orientada a produto e orientada a processo [Chung 93]. A primeira aborda RNFs de forma a classificar o quanto um sistema satisfaz os RNFs que dele são requeridos. Por exemplo, medir a visibilidade de um software pode incluir, entre outras coisas, a medição de quantos desvios (branches) existem em um

software. Isto pode ser obtido utilizando-se um critério como: “Não deve haver mais do que X desvios por 1000 linhas de código”. Quase todos os trabalhos em RNF utilizam esta abordagem, orientada a produto, a qual é bem descrita por Keller [Keller 90].

Boehm [Boehm 78] mostra que a qualidade geral do produto final de um software pode ser melhorada simplesmente tornando os desenvolvedores conscientes de que critérios de qualidade deviam ser cumpridos. Também baseados em uma abordagem quantitativa sob a ótica da qualidade de software, Basili e Musa [Basili 91] propõem modelos e métricas para o processo de engenharia de software de uma perspectiva gerencial.

Já a abordagem orientada a processo, que é por nós utilizada nesta tese, advoga o desenvolvimento de estratégias para justificar decisões de desenho *durante o processo de desenvolvimento de software*. Ao contrário da abordagem orientada a produto, nesta abordagem procura-se racionalizar o processo de desenvolvimento propriamente dito em termos de RNFs [Chung 00]. Frequentemente, quando adicionamos um RNF a uma especificação de requisitos, forçamos tomadas de decisões que poderão afetar positiva ou negativamente outros RNFs, efeito esse visualizado como interdependências entre RNFs. Estas interdependências podem ser positivas ou negativas, ou seja, um RNF pode influenciar positivamente outro RNF contribuindo assim para sua satisfação, bem como pode influenciar negativamente, ou seja, contribuir para que um destes RNFs não seja satisfeito ou satisfeito apenas parcialmente.

Utilizamos nesta tese a noção de que um RNF raramente pode ser considerado totalmente satisfeito. Em conformidade com o que é utilizado por Mylopoulos [Mylopoulos 92], onde é utilizado o termo “satisfice” ao invés de “satisfy”,

adotaremos daqui por diante a idéia de que um RNF quando dito satisfeito, não necessariamente terá sido totalmente satisfeito, e sim, satisfeito dentro de limites razoáveis de serem utilizados.

Ortogonalmente, podemos classificar o tratamento dispensado a RNFs como divididos em qualitativos e quantitativos. A maioria das abordagens orientada a produto é quantitativa, uma vez que elas propõem métodos quantitativos para medir o quanto um software satisfaz um dado RNF. As abordagens orientadas a processo são, por outro lado, inteiramente voltadas para o aspecto qualitativo, via de regra adotando idéias oriundas do raciocínio qualitativo [AI 84].

RNFs abordam importantes aspectos relacionados à qualidade de softwares. Eles são essenciais para que softwares sejam bem sucedidos. A não observância de RNFs pode resultar em: softwares com inconsistência e de baixa qualidade; clientes e desenvolvedores insatisfeitos; tempo e custo de desenvolvimento além dos previstos devido à necessidade de se consertar softwares que não foram desenvolvidos sob a ótica da utilização de RNFs.

RNFs são geralmente subjetivos, uma vez que podem ser vistos, interpretados e conceituados de forma diferente por diferentes pessoas. É verdade que os requisitos funcionais também sofrem do problema de diferentes conteúdos advindos de diferentes pontos de vista, porém, como os RNFs são por natureza mais abstratos e uma vez que NFRs são comumente relacionados de maneira breve e vaga este problema é potencializado [Chung 00].

Além disso, RNFs freqüentemente interagem entre si, uma vez que a tentativa de satisfazer um RNF pode prejudicar ou ajudar a satisfazer outros RNFs. Como vários RNFs possuem efeitos de natureza global nos softwares, uma solução localizada pode não ser adequada.

Por todas estas razões, RNFs são difíceis de se lidar e vitais de serem tratados para que possamos obter softwares de qualidade.

## 2.3 - Requisitos Funcionais versus Requisitos Não Funcionais

Os *requisitos funcionais* são requisitos que expressam funções ou serviços que um software deve ou pode ser capaz de executar ou fornecer. As funções ou serviços são, em geral, processos que utilizam entradas para produzir saídas.

Os *requisitos não funcionais (RNFs)* são requisitos que declaram restrições, ou atributos de qualidade para um software e/ou para o processo de desenvolvimento deste sistema. Segurança, precisão, usabilidade, performance e manutenibilidade são exemplos de requisitos não funcionais.

A distinção entre o que é um requisito funcional e o que é um não funcional nem sempre é clara. Parte da razão advém para o fato de que os RNFs estão sempre relacionados a um requisito funcional [EAGLE 95] [Chung 00]. A Grosso modo, partindo da definição acima podemos dizer que um requisito funcional expressa algum tipo de transformação que tem lugar no software, enquanto um RNF expressa como essa transformação irá se comportar ou que qualidades específicas ela deverá possuir.

Um exemplo de diferença entre requisito funcional e RNF pode ser visto a seguir.

Suponhamos que estejamos no domínio de laboratórios de análises clínicas: um requisito funcional desse sistema poderia ser expresso da seguinte forma:

“O sistema deve fornecer uma entrada de dados que possibilite a designação de resultados a exames admitidos para um paciente por técnicos, supervisores e chefes”.

Este mesmo requisito funcional poderá ter associado a ele o seguinte RNF:

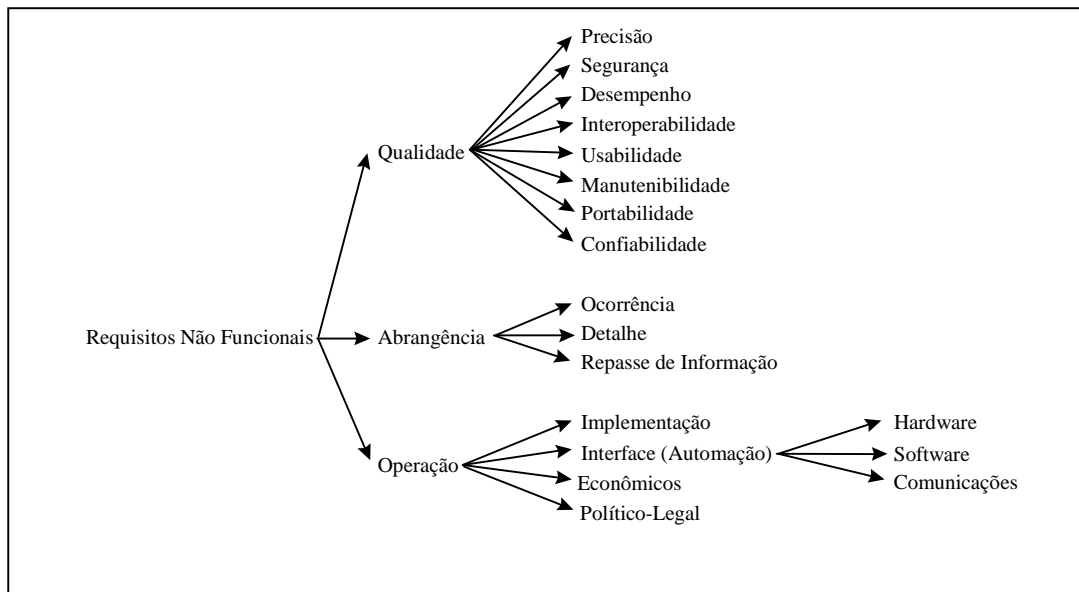
“Alguns exames deverão ter tratamento especial para a entrada de resultados. Para estes exames valores acima ou abaixo de pré-determinados valores só poderão ser digitados por chefes de seção”.

É importante ressaltar que esse último parágrafo não exprime uma função do sistema, e sim, uma restrição a uma função existente, entrar resultados de exames. O que se vê aqui é que essa funcionalidade do sistema deverá ser restrita de forma que, quando utilizada por pessoas que não sejam chefes, será restrita à condição de que essas pessoas estejam digitando um valor que se encontre dentro de limites pré-estabelecidos.

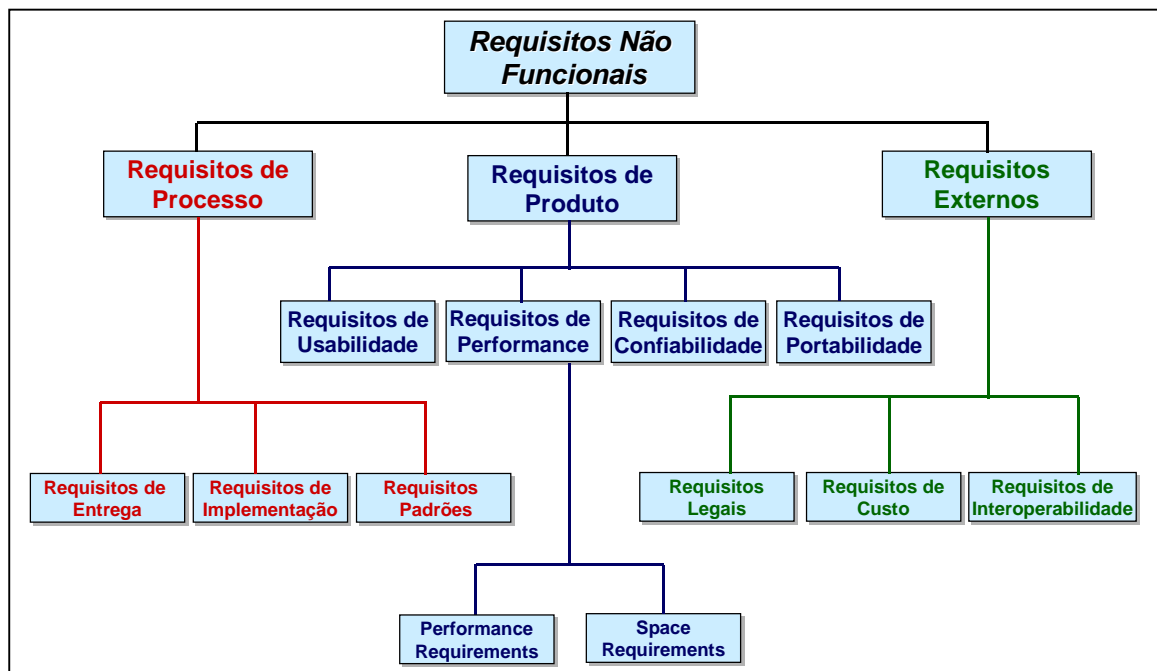
É visível que este RNF irá demandar um processo de segurança no tratamento de acesso a módulos do software, bem mais complexo daquele que seria implantado se apenas o requisito funcional tivesse sido especificado. A implementação de um processo de acesso a módulos, que critique não apenas a permissão de acesso ao se entrar em um módulo de software, mas também o conteúdo do que está sendo digitado dentro do módulo, é consideravelmente mais complexa. A tardia ou não elicitação deste requisito certamente implicaria num grande “retrabalho”.

## **2.4 - Classificação de RNFs**

Algumas classificações dos RNFs podem ser encontradas na literatura. Mamani propõe a classificação de RNFs apresentada na Figura 2.1 [Mamani 99] enquanto a Figura 2.2 apresenta a classificação de RNFs proposta por Sommerville [Sommerville 92] e a Figura 2.3 mostra uma classificação proposta por Boehm [Boehm 76] na qual ele define uma árvore mínima de padrões de qualidade que um software deveria apresentar.



**Figura 2.1 - Classificação de RNFs proposta por Mamani**

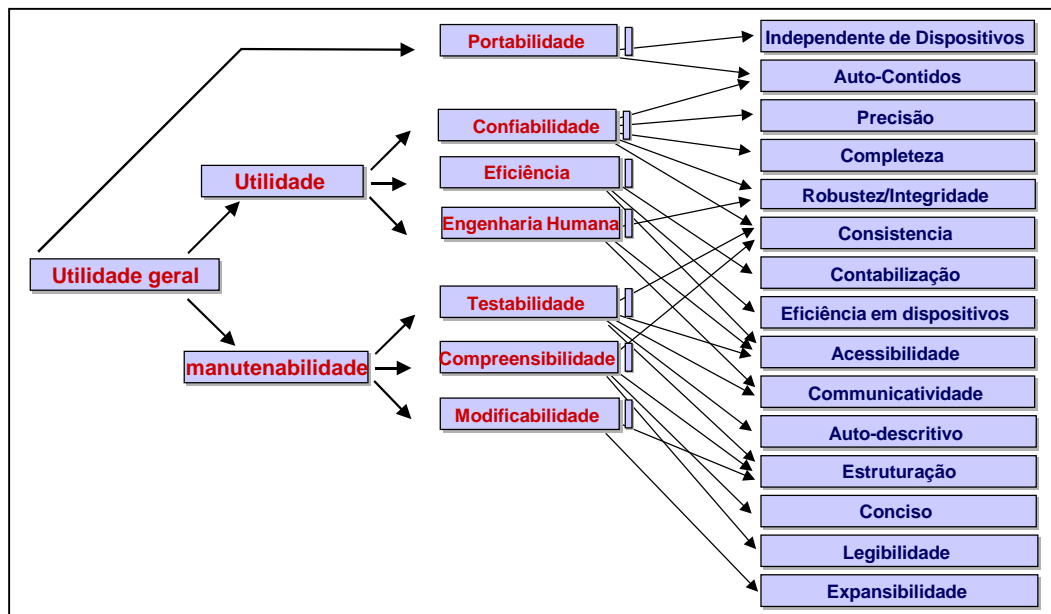


**Figura 2.2 - Classificação de RNFs segundo Sommerville**

Uma outra fonte que classifica alguns RNFs é o padrão internacional ISO 9126 [ISO9126]. Nesta norma são descritas seis características que definem a qualidade de software: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. Se observarmos as diversas definições de RNF, facilmente iremos considerar que estes requisitos de qualidade definidos na ISO 9126 são, em geral,



RNFs. O item funcionalidade é, porém, uma exceção que certamente não poderá ser enquadrado como RNF.



**Figura 2.3 - Árvore de Características de Qualidade de Software (Boehm 76)**

Proporemos aqui uma taxonomia que classifica os RNFs em primários e específicos. RNFs primários são aqueles mais abstratos que representam propriedades como: Confiabilidade, Rastreabilidade, Precisão, Segurança. Já os RNFs específicos são decomposições que se seguem aos RNFs primários e tendem a diminuir o nível de abstração de um determinado RNF, e portanto, atingir um nível de granularidade no tratamento da informação mais detalhado.

Um exemplo desta classificação seria o RNF Primário *Confiabilidade* que pode ser decomposto em validação, autorização e entrega do software.

RNFs primários podem ser decompostos em outros RNFs secundários até que se chegue ao que Chung [Chung 00] chama de operacionalização dos RNFs. Uma operacionalização de um RNF é uma ação ou informação que irá garantir a satisfação do RNF. Por exemplo, no caso de um sistema de informação para laboratórios de análises clínicas, a entrega do laudo ao paciente possui um RNF de *confidencialidade*

que seria seu RNF primário. Para que possamos detalhar o que isso implica, temos de decompor este RNF em um RNF específico de *segurança operacional* que por sua vez poderia ser decomposto na operacionalização *solicitar carteira de identidade*. Ou seja, para satisfazer o RNF *confidencialidade* teríamos de prever uma ação no sistema, que garantisse que o funcionário que entregou o laudo solicitou a carteira de identidade ao paciente antes de entregar-lhe o laudo.

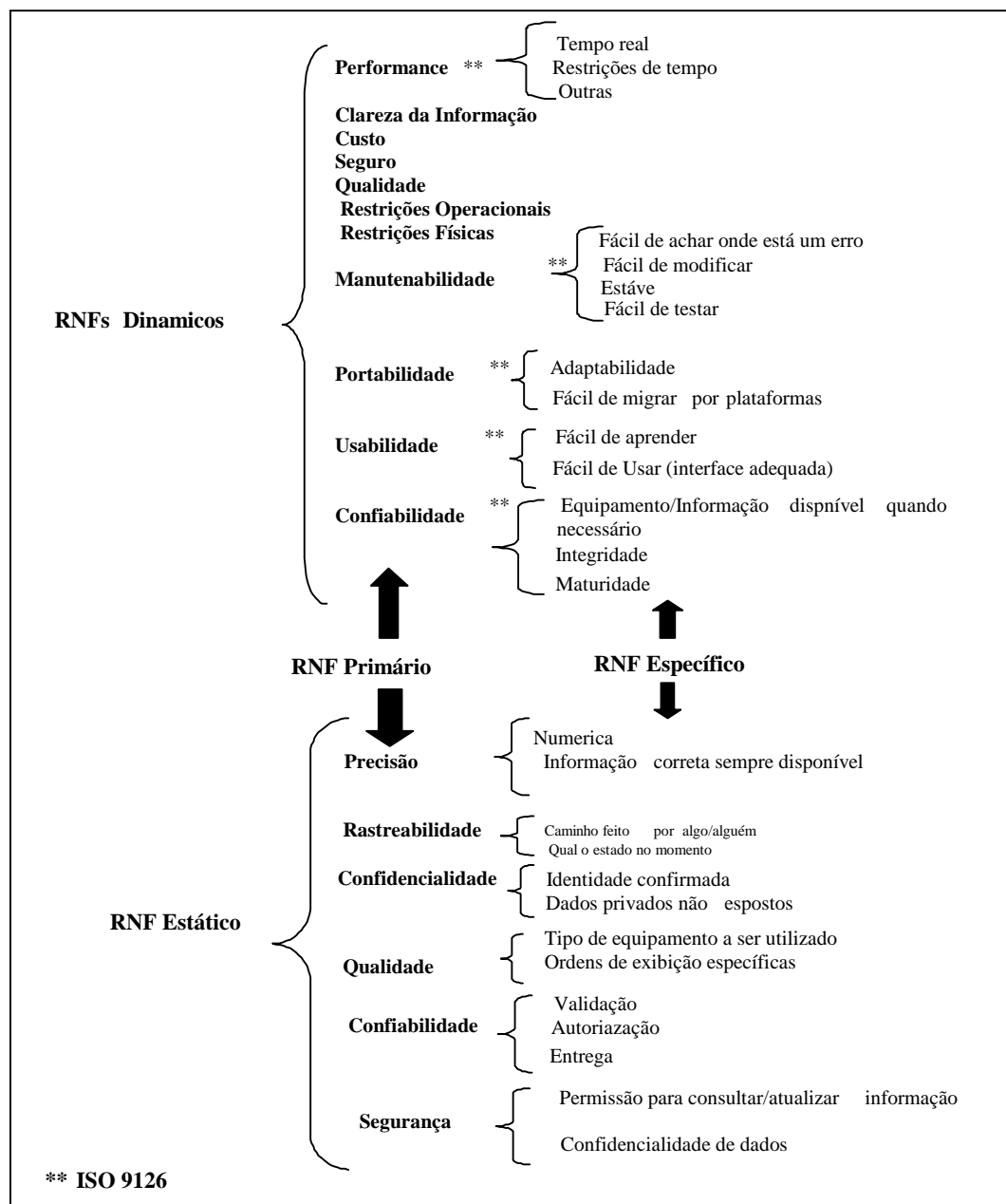


Figura 2.4 – Uma Taxonomia para RNFs

Ortogonalmente, separamos os RNFs em estáticos e dinâmicos. RNFs estáticos são aqueles que, quando presentes, **normalmente requerem o uso de dados** para validá-los como, por exemplo: segurança, precisão e rastreabilidade. RNFs dinâmicos, por outro lado, usualmente representam conceitos mais abstratos, **que normalmente envolvem ações ou critérios de qualidade** como, por exemplo: Qualidade, performance, manutenibilidade, restrições operacionais. Alguns RNFs podem ser tanto estáticos quanto dinâmicos dependendo do contexto do domínio que estejam inseridos. A Figura 2-4 mostra um resumo desta taxonomia que também pode ser utilizada como um checklist para a elicitação de RNFs.

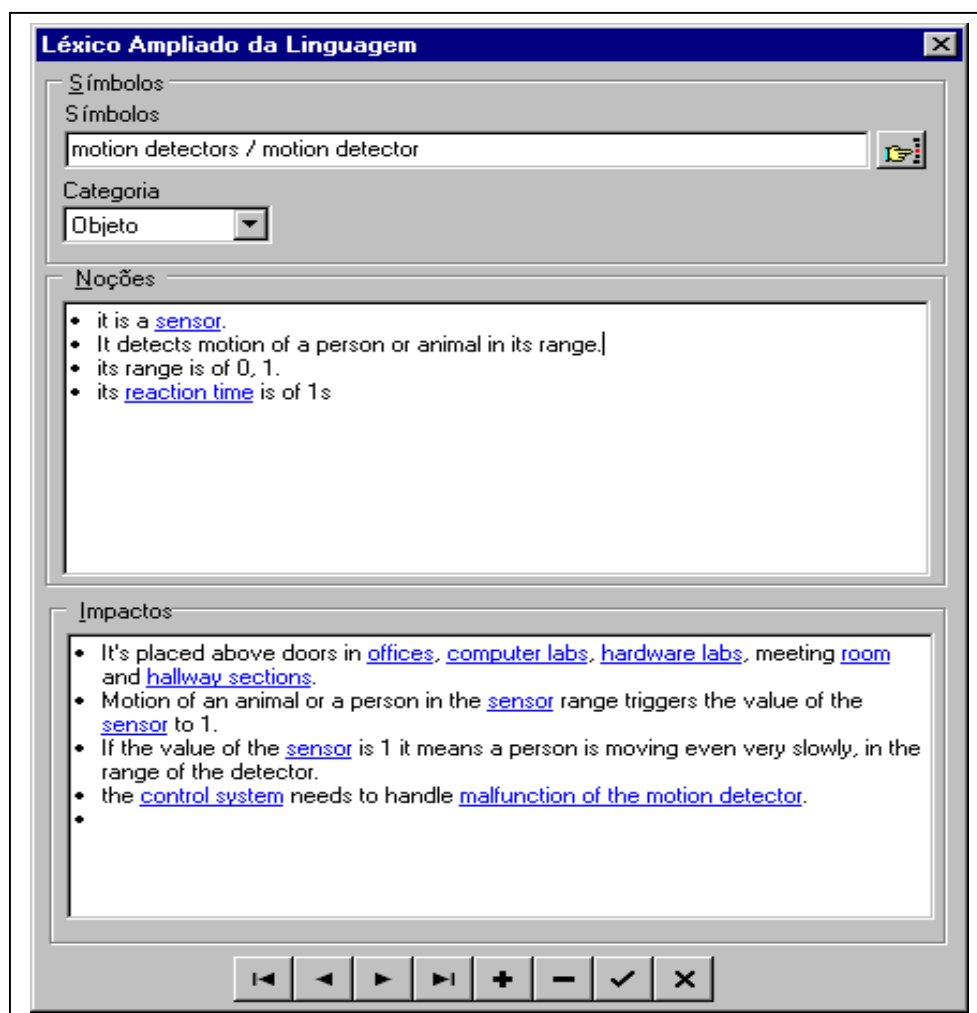
A lista apresentada na figura acima não pretende ser completa, mas sim, um ponto de partida para cada desenvolvedor desenvolver seu próprio conhecimento a respeito de RNFs. Uma lista mais exaustiva de RNFs pode ser encontrada em [Chung 00].

## 2.5 - O Léxico Ampliado da Linguagem

O principal objetivo do Léxico Ampliado da Linguagem (LAL) é registrar a linguagem utilizada pelos atores do UdI, sem contudo se preocupar com a funcionalidade [Franco92][Leite93]. O LAL do UdI é composto por entradas, onde cada entrada está associada a um símbolo (palavra ou frase) da linguagem do UdI. Cada símbolo pode possuir sinônimos e é descrito através de noções e impactos. As noções descrevem o significado e as relações fundamentais de existência do símbolo com outros símbolos (denotação). Os impactos descrevem os efeitos do uso ou ocorrência do símbolo no UdI ou os efeitos de outras ocorrências de símbolos no UdI sobre o símbolo (conotação). Dependendo do símbolo que descrevem, as entradas podem ser classificadas como sujeito, verbo, objeto e estado (predicativo).

Ao se descrever entradas do LAL, deve-se obedecer aos princípios de vocabulário mínimo e de circularidade. O princípio de vocabulário mínimo demanda que a

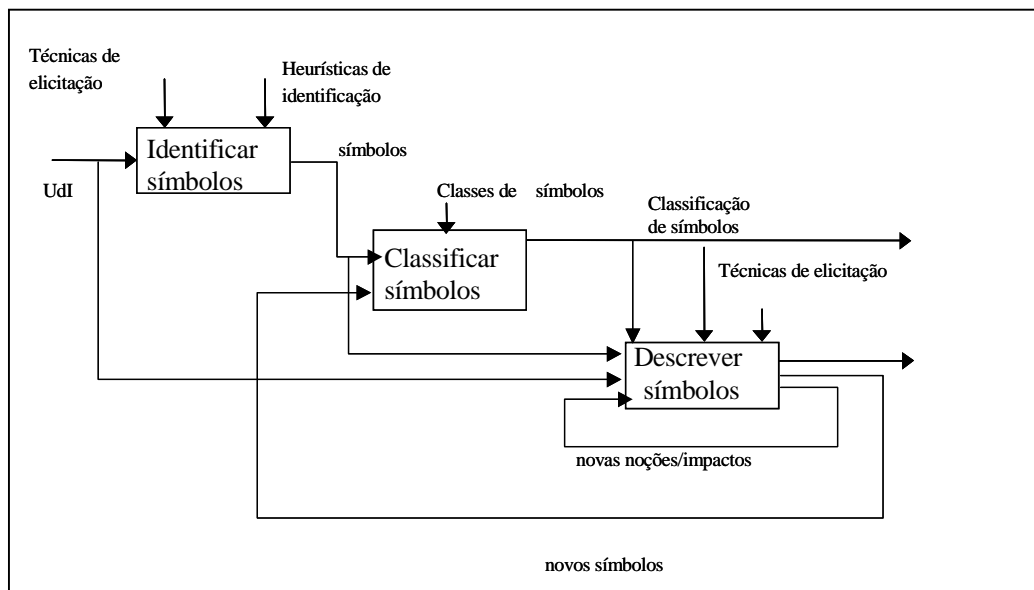
utilização de símbolos externos ao LAL do UdI seja minimizada ao máximo. O princípio de circularidade implica na maximização da utilização de símbolos do LAL do UdI na descrição de símbolos. Os símbolos do LAL, que aparecem na descrição de símbolos, devem ser sublinhados. Como decorrência do princípio de circularidade, a forma natural de representação do LAL é pela utilização de uma estrutura de hipertexto. As entradas do LAL são os nós do hipertexto, enquanto que os símbolos que aparecem nas descrições de símbolos, são os elos do hipertexto. A Figura 2-5 apresenta uma entrada do LAL para um sistema de controle automático de luzes. As palavras sublinhadas são elos para outros símbolos do LAL.



**Figura 2.5 - Exemplo do LAL de um Sistema de Controle de Luzes**

O processo de construção do LAL do UdI [Neto 00] (figura 2.6) engloba as seguintes atividades:

- O primeiro passo no processo de construção do LAL consiste na identificação dos símbolos. Para tal, devemos identificar as palavras ou frases da linguagem praticada no UdI que pareçam ter um significado especial no UdI. Em geral, estas palavras ou frases são utilizadas com frequência por atores do UdI ou outras fontes de informação. Palavras ou frases que parecem sem sentido, ou fora de contexto, têm grande probabilidade de serem símbolos do UdI. As técnicas de elicitación mais adequadas para esta atividade são a observação, leitura de documentos e entrevista não estruturada. As heurísticas de identificação se resumem a procurar sujeitos, verbos, objetos e estados (predicativos) presentes em frases que surgem no UdI e que possuem um significado especial no UdI. Antes de utilizar as heurísticas, deve-se separar os períodos compostos em orações simples, transportar orações na voz passiva para a voz ativa e transformar as formas substantivas de verbos para a forma verbal correspondente. Observou-se que a transformação das formas substantivas de verbos para formas verbais correspondentes é extremamente importante para que se possa utilizar as heurísticas de construção de cenários a partir do LAL do UdI, propostas por Hadad [Hadad97].
- O segundo passo deste processo consiste na classificação dos símbolos. Neste passo devemos determinar se os símbolos são sujeitos, verbos, objetos e estados (predicativos) de frases que surgem no UdI.



**Figura 2.6– Processo de Construção do LAL**

- Cada entrada tem zero ou mais sinônimos.
- Cada entrada tem uma ou mais noções.
- Cada entrada tem um ou mais impactos.
- Escreva noções e impactos usando frases simples, que expressam uma única idéia.
- A descrição de noções e impactos deve respeitar os princípios de circularidade e vocabulário mínimo.
- Para uma entrada que descreve um símbolo classificado como sujeito:
  - ♦ As noções devem dizer quem é o sujeito (pessoa ou coisa) do qual se diz alguma coisa;
  - ♦ Os impactos devem registrar as atividades que o sujeito realiza;
  - ♦ Os sinônimos devem conter termos que tenham o mesmo sentido do símbolo no UdI.
- Para uma entrada que descreve um símbolo classificado como verbo:
  - ♦ As noções devem descrever sucintamente a ação e dizer quem é o sujeito (pessoa ou coisa) que realiza a ação, quando a ação acontece e quais as informações necessárias para realizar a ação;
  - ♦ Os impactos devem registrar os procedimentos que fazem parte da ação, as condições/situações decorrentes da ação e outras ações que deverão ocorrer;
  - ♦ Os sinônimos devem registrar a forma substantivada, a voz passiva e a forma infinitiva do verbo.
- Para uma entrada que descreve um símbolo classificado como objeto:
  - ♦ As noções devem definir o ser (pessoa ou coisa) que sofre ações;
  - ♦ Os impactos devem descrever as ações que podem ser aplicadas ao objeto;
  - ♦ Os sinônimos devem conter termos que tenham o mesmo sentido do símbolo no UdI.

**Figura 2.7 - Heurísticas de representação de entradas de LAL**

- A última etapa do processo resume-se à descrição dos símbolos. Para realizarmos esta etapa temos de criar entradas do LAL referentes aos símbolos identificados. Para criar uma entrada do LAL para um símbolo, deve-se descrever noções e impactos, seguindo as heurísticas de

representação. Algumas destas heurísticas de representação são mostradas na figura 2.7. A utilização destas heurísticas deve ser observada ao se construir o LAL, para que as heurísticas de construção de cenários a partir do LAL, propostas por Hadad [Hadad97], possam ser utilizadas. Nesta atividade, identificam-se os sinônimos dos símbolos descritos por entradas do LAL. Eventualmente, são identificados novos símbolos nesta atividade, bem como novas noções e impactos de outros símbolos. As entradas do LAL assumem as mesmas classes - sujeito, verbo, objeto e estado (predicativo) – dos símbolos que descrevem.

Conforme mostrado na Figura 2.6, o processo de construção do LAL é continuado sendo retomado sempre que alterações surgirem no UdI que tragam novos símbolos para o domínio ou pela descoberta de novos símbolos durante a elicitacão de um dado símbolo (princípio da circularidade).

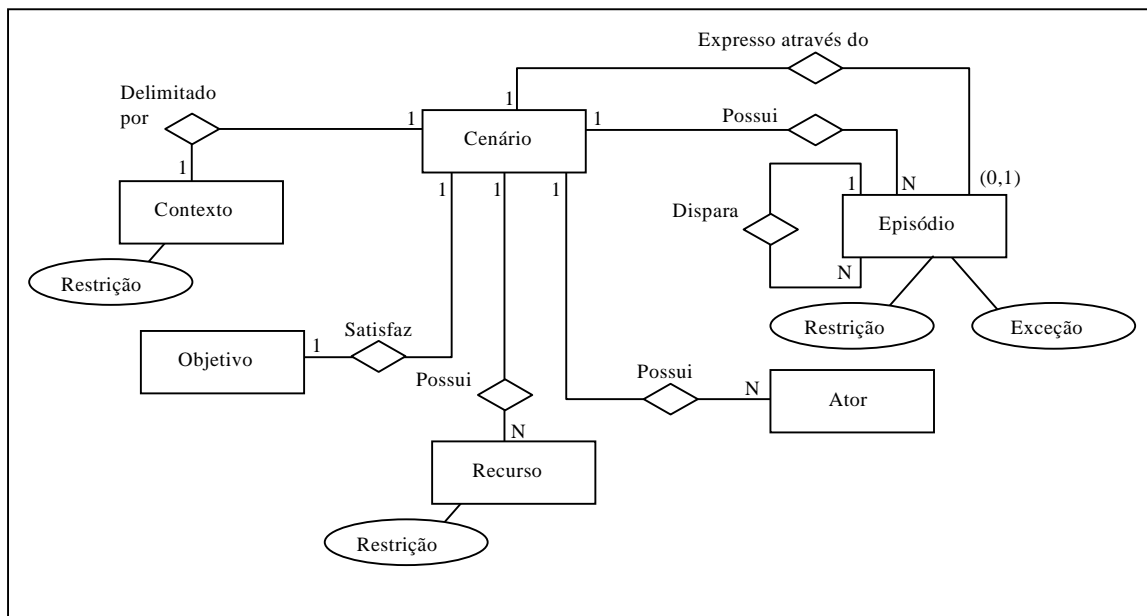
## 2.6 - Cenários

Cenários são construídos de diversas formas na literatura, indo da forma narrativa até a forma de protótipos escritos [Carrol 95] [Booch 91] [Jacobson 92] [Zorman 95] [Potts 94]. Nesta tese utilizaremos a abordagem proposta por Leite [Leite 97], onde os pontos centrais desta proposta são [Breitman 98]:

- Cenários descrevem situações que ocorrem no macrosistema e suas relações com o sistema;
- Cenários evoluem durante o processo de desenvolvimento de software;
- Cenários estão naturalmente ligados ao Léxico ampliado da linguagem (LAL do UdI);
- Cenários são descritos em linguagem natural.

O modelo de cenários visa representar aspectos comportamentais dos requisitos e faz parte de uma proposta maior, o *Requirements Baseline* [Leite97][Leite95]. A

Figura 2.8 mostra um diagrama de entidades e relacionamentos para a estrutura de cenários proposta por Leite [Leite97], enquanto a Figura 2.9 explica cada uma das entidades do cenário, mostradas na Figura 2.8, bem como a sintaxe<sup>2</sup> utilizada para descrever estas entidades.



**Figura 2.8 – Diagrama Entidade-Relacionamento Descrevendo Cenários**

O relacionamento "*expresso através do*" entre as entidades cenário e episódio, indica que um episódio pode ser expresso ou explicado através de um cenário, possibilitando a decomposição de cenários em sub-cenários. O atributo restrição indica, via de regra, requisitos não funcionais que se referem às entidades contexto, recurso e episódio do modelo ER. O atributo exceção indica falta ou mau funcionamento de um recurso necessário para se atingir o objetivo do cenário e deve ser representado por frases curtas ou pequenos parágrafos.

<sup>2</sup> + indica composição.  $n\{x\}m$  indica no mínimo  $n$  e no máximo  $m$  ocorrências de  $x$ .  $()$  indica agrupamento.  $|$  indica **ou**.  $[x]$  indica que  $x$  é opcional.



- **Título:** título do cenário, que serve também para identificá-lo. Em caso de sub-cenário, o título é o mesmo do episódio que o sub-cenário explica, sem as restrições e exceções.  
**Sintaxe:** frase | ( [ Ator | Recurso ] + verbo + predicado )
- **Objetivo:** meta que o cenário pretende atingir. O cenário descreve o alcance de um objetivo.  
**Sintaxe:** [ sujeito ] + verbo + predicado
- **Contexto:** localização geográfica/temporal e estado inicial do cenário  
**Sintaxe:** localização + estado, onde  
localização é nome,  
estado é ( [ ator | recurso ] + verbo + predicado + 0{restrições}n ) +  
0{ ([ e | ou ]+ [ ator | recurso ] + verbo + predicado + 0{restrições}n ) }n
- **Recursos:** meios de suporte, dispositivos e outras entidades passivas utilizadas pelos atores do cenário  
**Sintaxe:** 1 {nome + restrições}n , onde  
restrições é ( “**Restrição:** deve ter” + 1 {RNF específico}n )
- **Atores:** pessoas ou estruturas organizacionais que tem um papel no cenário  
**Sintaxe:** 1 {nome}n
- **Episódios:** ações que detalham o cenário e fornecem seu comportamento  
**Sintaxe:** episódios :: = < série >  
série :: = < sentença > | < série > < sentença >  
< sentença > :: = < sentença seqüencial > | < sentença não seqüencial > |  
                    < sentença condicional > | < sentença optativa >  
< sentença seqüencial > :: = < sentença episódio >  
< sentença não seqüencial > :: = # < série > #  
< sentença condicional > :: = Se < condição > , então < sentença episódio >  
< sentença optativa > :: = [ < série > ]  
< sentença episódio > :: = [ator | recurso] + verbo + predicado + restrição + 0{exceção}n,  
onde:  
restrição é (“**Restrição:** ” + 1 {recurso }n + “deve(m) ter ”  
                    + 1 {RNF específico}n  
                    + “, sendo a estratégia de satisfação ”+  
                    estratégia de satisfação)

**Figura 2.9 - Esquema para Descrição de Cenários**

A Figura 2.10 mostra um exemplo da linguagem de descrição de cenários utilizando a ferramenta OORNF para descrever um cenário para a definição de um esquema de luzes em um sistema para controle de iluminação ambiente. As palavras que aparecem sublinhadas são símbolos do LAL utilizados para descrever o cenário.

**Figura 2.10– Exemplo da Linguagem de Descrição de Cenários**

### 2.6.1 - Construção de cenários a partir do LAL do UdI

Hadad propõe uma estratégia baseada no uso do LAL para gerar cenários [Hadad97] [Hadad99]. Esta estratégia baseia-se em algumas heurísticas para realizar tal construção, descritas a seguir:

- **Passo 1: Identificar atores do UdI.**

Identificar os símbolos do LAL do UdI classificados como sujeito. Estes símbolos representam os atores do UdI, os quais são classificados em primários (atores que realizam ações diretas sobre o sistema) e secundários (atores que recebem/fornecem informações para o sistema, mas não executam ações diretas sobre a aplicação).

- **Passo 2: Identificar cenários candidatos.**

Recuperar os impactos das entradas do LAL do UdI associadas aos atores do UdI, os quais foram obtidos no passo 1. Eliminar os impactos repetidos. Os impactos

resultantes são os títulos dos cenários candidatos. Primeiramente obtemos os impactos das entradas do LAL do UdI associadas a atores principais e, em seguida, o mesmo é realizado para as entradas do LAL do UdI associadas aos atores secundários.

- **Passo 3: Descrever cenários candidatos.**

Para cada cenário candidato identificado no passo anterior, faça:

- ◆ Se o título do cenário candidato possui um símbolo do LAL do UdI classificado como verbo, então:
  1. Recuperar a entrada do LAL do UdI referente ao símbolo classificado como verbo.
  2. Definir o objetivo do cenário com base no seu título e nas noções da entrada recuperada.
  3. Verificar se existe alguma ordem de precedência entre o impacto que originou o cenário com os outros impactos. Caso exista, incluir no contexto o impacto que o precede como uma pré-condição.
  4. A partir de cada impacto da entrada, definir os episódios do cenário. Os episódios podem ser reescritos, utilizando a sintaxe para episódios, que permite expressar comportamento. O mesmo não é possível nos impactos de entradas de LALs do UdI.
  5. Identificar atores e recursos, os quais devem ser entradas do LAL do UdI classificadas respectivamente como sujeito e objeto.
- ◆ Se o título do cenário candidato não possui um símbolo do LAL do UdI classificado como verbo, então:
  1. Identificar os símbolos do LAL do UdI presentes no título do cenário candidato.

2. Definir o objetivo do cenário com base no seu título.
3. Verificar se existe alguma relação de precedência entre o impacto que originou o cenário com outros impactos. Caso exista, inclui-se no contexto o impacto que o precede como pré-condição.
4. Definir os atores e recursos com base nos símbolos recuperados. Os atores são entradas do LAL do UdI classificadas como sujeito. Os recursos são entradas do LAL do UdI classificadas como objeto. Neste caso, não se definem episódios a partir do LAL do UdI.

- **Passo 4: Completar os cenários com informações provenientes do UdI.**

Consiste em buscar informações adicionais, com os atores ou outras fontes de informação do UdI, para completar partes de cenários que não puderam ser preenchidas com as informações fornecidas pelo LAL do UdI. Em geral, este passo se aplica mais diretamente aos cenários para os quais não se definiram episódios.

- **Passo 5: Analisar cenários.**

Consiste em validar e verificar cenários. Os cenários são validados com os clientes a fim de identificar erros, omissões e/ou ampliar informações de episódios. Esta ampliação de informações de episódios engloba a substituição de um episódio por vários episódios no mesmo cenário e a descrição de episódios através de sub-cenários. A tarefa de verificação de cenários consiste em:

- ◆ **Unificar cenários.** Dois ou mais cenários, que possuem os mesmos episódios ou o mesmo objetivo e contexto, são agrupados em um único cenário. Quando necessário, utiliza-se a forma condicional para descrever episódios diferentes.
- ◆ **Detectar sub-cenários.** Se, em cenários provenientes de atores principais, um conjunto de episódios corresponde a um cenário

proveniente de um ator secundário, substituir aqueles episódios por este cenário.

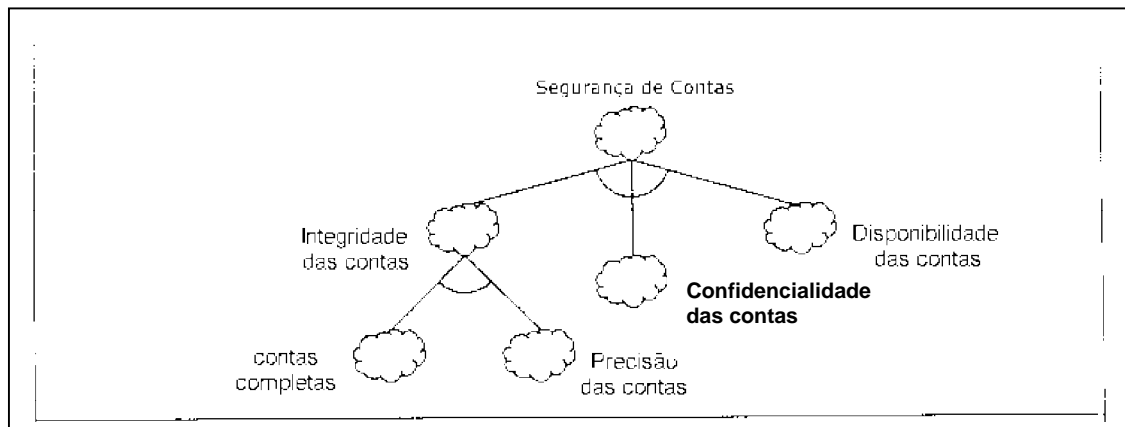
## 2.7 - Grafo de RNFs

A elicitação de RNFs demanda o uso de uma notação para representá-los de alguma forma para que a informação não se perca no tempo. Além disso, esta notação deve de alguma forma possibilitar que lidemos com os RNFs de uma maneira organizada e que facilite a lidarmos com as interdependências. Como propomos que os RNFs sejam elicitados desde as primeiras etapas do processo de desenvolvimento de software, necessitamos de uma maneira de representar estes RNFs.

Para representarmos os RNFs iremos utilizar o grafo de RNFs proposto por Chung [Chung 95] [Chung 00] com pequenas modificações que serão apresentadas no capítulo 4. Segundo o Framework apresentado por Chung, RNFs devem ser representados como *metas* a serem satisfeitas. Cada meta será decomposta em subseqüentes submetas até encontrarmos o que são chamadas de operacionalizações. Uma operacionalização corresponde a ações ou atributos que claramente identifiquem o que é necessário para satisfazer a meta principal.

Chung divide a decomposição dos RNFs em tipo e tópico. Na decomposição por **tipo** são usados fundamentalmente os métodos de decomposição existentes no framework, que indicam quais são as decomposições possíveis para um determinado RNF. No caso do RNF Segurança, por exemplo, integridade, confidencialidade e disponibilidade podem ser possíveis decomposições desta meta. A Figura 2.11, extraída de [Chung 00], mostra um exemplo da decomposição do RNF *Segurança* aplicado à conta corrente como parte de um estudo de caso de automação bancária. Neste exemplo pode-se ver que O RNF *Segurança* quando analisado sob a ótica de

contas correntes apresenta 3 possíveis decomposições *em integridade, confidencialidade e disponibilidade*. Percebe-se ainda, que a submeta *integridade* é novamente decomposta em submetas de *completeza e precisão*. As outras submetas não são aqui decompostas para tornar o grafo simples de ser visualizado ao nível de exemplo. Em uma abordagem completa todas seriam decompostas.

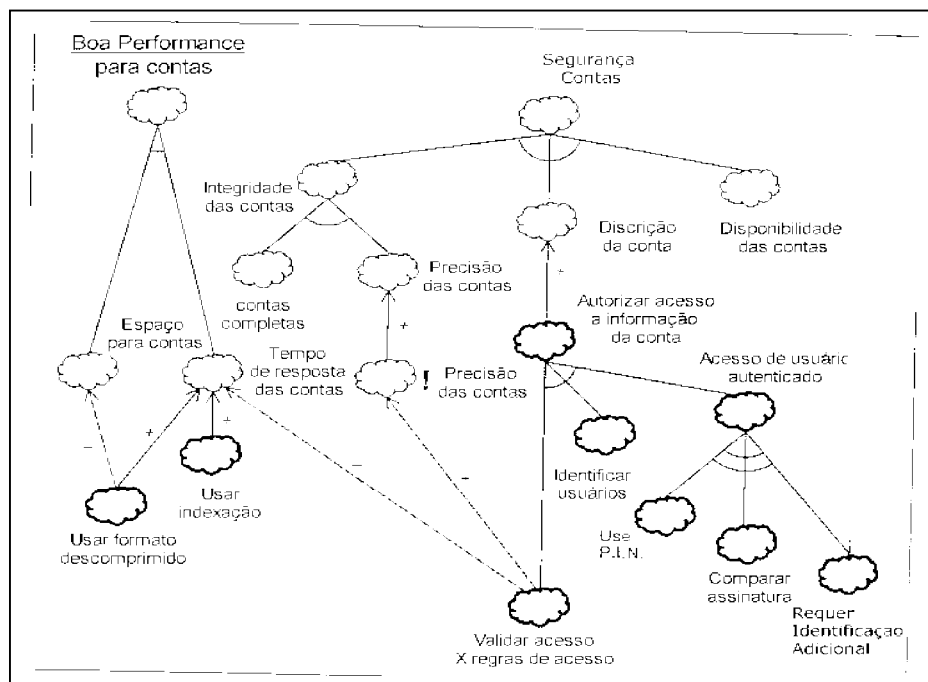


**Figura 2.11– Exemplo de Decomposição do Grafo de RNF**

A decomposição por **tópicos** trata da decomposição estrutural do problema. Por exemplo, no exemplo acima, o RNF Seguro aplicado a contas poderia ter sido decomposto em contas pessoa física e jurídica.

A Figura 2.12, extraída de [Chung 00], mostra um exemplo de refinamento até encontrarmos as necessárias e suficientes operacionalizações. Nesta figura podemos observar que, desta vez, a submeta decomposta é a de *confidencialidade*. Esta submeta é refinada na submeta *necessidade de autorização para acesso a informações sobre a conta*. Vemos então, que esta submeta é novamente decomposta em *Validar Acesso, Identificar Usuários e Acesso de Usuário Autenticado*, onde esta última é ainda decomposta em *Use P.I.N., Comparar assinatura e Requer Identificação Adicional*. No contexto do desenvolvimento deste sistema, estes últimos refinamentos foram considerados suficientes para expressar as ações e dados necessários para

satisfazer ao RNF *Segurança*. O momento de parar a decomposição é uma decisão pessoal e não existem regras fixas que determinem um limite.

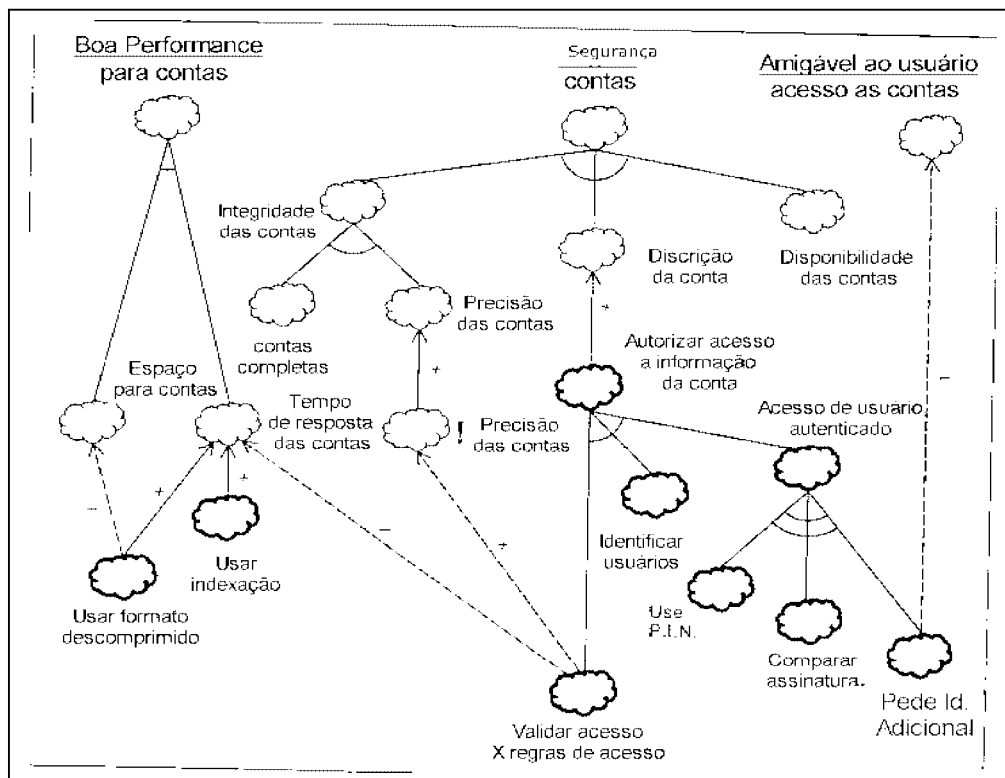


**Figura 2.12– Grafo de RNFs Decomposto Até Suas Operacionalizações**

É importante observar que todas as três submetas de *Autorizar Acesso a Informação da Conta* estão ligadas por um arco. Isto denota que as três são contribuições do tipo E, ou seja, para que a submeta seja satisfeita todas as três submetas terão de ser satisfeitas.

Por sua vez, a submeta de *Acesso de usuário Autenticado* possui outras três submetas que são ligadas por um arco duplo que desta feita denota uma contribuição do tipo OU, ou seja, se uma das três submetas for satisfeita a submeta imediatamente superior será satisfeita.

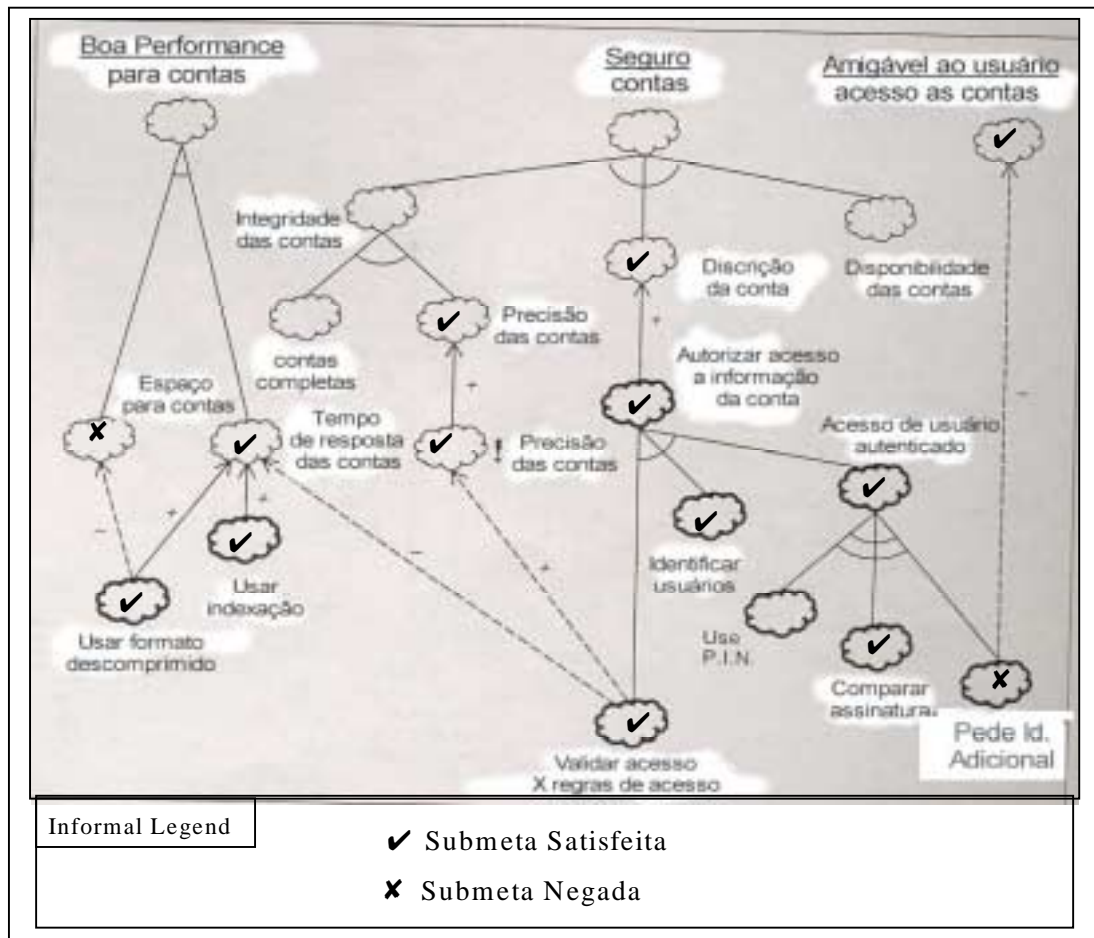
Neste novo grafo aparece ainda um outro RNF *Boa Performance* que também é decomposto até acharmos as operacionalizações: *Usar formato descomprimido* e *Usar indexação*.



**Figura 2.13– Exemplo de Grafo com Interdependências**

Uma análise mais detalhada deste grafo irá revelar algumas interdependências entre submetas, algumas positivas representadas com um sinal + e outras negativas, representadas com o sinal -. A figura 2.13 mostra estas interdependências. Nela, podemos observar que a operacionalização *Validar Acesso X regras de elegibilidade* contribui positivamente para a satisfação da submeta *Precisão* e negativamente para a Submeta *Tempo de Resposta das contas* do RNF *Boa Performance para contas* correntes. Podemos ainda observar um impacto negativo da operacionalização *Pede Id. Adicional* no RNF *Amigável ao Usuário* no acesso a contas correntes. Esta contribuição negativa é representada como uma possível futura contribuição negativa e adveio da consulta a uma base de conhecimentos de interdependências entre RNFs.





**Figura 2.14 – Grafo de RNFs com Interdependências Analisadas**

Uma vez que detectamos interdependências negativas temos que lidar com os conflitos que elas representam, ou seja, avaliar quais metas e submetas deverão não ser satisfeitas ou satisfeitas apenas parcialmente, ou ainda, se apesar das contribuições negativas todas as metas e submetas podem ser mantidas. A Figura 2.14 mostra o grafo acima após o engenheiro de requisitos ter tomado as decisões necessárias. Neste grafo o símbolo ✓ representa que esta operacionalização foi escolhidas para serem implementadas enquanto o X, representa operacionalizações que foram rejeitadas devido a seus impactos negativos em outras metas de maior importância relativa.

## 2.8 - Unified Modeling Language

A Unified Modeling Language (UML) é uma linguagem de modelagem visual de propósito geral que é utilizada para especificar, visualizar, construir e documentar artefatos de um software. Ela captura decisões e conhecimentos sobre o software a ser construído. A UML é para ser utilizada para compreender, desenhar, configurar, manter e controlar informações sobre o software sendo desenvolvido. Em momento algum ela é atrelada com algum método de desenvolvimento, ciclos de vida de sistema ou domínios específicos, e portanto, pretende suportar a maioria dos processos de desenvolvimento de software orientados a objeto existentes [Rumbaugh 99].

Rumbaugh [Rumbaugh 99] divide a UML em áreas que por sua vez contêm visões, onde uma visão é um subconjunto de construtores da UML que representam um aspecto de um sistema. A UML fica então dividida em três áreas: estrutural, comportamento dinâmico e gerência de modelos.

A classificação estrutural descreve coisas existentes nos sistemas e seus relacionamentos com outras coisas. Classificadores incluem, entre outros, classes, ator, componentes, interface, tipo de dado e nós, podendo ser encontrados nas visões: estática, casos de uso e de implementação.

A área de comportamento dinâmico descreve o comportamento de um sistema no tempo. Comportamento pode ser descrito como uma série de fotografias de mudanças no desenho do sistema retirados da visão estática. Esta área inclui as visões: máquinas de estado, atividade e interação.

A gerência de modelos descreve a organização dos modelos propriamente ditos através de unidades hierárquicas. Um pacote é uma unidade organizacional genérica para modelos. Pacotes especiais incluem modelos e subsistemas. A visão de gerência

cruza as outras visões e organiza-as objetivando o trabalho de desenvolvimento e controle de configuração.

Nesta tese estaremos especialmente voltados para as visões estáticas e de interação.

### **2.8.1 - Visão Estática**

A visão estática modela tanto conceitos pertinentes ao domínio da aplicação quanto conceitos inventados como parte da solução proposta para a implementação desta aplicação [Rumbaugh 99]. Esta visão é denominada de estática por não refletir os comportamentos dependentes de tempo do software, os quais são descritos em outras visões. Os principais participantes desta visão estão agrupados no diagrama de classes e são classes e seus relacionamentos, a saber: associação, generalização e vários outros tipos de dependência. A visão estática é mostrada através do diagrama de classes.

Um diagrama de classes descreve os tipos de objetos de um sistema e vários tipos de relacionamentos estáticos que existem entre eles. Existem dois tipos de relacionamentos que se destacam [Fowler 97]:

- Associações: um cliente pode alugar um certo número de vídeos;
- Subtipos: Uma enfermeira é um tipo de pessoa.

Diagramas de classe mostram também atributos e operações de uma classe e as restrições que se aplicam a como os objetos são interligados.

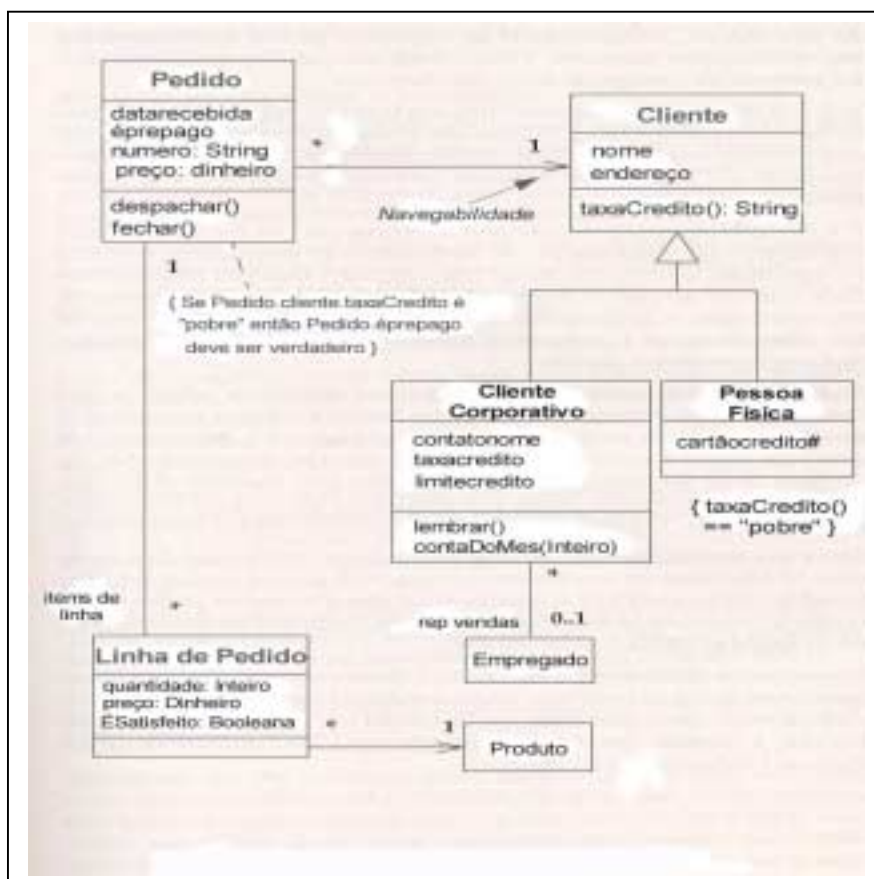
Fowler [Fowler 97] destaca ainda que um diagrama de classes pode ser construído utilizando-se três diferentes perspectivas:

1. Conceitual: representa basicamente conceitos do domínio em estudo e em geral é independente de linguagem ou arquitetura de implementação;

2. Especificação: representa um modelo já preocupado com a implementação do problema sob forma de um software, onde soluções de desenho são implementadas como, por exemplo, o uso de padrões [Gamma 94];
3. Implementação: Aqui todos os detalhes de tratamento de classes são utilizados e o desenho passa a ser fortemente dependente de software e arquitetura utilizados.

Classes são descritas utilizando-se um retângulo com três diferentes compartimentos onde são mostrados respectivamente, o nome da classe, seus atributos e as operações desta classe. A representação de atributos e operações é comumente suprimida das classes exceto em um único diagrama.

Relacionamentos entre classes são representados como caminhos conectando duas classes (ou uma classe consigo mesma). Os diferentes tipos de relacionamentos são distinguidos pela textura da linha e adereços nos caminhos e terminações destes [Rumbaugh 99]. A Figura 2.15, extraída de [Fowler 97], mostra um exemplo onde aparecem os diversos componentes do diagrama de classes. É necessário ressaltar que o que aparece entre { } são restrições impostas ao modelo, que podem obter a forma tanto de uma expressão formal quanto de uma sentença em linguagem natural.



**Figura 2.15 – Exemplo de um Diagrama de Classes em UML**

Podemos ver no exemplo mostrado na figura 2.15 que o diagrama lá mostrado indica uma associação entre as classes pedido e cliente onde o lado da associação referente ao cliente possui uma multiplicidade 1 e a de pedido uma multiplicidade \*, indicando que um pedido tem de vir de apenas um cliente, mas um cliente pode realizar diversas ordens. As multiplicidades mais comuns são 1, \*, 0..1 (podemos ter zero ou um). Esta associação especificamente é representada por uma seta que indica a navegabilidade desta associação. Neste caso significaria que um pedido tem a responsabilidade de dizer a quais clientes ele pertence, enquanto um cliente não tem a capacidade de dizer quais pedidos possui.

Uma associação pode ainda ter um papel associado a ela, como por exemplo, a classe empregado da Figura 2.15 que possui o papel de representante de vendas.

É possível ainda, verificar na Figura 2.15 que a classe cliente é uma generalização das sub-classes cliente corporativo e pessoa física. Isto significa que as sub-classes

irão herdar os atributos (nome e endereço) e operações (taxaCredito()) contidos na classe cliente, além de ter os seus próprios atributos e operações.

### **2.8.2 - Visão de Iteração**

Esta visão descreve seqüências de trocas de mensagens entre papéis que espelham o comportamento do software. Um classificador do papel é uma descrição de um objeto que desempenha uma atividade particular dentro de uma interação, que o distingue de outros objetos da mesma classe [Rumbaugh 99]. Um classificador possui identidade, estado, comportamento e relacionamentos. A UML define diversos tipos de classificadores entre eles: classes, ator, componentes, interface, tipo de dado e nós.

A visão de interação é mostrada através de dois diagramas que enfocam aspectos diferentes: o diagrama de seqüências e o diagrama de colaborações.

#### **2.8.2.1 - O Diagrama de Seqüência**

O diagrama de seqüência mostra um conjunto de mensagens dispostas de maneira temporal, ou seja, seqüencialmente no tempo. Cada classificador é mostrado através de uma linha da vida do objeto, a qual é representada através de uma linha vertical. Mensagens são mostradas como setas entre linhas da vida. Um diagrama pode mostrar um cenário, ou seja, a história individual de uma transação.

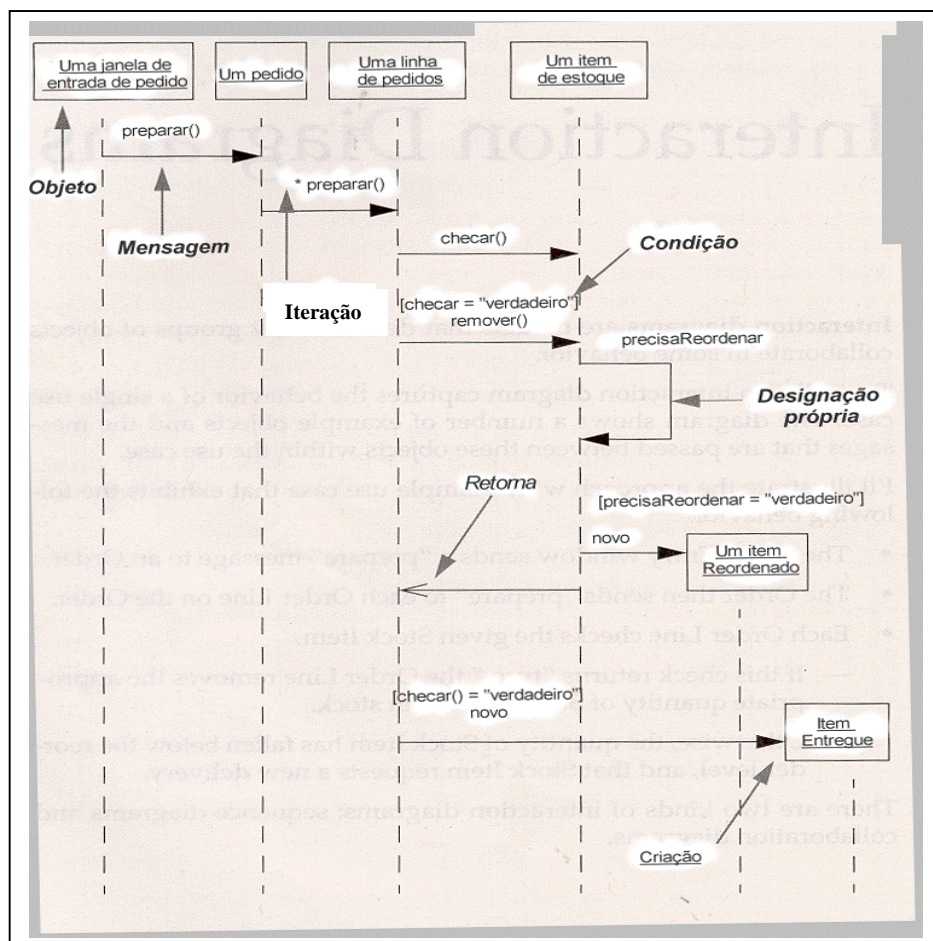
Cada mensagem possui pelo menos um nome de mensagem, sendo possível ainda incluir-se argumentos e mensagens de controle. É possível ainda representar-se auto-delegações que são mensagens enviadas de um objeto para ele próprio [Fowler 97].

Existem dois tipos de controles pré-definidos para serem utilizados no diagrama de seqüência, *condição* e *marcador de iteração*.

Condição é um controle utilizado para indicar que aquela mensagem só será enviada caso a condição expressa seja verdadeira, onde a condição estará especificada entre colchetes.

Marcadores de iteração são controles que estabelecem que uma mensagem é enviada mais de uma vez, o que caracteristicamente ocorrerá quando quisermos atuar sobre uma coleção.

A Figura 2.16 mostra um exemplo de um diagrama de seqüência extraído de [Fowler 97] onde podem ser vistos exemplos de condição e iteração.



**Figura 2.16– Exemplo de um Diagrama de Seqüência**

### 2.8.2.2 - Diagrama de Colaborações

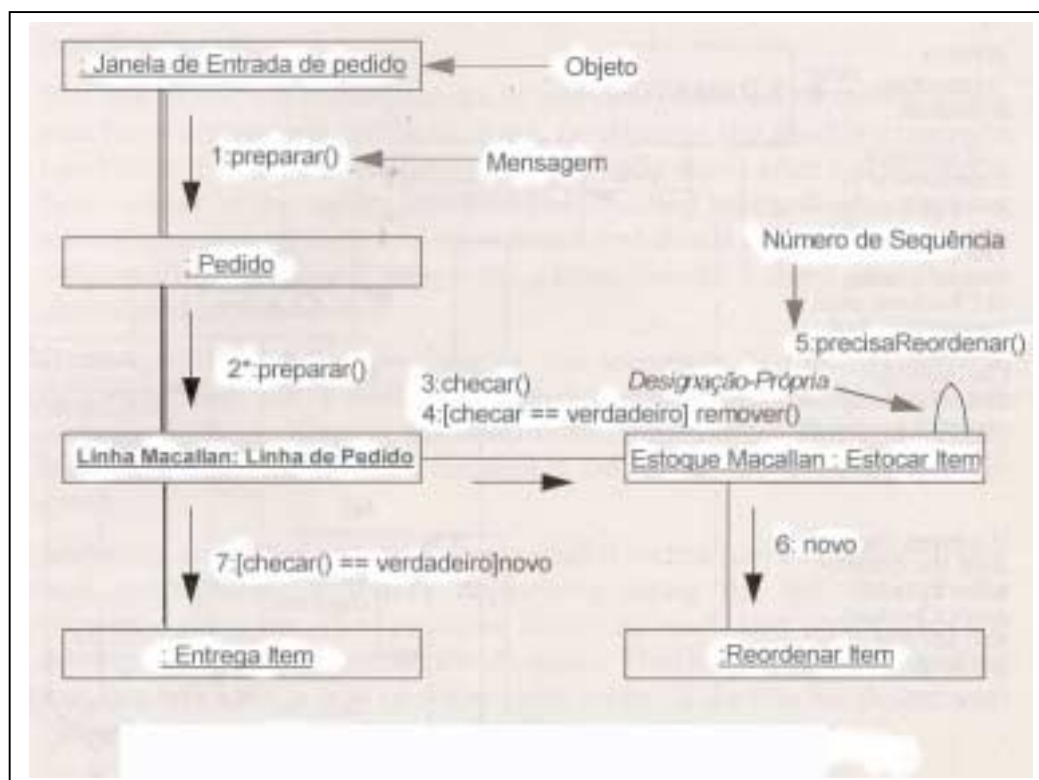
Em um diagrama de colaborações, ilustrado na Figura 2.17, os objetos são mostrados como ícones. Da mesma forma que no diagrama de seqüência, as setas

indicam as mensagens que são trocadas dentro de um dado cenário, sendo que no presente caso, a sequência é indicada por meio da numeração das mensagens.

O uso da numeração, se por um lado dificulta a visualização de seqüências, por outro favorece que outros detalhes sejam mostrados mais facilmente devido a liberdade de diagramação [Rumbaugh 99].

Vários tipos de numeração podem ser utilizados, inclusive qualquer um que seja criado por um desenvolvedor, contanto que a semântica da numeração utilizada fique bem clara para todos os envolvidos no projeto.

Os controles de condição e iteração relatados na seção anterior são aplicáveis também no diagrama de colaborações.



**Figura 2.17 –Diagrama de Colaborações**

### 2.8.3 - Object Constraint Language (OCL)

Modelos gráficos, como o diagrama de classes, não são suficientes para se gerar uma especificação precisa e não ambígua. Frequentemente necessitamos expressar restrições nestes modelos, as quais são em geral descritas pelo uso de linguagem



natural. Tal prática normalmente resulta em especificações ambíguas ou imprecisas [Rational 97]. Para evitar isso, alguns autores sugerem o uso de linguagens formais, as quais entretanto, não são fáceis de serem usadas por pessoas que não tenham uma sólida formação matemática.

A OCL surgiu da necessidade de preencher este espaço, ou seja, termos uma linguagem com viés formal que se mantém fácil de ler e escrever. A OCL foi desenvolvida como uma linguagem de modelagem de negócios dentro da divisão de seguros da IBM e tem suas raízes no Syntropy [Rational 97].

A OCL não é uma linguagem de programação, logo, não é possível criar programas ou controle de fluxos utilizando-a.

A OCL é uma linguagem tipada, logo toda expressão OCL tem um tipo, onde todos os tipos utilizados dentro de uma expressão devem estar em conformidade uns com os outros.

O uso da OCL é recomendável para os seguintes propósitos [Rational 97]:

- Especificar invariantes em classes;
- Especificar invariantes para estereótipos;
- Descrever pré e pós operações;
- Como uma linguagem de navegação;
- Para especificar restrições em operações.

Por padrão, quando uma expressão OCL é escrita utiliza-se a fonte courier e uma palavra sublinhada antes de uma expressão OCL, determina o contexto desta expressão. Por exemplo:

Pessoa

```
Self.age>0
```

No exemplo acima, a expressão OCL se dá no contexto da classe de objetos Pessoa e estabelece que a idade de uma pessoa tem de ser maior que zero. A utilização do *Self* indica que o atributo “age” pertence ao próprio contexto de pessoa e poderia ser omitido.

O exemplo a seguir demonstra o uso de pós-condição dentro de um determinado contexto:

Pessoa::Aniversário{ }

```
Post: idade = idade@pre + 1
```

No exemplo acima é demonstrado que na ocorrência da operação *Aniversário* na classe de objetos Pessoa é garantido que ao final da execução desta operação a expressão que segue a cláusula “post” será verdadeira. No presente caso, a expressão determina que o atributo idade terminará com o valor com o qual ele começa a operação somado de um.

O uso da cláusula “pre:” determina que para que aquela operação seja realizada é necessário que a expressão seguinte à cláusula seja verdadeira. A execução de uma operação pode ser parte integrante de uma cláusula “post”, significando que esta operação terá de ser feita antes da operação determinada pelo contexto ser realizada. Por exemplo:

Empresa::contrataPessoa{p:pessoa}

```
Pre: p not empregado
```

### **3 - Lidando com RNFs: da Elicitação ao Modelo Conceitual**

A estratégia proposta nesta tese não tem por objetivo enfocar um RNF específico, mas sim, demonstrar como identificar e representar RNFs de uma maneira geral, bastante próxima à abordagem orientada a processo proposta por Mylopoulos [Mylopoulos 92].

Observamos ao longo de nossos estudos de casos [Cysneiros 97] [Cysneiros 99] [Cysneiros 99b] que, apesar do simples fato de abordar e representar RNFs durante a etapa de elicitação de requisitos já ser um grande passo para a melhoria da qualidade do software obtido, a utilização de uma estratégia que permita a integração destes RNFs aos modelos que refletem os requisitos funcionais contribui para que estes RNFs estejam presentes no desenho do sistema desde suas primeiras versões, diminuindo a possibilidade de futuros conflitos entre as visões funcional e não funcional do problema.

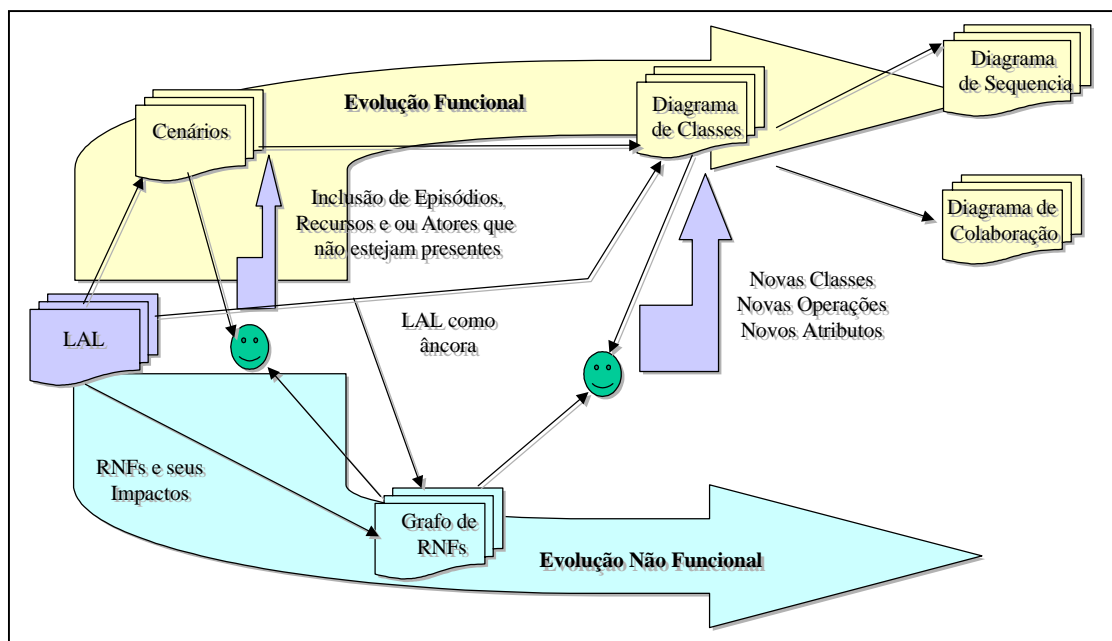
Para que possamos lidar com os RNFs durante grande parte do ciclo de vida do software, propomos uma estratégia que além de especificar uma maneira de se lidar com RNFs e representá-los, também os integre ao modelo conceitual que expressa os requisitos funcionais. Desta maneira, podemos obter um processo sistematizado para obter e representar RNFs e também para analisá-los, seja da ótica das interdependências entre eles como de suas interdependências com requisitos funcionais.

A estratégia propõe o uso do Léxico Ampliado da Linguagem como âncora para todo o processo. O LAL será utilizado para a construção tanto do modelo funcional como do não funcional facilitando, desta maneira, a posterior integração de ambos os



detalhamento da instanciación da estratégia para o modelo orientado a objetos. A instanciación desta estratégia para o modelo entidade relacionamento pode ser encontrada em [Cysneiros 99]. O capítulo sete irá detalhar o processo de integração dos modelos funcionais e não funcionais.

É importante ressaltar que a estratégia aqui proposta independe do tipo de método utilizado no processo de desenvolvimento de software bem como não está atrelada a nenhum tipo de ciclo de vida de sistemas específico, sendo, portanto uma estratégia de uso geral.



**Figura 3.2 – Detalhamento da Estratégia Proposta**

Deve-se ainda ressaltar que o uso do LAL, que poderia restringir a abrangência da independência do método acima mencionada pode, em ultimo caso, até ser relevado. Apesar de ter um papel de grande importância na estratégia, esse papel garante apenas o suave acoplamento das visões. Não utilizar o léxico como ancora irá levar apenas a um processo de integração mais difícil e menos preciso, mas ainda assim possível.

Em nossa proposta, enxergamos os requisitos evoluindo sob dois ciclos distintos: o funcional e o não funcional. O detalhamento da estratégia pode ser visto na Figura 3.2. Nesta figura pode-se observar que o ciclo funcional seria composto pelos modelos de cenários, de classes, de seqüência e de colaboração, enquanto o ciclo não funcional seria composto basicamente do grafo de RNFs. O LAL, como âncora de todo o processo não estaria classificado em nenhum dos dois ciclos, sendo a base de ambos.

É importante ressaltar que utilizamos na definição de artefatos utilizados na visão funcional a proposta apresentada por Hadad [Hadad 97] e Leonardi [Leonardi 97] que apresenta uma estratégia para usar o LAL e os cenários para gerar o diagrama de classes a partir de uma série de heurísticas.

Pela observação da Figura 3.2 podemos constatar que o primeiro passo desta estratégia é a definição do LAL do Udi. Esta definição será feita utilizando-se a proposta de Leite [Franco92][Leite93] e apresentada na seção 2.6 desta tese acrescida de extensões que serão mostradas no capítulo 4. Esta extensão mostra como representar no LAL as necessidades não funcionais e seus efeitos entre símbolos.

Uma vez obtido o LAL, parte-se para a construção dos modelos funcionais e não funcionais. Os dois são construídos separadamente, não havendo, a princípio, uma preocupação de atrelamento entre ambos. Estes procedimentos podem ser executados por equipes diferentes ou não. É claro que quanto maior a experiência do engenheiro de software em lidar com RNFs, maior será a facilidade de aplicar a estratégia. Na definição de ambos os modelos, o engenheiro de software deverá necessariamente utilizar símbolos definidos no LAL para especificar classes no diagrama de classes e tópicos de grafos de RNFs. O LAL será ainda fonte da definição dos cenários conforme mostrado na seção 2.7.

A construção do grafo de RNFs será facilitada pela inclusão no LAL das necessidades de RNFs e seus conseqüentes impactos nos diversos símbolos, extensão esta suportada pela ferramenta OORNF. Esta extensão favorece ainda o tratamento do aspecto evolutivo de RNFs que demandam contínuas mudanças no LAL devido ao tratamento de interdependências que surgem conforme vamos introduzindo maneiras de satisfazer RNFs necessários.

O tratamento de interdependências deverá ser feito ao nível do grafo de RNFs, uma vez que, por ser um artefato que lida com linguagem natural, o LAL não favorece esta tarefa. O capítulo 5 mostrará as extensões que propomos ao grafo de RNFs, bem como a sua geração a partir do LAL e uma sistematização para identificação de interdependências.

A construção do modelo funcional poderá ser feita através de qualquer processo que a equipe encarregada de sua construção achar adequado e não será alvo de estudo nessa tese. Escolhemos usar os cenários, diagrama de classes, seqüência e colaboração por serem largamente utilizados em diversas metodologias de desenvolvimento de software. Ressaltamos ainda que, a idéia associada a cenários nesta tese não é a de um artefato de refinamento de casos de uso, e sim, a de que cenários são descrições auxiliares para o processo de definição de requisitos, onde os cenários fornecem uma fonte de conhecimento na qual os requisitos podem ser encontrados e especificações podem usá-los como base [Leite 00].

A integração de ambos os modelos será feita através da incorporação dos RNFs aos modelos funcionais. Uma vez que o LAL foi utilizado como âncora para a construção tanto dos modelos da visão funcional quanto da não funcional.

Ao nível dos cenários isto será feito avaliando-se quais os símbolos do LAL aparecem no título de um cenário. Para cada símbolo que for encontrado iremos

procurar todas as ocorrências deste símbolo nos grafos de RNFs. Primeiramente avaliaremos se o grafo em questão realmente possui relação com o cenário em estudo. Uma vez que esta avaliação seja positiva, teremos de avaliar se os episódios deste cenário satisfazem as ações denotadas nos grafos. Se isso não ocorrer, atualizamos os cenários com os episódios necessários, e eventualmente, com novos atores e recursos. Este processo deve ser repetido para cada cenário existente e será mais detalhado no capítulo 5.

Para o diagrama de classes usaremos um processo semelhante que se baseia na idéia de que, para cada classe existente, iremos analisar o modelo não funcional a procura de RNFs que devam ser aplicados a esta classe. Novamente baseamo-nos no fato de que ambos os modelos foram construídos usando símbolos do LAL. Para cada classe existente no diagrama iremos procurar no conjunto de grafos de RNFs as ocorrências do símbolo que nomeia a classe. Para cada grafo encontrado iremos avaliar quais os dados e ações que são representados no grafo e verificar se os mesmos são satisfeitos na classe sendo analisada. Caso negativo, atualizamos a classe com os atributos e/ou operações que sejam necessários. Este processo será detalhado no capítulo sete.

Ambos os ciclos evoluem de maneira independente, ou seja, não existem ligações automáticas entre os modelos de cada visão. Faz-se portanto necessário, que o engenheiro de software realize os processos de integração das visões a cada alteração (ou conjunto de alterações conforme a velocidade destas) que seja encontrada em quaisquer das visões.

No caso da detecção de um novo RNF ou da alteração da condição de satisfação de RNFs já elicitados, após analisar os impactos destas alterações na visão não funcional o engenheiro de software deverá realizar a integração das mesmas na visão funcional.



Isto significa que ele deverá verificar quais os cenários que se relacionam com os grafos envolvidos e verificar se alguma alteração nos cenários será decorrente das alterações realizadas nos grafos de RNF.

Analogamente o mesmo deverá ser feito com o modelo de classes, validando-o novamente com os grafos de RNFs alterados.

No caso de mudanças ocorrendo na visão funcional, o processo inverso deverá ser realizado. Cada alteração deverá gerar alterações primariamente no LAL. Esta alteração possivelmente desencadeará mudanças nos cenários e nos modelos de classes, seqüência e colaboração. Caso novas classes sejam incluídas, devemos proceder ao processo de integração para estas classes, ou seja, analisar todos os grafos procurando pelo símbolo do LAL que nomeia a classe. Paralelamente, devemos aplicar o processo usado na visão não funcional para verificarmos se novas funções sendo introduzidas não demandam nenhum RNF. Se isso acontecer, teremos que realizar novamente a integração dos RNFs na visão funcional.

## 4 - Estendendo o Léxico ampliado da Linguagem

O LAL [Leite 93] atua em nossa estratégia de três formas diferentes. A primeira refere-se ao papel que ele desempenha no baseline de requisitos servindo como um instrumento para que se conheça a linguagem utilizada no domínio. A segunda refere-se ao papel de funcionar como uma âncora na produção dos modelos funcionais e não funcionais e como tal favorecer a integração futura dos dois modelos [Cysneiros 99]. Por fim, o LAL funciona como um agente na elicitação de RNFs.

É importante ressaltar que as figuras encontradas nesse capítulo e em capítulos futuros estarão fortemente representadas usando-se a língua inglesa. Isso se deve a termos extraído essas figuras dos estudos de caso que realizamos e os mesmo terem suas especificações escritas originalmente em inglês. Portanto, como trabalhamos com uma âncora que é a expressão léxica do domínio, decidimos manter as expressões em inglês para que nada se perdesse ou alterasse durante a tradução.

Um ponto importante de ser ressaltado é que o aspecto evolutivo de um domínio é particularmente potencializado quando estamos tratando com RNFs. Isto porque uma alteração no domínio pode levar-nos a diversas alterações nos RNFs. Isto advém do fato dos RNFs constantemente trazerem interdependências negativas e positivas que muitas vezes se traduzem em conflitos. Por consequência, é bastante comum que, durante o processo de desenvolvimento de software, decisões sobre satisfação parcial ou total de RNFs se alterem com frequência. Cada vez que isso acontece devemos ser capazes de rastrear o mais facilmente possível o impacto dessas mudanças.

Para que o LAL pudesse auxiliar na elicitação de RNFs, foram introduzidas algumas mudanças no mesmo. Como ferramenta para apoiar o registro do LAL,

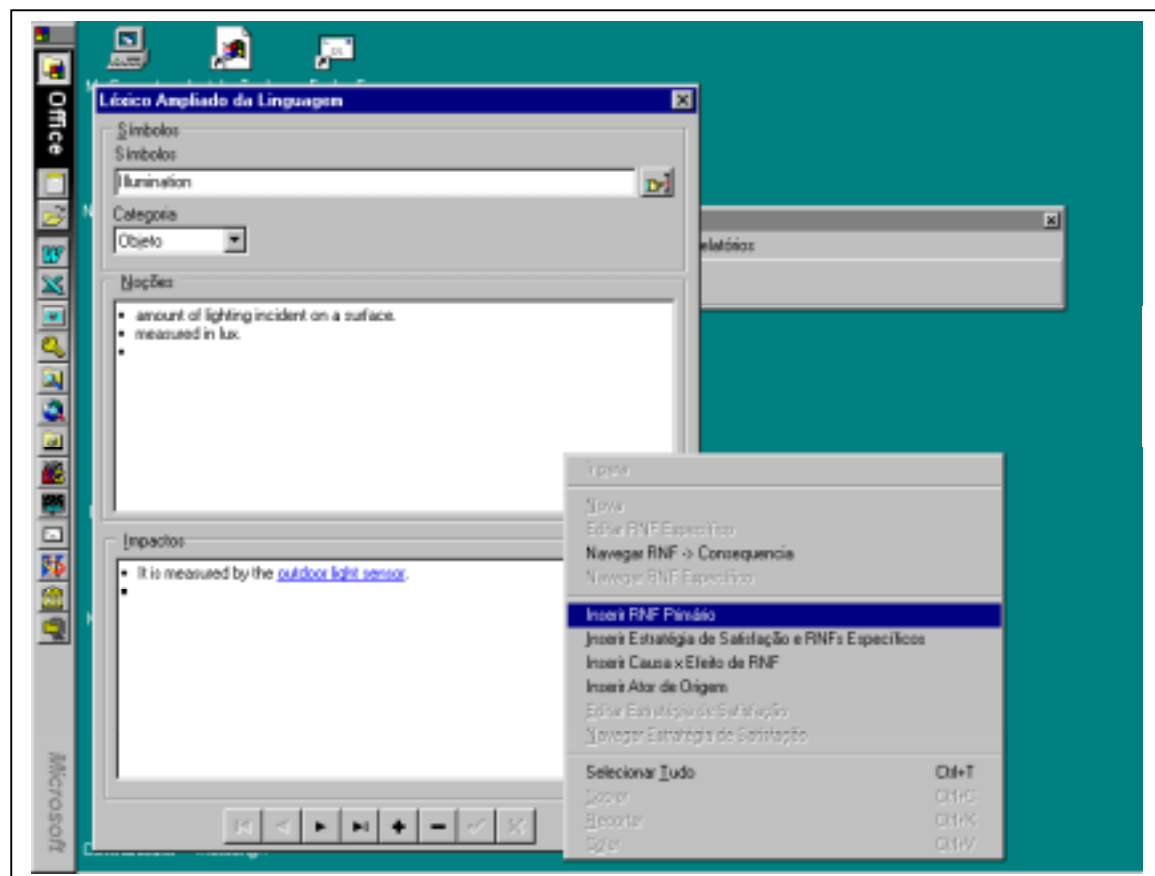
utilizamos a ferramenta OORNF proposta por Neto [Neto 00]. Algumas mudanças foram implementadas nesta ferramenta para que a mesma passasse a suportar as alterações propostas por nós para o LAL. Esta ferramenta contém uma base de conhecimentos de RNFs que espelha RNFs primários, suas operacionalizações e eventuais influências positivas e negativas entre cada operacionalização. Esta base de conhecimentos é dinâmica podendo ser acrescida de acordo com a experiência obtida pelo engenheiro de software. A ferramenta possibilita ainda o armazenamento de um LAL para vários projetos que estejam em andamento, onde este LAL pode conter RNFs e suas operacionalizações. Além do LAL a ferramenta OORNF possibilita o armazenamento de cenários e cartões CRC, bem como auxilia na elicitação dos cenários e cartões CRC utilizando-se de heurísticas propostas por Hadad e Leonardi [Hadad 97] [Leonardi 97].

A primeira extensão que propomos para o LAL é a de podermos explicitar no LAL que um determinado símbolo possui um ou mais RNFs a ele associados. Isso significa dizer que aquele determinado símbolo precisa destes RNFs para satisfazer totalmente as necessidades do cliente. Por exemplo, no caso de um sistema de controle de iluminação, para o símbolo *Iluminação* poderíamos determinar que toda iluminação tem de ser segura para o usuário do sistema, demandando assim que símbolo possua o RNF *Seguro*.

A segunda alteração proposta no léxico advém da necessidade de detalharmos o que é preciso para satisfazer um RNF existente em um símbolo. No exemplo anterior, poderíamos identificar que para uma iluminação ser segura, o nível de iluminação de um grupo de luzes não deve ser inferior a 14 lux, bem como o sistema deve se assegurar que, em caso de mau funcionamento dos sensores, todas as ações que o sistema tomar irão priorizar a segurança do usuário. Pode ser observado que para

satisfazer este RNF deveremos definir impactos em outros símbolos que não aquele onde o RNF aparece. Rastrear estas interdependências entre os símbolos será de grande valia na hora de montarmos os grafos de RNFs e na hora de tomarmos decisões de desenho que venham a restringir satisfações de um RNF em detrimento de outro.

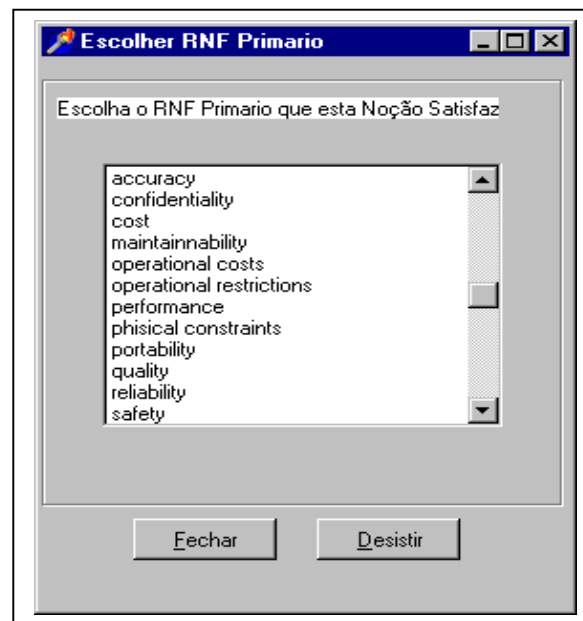
Estas duas extensões ao LAL foram introduzidas na ferramenta OORNF de forma a termos facilitado o processo de elicitação de RNFs, particularmente o acompanhamento de alterações realizadas no domínio e de negociações entre satisfação de RNFs.



**Figura 4.1 – Incluindo um RNF Primário em um Símbolo**

A primeira extensão que introduzimos na ferramenta OORNF foi a possibilidade de se incluir nas noções de um símbolo, a necessidade deste símbolo por um determinado RNF primário que conste da base de conhecimentos da ferramenta. Na

tela de inclusão de símbolos, ao clicarmos com o botão direito do mouse teremos a exibição de um menu que, entre outras escolhas, nos traz a opção de Inserir RNF Primário conforme pode ser visto na figura 4.1. Ao ser selecionada a opção de Inserir RNF Primário, a ferramenta irá exibir uma tela com os RNFs Primários existentes na base de conhecimento, exibida na Figura 4.2.



**Figura 4.2 – RNFs Primários contidos na Base de Conhecimento**

Uma vez que selecionemos um ou mais RNFs para um símbolo, estes aparecerão sublinhados em vermelho em seguida a uma frase padrão indicando que este símbolo tem um RNF com a seguinte sintaxe: **“Possui RNF <<nome do RNF>>”**. A Figura 4.3 mostra como ficaria o símbolo *Iluminação* pertencente a um sistema de controle de luzes, que possui os RNFs *Seguro* e *Custo Operacional*.

A escolha de um RNF a ser satisfeito em um símbolo deve ser seguida da avaliação do que será necessário para que este RNF seja satisfeito. Por exemplo, se o engenheiro de software decide que o símbolo *Iluminação* demanda o RNF *segurança* ele deverá avaliar que medidas serão necessárias para garantir uma iluminação segura. Esta avaliação poderia, por exemplo, levar a conclusão que uma iluminação segura é

toda aquela que possui mais do que 14 lux, e que, em qualquer caso, o sistema deve garantir que essa iluminação seja praticada.

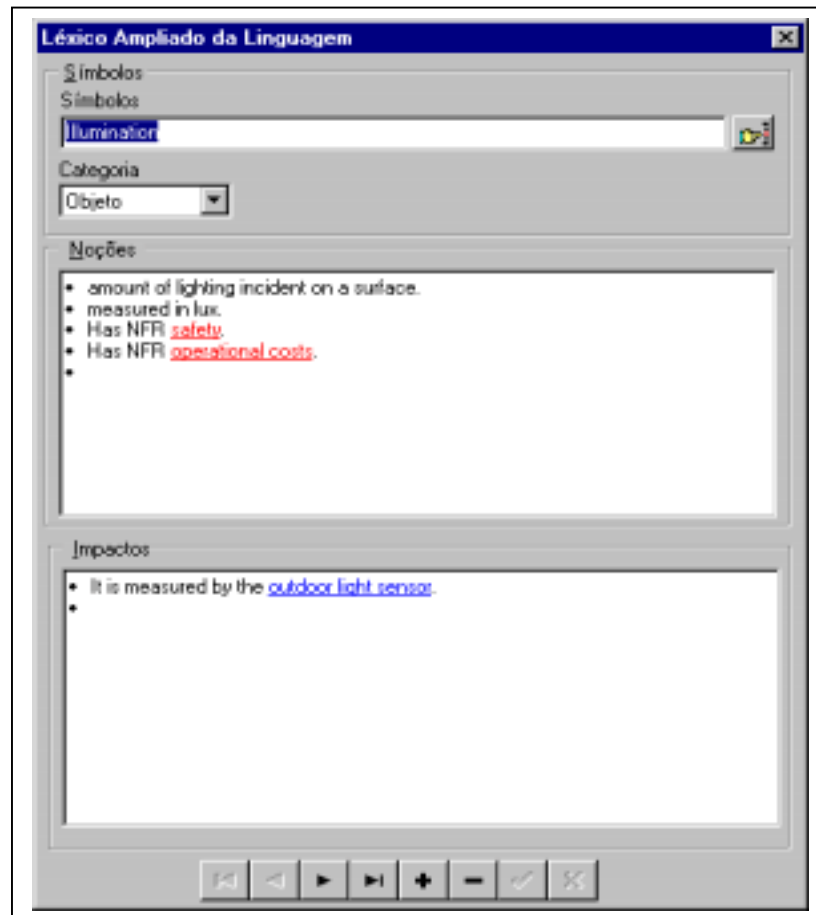
Apesar de lidarmos com interdependências de maneira mais detalhada com o uso do grafo de RNFs, detalhada no capítulo 5, já aqui é interessante que o engenheiro de software se pergunte se a satisfação deste RNF poderá ter impactos positivos ou negativos em outros símbolos. A ferramenta OORNF possui uma opção de se lidar com um léxico apenas de RNFs onde está armazenada a base de conhecimentos sobre RNF. Nesta base, além de RNFs primários e secundários, também teremos cadastradas estratégias de satisfação, aqui chamadas de operacionalização, com suas respectivas influências positivas e negativas em outros RNFs/estratégias de satisfação [Neto 00].



**Figura 4.3 – Influências entre RNFs Contidas na Base de Conhecimento**

A Figura 4.3 mostra o conteúdo da base de conhecimento para o caso da operacionalização (lá chamada de estratégia de satisfação) validação. Esta operacionalização denota que para satisfazermos um determinado RNF, precisão de valor, no caso, necessitamos de algum tipo de validação que garanta a precisão

(correção) da informação. Podemos ver nesta figura que a operacionalização de validar informações, em geral, contribui positivamente para precisão de valor e propriedade e negativamente para precisão de oportunidade e confidencialidade.

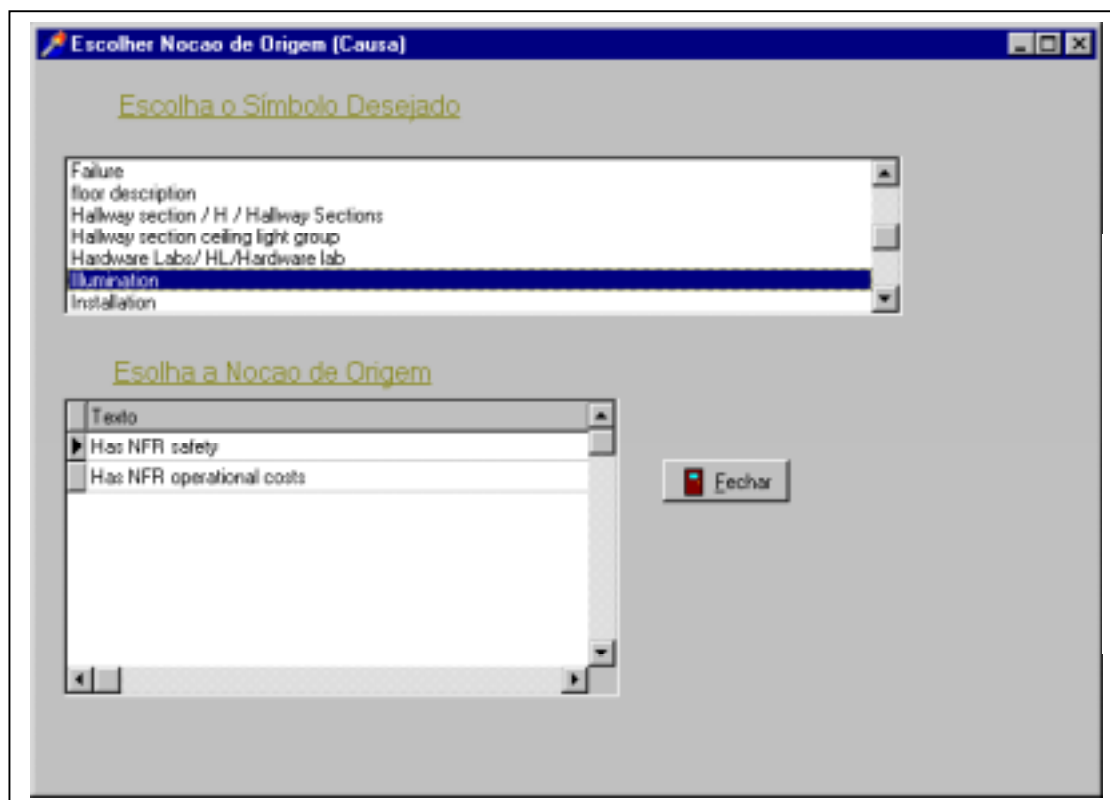


**Figura 4.4 – Entrada do LAL com RNFs Primários Assinalados**

Para suportar a segunda extensão que fizemos ao LAL, fornecer a capacidade de visualizarmos quais noções e impactos existem para satisfazer um RNF, estendemos a ferramenta OORNF para possibilitar que o engenheiro de software assinale o fato de uma noção ou impacto serem derivados da necessidade de satisfazer um RNF presente em um símbolo, que pode ser o mesmo ou outro qualquer. Isso é feito de forma semelhante à inclusão de um RNF Primário, ou seja, através de um menu que é acionado ao pressionarmos o botão direito do mouse. Quando isto ocorrer, uma das opções deste menu será Inserir Causa x Efeito de RNF. Esta opção deverá ser

realizada estando o cursor posicionado no fim da sentença que expressa o impacto ou noção que é efeito da necessidade de satisfazer um RNF deste ou de outro símbolo.

A seleção desta opção irá trazer uma tela, mostrada na Figura 4.5, que dará a opção de selecionarmos quaisquer símbolos do LAL e seus respectivos RNFs como sendo a origem desta noção ou impacto. Na figura 4.4, por exemplo, poderíamos precisar demonstrar que o impacto ali presente era devido a satisfação do RNF Seguro especificado nas noções desse mesmo símbolo. Neste caso, quando utilizando a janela mostrada na Figura 4.5 iríamos procurar pelo símbolo *Iluminação* e escolher o RNF *Seguro*.

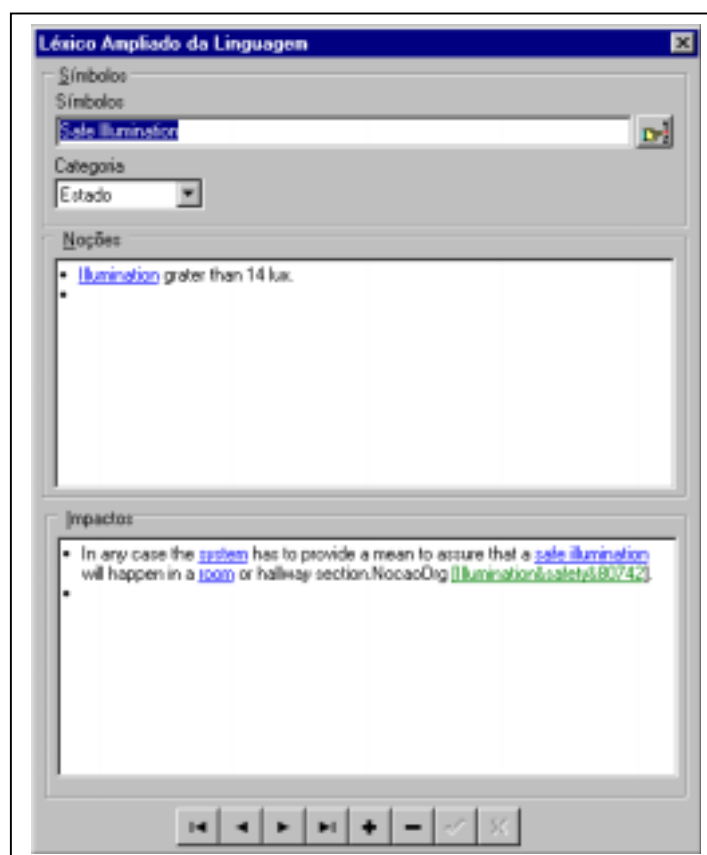


**Figura 4.5 – Selecionando o RNF que Originou uma Noção ou Impacto**

Uma vez que tenhamos selecionado o RNF de origem, a ferramenta OORNF irá inserir um identificador de origem composto do seguinte padrão: **NacaoOrg** **[Símbolo&RNF&IdEntrada]**, onde a parte entre colchetes estará em verde. A string NacaoOrg é utilizada pela ferramenta como identificador dentro do léxico de uma

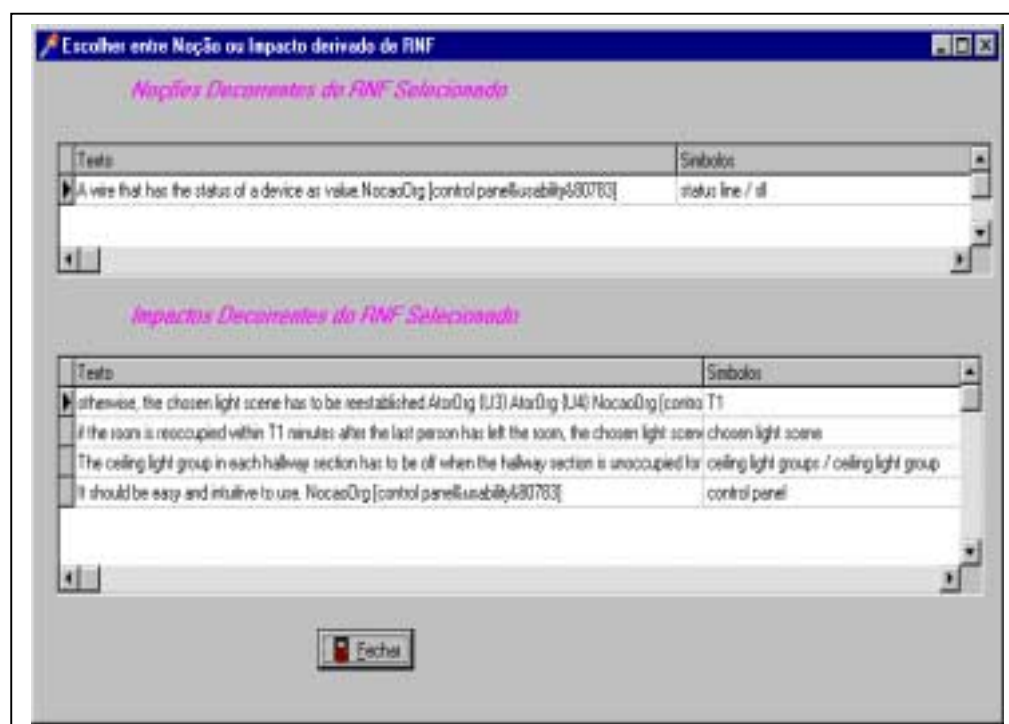


ligação de efeito causado por outro ponto do léxico. A utilização deste padrão visa a possibilitar o controle de integridade entre as entradas do léxico, de forma que se o RNF contido no símbolo de origem for eliminado, a ligação de consequência também o será. A figura 4.6 mostra um exemplo onde vemos que o impacto da entrada Iluminação Segura possui o padrão que identifica que ele é derivado da satisfação do RNF Seguro do símbolo Iluminação (NocaoOrg[Illumination&safety&80742]). Ou seja, esse impacto existe para satisfazer a necessidade que temos que o símbolo Iluminação seja atendido na sua necessidade de ser seguro. Logo, se mais a frente decidirmos que essa necessidade não mais existe e portanto temos de eliminar essa noção do símbolo, a ferramenta irá automaticamente eliminar as noções e impactos existentes que existem para satisfazer esse RNF, pois elas não mais serão necessárias.



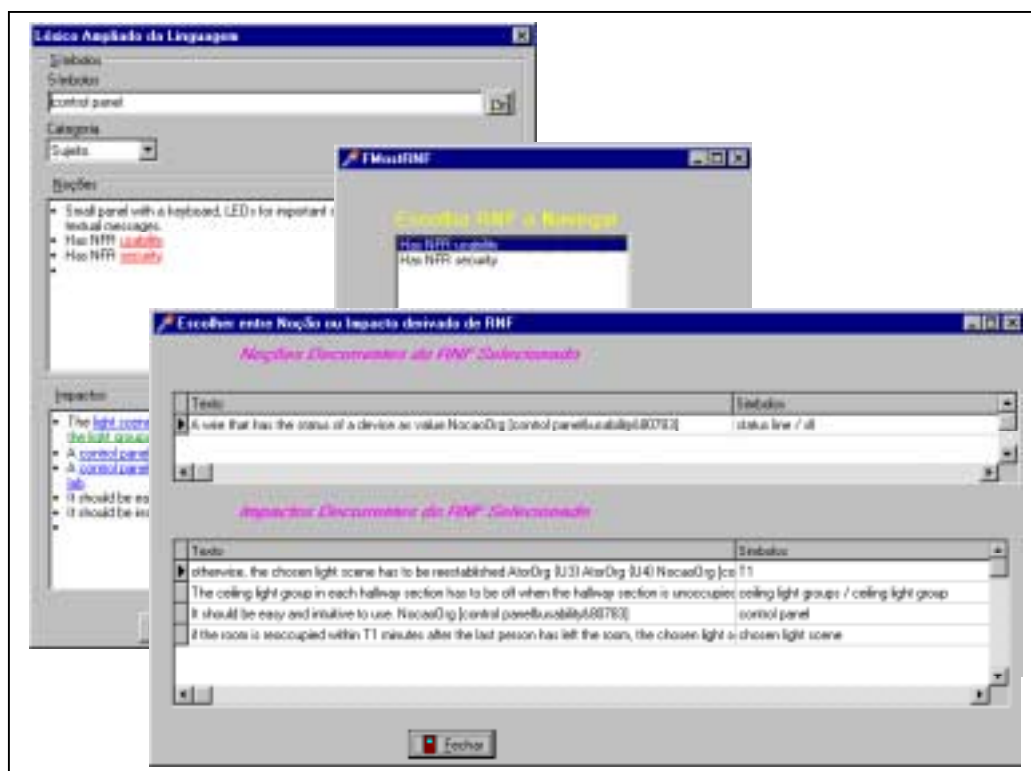
**Figura 4.6 – Exemplo de Entrada do Léxico com Identificação de RNF de Origem**

A ferramenta OORNF passou a fornecer também um mecanismo de navegação onde o engenheiro de software pode navegar do RNF para suas conseqüências. Para isto basta clicar com o botão direito do mouse sobre o símbolo do LAL e utilizar, no menu em que é mostrada (veja Figura 4.1), a opção Navegar RNF -> Conseqüência. Ao se fazer isso, a ferramenta exibirá uma tela mostrando todos os RNFs deste símbolo para que o engenheiro de software possa escolher um dos RNFs de forma a poder navegar pelas conseqüências deste RNF no léxico, ou seja, será possível ver quais noções e impactos de quais símbolos contêm definições que estão lá para satisfazer o RNF sendo pesquisado. Uma vez selecionado o RNF desejado, uma tela será mostrada, onde aparecerão dois mecanismos de seleção distintos. Um, mostrando as noções que são derivadas da satisfação deste RNF e, outra, com os impactos que resultam da satisfação deste RNF. Nesta tela aparecem parte do texto da noção/impacto e o símbolo ao qual esta noção/impacto pertencem. Com um clique duplo sobre uma destas entradas a ferramenta irá exibir o símbolo associado à noção/impacto selecionado. A ferramenta permite ainda, que ao se fechar uma janela, retorne-se à tela anterior e assim por diante até voltarmos ao símbolo onde começou a navegação.



### Figura 4.7 – Exemplo da Navegação Entre RNFs e suas Consequências

A Figura 4.7 mostra um exemplo da tela onde são exibidas as noções e impactos resultantes da satisfação do RNF *Usabilidade* do símbolo **Painel de Controle** de um sistema de controle de iluminação (Figura 4.8). Na Figura 4.7 podemos ver que, para satisfazer este RNF foi necessário incluir uma noção no símbolo linha de status, um impacto no próprio símbolo **Painel de Controle**, um impacto no símbolo **T1**, outro impacto no símbolo **Esquema de luz escolhido** e finalmente mais um impacto no símbolo **Grupo de iluminação do teto**.



**Figura 4.8 – Símbolo Painel de Controle**

A utilização desta navegação se mostrou bastante útil não apenas no acompanhamento de alterações no domínio, mas também na geração do Grafo de RNFs. Isto graças ao fato de podermos, através dela, facilmente determinar onde teremos de procurar no LAL para vermos quais noções e/ou impactos satisfazem um

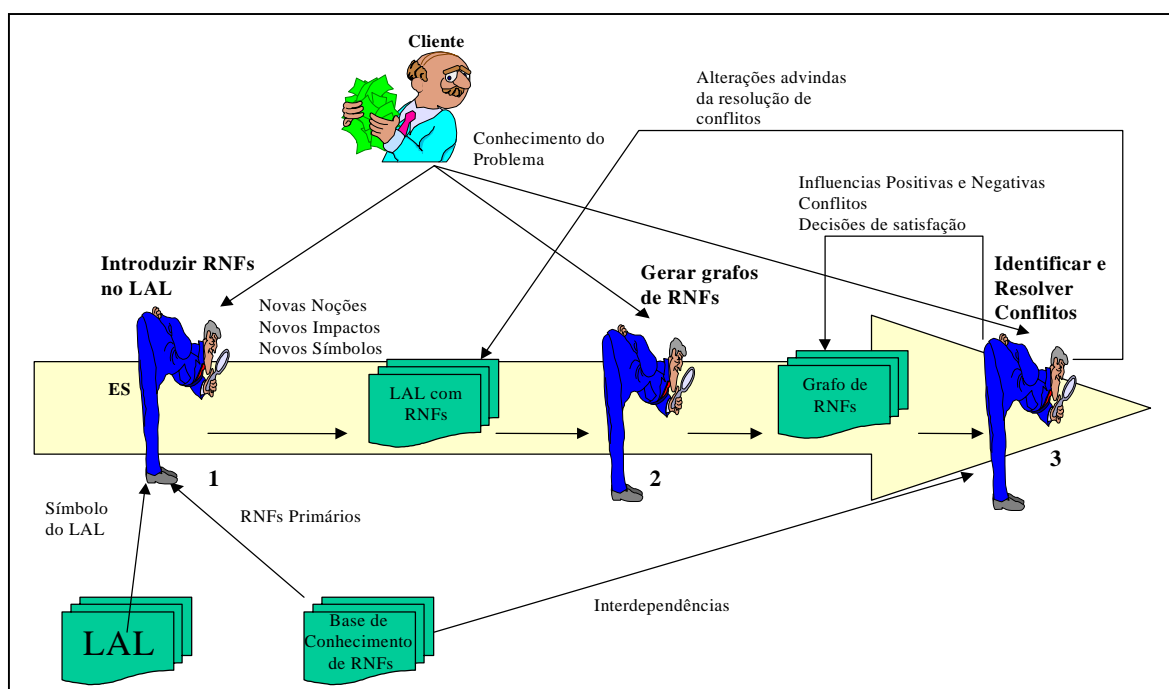
RNF. Estas noções e impactos serão fortes candidatos a operacionalizações que satisfazem RNFs no grafo de RNF. O capítulo 5 irá detalhar melhor a aplicação deste conceito.

## 5 – Desenvolvendo a Visão Não Funcional

A montagem do LAL não inclui, a princípio, uma preocupação com RNFs. Entretanto, alguns RNFs irão naturalmente aparecer durante o levantamento do LAL devido ao fato de que certos RNFs ficam em grande destaque em determinados domínios, como, por exemplo, o RNF Segurança no domínio bancário.

De uma maneira geral, podemos dizer que a primeira versão do LAL só irá captar os RNFs que se destaquem demasiadamente. Isto ocorre devido ao fato de normalmente não estarmos especialmente atentos aos RNFs na primeira definição do LAL. A inclusão dos RNFs neste, bem como a representação destes no grafo de RNF, fazem parte de um processo de elicitação de RNFs proposto nesta tese.

A Figura 5.1 mostra uma visão geral do processo de elicitação de RNFs, o qual será detalhado nas seções a seguir. Esta figura pode ser vista como um detalhamento do desenvolvimento da visão não funcional mostrado na Figura 3.2.



**Figura 5.1 – Processo de Elicitação de RNFs**

## 5.1 - Introduzir RNFs no LAL

Dentro da visão não funcional, a atualização do LAL disponível é o primeiro passo para começarmos a elicitar os RNFs. Começamos essa etapa varrendo todas as entradas definidas no LAL. Para cada entrada, utilizamos a base de conhecimento contida na ferramenta OORNF [Neto 00], ou qualquer outra fonte de conhecimento sobre RNFs, [Chung 00] por exemplo, para nos perguntarmos, e, quando possível, ao cliente, se algum RNF se aplica a este símbolo. Se algum RNF se aplica, devemos inserir este RNF na noção deste símbolo, conforme mostrado no capítulo 4, e passar então a avaliar quais os impactos deste RNF, ou seja, o que precisa ser feito para que este RNF seja satisfeito. Isto irá se traduzir na inclusão de novas noções e/ou impactos neste mesmo símbolo ou mesmo em outros símbolos.

Tomemos o caso de um sistema de automação laboratorial como exemplo. No LAL que representa a linguagem praticada neste domínio, encontraríamos o símbolo **Revisar Exame**. Este símbolo denota a necessidade de que médicos revisem os resultados dos exames de um paciente em busca de inconsistências. Ao contrapormos a lista de RNFs da base de conhecimentos com este símbolo, poderíamos nos aperceber, com ou sem a ajuda do cliente, que os RNFs *Precisão* e *Usabilidade* seriam aplicáveis, já que claramente um laudo precisa ser preciso e a tarefa de revisar um laudo, por sua inerente complexidade, tem de ser bem apoiada por uma boa usabilidade do sistema.

Ao buscarmos de que forma poderíamos garantir esses RNFs, encontraríamos a necessidade de que o sistema possa fazer arredondamentos diferenciados por exame, RNF precisão de valor, e que os exames com resultados fora da faixa de normalidade sejam alertados de alguma forma durante a revisão tendo os médicos acesso a esses

valores de normalidade, RNF de usabilidade. Isto iria implicar no fato de incluir no símbolo **Exame** as noções:

- Possui RNF Precisão
- Possui **fator de arredondamento**
- Possui RNF Usabilidade
- Possui **valores de normalidade**

A inclusão destas novas noções levaria a inclusão dos novos impactos:

- Cadastrar **valores de normalidade** de exame
- Cadastrar **fator de arredondamento**

Quando introduzir noções e impactos originados da necessidade de satisfazer RNFs, é imperativo que o engenheiro de software identifique essa ligação entre as partes utilizando o mecanismo introduzido na ferramenta OORNF, descrito no capítulo 4 e exemplificado na figura 4.6.

Eventualmente poderemos nos deparar com uma situação inversa à descrita anteriormente, ou seja, ao analisarmos um símbolo do LAL poderemos vislumbrar alguma operacionalização necessária para satisfazer um RNF, sem, contudo, vislumbrar o RNF propriamente dito. Por exemplo, no domínio do sistema laboratorial descrito anteriormente, ao revisarmos o símbolo **Laudo** poderemos eventualmente vislumbrar que necessariamente todo laudo deve ser impresso em impressora Laser, para somente depois atrelar este impacto à necessidade de termos o RNF *Qualidade* associado a este símbolo.

O engenheiro de software não deve estar aqui preocupado em identificar interdependências. Isso será feito de maneira sistemática numa etapa posterior. Entretanto, algumas interdependências poderão ser encontradas já nesta etapa e poderão ser registradas no próprio léxico através de noções que as deixem claras. Por

exemplo, ao definirmos o símbolo **Assinar Laudo Eletronicamente** em um sistema de informações laboratoriais, identificamos junto ao cliente que deveríamos ter intervalos de valores dentro dos quais o sistema pudesse assinar eletronicamente o laudo para cada exame. Isto iria ser refletido no símbolo **Exame** que teria então uma noção “Possui faixa de assinatura eletrônica”.

## **5.2 - Gerar Grafos de RNFs**

Uma vez que temos os RNFs expressos no LAL deveremos agora representá-los de forma mais organizada e propícia para lidar com eles. Para isso, propomos a utilização do grafo de RNFs proposto por Chung [Chung 93] [Chung 00] com algumas alterações definidas por nós para auxiliar a rastreabilidade de origem e a integração com o modelo funcional.

### **5.2.1 - Alterações no Grafo de RNF**

Conforme dito anteriormente, as alterações propostas para o grafo de RNFs detalhado na seção 2.7 visam a introduzir algum mecanismo de rastreabilidade do RNF à sua origem e a facilitar a integração deste à visão funcional.

#### **5.2.1.1 Alteração para Facilitar a Rastreabilidade Reversa**

O processo de elicitação de RNFs constantemente remete-nos a reavaliar decisões de desenho anteriores, fazendo com que RNFs anteriormente satisfeitos passem agora a serem apenas parcialmente satisfeitos ou mesmo negados ou vice-versa.

Um exemplo disto, extraído de um sistema de controle de iluminação, seria o RNF *Custo Operacional* associado ao símbolo do LAL **Mau funcionamento do Sensor de Luz**. Cabe ressaltar que na notação de Chung [Chung 00] (Ver seção 2.7), este símbolo seria o tópico do RNF *Custo Operacional*. Originalmente, observando fontes de informação do Udi, determinou-se que para este RNF ser satisfeito era necessário e



suficiente que o sistema, ao detectar um mau funcionamento do sensor de luz, estabelecesse como medida atual recebida do sensor a última medida recebida. Posteriormente, avaliamos que o RNF *Seguro* se aplicava ao símbolo *Iluminação* e que para satisfazer este RNF era necessário garantir que, a qualquer momento, uma sala ocupada teria de ter uma iluminação de pelo menos 14 lux. Fica claro que, manter-se a operacionalização que satisfaz a *Custo Operacional*, ou seja, atribuir o último valor recebido do sensor de luz claramente conflita com a operacionalização do RNF *Seguro*, já que a iluminação externa pode ter caído de tal maneira que mantendo a última leitura o ambiente passe a ter uma iluminação inferior a 14 lux.

Exemplificando, suponhamos que num determinado momento a iluminação externa medida pelo sensor de luz produza o equivalente a 10 lux e que o esquema de luz escolhido especifica uma iluminação total equivalente a 18 lux. O sistema de controle deverá então controlar os grupos de luzes do ambiente de forma a que eles forneçam 8 lux para que o esquema de luz escolhido seja obtido, uma vez que a iluminação do ambiente propriamente dita será a soma da iluminação dos grupos de luzes com a iluminação vinda de fonte externa. Suponhamos então que há um mau funcionamento no sensor de luz e, por consequência, o sistema de controle estabeleça que o sensor de luz continue a ler a última leitura, ou seja, 10 lux. Suponhamos ainda, que nuvens tenham coberto o céu e neste momento o total de iluminação vindo de fora do ambiente tenha caído para 4 lux. O sistema continuará a indicar aos grupos de luzes que forneçam o equivalente a 8 lux, o que resultará num total de apenas 12 lux no ambiente, resultado da soma dos 8 lux dos grupos de luzes com os reais 4 lux vindos de fonte externa, inferior portanto, ao mínimo estabelecido pelo RNF *Seguro*.

Para resolvermos este conflito temos que consultar os clientes que estão diretamente envolvidos com estes RNFs, ou seja, temos de ser capazes de identificar

de onde o conhecimento sobre o domínio se originou de forma a gerar a necessidade dos RNFs e suas respectivas operacionalizações, para então determinamos qual RNF será comprometido, e em que extensão, para solucionar o conflito.

De forma a termos este tipo de informação e facilitar tomadas de decisões em casos de conflitos, passamos a representar acima do grafo de RNFs qual o ator do UdI de onde se originou o conhecimento do domínio que levou-nos a estes RNFs e operacionalizações. A Figura 5.2 retrata este procedimento. As letras D e S no interior de cada nó do grafo representam a negação (Denied) e a satisfação (Satisfied) de cada meta/submeta.

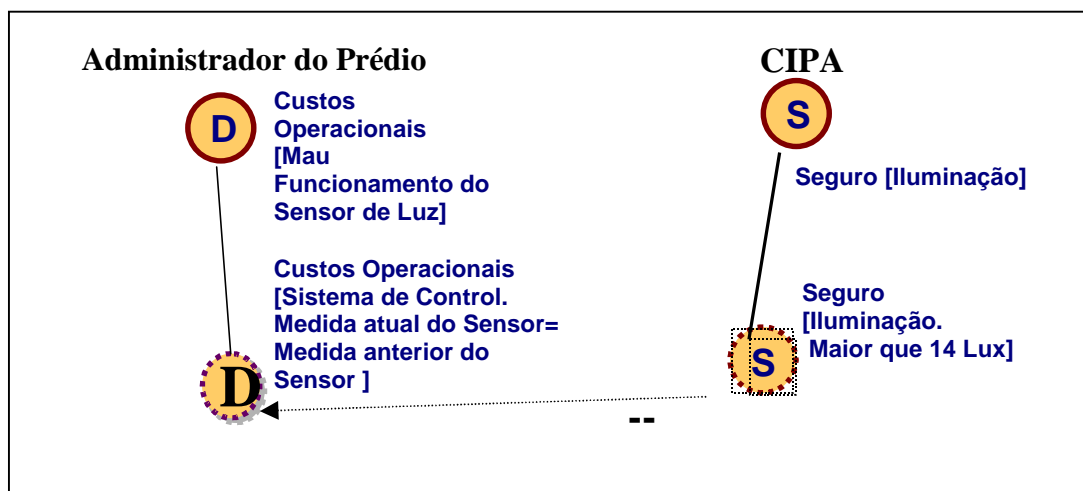


Figura 5.2 – Exemplo de Grafo de RNFs com Identificador de Origem do RNF

### 5.2.1.2 Alterações para Facilitar a Integração das Visões Funcional e Não Funcional

Uma das dificuldades que enfrentamos nas primeiras tentativas de integrar a visão não funcional à funcional residiu na identificação do ponto de convergência entre ambas as visões. Relacionar a que entidade ou classe que um determinado RNF estaria ligado muitas vezes era pouco claro.

Para facilitar este processo de integração, estabelecemos como regra de construção que ambas as visões deveriam ser construídas tendo o LAL como âncora. Isto significa dizer que tanto o Grafo de RNFs quanto o diagrama de classes devem ser

construídos usando apenas símbolos do LAL para expressar nomes. No caso do Grafo de RNFs, significa dizer que todo tópico de um RNF será, obrigatoriamente um símbolo do LAL. No caso da Figura 5.2, por exemplo, tanto *Mau funcionamento do Sensor de luz*, quanto *Iluminação* são símbolos do LAL.

No caso de efetivamente identificarmos algum RNF com um tópico (ver descrição de grafo de RNF seção 2.7) que não seja símbolo do LAL, devemos revisar o LAL para incluir este símbolo com suas noções e impactos, uma vez que se esta palavra/expressão é necessária como origem de um RNF, certamente terá importância particular na linguagem do UdI e, portanto deve ser um símbolo do LAL.

Quando estivermos refinando as metas em submetas esta restrição não mais se aplica, pois nem sempre poderemos decompor um RNF utilizando símbolos do LAL. Contudo, sempre que um símbolo do LAL puder ser usado isto deverá ser feito já que isso resultará numa diferente abordagem a ser utilizada durante a integração das visões.

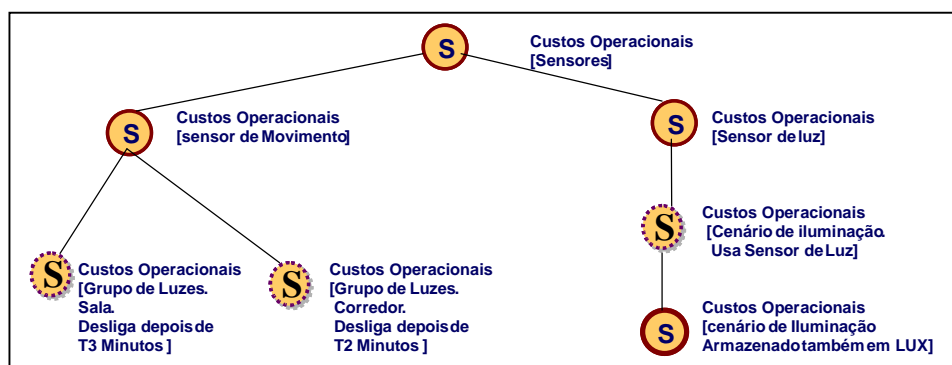
A segunda alteração que visa a facilitar a integração reside na classificação das operacionalizações em dinâmicas e estáticas. Operacionalizações dinâmicas são aquelas que denotam ações a serem realizadas, por exemplo, *desligar a luz depois de T2 minutos* indica a necessidade de uma operação a ser realizada (Figura 5.3), enquanto operacionalizações estáticas indicam, via de regra, a necessidade de um dado específico para satisfazer um RNF, por exemplo, *a iluminação deve ser armazenada em lux* determina a necessidade de um atributo para armazenar esta informação (Figura 5.3). Operacionalizações que não expressem nem necessidades de dados nem ações serão classificadas como estáticas. Por exemplo, num sistema de informações laboratorial, poderemos encontrar um RNF *Usabilidade* aplicado a *Listas* cuja operacionalização estabelece a necessidade de a interface de requisição

ser a mesma para os diversos tipos de mapas. Esta operacionalização será classificada como estática.

Classificando as operacionalizações em dinâmicas e estáticas, ficará claro durante a integração dos RNFs ao diagrama de classes, que operacionalizações irão ser refletidas através de operações nas classes (operacionalizações dinâmicas) e quais serão refletidas através de atributos nas classes (operacionalizações estáticas).

As operacionalizações dinâmicas serão representadas por um círculo pontilhado, enquanto as estáticas serão representadas por um círculo com borda mais espessa que a normal para diferenciá-la das submetas. Apesar da notação atual [Chung 00] utilizar nuvens representando as metas e submetas, mantivemos o uso de círculo conforme usado por Chung em sua proposta anterior [Chung 93] por questão de simplicidade de desenho.

Operacionalizações estáticas estarão sempre no nível folha do grafo, enquanto as dinâmicas podem aparecer em qualquer nível, inclusive sendo decomposta em uma estática. A Figura 5.3 mostra um exemplo onde aparecem ambas as operacionalizações. Nesta figura podemos ver duas operacionalizações estáticas que refinam a submeta de custos operacionais relacionada a sensores de movimento. Uma delas determina que as luzes devem ser apagadas nas salas depois de T3 minutos vazia e a outra determina que as luzes devem ser apagadas no corredor depois de T2 minutos vazio.



**Figura 5.3 – Exemplo de Operacionalizações Estáticas e Dinâmicas**

Observamos também que na decomposição de custos operacionais relacionados com os sensores de luz, teremos uma operacionalização dinâmica enfatizando a necessidade do sistema usar a luz externa, captada pelos sensores de luz, para manter o ambiente iluminado. Isto nos leva à operacionalização estática que mostra que o esquema de luzes deve ser armazenado também sob forma de quantidade de iluminação, e não apenas na forma de percentual de atenuação das luzes, ou seja, denota a necessidade de um atributo que armazene esta informação.

A notação do grafo por nós utilizada é bem próxima da proposta por Chung [Chung 93] e baseia-se na representação do RNF seguido das metas/submetas entre colchetes, ver seção 2.7, onde as decomposições estão separadas por ponto, podendo esta submetas serem omitidas. Da mesma forma que proposto por Chung e explicado na seção 2.7, podemos também decompor metas sob ótica de tipo de RNF (Seguro em integridade e confiabilidade) ou do tópico ao qual ela se aplica (Sensores em sensor de movimento e sensor de luz).

A Figura 5.3 ilustra as diversas formas de representação que podemos utilizar. Nesta figura podemos ver que o RNF *custo operacional* já decomposto para **sensor de movimento** é representado sem a sua meta anterior (sensores), sendo representada como Custo Operacional [Sensor de Movimento]. A outra forma possível de representarmos esse mesmo nó do grafo seria Custo Operacional [Sensores. Sensor de Movimento] Já a decomposição do mesmo é feita representando-se a nova submeta junto com as antigas separadas por ponto, ficando representada por Custo Operacional [Sensor de Movimento. Sala]. De uma maneira geral a inclusão das metas/submetas anteriores pode e deve ser evitada para efeito de simplicidade da representação do grafo. Entretanto, muitas vezes a representação das mesmas pode exercer um papel de

ressaltar ao que se aplica a submeta sendo representada, o que por vezes auxilia na confecção do grafo.

### **5.2.2 - Construção do Grafo de RNFs**

No intuito de auxiliar na sistematização da construção de grafos de RNFs, desenvolvemos algumas heurísticas para auxiliar a construção do grafo de RNFs que são apresentadas a seguir:

1. Percorrer todos os símbolos do LAL já com RNFs representados, verificando quais possuem RNFs;
2. Para cada símbolo que possuir um RNF faça:
  - 2.1 Definir a raiz do RNF utilizando o RNF seguido do símbolo do LAL como tópico.
  - 2.2 Decompor esta meta em submetas seja pela ótica de decomposição em tipos de RNFs (primários em secundários) ou em tópico (sensor em tipos de sensores). Representar resultado obtido no grafo.
  - 2.3 Continuar a decomposição até que se possa visualizar o que é necessário para satisfazer este RNF, ou seja, suas operacionalizações. Representar as operacionalizações no grafo de RNF classificando-as em estáticas ou dinâmicas conforme o caso.
  - 2.4 Nas decomposições anteriores utilizar o conhecimento próprio do domínio e sempre que possível envolver o cliente na definição destas decomposições, avaliando em conjunto com este se as decomposições estão corretas e são suficientes.

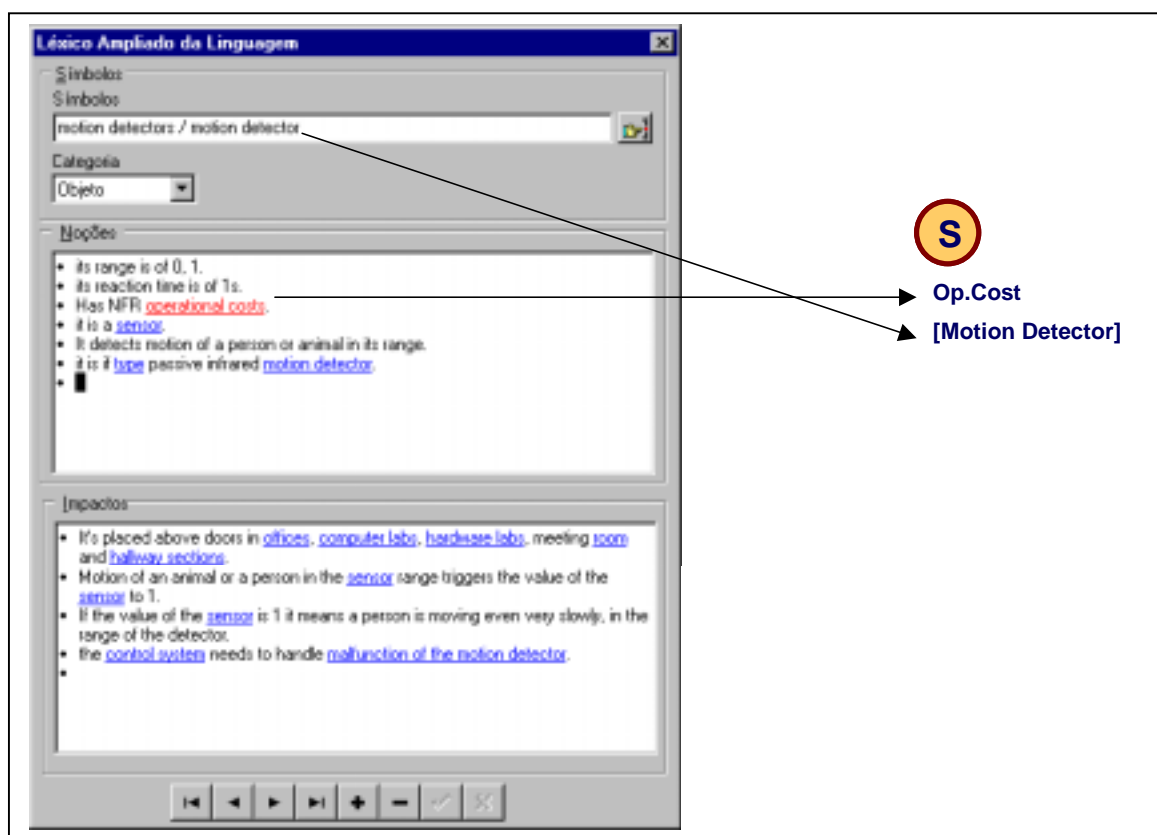
2.5 Verificar no LAL se o RNF não tem conseqüências (operacionalizações) já definidas seja neste mesmo símbolo, seja em outros símbolos através do uso de opção da ferramenta OORNF de Navegar RNF -> Conseqüências (ver capítulo 4).

2.6 Confrontar operacionalizações existentes no LAL com as que foram achadas via passos 2.2 a 2.4. As operacionalizações presentes no LAL que não estiverem no grafo devem ser representadas no grafo, o que eventualmente pode demandar novas decomposições. As operacionalizações existentes no grafo que não se encontrem no LAL devem ser incluídas no LAL. De uma maneira geral, operacionalizações estáticas se traduzem em noções de símbolos e dinâmicas em impactos.

2.7 No caso de dificuldades em se decompor o RNF (passos 2.2 a 2.4) a identificação de possíveis operacionalizações já descritas no LAL (passo 2.5) pode servir como uma ajuda num processo que alterne abordagens bottom-up e top-down de decomposição deste RNF.

A construção do grafo de RNFs é feita caminhando-se por todas as entradas definidas no LAL que já contenha os RNFs representados. Todo símbolo que possuir RNFs irá gerar um grafo de RNF, sendo que este RNF junto com o símbolo formarão a raiz de um grafo. Em um momento futuro dois ou mais grafos poderão ser agregados em um só por ter o engenheiro de software verificado que estes grafos são decomposições de um grafo maior. Um exemplo disto pode ser visto na Figura 5.3, onde originalmente tínhamos dois grafos distintos, um para o **sensor de movimento** e outro para o **sensor de luz**. Posteriormente avaliamos que na verdade ambos seriam a decomposição do RNF *Custo Operacional* aplicado a **sensores**.

A Figura 5.4 mostra um exemplo da formação de um grafo de RNF começando de um símbolo do LAL, extraído da ferramenta OORNF, para o símbolo **Sensor de Movimento**. Uma vez definida a raiz deve-se então decompor esta meta até chegar as operacionalizações que irão satisfazer o RNF. Esta decomposição irá seguir a utilização de métodos de decomposição utilizados por Chung [Chung 00], onde a decomposição de um RNF pode se dar tanto ao nível de tipo como de tópico, conforme explicado anteriormente.



**Figura 5.4 – Raiz de um Grafo de RNF Elicitada com o Uso da Ferramenta OORNF**

Quando estivermos utilizando a decomposição por tipos, devemos utilizar alguma base de conhecimento ou checklist de RNFS (Ver [Chung 00]) para nos orientar em como decompor RNFS. Em nossa estratégia utilizamos a base de conhecimento suportada pela ferramenta OORNF, que mostra quais os RNFS secundários que decompõe um dado RNF primário para nos auxiliar nesta decomposição (ver Figura



4.3). É importante, entretanto, ter em mente que esta base não é completa e, portanto, devemos sempre explorar possíveis novas fontes de conhecimento em RNFs, com a conseqüente atualização da base.

Tomando como exemplo o RNF *Custo Operacional* relacionado a sensor de movimento, representado por Custo Operacional [Sensor de Movimento], poderíamos pensar em decompor esta meta nas submetas de sala e corredor, ou seja, teremos que ter operacionalizações diferentes dependendo da localização do sensor de movimento. É necessário ressaltar que, neste caso, as decomposições também são símbolos do LAL, mas isto não é mandatório.

Uma vez decomposta em sala e corredor, temos de verificar se alguma outra decomposição é necessária. Notamos então que temos ainda que especificar como a necessidade de satisfazer a baixos custos operacionais pode ser satisfeita quando observada da ótica do uso de sensores de movimento em salas e corredores. Verificamos, junto ao cliente ou por conhecimento do domínio, que para tal é necessário existirem as ações de desligar as luzes das salas após T3 minutos destas estarem vazias e desligar as luzes do corredor após T2 minutos deste estar vazio. A representação deste processo pode ser vista na Figura 5.5

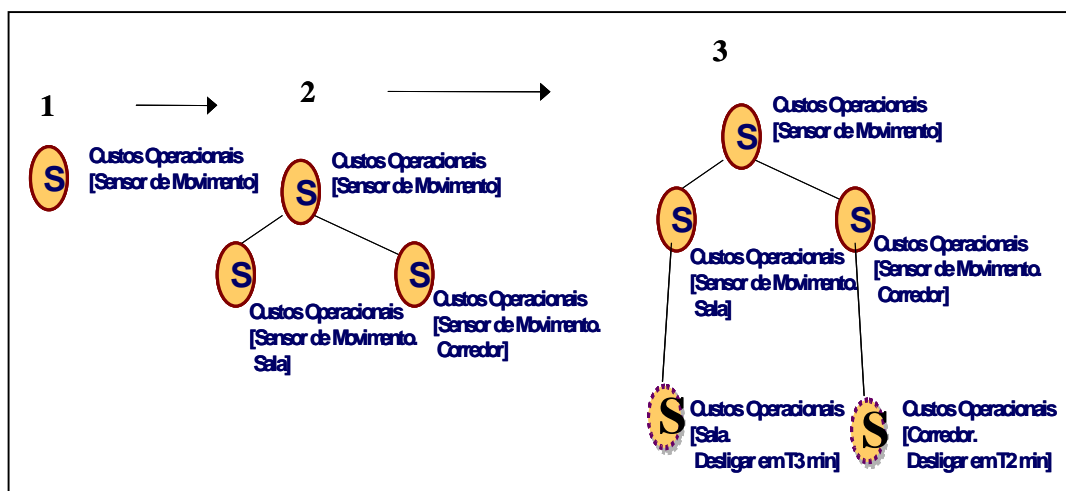
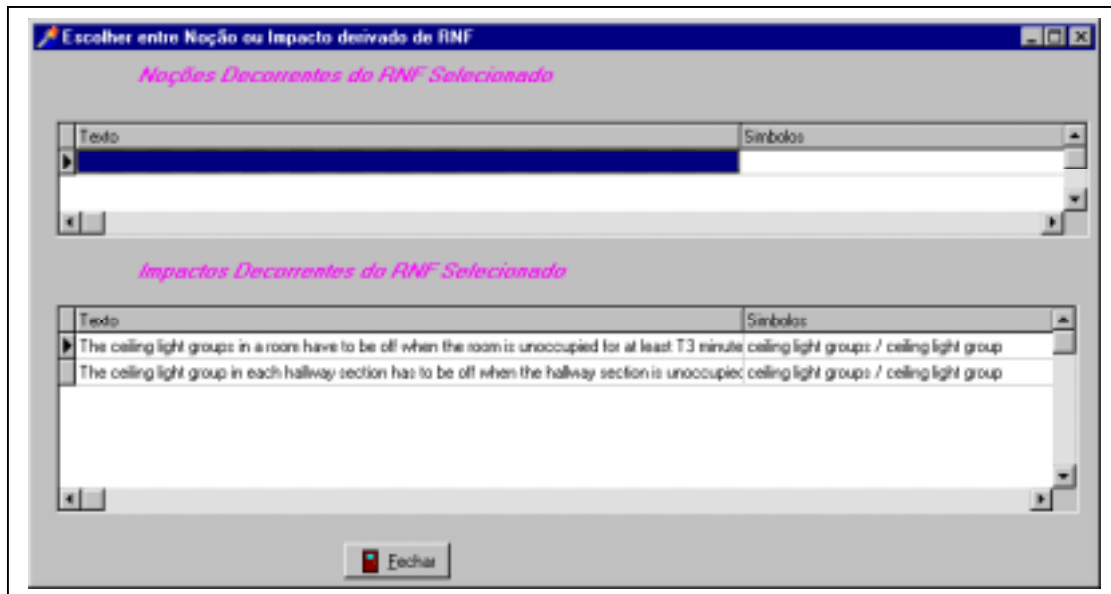


Figura 5.5 – Exemplo da Construção do Grafo de RNFs

Por fim, verificamos no LAL quais as noções e impactos existentes que são consequência da necessidade de satisfazer o RNF *Custo Operacional* para o símbolo **Sensor de Movimento**. A Figura 5.6 mostra as consequências advindas deste RNF extraídas da ferramenta OORNF, consultando-se as consequências advindas do RNF *Custos Operacionais* aplicado ao símbolo **Sensor de Movimento**. Verificamos nesta figura que este RNF desencadeia a necessidade de dois impactos em outro símbolo e nenhuma noção. Estes impactos são: *O grupo de luzes de uma sala deve estar desligado depois que a sala esteja desocupada há T3 minutos* e *O grupo de luzes de cada corredor deve estar desligado depois que o corredor esteja desocupado há T2 minutos*. Ou seja, coincidem com as operacionalizações já encontradas durante a decomposição do grafo.

Suponhamos, entretanto, que ao tentar decompor o RNF *Custo Operacional* para o símbolo **Sensor de Movimento**, o engenheiro de software tivesse encontrado dificuldades de prosseguir. Utilizando-se da heurística 2.7 ele poderia buscar as consequências advindas deste RNF conforme abordado no parágrafo anterior chegando, por consequência, às mesmas operacionalizações necessárias. Ao analisarmos estes textos, podemos ver os símbolos do LAL **Sala** e **Corredor** e ver que para cada um deles teremos um comportamento específico. Como ambos os símbolos são símbolos do LAL e de cada um decorrem ações diferentes, a melhor maneira de se representar esse RNF seria realmente decompondo a meta raiz em duas submetas (sala e corredor) e então novamente decompondo cada uma nas suas respectivas operacionalizações, obtendo novamente o grafo da Figura 5.4.



**Figura 5.6 – Analisando as Conseqüências de um RNF**

É necessário que o engenheiro de software tenha atenção para verificar ao incluir uma operacionalização se esta é refletida pelo LAL. Se isto não acontecer o mesmo deverá ser atualizado para refletir essa operacionalização. Por exemplo, ao decompor a raiz do grafo mostrado na Figura 5.4 chegamos à operacionalização de desligarmos as luzes da sala após esta estar desocupada por mais de T3 minutos, como mostrado na Figura 5.5. Devemos então verificar se o símbolo *Sala* do LAL espelha esta operacionalização.

### 5.3 - Identificar interdependências e Resolver Conflitos

A ultima etapa da elicitação dos RNFs consiste em identificar quais as interdependências, tanto positivas quanto negativas, e resolver os possíveis conflitos que venham a ser identificados.

Muitas interdependências poderão ser identificadas logo após representarmos uma ou duas operacionalizações no grafo. Isto pode acontecer simplesmente pelo fato de que, a representação destas operacionalizações visualmente próximas facilita que o

engenheiro de software tenha sob sua atenção duas ou mais operacionalizações que, quando representadas separadamente, não teriam suas interdependências tão notadas. Ou seja, que pelo simples fato de as vermos representadas lado a lado nos apercebemos de que a semântica expressada em uma operacionalização terá influência negativa ou positiva na outra. Outra fonte de identificação de interdependências reside no conhecimento do domínio que o engenheiro de software tem. Quanto maior este conhecimento mais facilmente ele irá identificar possíveis interdependências com outros grafos enquanto ainda estiver representando uma dada operacionalização.

Para que não fiquemos na dependência de que a organização dos grafos favoreça a identificação de interdependências nem do conhecimento que o engenheiro de software possa ter do domínio, propomos algumas heurísticas para sistematizar a análise dos grafos de RNF. São elas :

1. Comparar todos os grafos de um mesmo tipo. Por exemplo, todos os grafos relativos ao RNF *Seguro*.
2. Comparar grafos de tipos possivelmente conflitantes. Por exemplo, grafos que envolvam *Segurança* e *Performance*, mesmo quando aplicados a diferentes tópicos.
3. Comparar os grafos por pares de forma a comparar um dado grafo com todos os outros, sem contudo, repetir comparações que já tenham sido feitas anteriormente.

A terceira heurística poderá ser utilizada junto com as 1 e 2 no caso de existirem muitos grafos a serem comparados.

Em seguida detalharemos cada uma dessas heurísticas.





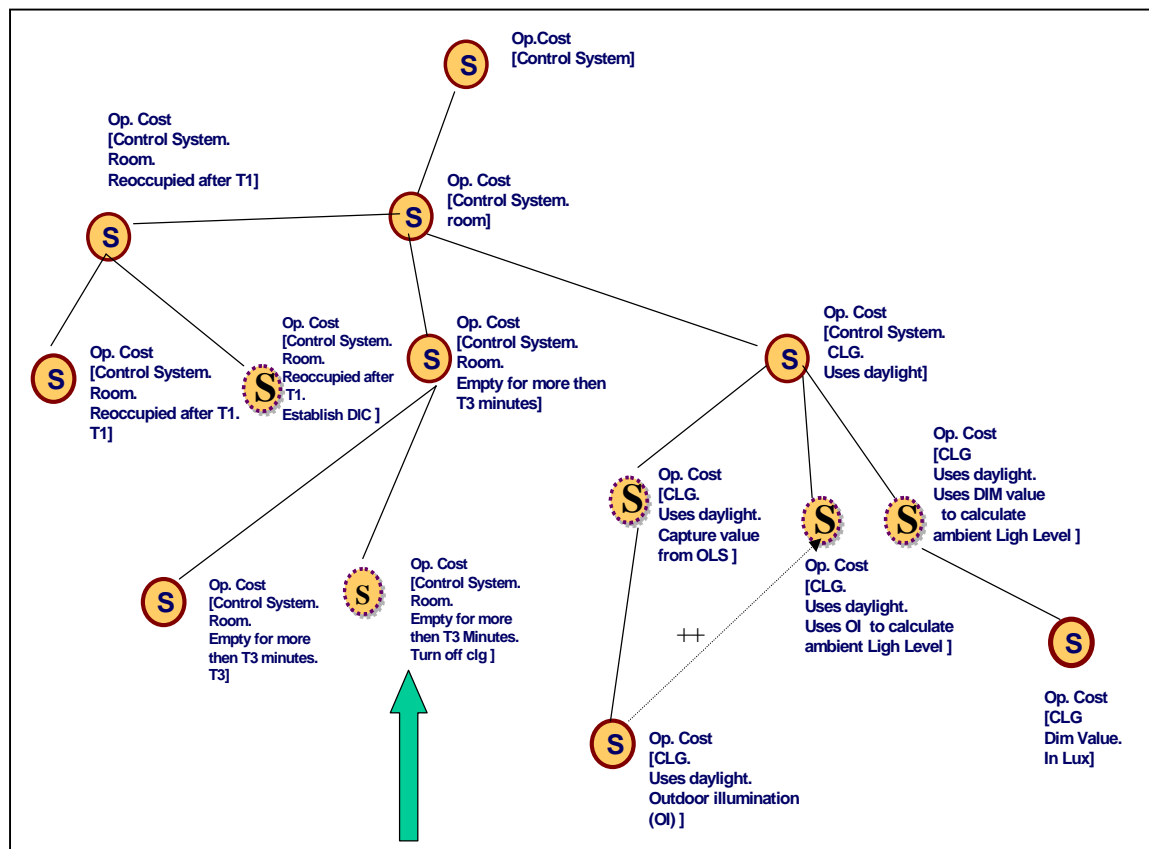
Um exemplo de RNFs que constantemente são conflitantes é os RNFs de *Performance* e *Segurança*. A introdução de mecanismos de controle de acesso frequentemente acarreta ônus à performance do sistema. Portanto, o engenheiro de software deve confrontar os grafos de RNF que sejam do tipo *Performance* com os do tipo *Segurança*. Um exemplo disto pode ser visto na Figura 5.8.

Outro exemplo de influência conhecida pode ser encontrado na necessidade de autenticação de acesso. Quando levada a níveis de validação muito rígidos esta validação pode influir negativamente para a usabilidade do sistema bem como para sua performance. Entretanto, poderá contribuir positivamente para a confiabilidade da informação.

### **5.3.3 - Comparação Por Pares de RNFs**

A comparação por pares tem por objetivo garantir que todos os RNFs foram avaliados quanto à influência de um no outro. Quando lidamos com um grande número de grafos, passa a ser fácil a não observância de possíveis influências devido ao grande número de informações com que estamos lidando. Conseqüentemente, o uso da comparação por pares pode trazer um aumento na detecção de possíveis influências.

Para efetivarmos esta comparação elegemos um grafo para começarmos e o comparamos individualmente com todos os outros grafos. Em seguida elegemos um segundo e repetimos o procedimento tomando o cuidado de não repetir a comparação feita na iteração anterior. É interessante também evitar comparações que já tenham sido feitas em conseqüência da aplicação das outras duas heurísticas de forma a minimizarmos o esforço.

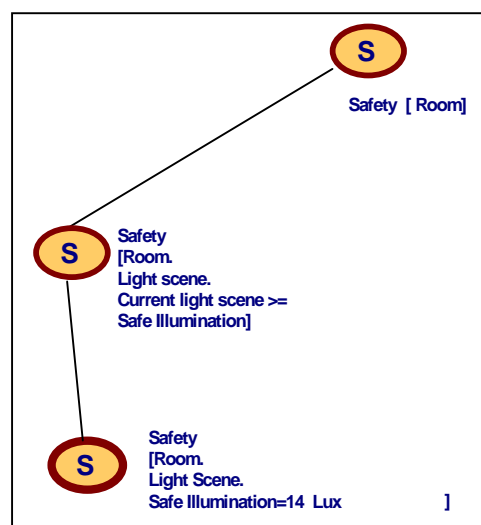


**Figura 5.9 – Grafo Antes de Comparação por Pares**

Esta heurística sem dúvida implica num considerável tempo a ser consumido nesta tarefa e apesar de ter demonstrado um bom retorno em nossos estudos de caso, deve ter sua utilização avaliada com cuidado sob pena de nos estendermos demasiadamente nesta tarefa com conseqüências não desejadas ao nível de prazo e custo de desenvolvimento. Uma maneira de diminuirmos o custo da implementação desta heurística é manter um bom registro de grafos já comparados não realizando nesta etapa comparações que já tenham sido efetuadas.



Um exemplo de conflito identificado a partir do uso desta heurística pode ser visto nas Figuras 5.9 e 5.10 e 5.11. A Figura 5.9 mostra o grafo referente ao RNF *Custos Operacionais* aplicado ao símbolo **Sistema de Controle**. Podemos ver nesta figura, que a operacionalização, destacada por uma seta, de desligar as luzes após T3 minutos esta marcada como satisfeita, ou seja, neste momento havíamos decidido que o sistema iria implementar sem restrições uma ação desligar as luzes de uma sala após a **mesma estar desocupada por mais de T3 minutos**.



**Figura 5.10 – Segundo Grafo para Comparar**

Quando realizávamos a comparação par a par, acabamos por colocar o grafo da Figura 5.9 junto ao da 5.10, verificamos que existiria um conflito entre desligar as luzes de uma sala e o RNF *Seguro* aplicado a **Sala** que estabelece que a menor iluminação de uma sala ocupada deve ser 14 lux. Verificamos nesse momento que simplesmente apagar as luzes de uma sala confiando cegamente nos sensores poderia ser extremamente conflitante com a segurança do sistema. Decidimos então, que antes de desligar a luz de uma sala o sistema deveria emitir um aviso e solicitar uma confirmação de algum usuário. Só na falta desta confirmação é que as luzes seriam



O engenheiro de software deve ter particular atenção para verificar se a resolução de conflitos não levará a mudanças no léxico por causa de uma eventual adição de uma nova operacionalização que não esteja ainda refletida no LAL. Se isto acontecer o LAL deve ser então atualizado.

As Figuras 5.8, 5.9, 5.10 mostradas na seção anterior exemplificam uma típica resolução de conflitos. É importante ressaltar que a submeta de desligar as luzes após T3 minutos, que anteriormente era marcada como satisfeita, passa a ser apenas parcialmente satisfeita, já que isso só será executado caso não haja cancelamento por parte de um usuário.

## 6 - Estendendo os modelos da visão funcional

Um importante aspecto no processo de lidar com RNFs reside na capacidade de avaliar as influências da satisfação dos RNFs nos modelos tanto funcionais quanto não funcionais. No capítulo 5 demonstramos como lidar com esses problemas ao nível de grafos de RNFs, ou seja, interdependências entre RNFs.

Entretanto, existe também a necessidade de se lidar com impactos advindos da satisfação de RNFs nos modelos utilizados para expressar a visão funcional. Este tema será detalhado no capítulo 7.

Para permitir que possamos lidar com RNFs ao nível de modelos que expressam a visão funcional, propomos algumas extensões aos cenários, diagrama de classes, seqüência e colaboração. Estas extensões visam a facilitar a representação dos impactos advindos da satisfação de RNFs, analisar seus impactos e ainda rastreá-los reversamente para os grafos de RNFs e o LAL.

### 6.1 - Extensão aos Cenários

No que tange aos cenários, a única extensão proposta por nós visa a facilitar o rastreamento da origem de determinados episódios, recursos ou atores utilizados em um cenário que tenha sido originado da visão não funcional.

Para isto, propomos que toda a vez que um RNF tenha alguma consequência em um cenário, um novo recurso, novo ator ou episódio, esta influência seja representada imediatamente após esta ocorrência. Iremos representar essa influência explicitando o ponto no grafo de RNFs que originou esta inclusão.

Esta representação irá obedecer a seguinte sintaxe: *{RNF [Símbolo do LAL]}*, e será colocada como uma restrição conforme mostrado na Figura 2.9.

A representação dos impactos advindos da satisfação de RNFs nos cenários utilizará a mesma sintaxe proposta por Leite [Leite 97] [Breitman 98]. Ou seja, a satisfação de RNFs irá impactar na inclusão de novos episódios, atores e recursos.

**Cenários**

**Título**  
Define [Light Scene](#).

**Objetivo**  
Define [user](#) preferred [light scene](#).

**Contexto**  
[4th floor of building 32](#), [user](#) in [room](#).

**Atores**  
[user](#), [control system](#).

**Recursos**  
default [light scene](#) value, [control panel](#).

**Episódios**  
 1. [user](#) places himself/herself near [Control Panel](#).  
 2. [user](#) retrieves cdefault [light scene](#).  
 3. [user](#) changes desired values for his [chosen light scene](#).  
 4. [Control System](#) determines how much lux does this [light scene](#) represents.  
 Constraint: {Safety [[Room](#)]}.  
 5. If [light scene](#) < [safe illumination](#) then [system](#) issues a warning.  
 Constraint: {Safety [[Room](#)]}.  
 |

Navigation buttons: [Previous] [Next] [First] [Last] [Add] [Remove] [Check] [Close]

**Figura 6.1 – Exemplo de Cenário com a Representação do RNF de Origem**

Um exemplo disto pode ser extraído de um dos estudos de caso, sistema de controle de iluminação, onde acharemos um RNF *Seguro* associado ao símbolo do LAL Sala. A satisfação deste RNF determina que em nenhum momento a quantidade de luz em um cômodo pode ser inferior a 14 Lux. Ao avaliarmos o cenário Definir Esquema de Luz, verificamos que para satisfazer este RNF seria necessário acrescentar dois episódios. O primeiro para calcular o equivalente em lux do esquema de luz escolhido pelo usuário e o segundo para verificar se este valor é superior ou igual a 14 lux. A Figura 6.1 mostra este cenário utilizando-se a ferramenta OORNF

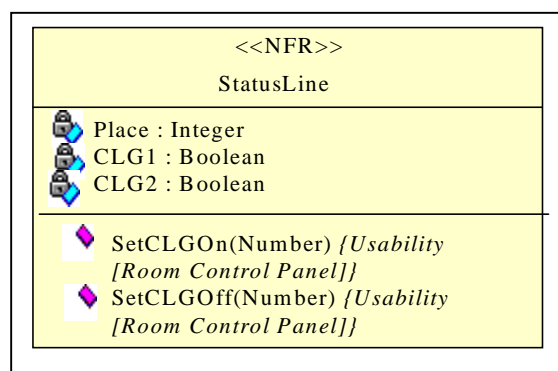
[Neto 00] com os novos episódios incluídos, junto com a notação de origem destes novos episódios.

## 6.2 - Extensão ao Diagrama de Classes

O diagrama de classes foi estendido para suportar a representação dos efeitos advindos da satisfação dos RNFs encontrados durante o ciclo não funcional. A primeira extensão proposta tem o intuito de representar classes que tenham sido incluídas no diagrama por força da necessidade de se satisfazer um determinado RNF.

A necessidade de se representar uma classe que não antes existia, advinda da integração dos RNFs ao diagrama de classes, pode ocorrer tanto pelo fato de termos identificado classes que foram esquecidas durante o desenho do sistema, quanto por decisão de desenho particular do engenheiro de software. Para representar esta classe, usaremos um estereótipo representando que aquela classe é advinda da necessidade de satisfazer um RNF.

É importante ressaltar que não existe a obrigatoriedade de se criar classes para satisfazer RNFs. Essa decisão será de total responsabilidade do engenheiro de software e poderá e deverá ser evitada para evitar comprometimento da qualidade do desenho final do sistema.



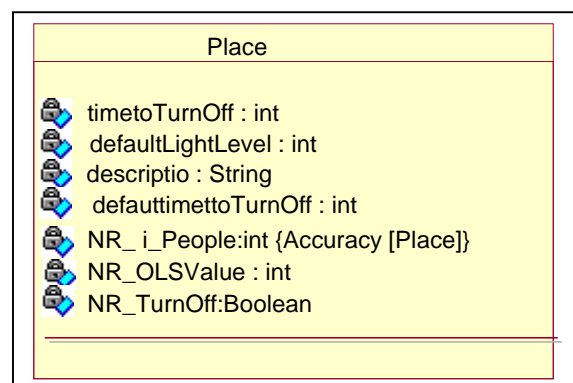
**Figura 6.2 – Exemplo de um Classe Estereotipada para Representar um RNF**

A Figura 6.2 nos mostra um exemplo, extraído do estudo caso de controle de iluminação, onde a classe Linha de Estado passa a ser incluída no diagrama de classes

para atender ao RNF *Usabilidade* relacionado ao tópico (classe) **Sala**. Esta classe é necessária para que o ocupante de uma sala possa saber de maneira fácil quais os grupos de luzes que estão acesos.

A representação de atributos derivados da necessidade de satisfação de RNFs é representada da mesma forma que descrito por Rumbaugh [Rumbaugh 99], com exceção do fato de que todo atributo advindo da satisfação de um RNF deverá ser precedido de “NR\_”.

Caso o engenheiro de software não considere óbvia a origem deste atributo, ele poderá representá-la da mesma forma como descrito na sessão 6.1 para os cenários, ou seja, *{RNF [Símbolo do LAL]}*, em itálico. Via de regra, a representação de novos atributos leva a inclusão de novas operações que os manipulem, portanto, toda vez que um atributo for incluído, deve-se estar atento para a eventual necessidade de novas operações serem adicionadas à classe. Como veremos no capítulo 7, atributos serão adicionados a uma classe ou para suportarem uma nova operação ou pela presença de operacionalizações estáticas em grafos de RNFs. A Figura 6.3 mostra um exemplo da representação de atributos que satisfazem RNFs.



**Figura 6.3 – Exemplo da Representação de Atributos advindos de RNFs**

A representação de operações advindas da satisfação de RNFs será representada segundo o padrão apresentado por Rumbaugh [Rumbaugh 99] com duas representações adicionais.

A primeira segue o padrão descrito acima para atributos e visa a representar a origem desta operação em termos de visão não funcional, de forma a podermos rastrear uma operação até o grafo de RNFs de onde ela foi originada. Este rastreamento é muito importante durante a evolução do desenho do software e sua representação torna-se, portanto, obrigatória. A importância desta representação advém do fato de que, ao representarmos novas classes, operações e atributos, muitas vezes acabamos por criar conflitos com o desenho do software existente e poder rastrear a origem deste conflito de volta à sua origem é importante para podermos tomar decisões sobre o que pode ser negociado e até que ponto.

Além de representar o RNF que originou a operação, podemos necessitar representar condições de contorno que envolvem a execução desta operação. Isto será feito representando-se estas condições sob forma de restrições entre chaves com fonte courier. O conteúdo destas condições pode ser escrito em formato livre, porém, recomenda-se o uso, sempre que possível, de expressões em OCL [Rational 97].

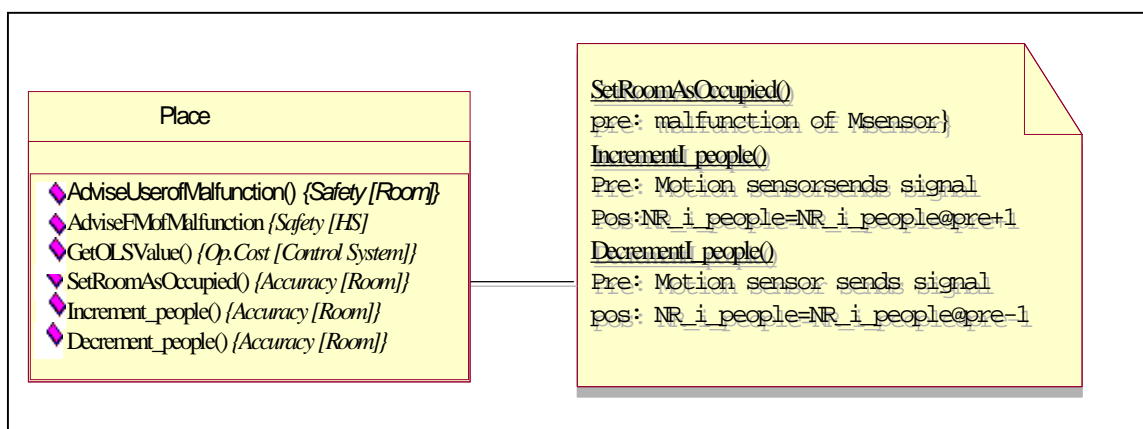
Novamente utilizando o estudo de caso de controle de iluminação, vemos que existe um RNF *Precisão* associado à **Sala** que determina que deve haver precisão na determinação de que uma sala está vazia, e que isto é obtido contabilizando-se a quantidade de pessoas que entram e saem de uma sala, ao invés de somente detectar falta de movimento na sala. A satisfação deste RNF levou à inclusão de uma operação “*incrementaPessoa()*” na classe **Lugar**, onde esta operação será realizada a partir do recebimento de um sinal do sensor de movimento e resultará no aumento em uma unidade no atributo que registra a quantidade de pessoas na sala. Como a simples detecção de movimento na sala não pode determinar a saída ou entrada de uma pessoa, a determinação de se há realmente a entrada de uma pessoa ou não será feita



em conjunto com a leitura da sinalização dos sensores de abertura de porta. Isto seria representado da seguinte forma:

```
Increment_people() { Pre: Motion sensor sends signal
                    Door Sensor signs
                    Post:NR_i_people=NR_i_people@pre+1 }
```

No caso do engenheiro de software achar que a inclusão destas condições representa uma demasiada sobrecarga visual ao modelo, ele poderá representá-la dentro de um comentário associado à classe conforme mostrado na Figura 6.4.



**Figura 6.4– Exemplo de Condições de Exceção Representadas em Comentários**

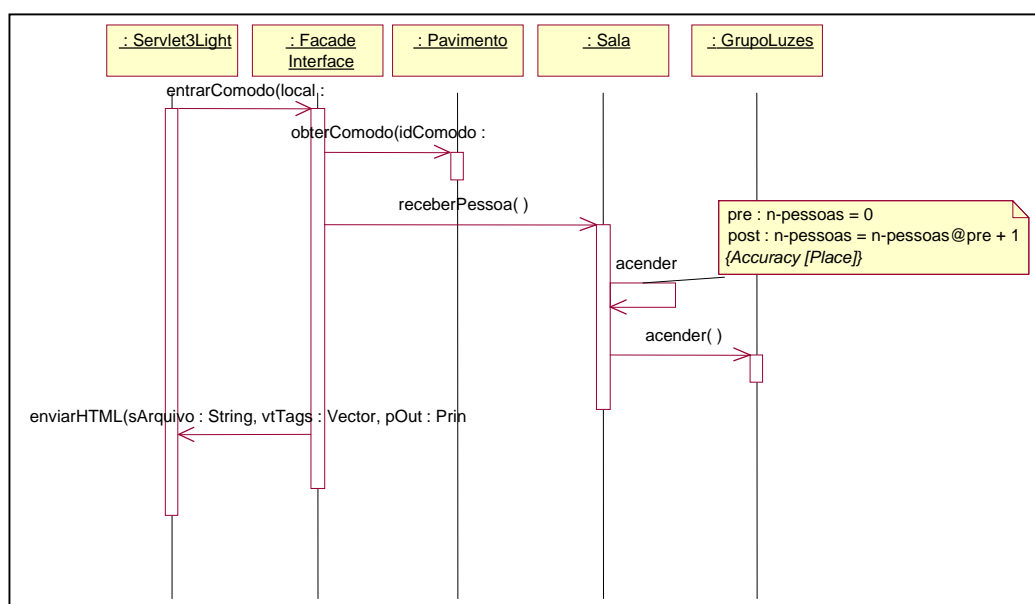
## 6.3 - Extensão aos Diagramas de Sequência e Colaboração

A extensão proposta para os diagramas de sequência e colaboração visa, fundamentalmente, incluir nestes diagramas condições de contorno envolvidas no envio de mensagens que advenham ou tenham colaboração na satisfação de um RNF.

Da mesma forma que no diagrama de classes, representaremos estas condições em comentários de maneira livre, ou preferencialmente, pelo uso de expressões OCL [Rational 97]. Este comentário estará ligado à mensagem a qual se aplica a condição de contorno. Adicionalmente representaremos o RNF de onde se originou esta condição utilizando a mesma sintaxe do diagrama de classes, ou seja, { *RNF [Símbolo do LAL]* } em itálico. A representação da origem do RNF, apesar de redundante com a

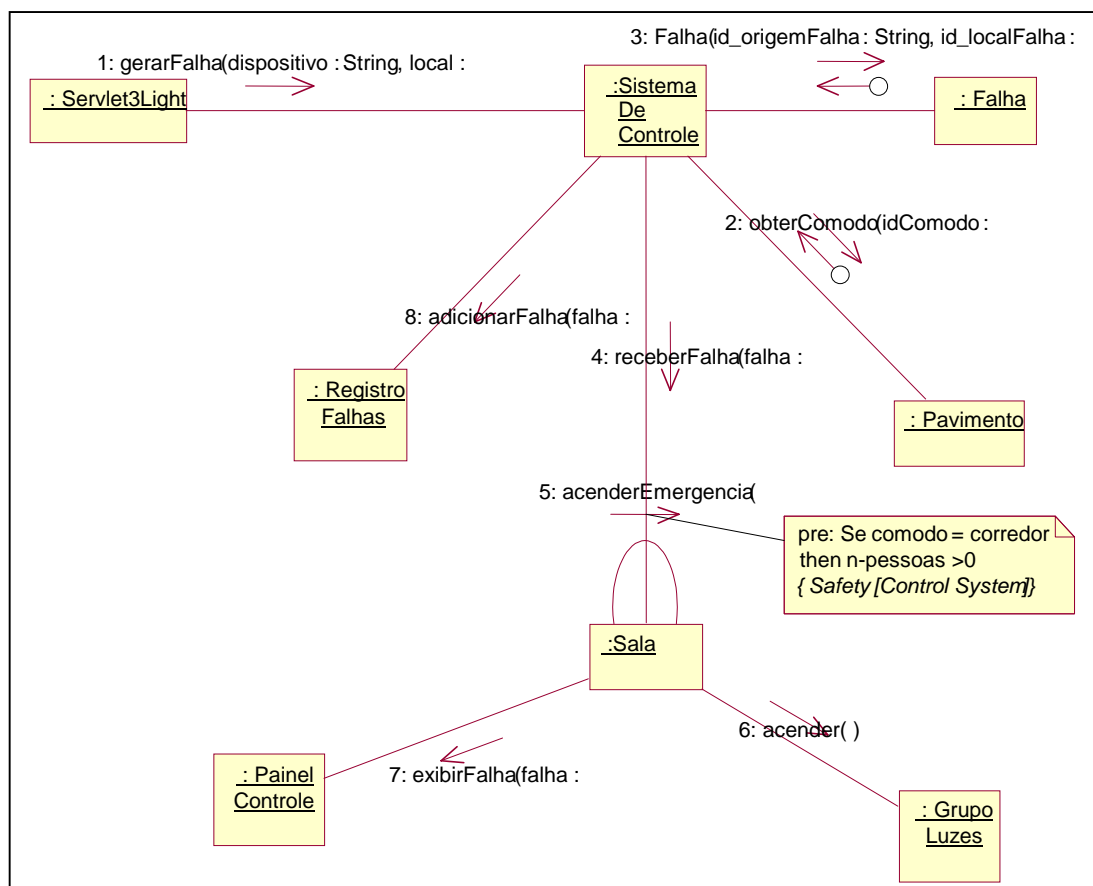
existência da mesma na especificação da operação da classe à qual a mensagem está associada, auxilia na pronta identificação da origem daquela mensagem ou das condições impostas à mesma. Essa pronta identificação facilita o processo de revisão e atualização de especificações, visto que pelo fato de RNFs serem usualmente abstratos, estes tendem a não ficar claros na mente dos engenheiros de software, e portanto, serem esquecidos durante o processo de revisão ou durante a inclusão de nova funcionalidade no sistema.

A Figura 6.5 mostra um exemplo da aplicação desta extensão a um diagrama de seqüência, “usuário entra em sala”, extraído do estudo de caso de controle de iluminação. Neste diagrama podemos observar que, para que a mensagem “acender” seja enviada de sala para a própria sala, é necessário que o número de pessoas na sala seja maior que zero ( $n\text{-pessoas} > 0$ ). Podemos ainda observar que ao final desta mensagem o atributo  $n\text{-pessoas}$  obrigatoriamente terá sido incrementado de um. Vemos ainda que esta condição foi determinada pelo RNF *Precisão* aplicado à *Sala*, de forma que quaisquer dúvidas sobre a razão da existência deste controle poderão ser rastreadas de volta ao grafo de RNFs que a originou.



**Figura 6.5 – Exemplo de Extensão ao Diagrama de Seqüência para Expressar RNFs.**

A Figura 6.6 mostra um exemplo aplicado ao diagrama de colaboração, “ocorre mau funcionamento do sensor de luz”, também extraída do estudo de caso de controle de iluminação. Neste exemplo, podemos ver que no caso de falhas ocorridas no sensor, a mensagem “acenderEmergência()”, para ser executada, deve ter como pré-requisito o fato do cômodo onde houve a falha ser um corredor e este estar ocupado. Isto vem da observação do RNF *Seguro* aplicado à classe Sistema de Controle, e advém do fato de que, no caso do mau funcionamento ser numa sala, a mensagem será enviada independente de estar ocupada ou não.



**Figura 6.6– Exemplo da Extensão ao Diagrama de Colaboração para Representar RNFs**

## **7 – Integrando as Visões Funcionais e Não Funcionais**

Uma vez que os ciclos funcionais e não funcionais existem independentemente um do outro, torna-se necessário que existam pontos de convergência onde a integração de ambas as visões seja feita, mais especificamente, onde possamos representar na visão funcional as necessidades da visão não funcional.

O processo de integração baseia-se no uso do LAL como âncora para a construção tanto do grafo de RNFs quanto dos cenários e do diagrama de classes. Através do uso desta âncora, o processo de integração das visões é facilitado, ficando centrado na busca da convergência de um símbolo que apareça num modelo e em outro e na avaliação dos impactos decorrentes da integração de ambos.

Em seguida iremos detalhar o processo para cada um dos artefatos utilizados na visão funcional.

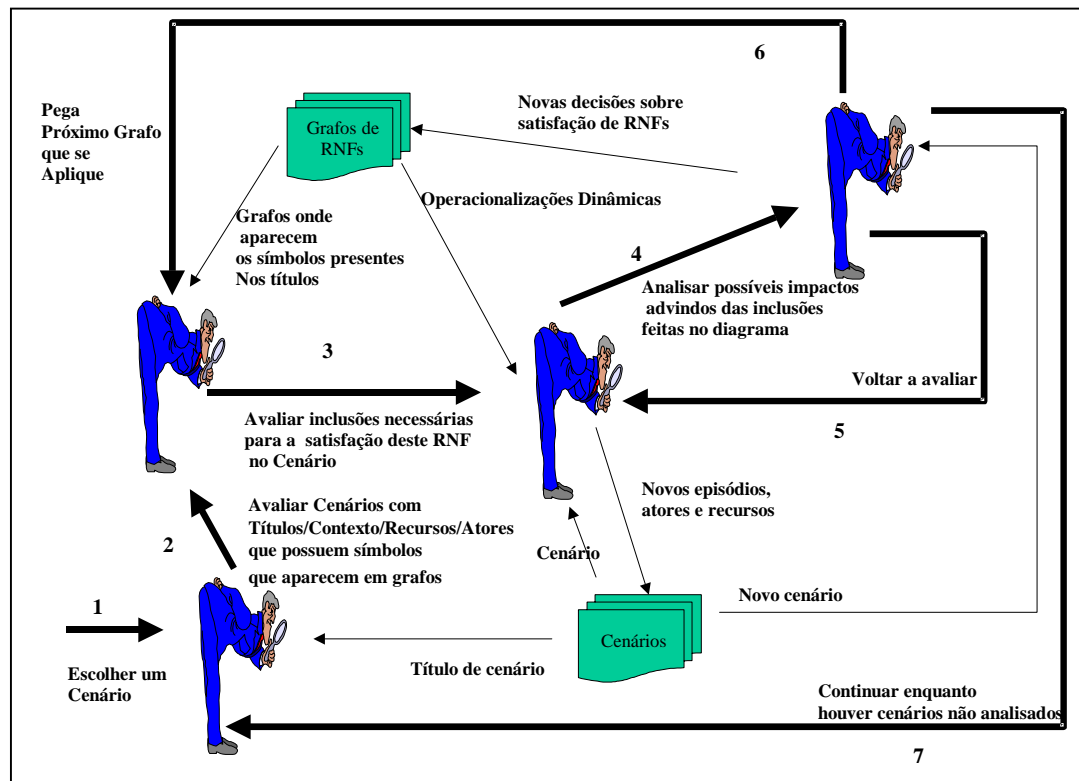
### **7.1 - Integrando RNFs aos Cenários**

A Figura 7.1 ilustra o processo de integração de RNFs aos cenários elicitados que representam a visão funcional do domínio.

Podemos ver nesta figura que o primeiro passo deste processo, etapa 1, é escolher um cenário para analisarmos. Os passos seguintes serão repetidos para cada um dos cenários elicitados até que todos tenham sido analisados, representados pela etapa 7.

Uma vez escolhido um cenário, verificamos quais os símbolos do LAL que aparecem no título deste cenário, etapa 2. Para cada símbolo, verificamos se no conjunto de grafos de RNF existe algum onde o símbolo apareça. Se existir,

avaliaremos cada uma das operacionalizações dinâmicas deste RNF, etapa 3, para ver se elas se aplicam ao cenário sendo analisado. Caso se apliquem, analisamos o cenário para ver se já existem episódios e por consequência, atores e recursos que garantam a satisfação destas operacionalizações. Este procedimento é repetido para o contexto, recursos e atores constantes do cenário.



**Figura 7.1 – Integração de RNFs à Cenários**

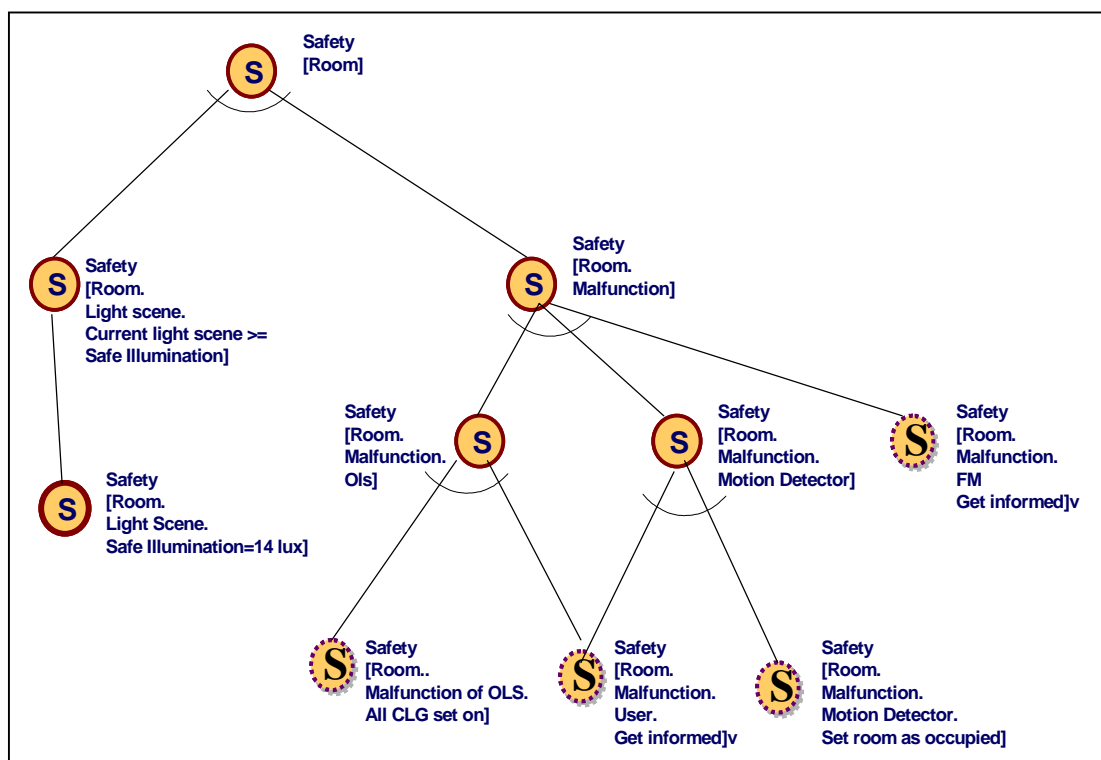
Se os episódios existentes não forem suficientes para garantir a satisfação deste RNF, incluiremos um ou mais episódios de forma a satisfazer este RNF, seguindo o padrão mostrado na seção 6.1, bem como os possíveis novos atores e recursos.

Uma vez que tenhamos feito as inclusões necessárias, passamos a analisar, etapa 4, que possíveis conflitos possam decorrer da inclusão destes novos episódios neste cenário. Caso algum conflito seja detectado, deveremos retornar ao grafo de RNF para verificar o quanto prejudicial seria a não satisfação deste RNF, e avaliar junto com o cliente quais as opções possíveis para contornarmos o conflito. As opções

podem ir desde a simples não satisfação deste RNF até a alterações na funcionalidade do software em detrimento de um RNF, passando por soluções intermediárias que possam comprometer ambas as visões dentro de limites razoáveis.

Quaisquer alterações ao nível de RNFs deverão ser representada no grafo de RNFs e novamente procederemos à avaliação, etapa 5, de quais impactos este RNF terá agora neste cenário. Este procedimento continuará até que uma solução de compromisso entre ambas as visões chegue a bom termo.

Repetiremos este procedimento, etapa 6, até não encontrarmos mais grafos de RNF que contenham símbolos do LAL existentes no título do cenário.



**Figura 7.2 – Exemplo de Uso de Grafos de RNFs para Avaliar Cenários**

É importante ressaltar que quando procuramos por um símbolo do LAL nos grafos, apesar de procurarmos primeiramente na raiz dos grafos, iremos procurar em todas as submetas que decomponham uma determinada meta. A Figura 7.2 ilustra um exemplo disto. Se estivermos analisando um cenário que tenha o símbolo Sala como parte do título, dos recursos desse cenário, do contexto ou dos atores, ao avaliarmos o

grafo mostrado na Figura 7.2, teríamos que verificar, entre outras coisas, se o cenário em questão satisfaz as necessidades de que um cenário de iluminação que seja escolhido para esta sala satisfaça o RNF *Seguro* de forma a que a iluminação escolhida seja superior a 14 lux.



Figura 7.3 – Cenário Elicitado na Visão Funcional

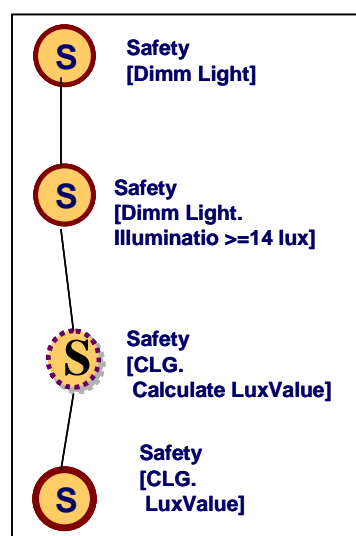
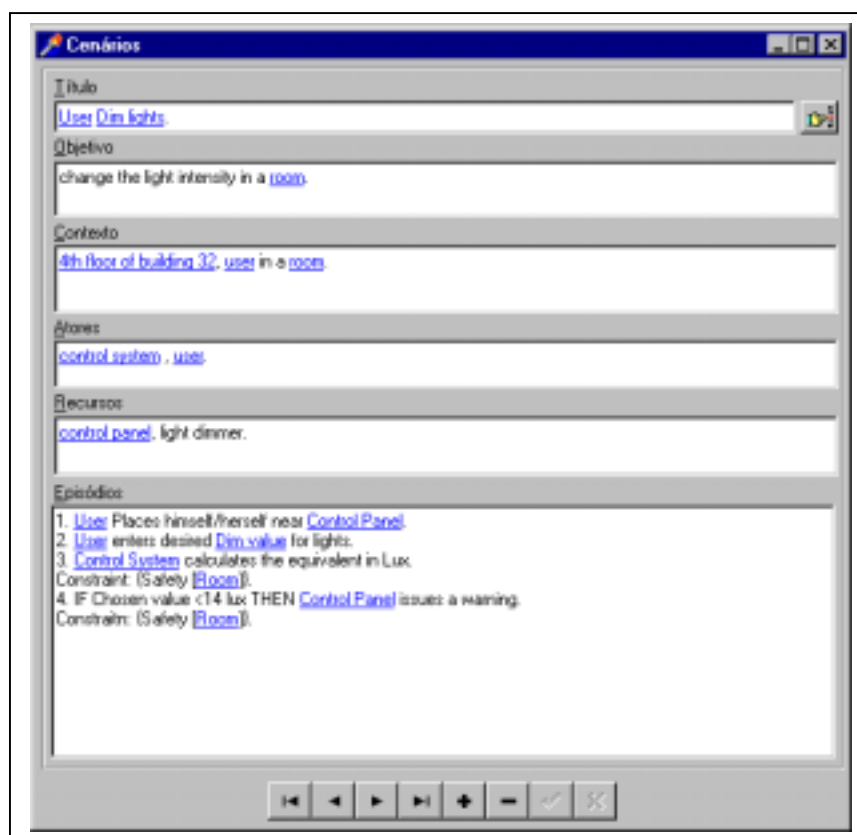


Figura 7.4 – Grafo de RNF com Símbolo Presente no Cenário em Avaliação

Um exemplo da aplicação desta heurística de integração pode ser visto a partir do cenário “Usuário atenua luzes”, extraído do estudo de caso do sistema de controle de iluminação, que é mostrado na Figura 7.3. Neste exemplo, vemos que o título do cenário possui o símbolo **Atenuar Luzes**. Procuramos então grafos de RNF onde este símbolo apareça e encontramos o grafo mostrado na Figura 7.4. Verificamos que este cenário não satisfaz a necessidade de conferir se o usuário não tentar atenuar as luzes para um valor inferior a 14 Lux. Uma vez que isso é constatado incluímos dois novos episódios no cenário para satisfazer essa condição. O primeiro trata da conversão do valor de atenuação expresso naturalmente em percentual para um valor em Lux, e o segundo em comparar esse valor ao valor limite. A Figura 7.5 mostra o cenário resultante



**Figura 7.5 - Exemplo da Aplicação da Heurística de Integração de RNF a Cenários**



## 7.2 - Integrando RNFs ao Diagrama de Classes

O processo de integração de RNFs ao diagrama de classes é bastante semelhante ao processo de integração aos cenários como pode ser visto na Figura 7.6.

Este processo se baseia no fato de o LAL ter sido usado como âncora para a construção de ambos os modelos, ou seja, toda classe deve ser um símbolo do LAL, à exceção de agregações que não necessariamente precisam ser símbolos, e todos os tópicos de RNFs também devem ser símbolos do LAL. As generalizações ficam livres dessa restrição por não necessariamente representarem uma situação existente no domínio. Muitas vezes são abstrações criadas para atender a desenhos específicos ou a padrões de desenho, e portanto não devem se encaixar nessa restrição. Desta forma, analisar posteriormente um diagrama de classes e saber quais os RNFs se aplicam a uma determinada classe se torna um trabalho mais sistemático e menos intuitivo.

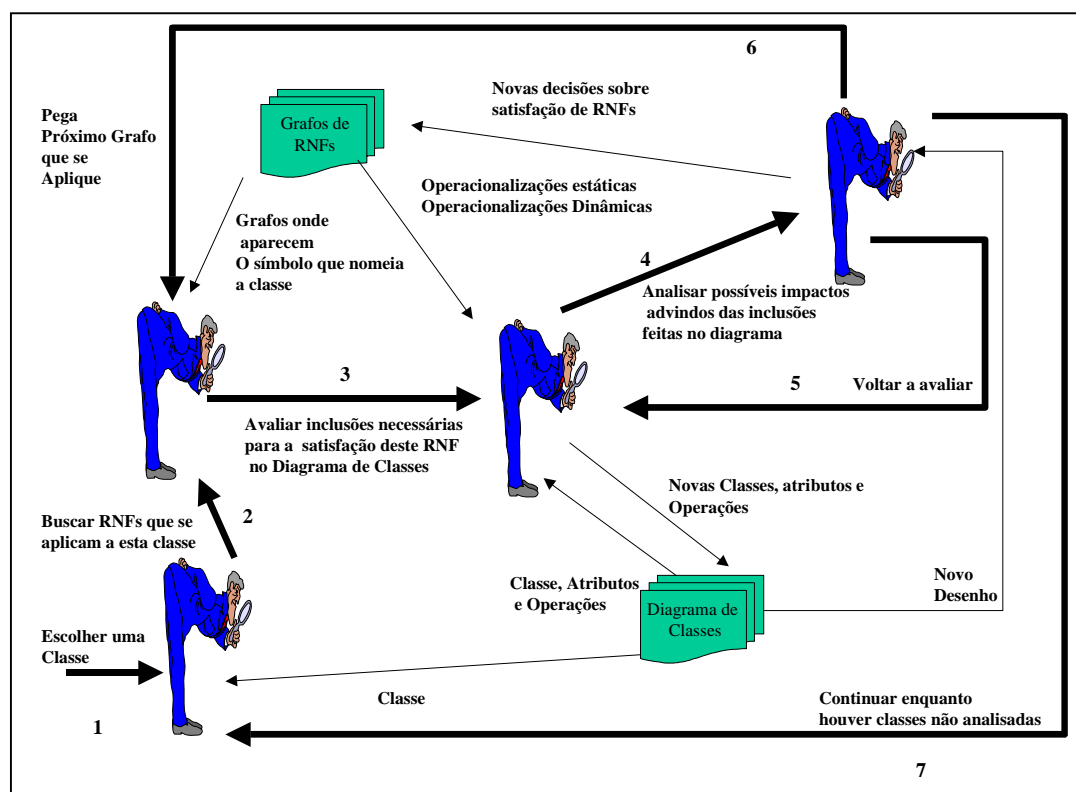


Figura 7.6 – Integração de RNFs ao Diagrama de Classes

Conforme podemos ver na Figura 7.6, o processo de integração começa quando o engenheiro de software escolhe uma classe do diagrama de classes, etapa 1, para avaliar se algum RNF deve ser satisfeito por esta classe. Os passos seguintes serão repetidos para cada uma das classes existentes até que todos tenham sido analisados, conforme indicado pela etapa 7.

Uma vez escolhida a classe, procuramos no conjunto de grafos de RNFs, etapa 2, se existe algum grafo onde o símbolo, ou seja, a classe apareça. Se existir, avaliaremos se cada uma das operacionalizações, tanto estáticas quanto dinâmicas deste RNF, etapa 3, são satisfeitas pela classe, ou seja, **se as operacionalizações estáticas possuem atributos correspondentes e se as dinâmicas possuem operações correspondentes**. Se não existirem, incluiremos os atributos e operações que forem necessários, tendo o cuidado de verificar se, para cada atributo incluído, existem as operações necessárias para manipulá-los, e se, para cada operação incluída, não necessitamos de outros atributos que não tenham ainda sido identificados. Aqui devemos prestar especial atenção no caso desta classe ser uma subclasse de outra classe. *Neste caso, antes de procedermos à inclusão de atributos e operações nesta classe, devemos primeiro nos certificar se a sua superclasse não implementa os mesmos atributo e operações.*

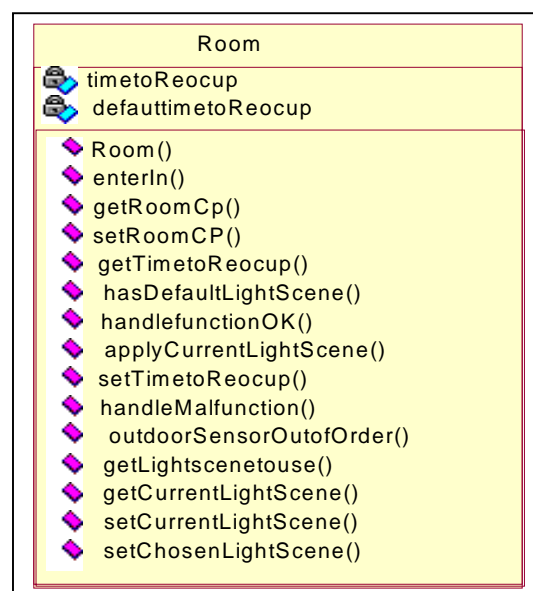
Uma vez que tenhamos feitos as inclusões necessárias, passamos a analisar, etapa 4, que possíveis conflitos possam decorrer da inclusão destes novos atributos e operações. Caso algum conflito seja detectado, deveremos retornar ao grafo de RNF para verificar o quão ruim seria a não satisfação deste RNF e avaliar junto com o cliente, quais as opções possíveis para contornarmos o conflito. As opções podem ir desde uma simples não satisfação deste RNF até a alterações na funcionalidade do

software em detrimento de um RNF, passando por soluções intermediárias que possam comprometer ambas as visões dentro de limites razoáveis.

Quaisquer alterações ao nível de RNFs deverão ser representadas no grafo de RNFs e novamente procederemos à avaliação, na etapa 5, de quais impactos este RNF terá agora na classe que está sendo analisada. Este procedimento continuará até que uma solução de compromisso entre ambas as visões chegue a bom termo.

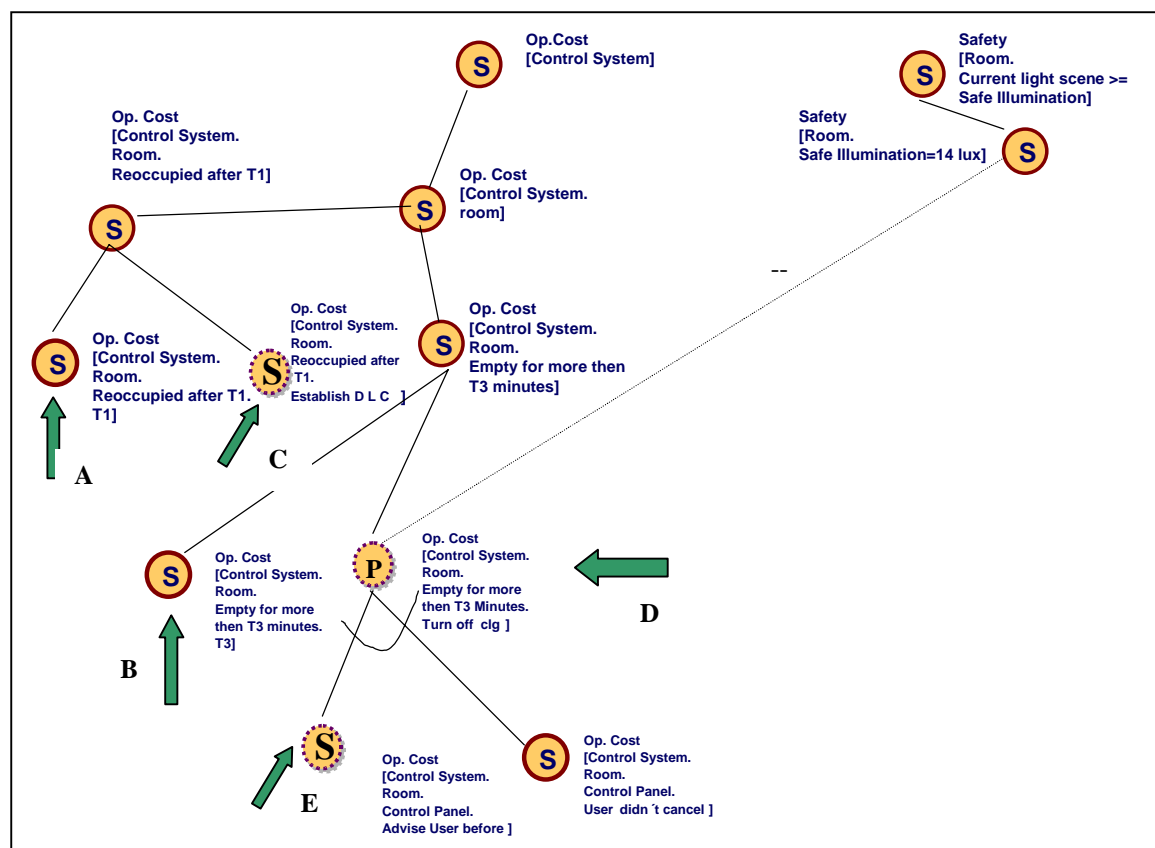
Repetiremos este procedimento, na etapa 6, até não encontrarmos mais grafos de RNF que contenham o símbolo do LAL que nomeia a classe.

As Figuras 7.7, 7.8, 7.9, 7.10 e 7.11 ilustram a aplicação desta heurística utilizando um exemplo extraído do estudo de caso do sistema de controle de iluminação. A Figura 7.7 mostra a classe **Sala** extraída do diagrama de classes gerado pela visão funcional, enquanto a Figura 7.8 mostra um grafo de RNF que detalha necessidades de controlar *Custos Operacionais* a ser aplicado inicialmente à classe **Sistema de Controle**. Consideramos este grafo para efeitos de integração com a classe **Sala** pelo fato de **Sala** ser uma submeta que decompõe este grafo.



**Figura 7.7 – Classe Sala Extraída da Visão Funcional**

Podemos ver no grafo que precisaríamos de dois atributos, T1 e T3 (identificados pelas setas A e B), para satisfazer o RNF, bem como de operações que estabelecessem um cenário de luz escolhido (identificada pela seta C), desliguem as luzes da sala (identificada pela seta D) e que avisem o usuário de que estas luzes serão desligadas (identificadas pela seta E). É importante observar que a operacionalização de desligar as luzes somente será satisfeita no caso de o usuário ser avisado pelo painel de controle e de não haver um cancelamento do procedimento. Isto é necessário para que o RNF *Seguro* seja satisfeito.

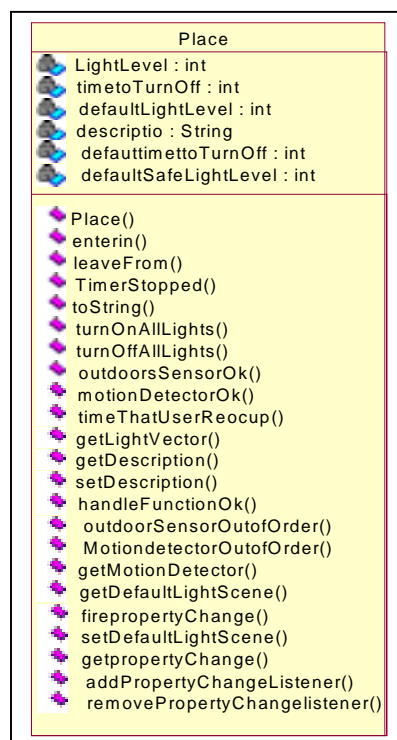


**Figura 7.8 – Grafo de RNF Sendo Integrado à Classe Sala**

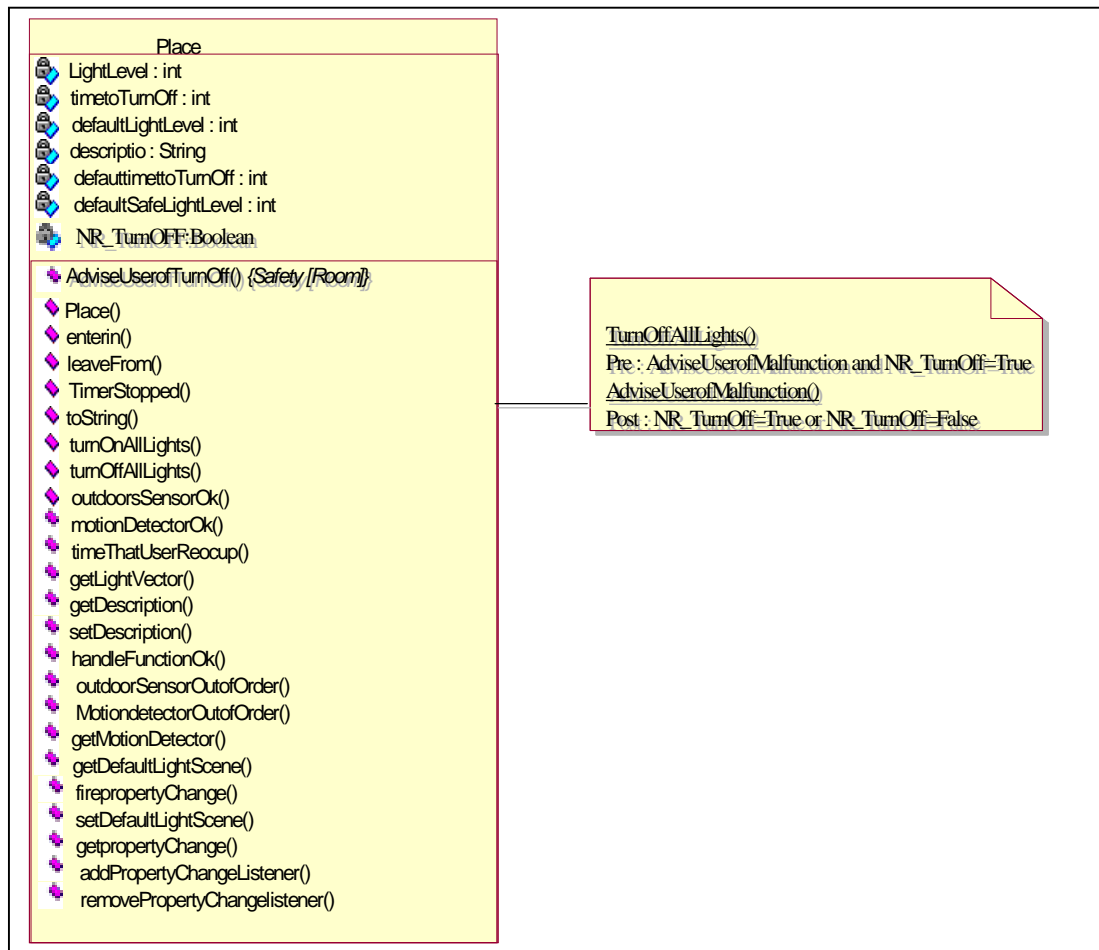
A classe *Sala*, mostrada na Figura 7.7, satisfaz o atributo T1, aqui chamado de “TimeToReocup”, mas não o atributo T3 nem as operações em questão. Entretanto, *Sala* é uma subclasse de *Lugar*, mostrada na Figura 7.9, que possui o atributo T3, lá chamada de “TimeToTurnOff” e as operações “SetDefaultLightScene” e

“TurnLightsOff” que satisfazem as duas primeiras operacionalizações dinâmicas citadas acima.

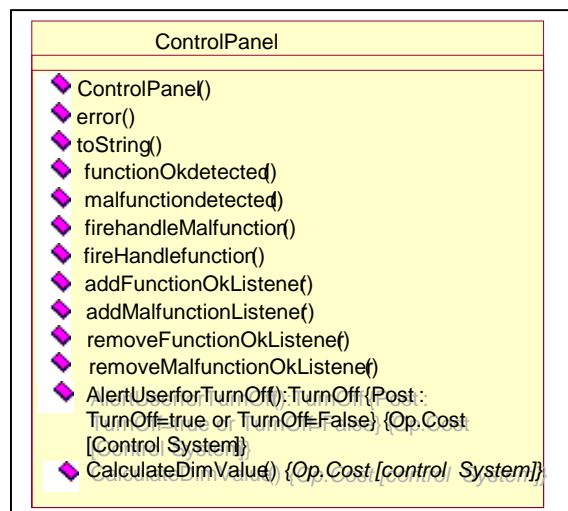
Teremos então de acrescentar uma operação, “AdviseUserofTurnOff”, que na verdade irá chamar uma operação na classe **Painel de Controle** para determinar se o usuário irá ou não cancelar o comando de desligar a luz. Além de acrescentar esta operação devemos condicionar a execução da operação “TurnOffallLights” ao fato de o usuário não ter cancelado este procedimento. A Figura 7.10 mostra as alterações na classe **Lugar**, incluindo a inclusão do atributo “NR\_TurnOff”, enquanto a Figura 7.11 mostra as alterações necessárias à classe **Painel de Controle**. As alterações nesta última classe, se não forem feitas neste momento, certamente irão acontecer ao examinarmos esta classe e conseqüentemente nos depararmos com este mesmo grafo e suas implicações. Entretanto, uma vez que já identificamos a necessidade de se alterar esta classe devemos proceder a alteração de imediato.



**Figura 7.9 – Classe Lugar, Superclasse de Sala**



**Figura 7.10 – Alterações na Classe Lugar para Satisfazer um RNF**



**Figura 7.11 – Alterações na Classe Paine de Controle para Satisfazer o mesmo RNF**

Vale a pena ressaltar que, no caso mostrado acima, os RNFs acabam por se decompor em algumas operacionalizações dinâmicas que na verdade são também

requisitos funcionais. Isto decorre do fato de que os RNFs, em sua grande maioria, estão associados a requisitos funcionais e por isso mesmo constantemente acabamos por ter operacionalizações que exprimem necessidades funcionais aliadas a outras não funcionais.

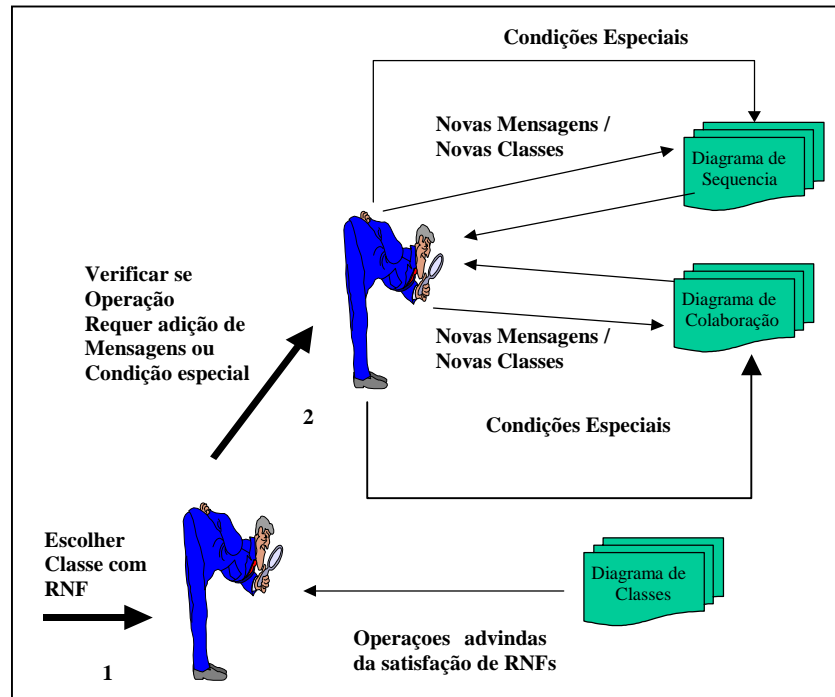
Este ponto mostra-se bastante importante, pois, o uso desta estratégia acaba por contribuir também para a validação dos requisitos funcionais. No caso mostrado acima, por exemplo, se os requisitos funcionais que demandam que as luzes sejam desligadas depois de T3 minutos não estivessem ainda presentes na classe, através de atributos e operações compatíveis, quando aplicássemos as heurísticas de integração, acabaríamos por detectar a necessidade destes atributos e operações e por consequência preencheríamos uma lacuna deixada na visão funcional.

Uma vez terminadas as análises de impactos advindos da integração do RNF Custo Operacional à classe **Sala**, etapas 3 a 6 da figura 7.6, passaríamos então para uma próxima classe, por exemplo, **Painel de Controle**, e repetiríamos todo o processo novamente.

### **7.3 - Integrando RNFs aos Diagramas de Seqüência e Colaboração**

A integração dos RNFs aos diagramas de seqüência e colaboração visa a acrescentar eventuais mensagens entre classes, cuja necessidade não tenha sido identificada, bem como condições especiais para a execução destas mensagens.

A Figura 7.12 ilustra o processo de integração dos RNFs aos diagramas de seqüência e colaboração.

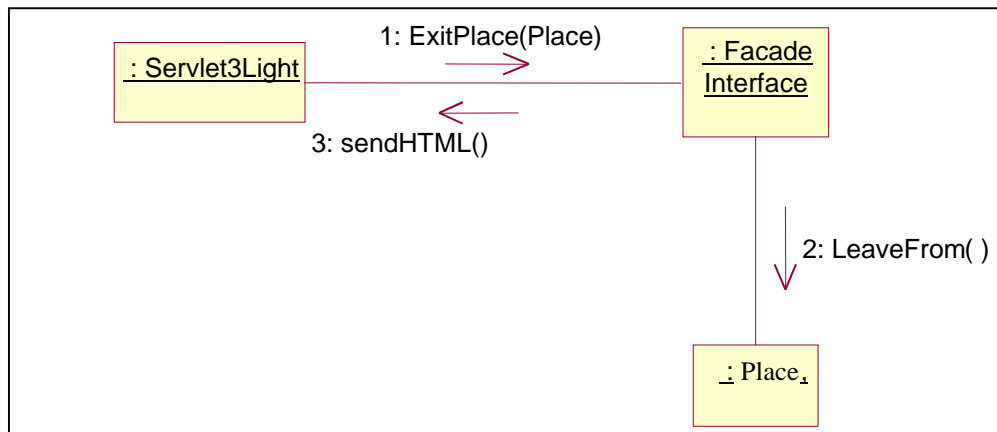


**Figura 7.12 – Processo de Integração de RNFs aos Diagramas de Seqüência e Colaboração**

Como podemos observar a partir da Figura 7.12, o processo de integração dos RNFs parte da idéia de que os diagramas de seqüência e colaboração serão espelhos do que ocorre nas classes. Começamos então por percorrer cada classe, e suas superclasses, verificando em cada uma quais as operações que foram adicionadas por conta da satisfação de RNFs. A identificação do RNF de origem,  $\{RNF [Tópico]\}$ , irá ajudar nesta etapa. Para cada operação derivada da satisfação de um RNF verificamos nos diagramas de seqüência e colaboração, que lidam com a classe sendo analisada, se esta nova operação implica alguma alteração nestes diagramas. Se implicar, realizamos a inclusão de novas mensagens, classes e condições especiais conforme for necessário. Para tal, devemos seguir o mostrado na sessão 6.3. As novas operações podem não afetar diagramas existentes, mas sim requerer que novos diagramas sejam feitos, o que deverá ser feito da maneira usual.

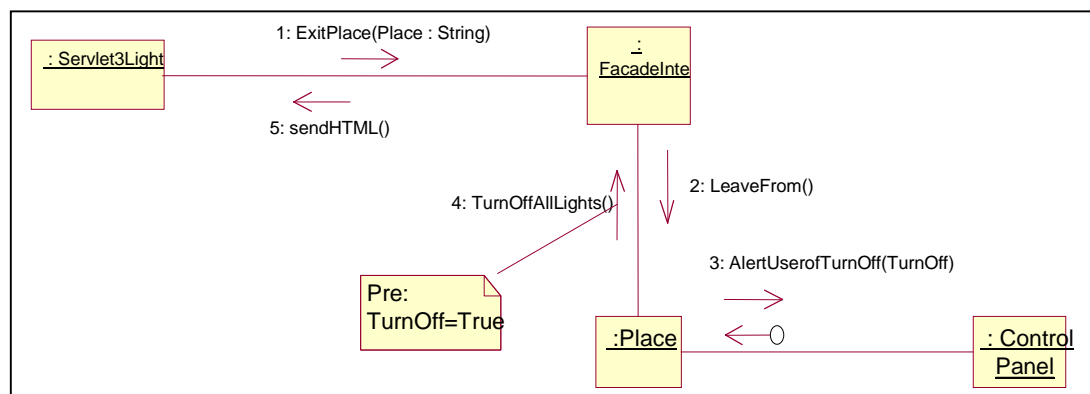


A necessidade de condições especial, via de regra, estará expressa sob forma de comentários no diagrama de classes e preferencialmente, conforme dito na seção 6.2, terão o formato de expressões OCL [Rational 97].

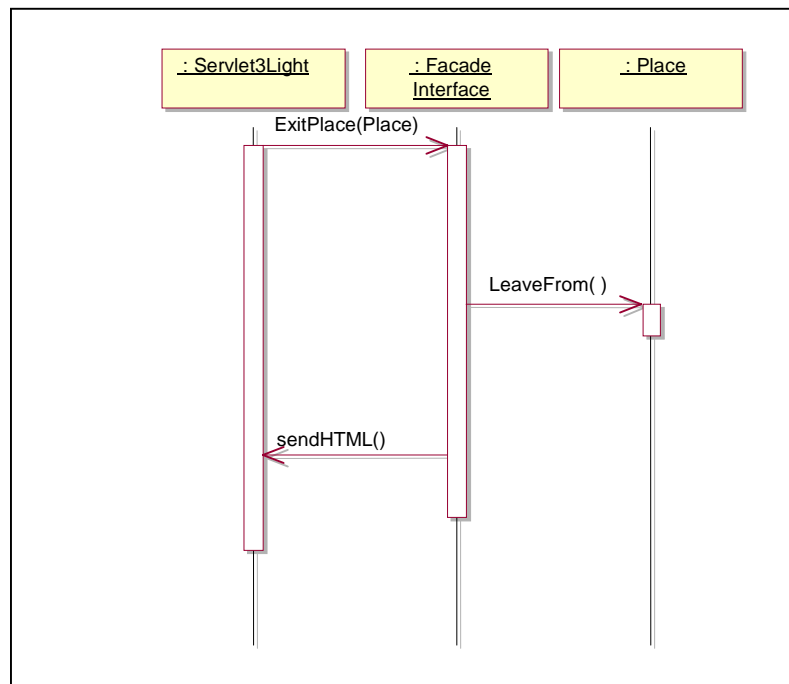


**Figura 7.13 – Diagrama de Colaboração antes da Integração dos RNFs**

Como a decisão de produzir ou não diagramas de sequência e colaboração é bastante pessoal e varia para cada desenvolvedor, não nos preocupamos aqui em tentar estabelecer condições específicas nas quais isto deva ocorrer ou não, deixando a cargo da equipe encarregada do desenho do software determinar sua necessidade.



**Figura 7.14 – Diagrama de Colaboração Exibindo as Alterações Advindas de RNFs**

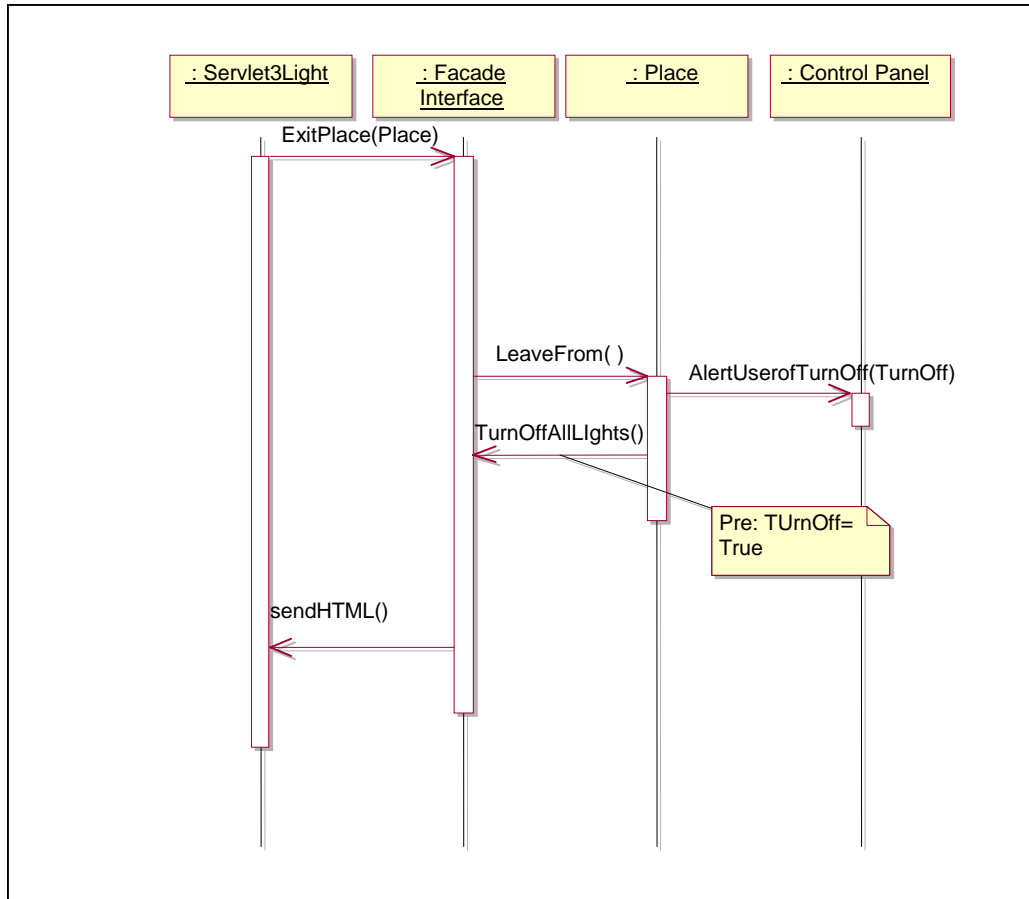


**Figura 7.15 – Diagrama de Seqüência**

Utilizando a classe `Lugar` mostrada na sessão anterior, Figura 7.10, temos que procurar nos diagramas pela ocorrência desta classe. Verificamos então que na classe `Lugar` temos uma operação “`AdviseUserofTurnOff`” que tem por objetivo avisar a um usuário de um cômodo, via painel de controle, que as luzes serão apagadas. Verificamos também que para que a operação “`TurnOffAllLights`” seja executada é necessário que o usuário não tenha cancelado, via painel de controle, esta ação (Ver comentário associado à classe da Figura 7.10). As Figuras 7.13, 7.14, 7.15 e 7.16 mostram respectivamente os diagramas de seqüência e colaboração antes da integração e os diagramas de seqüência colaboração já espelhando as necessidades advindas das novas operações.

A integração dos RNFs aos diagramas de seqüência e colaboração mostrou-se particularmente importante quando necessitamos demonstrar condições de controle, como por exemplo, a exibida na Figura 7.16. Nesta figura vemos que a operação de apagar todas as luzes só será realizada se a variável `desligar` for verdadeira. Esse tipo de controle, freqüentemente omitido nestes diagramas, quando não representado pode

resultar que o software apresente comportamento errôneo em determinadas situações, apesar de estar funcionalmente correto.



**Figura 7.16 – Diagrama de Sequência Exibindo as Alterações Advindas de RNFs**

## 8 – Validação da Estratégia

Este capítulo apresenta os três estudos de caso que realizamos com o intuito de validar a estratégia apresentada nesta tese. Para tal, elaboramos um projeto de experimentação que possibilitasse, na medida do possível, avaliarmos os possíveis ganhos advindos da utilização da estratégia proposta. Nos apoiamos em diversos trabalhos sobre experimentação em software [Pfleeger 95] [Basili 92] [Basili 84][Basili 88] [Basili 86] para elaborar um projeto para este experimento.

Como nosso objetivo é medir o ganho de um processo de desenvolvimento de software contra outros, sob a ótica de desenvolvedores e clientes, projetamos um experimento baseado no uso de replicação de projetos [Basili 86]. Visto que nossa estratégia pode ser vista como uma adição a qualquer processo de desenvolvimento de software, utilizamos três diferentes projetos onde cada um deles teve uma equipe gerando o modelo conceitual que sob suas óticas espelhava as necessidades do cliente e uma outra equipe, representada pelo autor dessa tese, avaliando a correção do modelo conceitual obtido.

Utilizamos aqui o conceito de que quanto mais completo um modelo conceitual for, ou seja, quanto mais ele estiver conforme com as expectativas que o cliente nutre em relação a ele, melhor qualidade ele terá. Embora reconheçamos que outros critérios de qualidade podem ser aplicados a um modelo conceitual, adotamos o da completeza como sendo o utilizado por nós para confrontar modelos conceituais gerados por processos diferentes. De fato, a qualidade de um projeto quanto a observância de critérios diversos como acoplamento e coesão não são invalidados pelo que está aqui proposto, ou seja, nada impede que revisões sejam feitas no modelo conceitual obtido aqui para adequá-lo a esse critérios.

A avaliação de correção aqui conduzida foi feita gerando-se o modelo não funcional de cada software avaliado e posteriormente integrando os RNFs encontrados aos modelos conceituais existentes, encontrando assim, diversas classes, atributos e operações que deveriam ter sido especificadas pelas outras equipes e não o foram. Essas classes, atributos e operações encontradas foram anotadas como falhas existentes nos modelos conceituais gerados pelas outras equipes, uma vez que se estes projetos tivessem sido entregues ao cliente sem essas alterações, estas teriam sido apontadas como falhas do software.

Como os estudos de caso I e II tiveram a mesma fonte de conhecimento para sua confecção, a especificação de um software para controle de iluminação de um andar de um prédio proposto em um seminário em Dagstuhl [Dagstuhl 99], para estes dois casos utilizamos o mesmo modelo não funcional, realizando a integração deste modelo aos dois diferentes modelos conceituais encontrados em cada um dos estudos de caso, conseqüentemente, um estudo de caso não teve qualquer influência no resultado do outro.

O uso desta abordagem possibilitou observarmos apenas o que foi descoberto devido à utilização da estratégia, uma vez que todas as classes, atributos e operações que foram encontradas pelo autor desta tese foram derivados apenas da aplicação da visão não funcional ao domínio em questão, ou seja, minimizamos eventuais distorções de medições que poderiam advir de um conhecimento anterior do domínio caso tivéssemos desenvolvido todo o software novamente.

Como a inclusão de novas classes, na maioria das vezes, será conseqüência de decisão de desenho derivada da inclusão de novos atributos e/ou operações em uma classe que justifiquem a criação de uma nova classe, acreditamos que *o número de operações e atributos acrescentados ao modelo conceitual anterior* será o melhor

indicador do grau de trabalho que teria sido evitado se a estratégia tivesse sido utilizada a priori.

Uma avaliação qualitativa das alterações necessárias torna-se bastante subjetiva e foi preterida nesse estudo em benefício de uma avaliação quantitativa indo de encontro ao critério de completeza de um software. Embora a avaliação quantitativa não reflita necessariamente toda a dificuldade que encontraríamos para redesenhar o software para atender aos RNFs, acreditamos que ela é suficiente em si só para mostrar a validade do uso da estratégia aqui proposta.

É verdade que a inclusão de novas classes, operações e atributos podem contribuir para uma deterioração da qualidade do desenho do software, tanto pela introdução de novos erros, como por prejudicar critérios como acoplamento e coesão que medem bons desenhos de software.

Entretanto, se os RNFs que deram origem a essas mudanças tivessem sido considerados desde o princípio do processo de desenvolvimento os problemas advindos da inclusão dessas classes, atributos e operações possivelmente estariam também presente.

Além disso, conforme mencionado anteriormente, nada impede que revisões sejam feitas no modelo conceitual obtido aqui para adequá-lo a outros critérios de qualidade de um projeto como coesão e acoplamento, processo aliás, comumente realizado durante o processo de desenvolvimento de software.

Por outro lado, temos a certeza de que o fato de o software não estar atendendo a necessidades reais do cliente constituem de fato uma falha do software. Conseqüentemente, acreditamos que as classes, atributos e operações que identificamos como faltantes nos modelos conceituais considerados constituem uma

boa medida de falhas ocorridas na geração desses modelos e que teriam sido evitadas se a estratégia aqui proposta tivesse sido utilizada.

A seguir detalhamos cada um dos estudos de caso realizados.

### **1) Sistema de Iluminação I:**

#### ***Escopo da Avaliação:***

- Replicação de projeto

#### ***Comparação efetuada:***

Avaliar as variações quanto ao conteúdo dos Cenários e Modelo de Classes obtido como um dos estudos de caso da tese de doutorado por Breitman [Breitman 00] com o obtido com a aplicação da estratégia proposta no capítulo 7 pelo autor desta tese.

**Objetivo da comparação:** Através da verificação quantitativa de classes/métodos/atributos que tenham sido identificados pelo autor da tese que não constavam do modelo gerado pelo outro time, avaliar melhorias introduzidas pela estratégia proposta.

#### ***Projeto do Experimento***

Aplica-se a um software para controle de iluminação de um andar de um prédio proposto em um seminário em Dagstuhl [Dagstuhl 99] utilizado na tese de Breitman [Breitman 00]. De posse dos modelos utilizados na tese de Breitman, reavaliamos estes modelos usando o enfoque não funcional proposto no capítulo 7. Ao final quantificamos o número de atributos, operações e classes que foram achadas com a aplicação da estratégia.

#### ***Ambiente***

- PUC-Rio
- Sistema de Médio Porte

- In Vitro<sup>3</sup>, sênior, Correlacional

## 2) Sistema de Iluminação II:

### *Escopo da avaliação:*

- Replicação de projeto

### *Comparação efetuada:*

Avaliação das variações quanto ao conteúdo do Modelo de Classes obtido por um grupo de alunos de graduação da cadeira de Projeto de Sistemas de Software como avaliação de final de curso com o obtido com a aplicação da estratégia proposta pelo autor desta tese.

**Objetivo da comparação:** Através da verificação quantitativa de classes/métodos/atributos que tenham sido identificados pelo autor da tese que não constavam do modelo gerado pelo outro time, avaliar melhorias introduzidas pela estratégia proposta.

### *Projeto do Experimento*

Aplica-se a um software para controle de iluminação de um andar de um prédio proposto em um seminário em Dagstuhl [Dagstuhl 99]. por um grupo de alunos de graduação da cadeira de Projeto de Sistemas de Software como avaliação de final de curso. De posse dos modelos gerados ao fim deste trabalho, reavaliemos estes modelos usando o enfoque não funcional proposto no capítulo 7. Ao final quantificamos o número de atributos, operações e classes que foram achadas com a aplicação da estratégia.

### *Ambiente*

- PUC-Rio
- Sistema de Médio Porte

---

<sup>3</sup> Validação efetuada em ambiente controlado, geralmente laboratórios acadêmicos envolvendo apenas o meio acadêmico. [Basili 92]



- In Vitro, sênior, Correlacional

### 3) Sistema de Informações Laboratoriais

#### *Escopo da avaliação:*

- Replicação

#### *Comparação Efetuada*

Variações quanto a diferenças entre Modelo de Classes apresentadas por um time de desenvolvimento e o autor desta tese.

**Objetivo da comparação:** Através da verificação quantitativa de classes/métodos/atributos que tenham sido identificados pelo autor da tese que não constavam do modelo gerado pelo outro time, avaliar melhorias introduzidas pela estratégia proposta.

#### *Projeto do Experimento*

Aplica-se ao desenvolvimento de um sistema de informações laboratoriais. O time pertencente ao laboratório gerou um modelo de classes que refletia as necessidades dos clientes levantadas por eles. Posteriormente o autor desta tese revisou o modelo de classes utilizando-se da estratégia detalhada no capítulo 7 para reavaliar o modelo de classes existente. Este modelo tinha como escopo o software a ser desenvolvido para a área de processamento do laboratório. Ao final quantificamos o número de atributos, operações e classes que foram achadas com a aplicação da estratégia.

#### *Ambiente:*

- Empresa de Desenvolvimento de Software
- Sistema de Grande Porte
- In Vivo<sup>4</sup>

---

<sup>4</sup> Validação efetuada em ambiente real de desenvolvimento com a supervisão/participação de pesquisadores. [Basili 92]

- Time da Empresa: Responsável pelo software da área de processamento - 2 Analistas Pleno e 1 Analista Júnior - utilizando estratégia de desenvolvimento baseada em ferramentas UML (Baixo conhecimento em UML). O analista júnior participou de projeto semelhante há 4 anos atrás tendo identificado à época alguns RNFs.
- Autor da Tese: Sênior - utilizando estratégia proposta na tese para lidar com RNFs.

## 8.1 Sistema de Iluminação I

Este estudo de caso utilizou-se da abordagem proposta por Basili [Basili 84] que utiliza a idéia de mais de um time replicando um mesmo projeto. Aqui, um dos times foi caracterizado pelo conjunto de especificações geradas por Breitman [Breitman 00] como um dos estudos de casos utilizados em sua tese de doutorado. Este estudo de caso referia-se a um software para controle de iluminação de um andar de um prédio proposto em um seminário em Dagstuhl [Dagstuhl 99]. O segundo time foi personificado pelo autor desta tese.

O objetivo deste estudo de caso era avaliar se a estratégia proposta nesta tese efetivamente trazia ganhos de qualidade ao processo de desenvolvimento de software. Para tal, aplicamos a parte da estratégia detalhada no capítulo 5, redefinindo o LAL especificado por Breitman sob a ótica não funcional e gerando os grafos de RNF. Posteriormente, integramos os RNFs achados aos modelos de cenários e de classes que haviam sido encontrados por Breitman durante sua tese, usando para isso a parte da estratégia descrita no capítulo 7. Cabe ressaltar que a especificação do problema utilizada por Breitman [Dagstuhl 99] continha explicitamente alguns RNFs que eram esperados estarem presentes no software.

Conforme explicado no início desse capítulo, utilizamos como fator de comparação o número de novas classes, operações e atributos encontrados pelo autor da tese em contraposição ao número de classes, operações e atributos existentes antes da utilização da estratégia.

A Tabela 8.1 ilustra os dados obtidos ao final do estudo de caso.

	Classes	Operações	Atributos
Modelo conceitual original	8	105	22
Derivadas da aplicação da estratégia	2	50	10
Variação Percentual	25%	47,6%	45,4%

**Tabela 8.1 – Resultados Obtidos no Estudo de Caso I**

Das 8 classes existentes 6, ou seja, 75%, sofreram alterações em decorrência da aplicação da estratégia.

### 8.1.1 - Desenvolvendo a Visão Não Funcional

Utilizamos o LAL definido por Breitman como ponto de partida para o desenvolvimento da visão funcional conforme descrito no capítulo 5. Num primeiro momento, percorremos cada símbolo do LAL aplicando a este símbolo o checklist contido na base de conhecimento representada no LAL-RNF da ferramenta OORNF [Neto 00]. No caso do símbolo **Sala**, por exemplo, verificamos que o RNF *Seguro* se aplicava a este símbolo e que para satisfazer este RNF seriam necessárias as seguintes alterações no LAL:

- A escolha de um esquema de luz não pode resultar numa quantidade de luz inferior a 14 lux – Símbolo **Esquema de Luz**;
- Se ocorrer algum mau funcionamento do sensor de luz externa o esquema de luz default deve ser colocado como todas as luzes acesas – Símbolo **Mau funcionamento do Sensor de Luz**;

- Se algum sensor de uma sala não funcionar corretamente o usuário da sala deve ser avisado – Símbolo **Usuário**.
- Em todas as salas o painel de controle deve ser instalado próximo a porta que leva ao corredor – Símbolo **Painel de Controle de Sala**

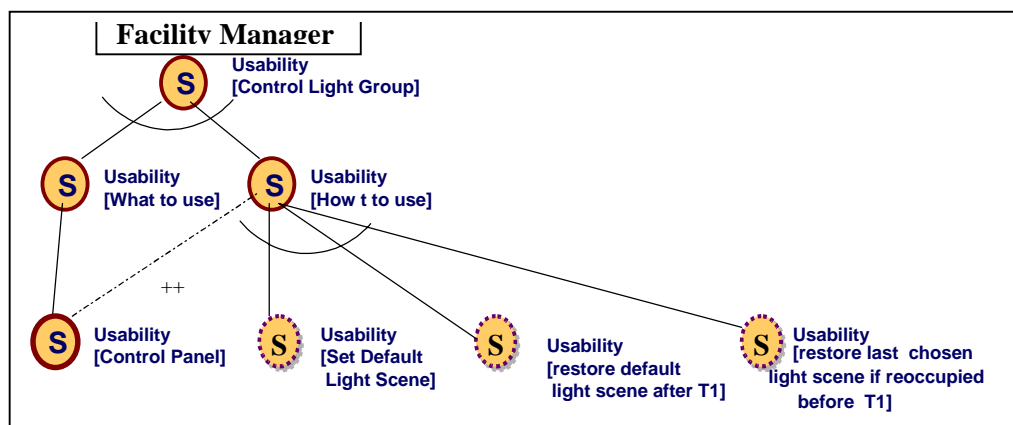
Já para o símbolo **Sistema de controle** identificamos três RNFs: *Seguro*, *Custos Operacionais* e *Confiabilidade*. A satisfação do RNF *Seguro*, por exemplo, levou às seguintes alterações no LAL:

- Se algum sensor de luz externo não funcionar corretamente e o corredor estiver ocupado, o grupo de luzes do teto deve permanecer ligado – símbolo **Grupo de luzes do corredor**;
- Se um mau funcionamento ocorre o administrador deve ser notificado – símbolo **Administrador (Facility Manager)**;
- Se algum sensor não funcionar corretamente em uma sala o esquema de luzes default deve especificar todos os grupos de luzes acesos – símbolo **Sistema de Controle**
- Se um grupo de luzes de um corredor não puder ser controlado nem manualmente nem pelo sistema de controle ele deve ser mantido ligado – símbolo **Controlar grupo de luzes**
- Se as luzes atenuadas não receberem um sinal a cada 60 segundos, elas devem passar a funcionar sem atenuação – símbolo **Luzes atenuáveis**.

Algumas vezes, incluíamos a noção de que um determinado símbolo precisava de um determinado RNF sem termos certeza de que isso era realmente verdade ou de como isso se daria, ou seja, quais os impactos da inclusão desta noção. Isto ficava

para ser resolvido na etapa seguinte quando montávamos os grafos de RNF e então voltávamos a perguntar se esse RNF era mesmo necessário e se era quais as decomposições que poderíamos fazer para chegar às operacionalizações.

Um exemplo disto é o RNF *Usabilidade* em relação ao símbolo Controlar grupo de luzes. Num primeiro momento era claro que o símbolo Controlar grupo de luzes necessitava do RNF *Usabilidade*, sem contudo, ser claro para nós como isso seria operacionalizado. Quando realizamos a decomposição do grafo de RNF mostrada na Figura 8.1, encontramos as seguintes operacionalizações que foram então adicionadas ao LAL:



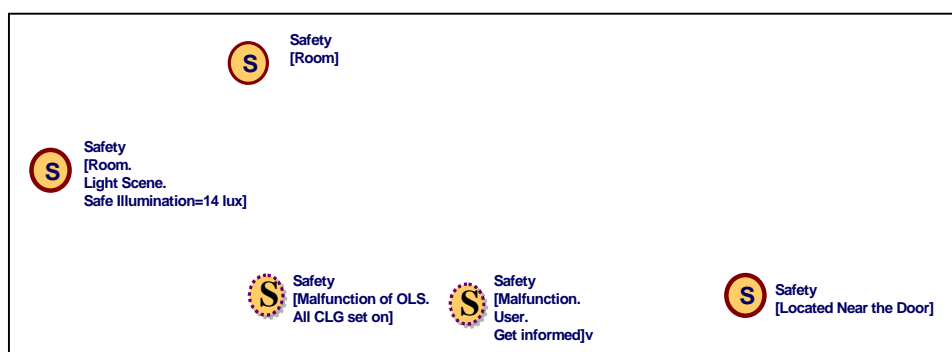
**Figura 8.1 – Grafo do RNF Usabilidade Aplicada a Controle de Grupo de Luz**

- O painel de controle deve possibilitar que um esquema de luz default seja estabelecido – Símbolo Painel de controle;
- Se uma sala é reocupada depois de T1 minutos, o esquema de luz default deve ser estabelecido – Símbolo Esquema de luz default;
- Se uma sala é reocupada antes de T1 minutos, o último esquema de luz escolhido deve ser restabelecido – Símbolo Esquema de luz escolhido.

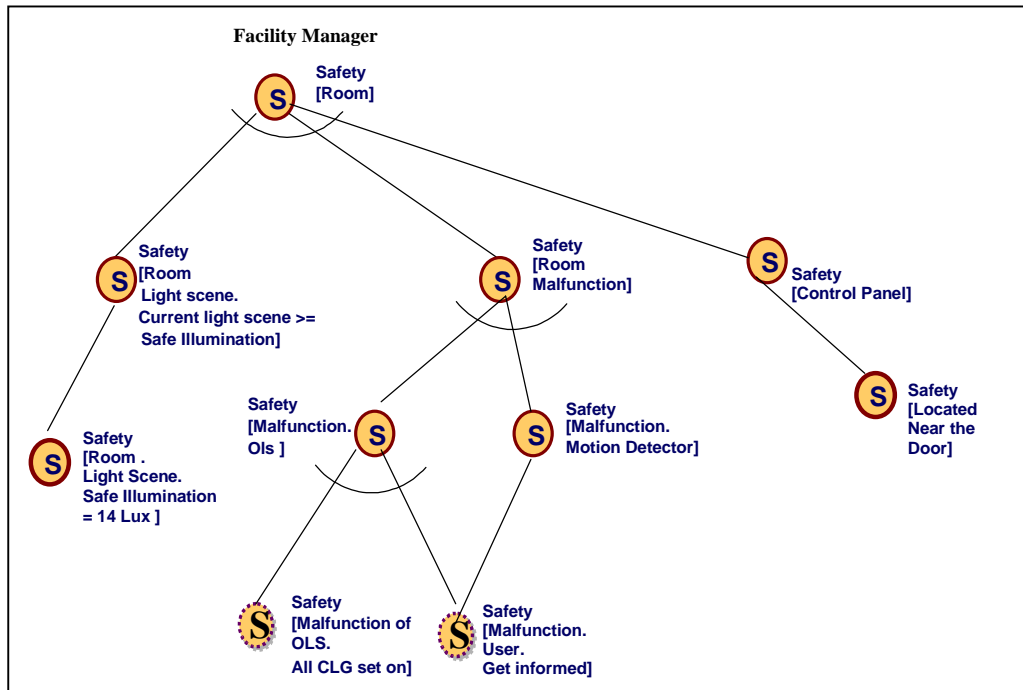
Uma vez tendo percorrido todos os símbolos do LAL partimos para a definição dos grafos de RNF, conforme detalhado no capítulo 5. Voltamos então a percorrer cada

símbolo que possuía RNF. Cada RNF de cada símbolo dava origem a um nó inicial de um grafo de RNF. Dessa forma quando analisamos o símbolo **Controlar grupo de luzes**, nos deparamos com o RNF *Usabilidade*, gerando a raiz do grafo mostrado na Figura 8.1..

Tendo gerado a raiz do grafo verificamos se já existiam outras noções ou impactos neste símbolo ou em outros que fossem consequência da satisfação deste RNF. Se existiam, essas consequências se tornavam automaticamente as operacionalizações deste grafo, e partíamos então para uma análise que pode ser encara como sendo parte bottom-up e parte top-down deste grafo, ou seja, tanto realizávamos grupamentos de operacionalizações em submetas do RNF quanto realizávamos decomposições a partir da raiz para chegarmos a essas folhas. A Figura 8.2 mostra um exemplo disto, onde aparecem a raiz e as operacionalizações do RNF *Seguro* para o símbolo **Sala**. Estas operacionalizações vieram dos impactos resultantes da inclusão deste RNF no LAL conforme citado anteriormente. A Figura 8.3 mostra o grafo depois de realizarmos o procedimento anteriormente citado para a construção do grafo.



**Figura 8.2 – Início da Construção do Grafo para o RNF Seguro, Aplicado à Sala**



**Figura 8.3 – Grafo Resultante**

Uma vez tendo obtido o grafo voltávamos então a realizar a abordagem de decomposição top-down, em busca de eventuais decomposições que não haviam sido identificadas inicialmente que resultariam em novas operacionalizações. No caso do exemplo anterior, verificamos que a decomposição de maus funcionamentos ocorridos na sala estava incompleta por não apontar a necessidade de que o administrador seja avisado. A inclusão desta decomposição levou ao grafo final mostrado na Figura 8.4 onde incluímos também a origem do conhecimento referente a estes RNFs. No caso específico, como estávamos tratando com uma só origem, que era a especificação existente em [Dagstuhl 99], resolvemos para efeito de ilustração da estratégia atribuir a origem do conhecimento ao único ator que estaria ligado a especificação do software, no caso o Administrador. Como o ator se repete para todos os grafos decidimos omiti-lo dos grafos para simplificar o desenho dos grafos.





seja usado pelo sistema de controle como temporizador de envio do sinal. Esta solução pode ser vista na Figura 8.6.

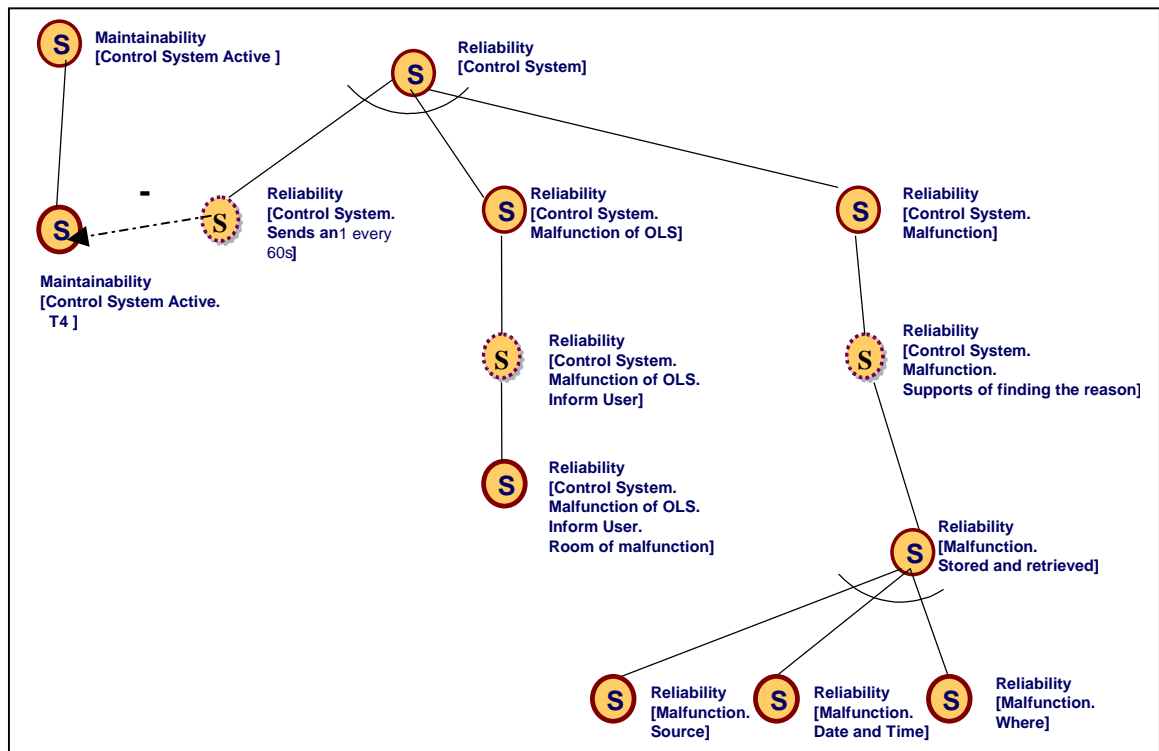


Figura 8.5 – Grafo com Interdependência Negativa Identificada

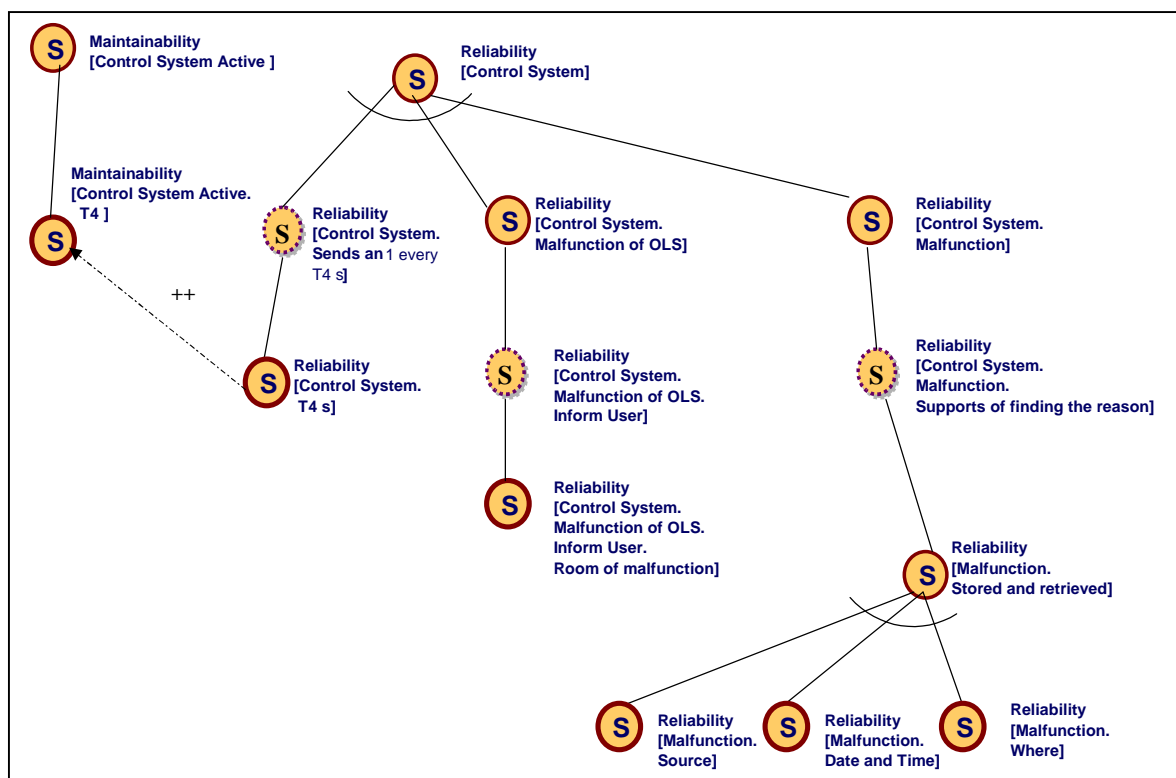


Figura 8.6 – Grafos Após Resolução de Conflitos

Outro exemplo de interdependências identificadas pode ser visto nas Figuras 8.7 e 8.8. A Figura 8.7 mostra o conflito surgido quando representamos lado a lado os grafos referentes ao RNF *Usabilidade* aplicada ao símbolo Painel de controle do quarto e o RNF *Seguro* referente ao símbolo Iluminação. Como o RNF *Seguro* estabelece que a iluminação em um ambiente não pode ser inferior a 14 lux, a simples representação de percentual de atenuação de grupos de luzes não é suficiente para garantir que este RNF seja satisfeito. Precisamos para tanto, que a quantidade de iluminação em lux seja também armazenada para cada grupo de luzes. Esta solução de desenho é representada na Figura 8.8

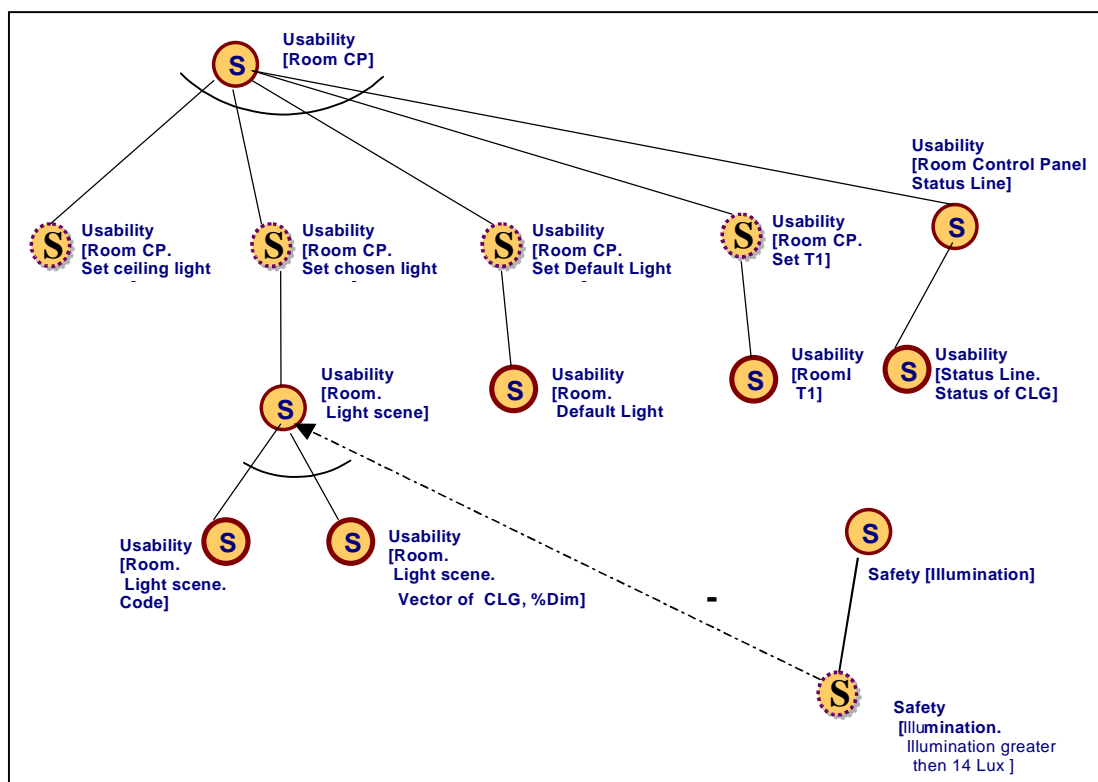


Figura 8.7 – Conflitos Identificados entre dois RNFs

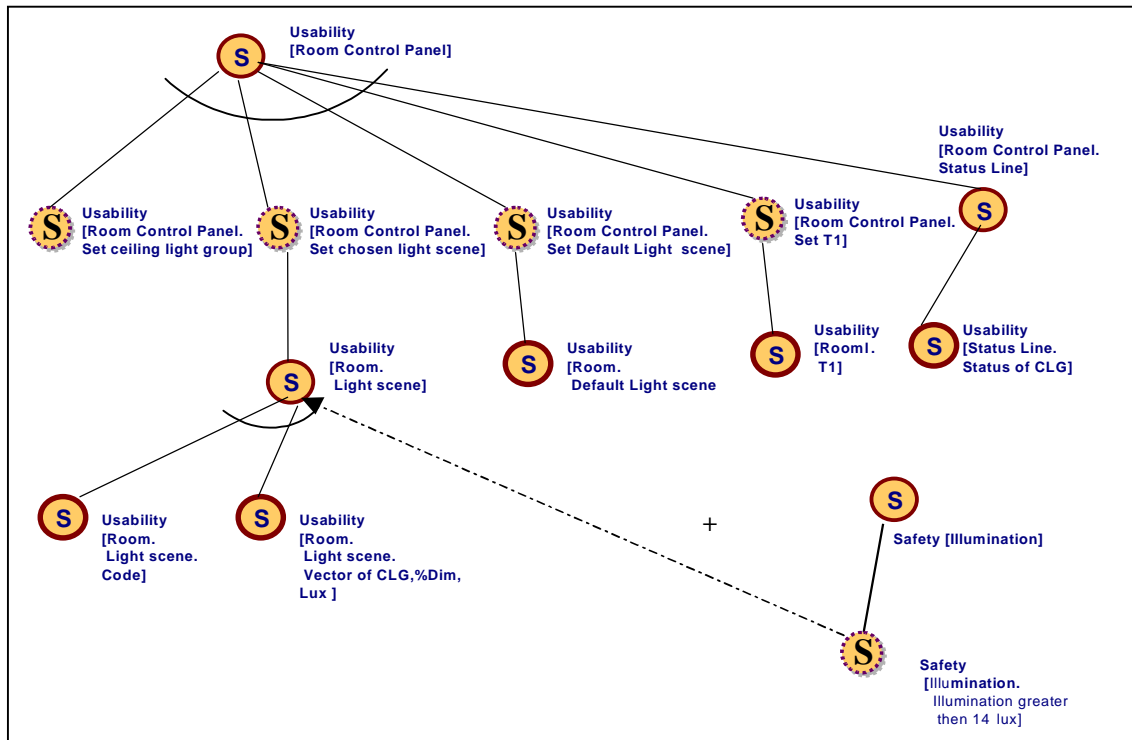


Figura 8.8 – Grafo Representando Nova Solução de Desenho

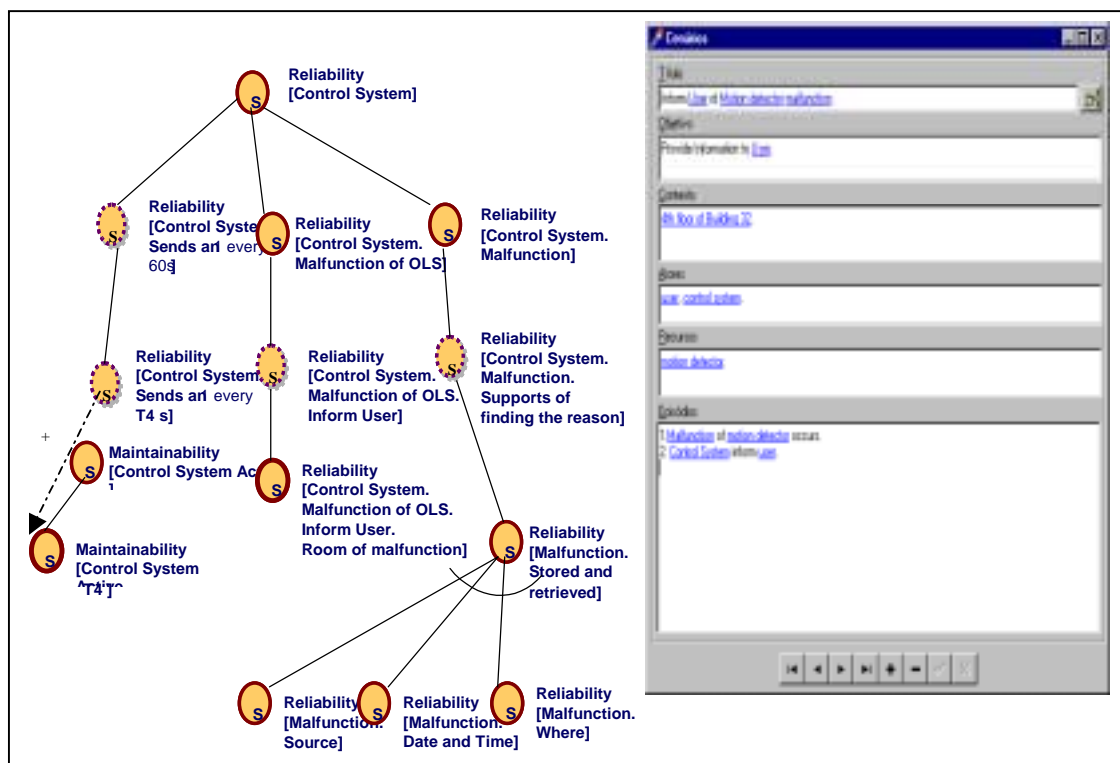
## 8.1.2 - Integrando as Visões Funcional e Não Funcional

### 8.1.2.1 Integrando aos Cenários

Após termos terminado a construção da visão não funcional, realizamos a integração desta com a visão funcional gerada por Breitman [Breitman 00], representada neste caso através de cenários e diagrama de classes. O Apêndice A mostra o conjunto de grafos de RNF obtidos para este estudo de caso.

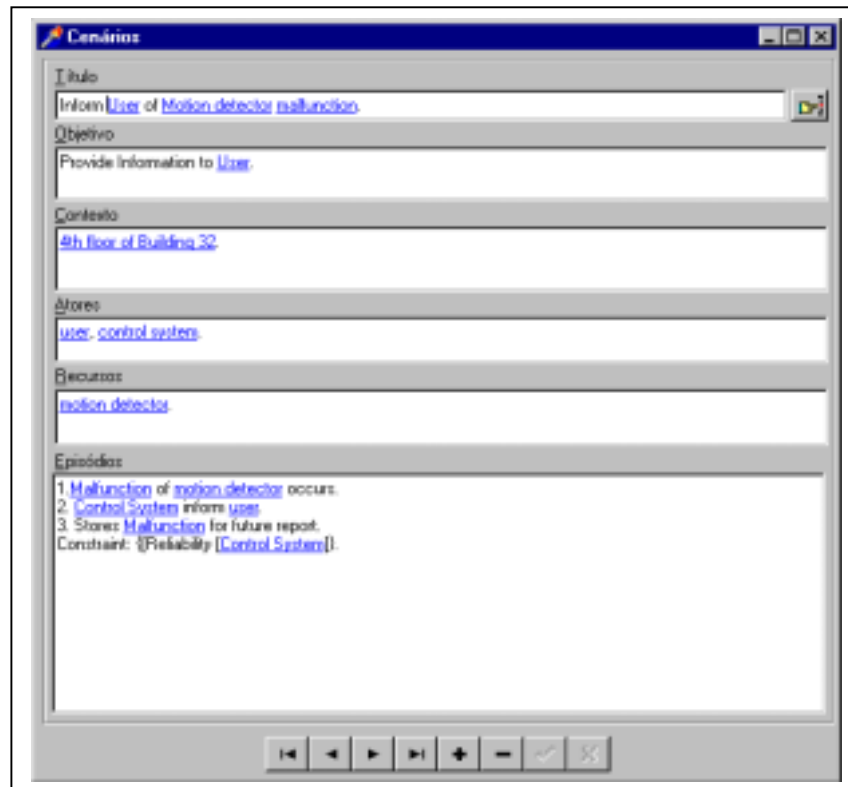
Para integrar os RNFs aos cenários seguimos o procedimento detalhado na seção 7.1. Para tal, percorremos cada cenário definido procurando por símbolos do LAL presentes no título, no contexto, nos recursos e nos atores. Para cada símbolo encontrado percorremos os grafos de RNF procurando pela ocorrência deste símbolo. Uma vez achado, analisamos se os episódios representados neste cenário são suficientes para satisfazer as operacionalizações existentes no RNF achado. A Figura

8.9 ilustra o exemplo do cenário Informa Usuário de mau funcionamento do detector de movimento.



**Figura 8.9 – Analisando um Cenário Contra Grafos de RNF**

Analisando este cenário encontramos o símbolo Mau funcionamento, que está presente no grafo do RNF *Confiabilidade* aplicado ao símbolo Sistema de Controle como uma decomposição. Podemos ver então que para satisfazermos este RNF precisamos armazenar informações sobre o mau funcionamento para posterior recuperação. Vemos também, que não há episódio no cenário que satisfaça essa condição. Representamos então um novo episódio para satisfazer esta operacionalização e representamos em seguida a este episódio a origem da necessidade deste episódio. O resultado pode ser visto na Figura 8.10.



**Figura 8.10 – Cenário Atualizado para Satisfazer a RNF**

Outro exemplo pode ser visto nas Figuras 8.11 e 8.12. A Figura 8.11 apresenta o cenário Usuário Atenua Luzes antes da aplicação da estratégia junto ao grafo de RNFs onde aparece o símbolo Atenua (Dim) presente no título de cenário. Podemos ver no grafo que para mantermos a segurança necessária nas salas, a iluminação mínima deve ser equivalente a 14 lux. Vemos ainda, que devemos armazenar o valor de atenuação das luzes não apenas em forma percentual, forma utilizada pelos atenuadores, mas também o seu equivalente em lux, de forma a podermos avaliar se a iluminação mínima está sendo respeitada. Podemos ver que o cenário atual não contempla nenhum esforço no sentido de converter o percentual de atenuação para um valor em lux e nem em verificar se o valor resultante é compatível com os critérios de segurança. A Figura 8.12 ilustra o cenário resultante da integração dos RNFs ao cenário.

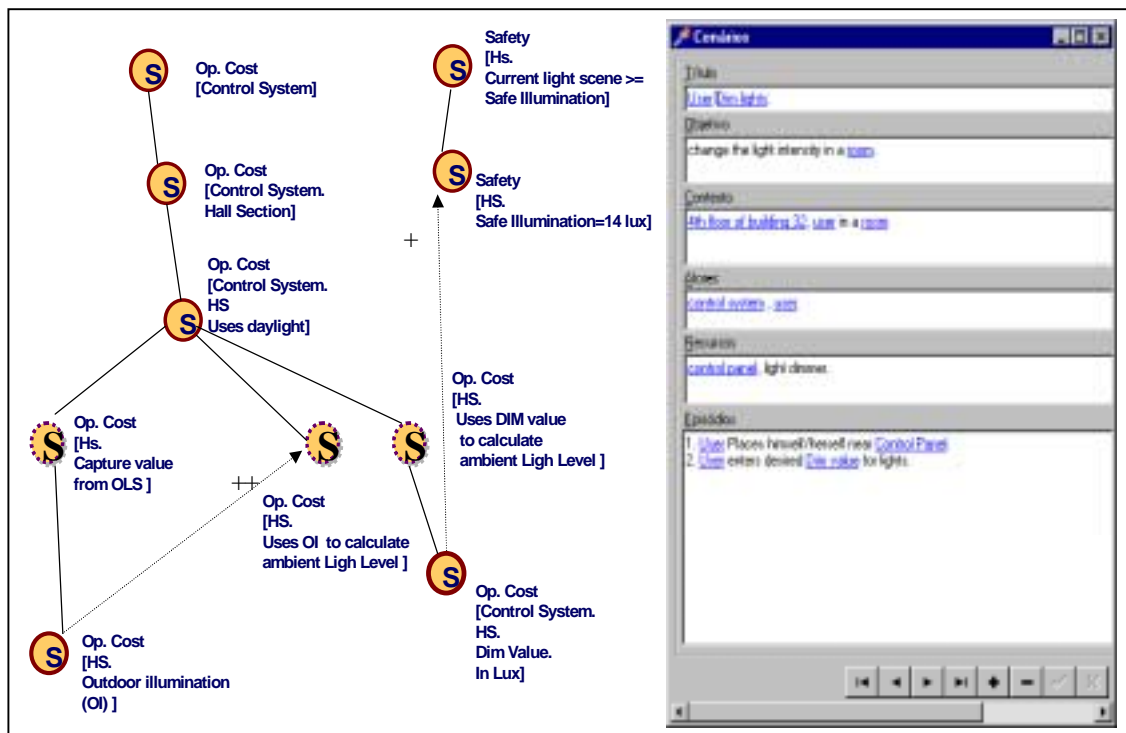


Figura 8.11 – RNFs Sendo Integrado a um cenário

The screenshot shows the 'Cenários' (Scenarios) window with the following details:

- Título:** User Dim lights.
- Objetivo:** change the light intensity in a [room](#).
- Contexto:** 4th floor of building 32, user in a [room](#).
- Atores:** [control system](#), [user](#).
- Recursos:** [control panel](#), light dimmer.
- Episódios:**
  1. [User](#) Places himself/herself near [Control Panel](#).
  2. [User](#) enters desired [Dim value](#) for lights.
  3. [Control System](#) calculates the equivalent in Lux.
  4. IF Chosen value <14 lux THEN [system](#) issues a warning.

Figura 8.12 – Cenário Resultante

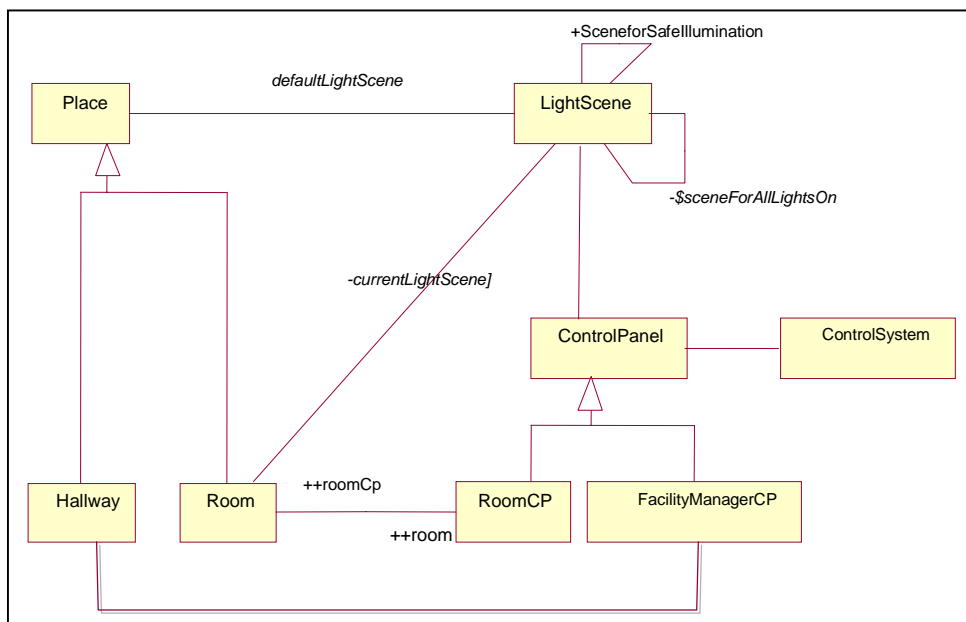
### 8.1.2.2 Integrando ao Diagrama de Classes

O primeiro passo nessa parte do estudo de caso foi verificar se todas as classes definidas eram símbolos do LAL, como isso era verdade, não foi necessária nenhuma adaptação.

Além de integrarmos os RNFs aos cenários, realizamos também a integração destes ao diagrama de classes que foi gerado por Breitman [Breitman 00]. Esta integração seguiu os passos detalhados na seção 7.2. Para cada classe existente no diagrama de classes, procurávamos pela ocorrência do símbolo que nomeia a classe em cada um dos grafos de RNFs constantes da visão não funcional. Para cada ocorrência encontrada verificávamos se a classe possuía os atributos requeridos no grafo, operacionalizações estáticas, e se ela implementava todas as operações necessárias em consonância com o grafo, operacionalizações dinâmicas. Caso isso não fosse verdade, incluíamos novos atributos ou operações conforme o caso. A cada inclusão de um atributo novo, verificávamos quais seriam as operações necessárias a serem introduzidas para lidar com esses atributos. A cada inclusão de uma nova operação verificávamos se não era necessária a inclusão de novos atributos requeridos pela operação. Verificávamos ainda se a inclusão destas novas operações e atributos não causava nenhum conflito com o desenho existente.

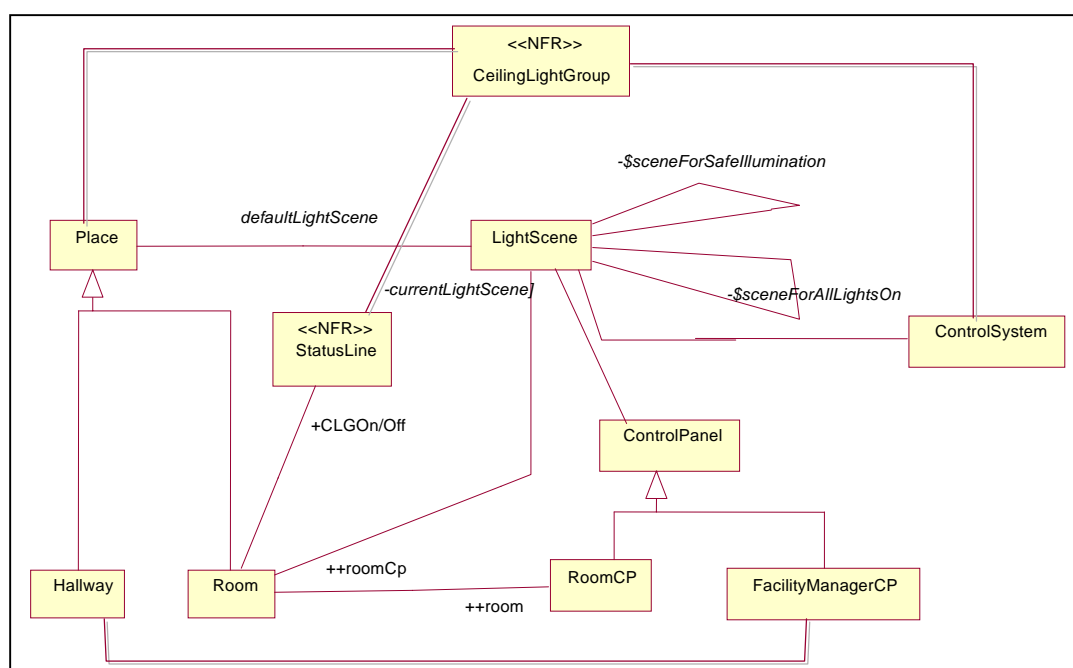
É importante ressaltar que as alterações realizadas no diagrama de classes procuraram ao máximo respeitar o desenho original, sem entrar no mérito de quanto adequado ou não era o desenho existente. Agindo dessa forma procuramos minimizar a contaminação do estudo com possíveis decisões de desenho dissonantes do estudo original.

A seguir iremos detalhar algumas das alterações realizadas no estudo de caso.



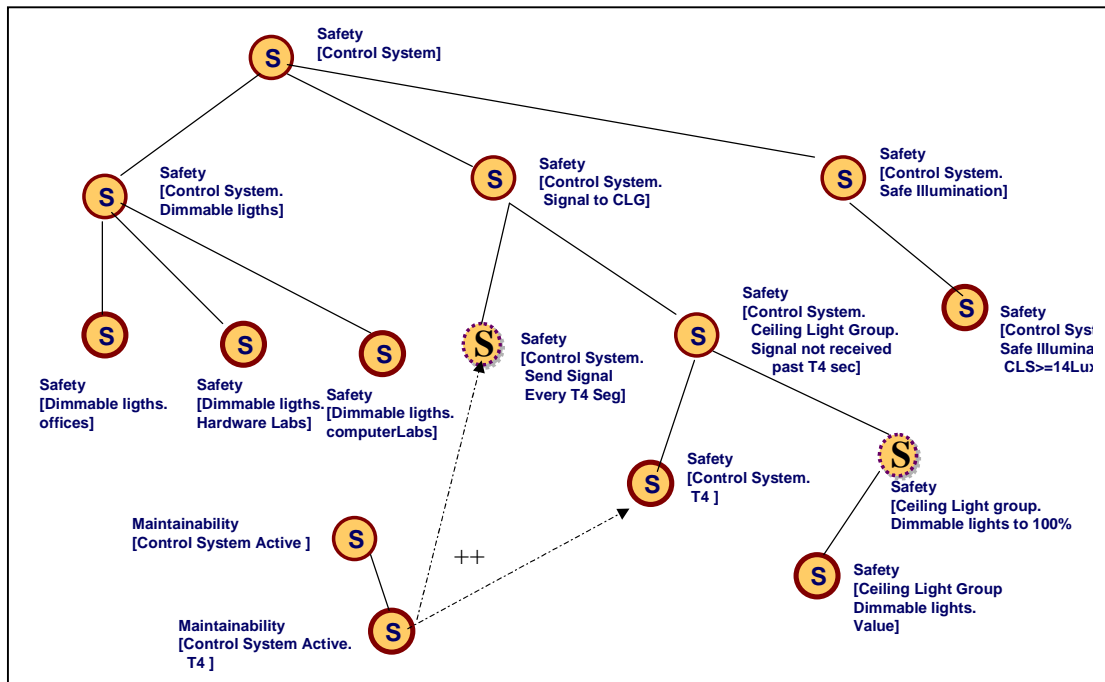
**Figura 8.13 – Diagrama de Classes Definido por Breitman [Breitman 00]**

A Figura 8.13 mostra o diagrama de classes gerado por Breitman [Breitman 00], enquanto a Figura 8.14 mostra o diagrama de classes após termos realizado a integração dos RNFs ao diagrama de classes. Mais adiante iremos detalhar como novas classes foram introduzidas, bem como detalharemos atributos e operações encontrados em algumas das classes existentes. Neste estudo de caso, 6 das oito classes existentes foram afetadas pelo uso da estratégia proposta.



**Figura 8.14 – Diagrama de Classes Após a Integração**



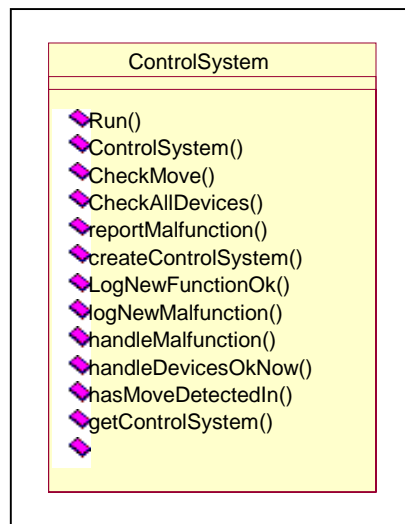


**Figura 8.15 – Um dos Grafos de RNF Usado Quando Avaliando a Classe Sistema de Controle**

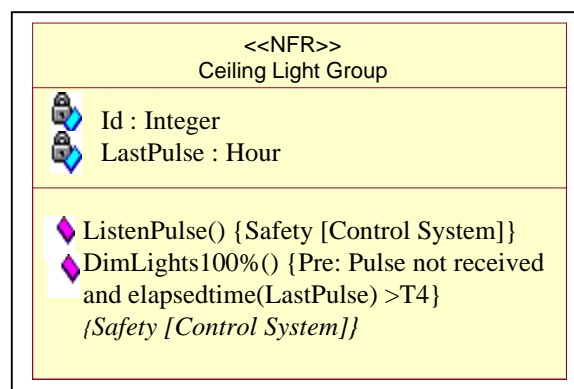
A classe Grupo de Luzes do Teto (CeilingLightGroup), por exemplo, foi acrescentada ao modelo quando estávamos avaliando a classe Sistema de Controle. Ao avaliarmos a classe Sistema de Controle procuramos todos os grafos de RNF onde aparecesse alguma referência ao símbolo Sistema de Controle.

A Figura 8.15 mostra um dos grafos achados onde vemos a necessidade de que o Sistema de Controle seja seguro. Para satisfazer este RNF, uma das necessidades demonstrada no grafo era a de que, se o sinal do sistema de controle que é enviado a cada T4 segundos por um determinado grupo de luzes não fosse recebido, este grupo de luzes deveria colocar a atenuação da luz em 100% (aqui significando que as luzes ficam no máximo). Ao avaliarmos a definição das operações da classe sistema de controle, Figura 8.16, verificamos que esse RNF não era atendido. Como não havia nenhuma classe definida para grupo de luzes, avaliamos que seria necessária a criação da mesma para satisfazer este RNF. A Figura 8.17 mostra o detalhamento de atributos e operações pertencentes a essa classe para satisfazer às necessidades deste RNF. É

importante ressaltar que como esta classe surgiu da necessidade da satisfação de um RNF ela está estereotipada para representar isso.



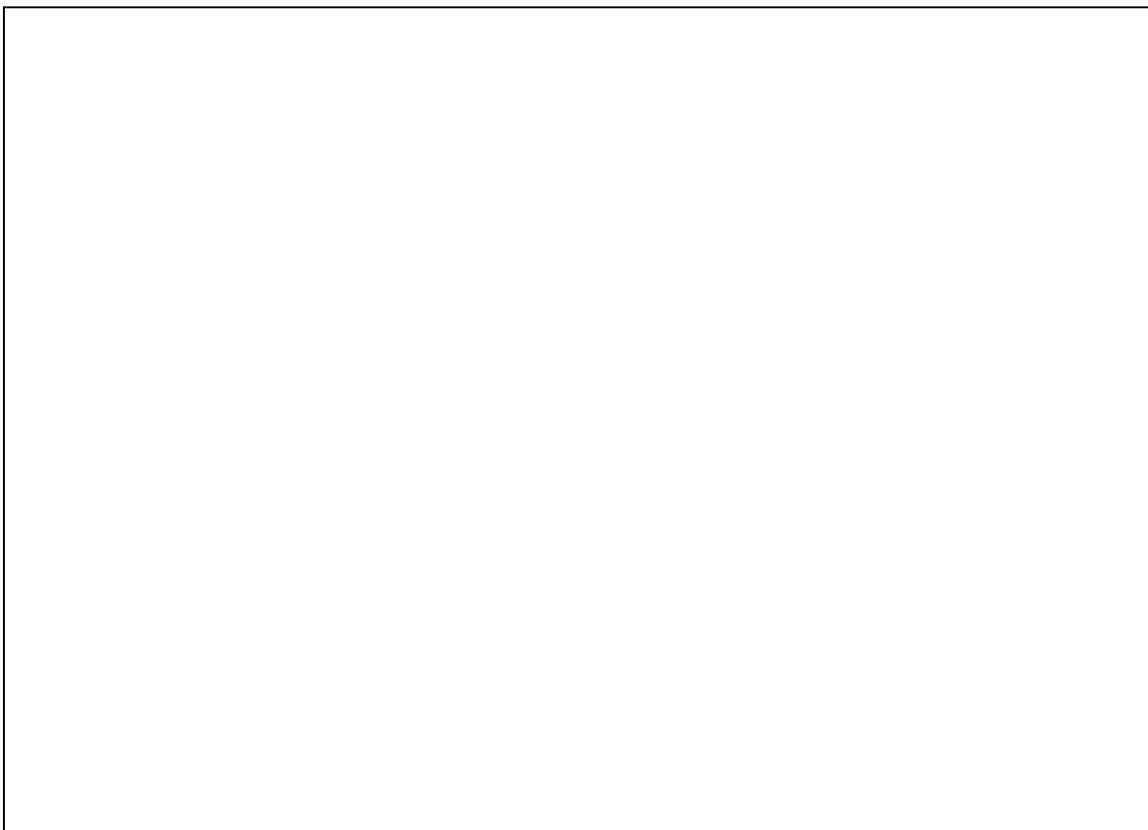
**Figura 8.16 – Classe Sistema de Controle Antes dos RNFs**



**Figura 8.17 – Classe Grupo de Luzes do Teto Resultante da Satisfação do RNF *Seguro* Aplicado ao Símbolo Sistema de Controle**

Adicionamos então essa classe ao diagrama e mais à frente procedemos para esta nova classe do mesmo jeito que fizemos para todas as outras, ou seja, percorremos os grafos de RNF para identificar possíveis RNFs não satisfeitos. A figura 8.18 mostra um grafo onde a classe grupo de luzes do teto aparece, aqui abreviada para CLG. Neste grafo de RNF o que aparece é a necessidade de que cada sala leve em consideração a quantidade de luz que vem de fora do ambiente para gerar a iluminação desejada utilizando-se desta luminosidade para atenuar mais ou menos as luzes, e desta forma economizar energia. Necessitaremos para isso de dois atributos

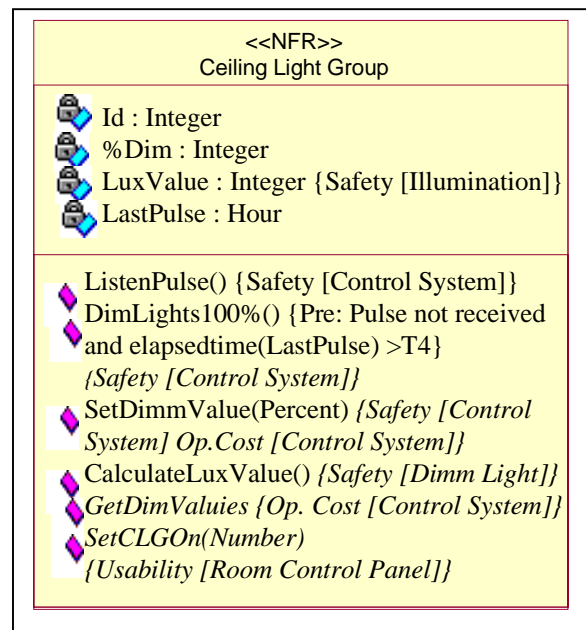
um para armazenar a atenuação em percentual, unidade que é utilizada para comunicar ao atenuador o quanto ele deve atenuar as luzes, e outro para armazenar o equivalente em lux da iluminação fornecida. A necessidade deste último atributo advém do fato de precisarmos usar as leituras do sensor de luzes externas para compor a iluminação interna, e do fato de que este sensor fornece um valor em lux. Além disso, para satisfazer o RNF *Seguro* aplicado à Sala, também precisamos saber o valor da iluminação de um CLG em lux. A Figura 8.19 mostra como ficou ao final a classe grupo de luzes do teto.



**Figura 8.18 – Grafo de RNF Aplicável à Classe Grupo de Luzes do Teto**

Após incluirmos a classe Grupo de Luzes do Teto, voltamos à classe Sistema de Controle para continuar a avaliar o grafo mostrado na Figura 8.15. Faltavam ainda serem satisfeitas as operacionalizações que diziam respeito ao envio de um sinal a cada T4 segundos, à necessidade da definição de um atributo (operacionalização

estática Iluminação Segura – Safe Illumination) para a quantidade mínima de iluminação, e ainda de o valor T4 estar definido sob a forma de um atributo que possa ser atualizado a partir do painel de controle com o intuito de simplificar futuras manutenções advindas do avanço da tecnologia.



**Figura 8.19 – Resultado Final da Classe Grupo de Luzes do Teto**

A Figura 8.20 mostra a classe Sistema de Controle após introduzirmos as operações e atributos necessários. Cabe ressaltar que as operações “SetT4”, “GetT4”, “SetSafeIllumination” e “GetSafeIllumination, aparecem para satisfazer a necessidade de que outras classes usem os atributos “T4” e “Safe Illumination” aqui definidos.

A Figura 8.21 mostra o grafo de RNF relativo à usabilidade sob a ótica do uso do painel de controle de sala. Verificamos por este grafo que cada sala necessita de uma linha de status que mostre o estado de cada um dos dois grupos de luzes da sala, ou seja, se está ligado ou desligado. Como a classe Sala não implementava nada neste sentido, decidimos que a melhor opção seria a criação de uma nova classe que desempenhasse tal papel. Esta classe necessita, além do identificador de a qual sala ela está associada, de um atributo para cada um dos dois grupos de luzes da sala, cada

qual sinalizando se o mesmo está aceso (True) ou apagado (False). Conseqüentemente tivemos que criar operações que manipulassem os atributos dessa classe conforme pode ser visto na Figura 8.22.

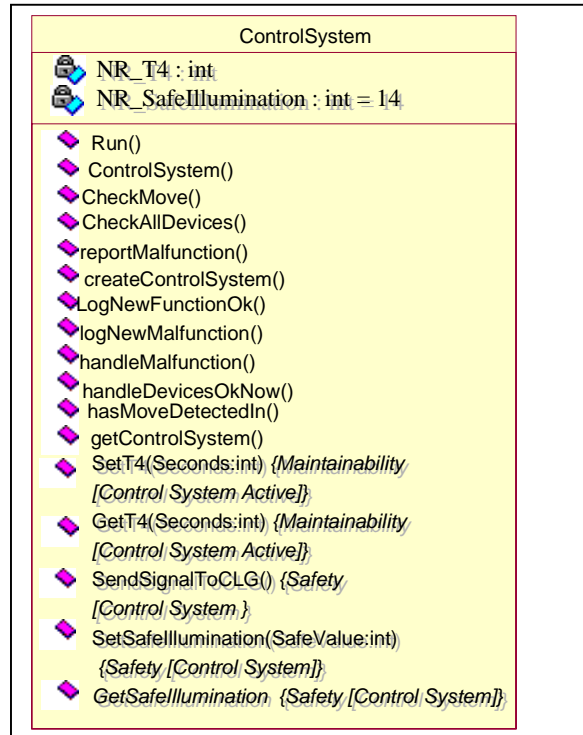


Figura 8.20 – Classe Sistema de Controle com RNFs

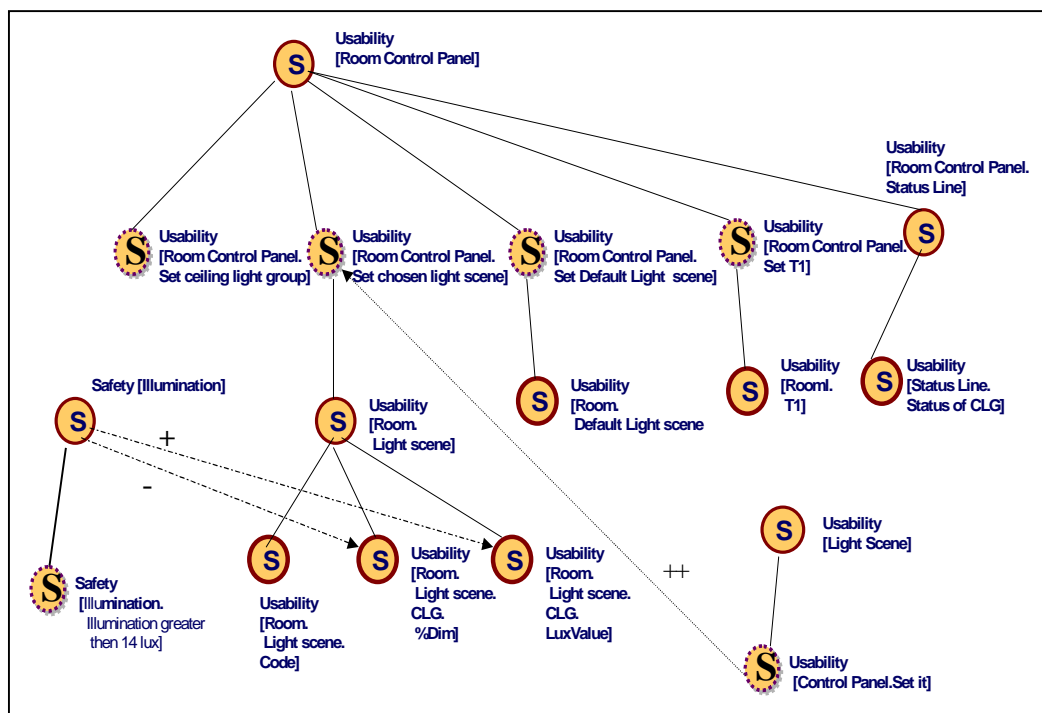
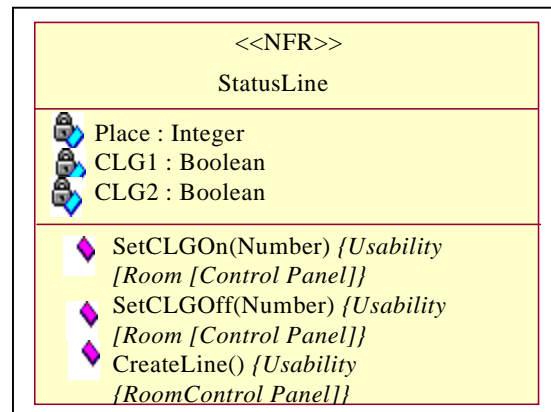


Figura 8.21 – Grafo de RNF para Usabilidade Aplicada á Painel de Controle



**Figura 8.22 – Detalhamento da Classe Linha de Status**

Ao analisarmos a classe Painel de Controle da Sala, verificamos que o grafo que especificava o RNF *Seguro* relacionado ao símbolo Painel de Controle da Sala, Figura 8.23, estabelece que é necessário que o Painel de Controle garanta que nem o esquema de luzes default, nem o esquema de luzes escolhido tenham iluminação resultante inferior à iluminação segura.

Como a classe Painel de Controle possui duas subclasses, Figura 8.24, temos que analisar em todas as três classes envolvidas se alguma delas satisfaz este RNF. No caso em questão isso não é verdade, ou seja, não existem críticas para os valores entrados em relação a eles serem seguros ou não. Em consequência, temos que adicionar operações que façam este controle. Como essas operações são necessárias tanto para dados entrados pelo painel de controle das salas como do administrados, iremos colocar as novas operações na classe painel de controle. A Figura 8.25 mostra como ficou definida a classe painel de controle após o processo de integração de RNFs.

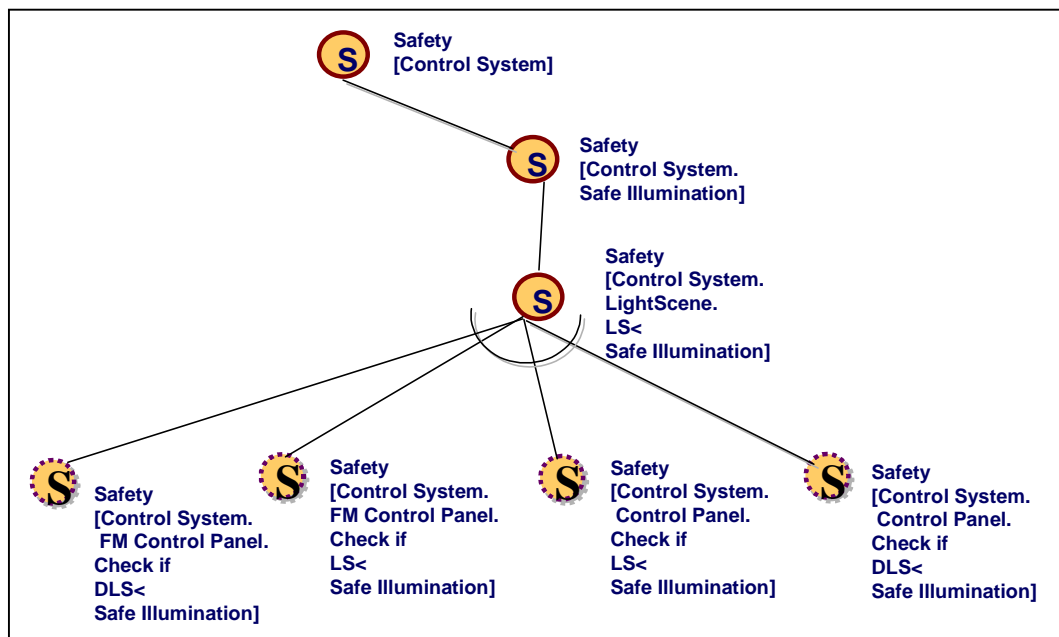


Figura 8.23 – Grafo do RNF Seguro Aplicado a Sistema de Controle

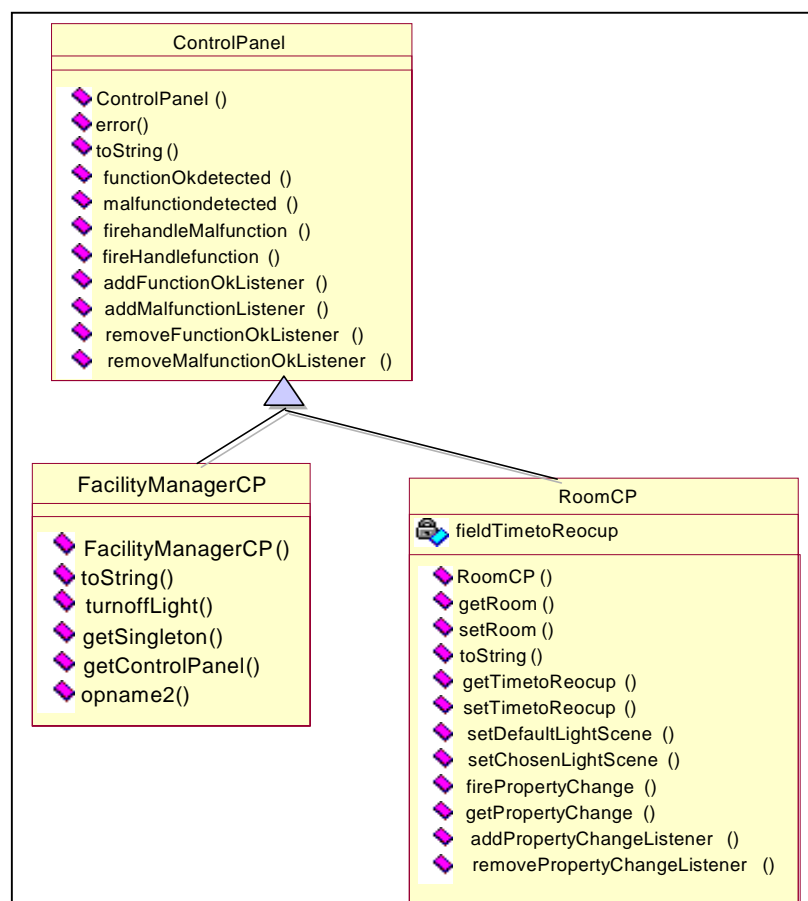
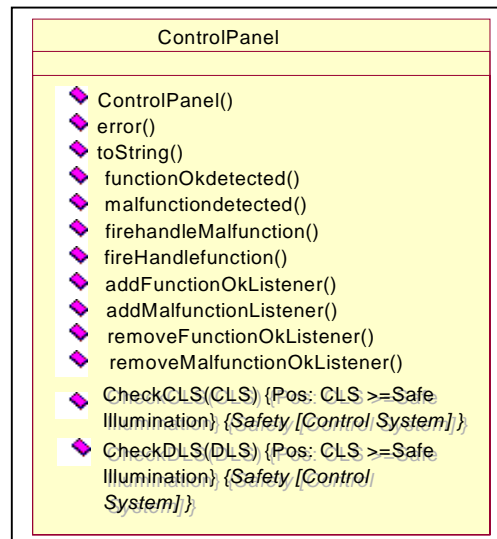


Figura 8.24 – Classe Painel de Controle e suas Subclasses



**Figura 8.25 – Classe Painel de Controle Após Integração com RNFs**

Uma lista completa dos grafos e classes deste estudo de caso encontra-se no Apêndice A.

## 8.2 - Sistema de Iluminação II

Este estudo de caso também utilizou a abordagem do estudo de caso mostrado na seção 8.1. Aqui, um dos times foi representado por um grupo de alunos de graduação da cadeira de Projeto de Sistemas de Software do curso de engenharia de computação da PUC-Rio. Esse time gerou um conjunto de especificações entregues como resultado do trabalho de final de curso submetido a avaliação para definição de grau. Este conjunto de especificações referia-se ao mesmo software para controle de iluminação de um andar de um prédio proposto em um seminário em Dagstuhl [Dagstuhl 99] e consistia de diagrama de classes, diagramas de seqüência e diagrama de colaborações. O segundo time foi personificado pelo autor desta tese.

Como já tínhamos gerado os modelos da visão não funcional no estudo de caso I e o objeto de estudo permanecia sendo o mesmo, *utilizamos o LAL e os grafos de RNF definidos no estudo de caso anterior* e passamos direto à integração das visões funcional e não funcional. É importante enfatizar que *ambos os estudos*



*utilizaram a mesma especificação de problema*, mas que originalmente chegaram a diferentes modelos conceituais.

Aqui, novamente utilizamos como fator de comparação o número de novas classes, operações e atributos encontrados pelo autor da tese em contraposição ao número de classes, operações e atributos existentes antes da utilização da estratégia

A Tabela 8.2 ilustra os dados obtidos ao final do estudo de caso.

	Classes	Operações	Atributos
Modelo conceitual original	15	115	32
Derivadas da aplicação da estratégia	3	45	8
Variação Percentual	20%	39,1%	25%

**Tabela 8.2 – Resultados Obtidos no Estudo de Caso II**

Das 15 classes existentes 4, ou seja, 27%, sofreram alterações em decorrência da aplicação da estratégia. Em seguida, mostraremos as alterações introduzidas pela estratégia no diagrama de classes e nos deteremos em algumas das classes afetadas para observarmos mais detalhadamente alterações surgidas da aplicação da estratégia.

## **8.2.1 - Integrando as Visões Funcional e Não Funcional**

### **8.2.1.1 Diagrama de Classes**

Novamente, o primeiro passo neste estudo de caso foi avaliar se todas as classes definidas eram símbolos do LAL. Aqui encontramos algumas classes que não eram símbolos do LAL, isto porque o desenho utilizado neste estudo de caso possuía um viés de modelo de especificação incluindo o uso de patterns [Gamma 94]. Como estas classes sempre se comunicavam com classes que eram símbolos do LAL verificamos que usar apenas estas últimas não comprometeria o resultado final da integração.

Seguimos então o mesmo padrão utilizado no estudo de caso anterior e detalhado na seção 7.2 para integrarmos as visões funcional e não funcional. Classe a classe procuramos pela ocorrência do símbolo que nomeava a classe nos grafos de RNF e ao

encontrarmos avaliamos se a classe já satisfazia ou não às operacionalizações expressadas no grafo.

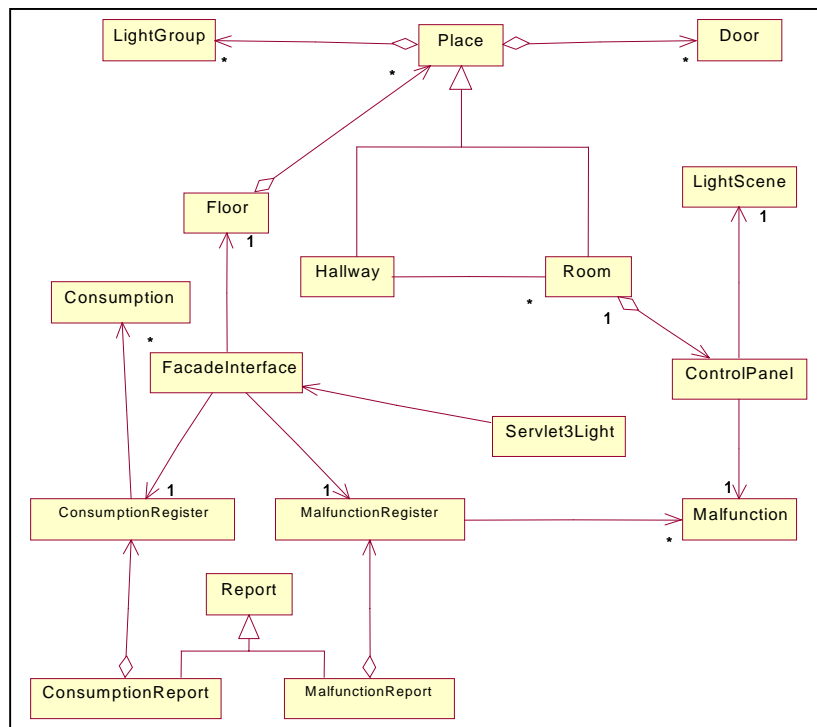


Figura 8.26 – Diagrama de Classes Antes dos RNFs

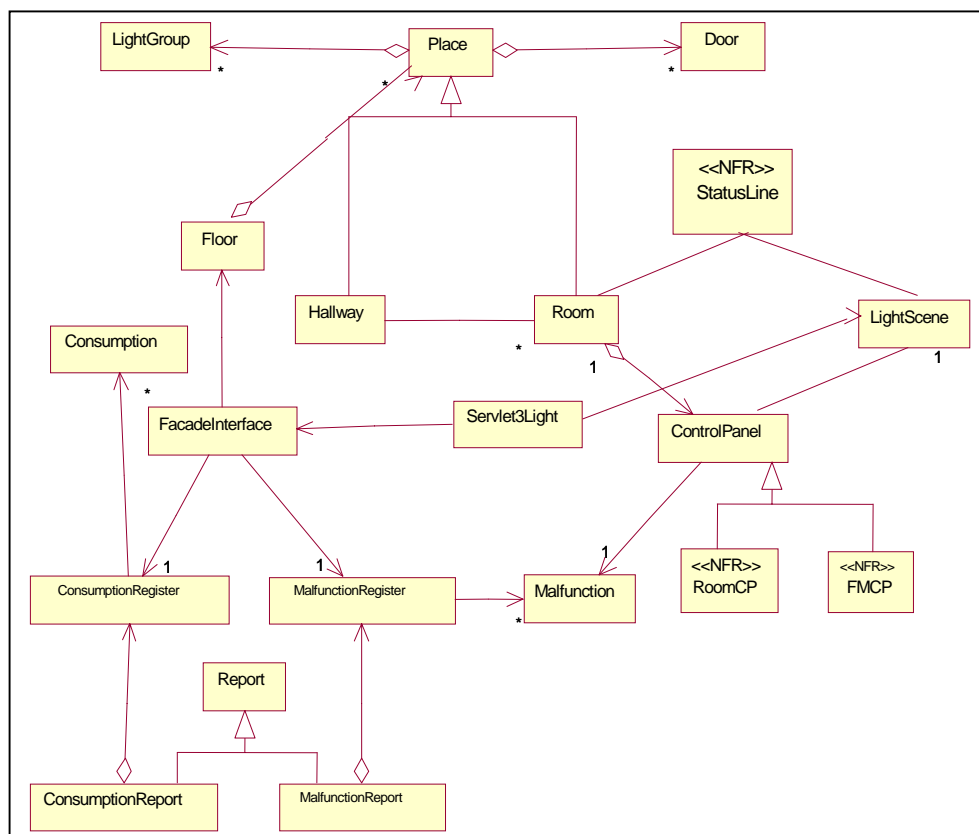
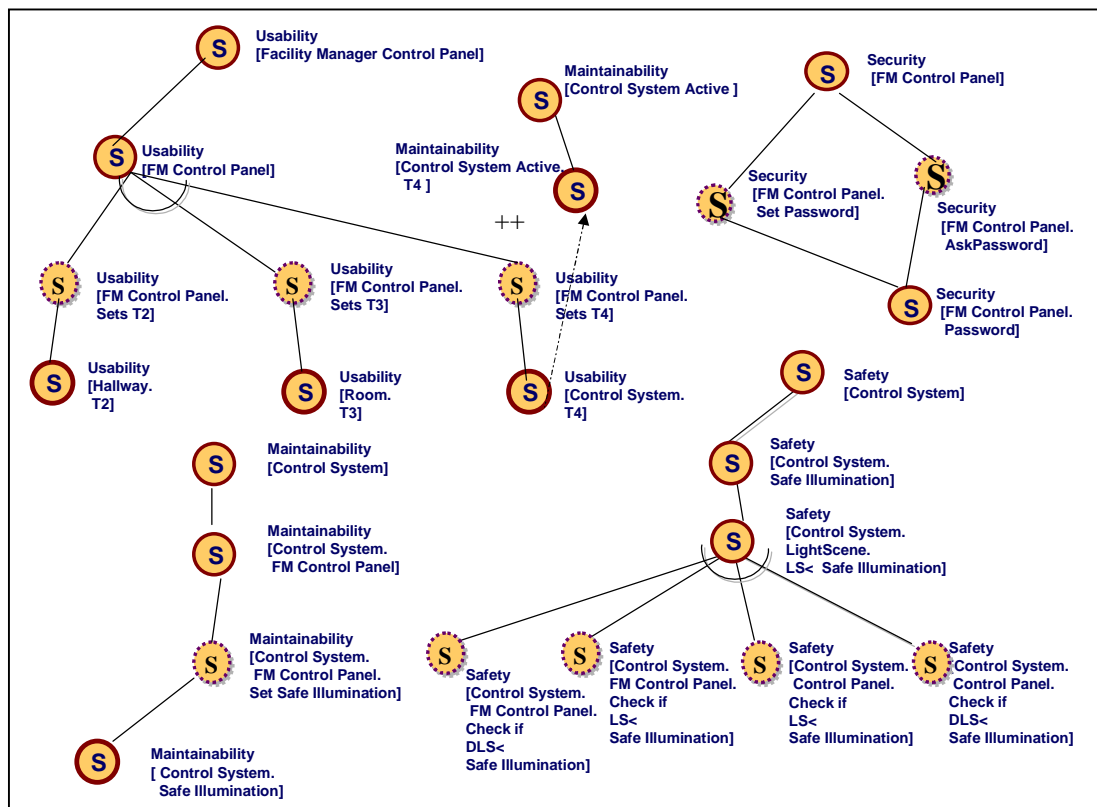


Figura 8.27 – Diagrama de Classes com RNFs

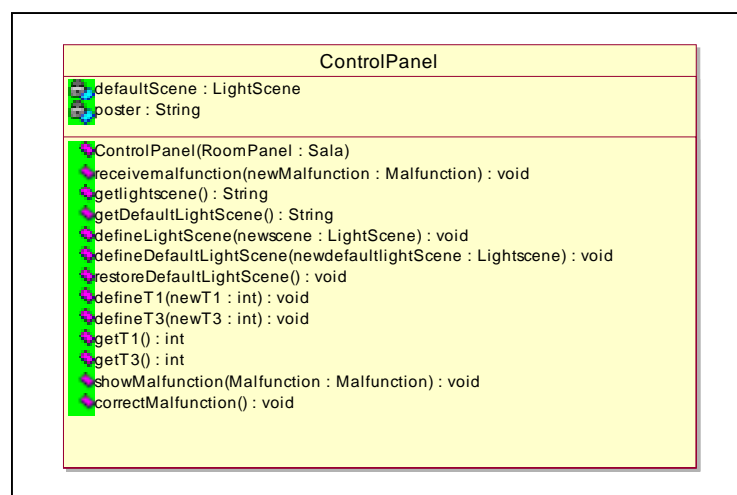
As Figuras 8.26 e 8.27 mostram os diagramas de classe antes e depois do processo de integração respectivamente.

Podemos observar pela comparação das duas figuras que as subclasses da classe Painel de Controle foram incluídas no diagrama após a integração dos RNFs ao mesmo. Isso se deveu ao fato de haverem determinadas atribuições que só podem ser realizadas pelo administrador via um painel de controle específico. Esta necessidade foi identificada quando estávamos avaliando a classe painel de controle. Nesta etapa examinamos todos os grafos de RNF que continham o símbolo Painel de Controle e verificamos se todas as operacionalizações eram satisfeitas pela classe. Não apenas verificamos que não era, como verificamos ainda que por questões de segurança uma classe Painel de Controle Adm. (FMCP) deveria ser criada para satisfazer essas necessidades de segurança.



**Figura 8.28 – Conjunto de Grafos de RNF Usados para a Classe Painel de Controle**

A Figura 8.28 exibe diversos grafos utilizados para avaliar a classe Painel de Controle, Figura 8.29 . Neste conjunto de grafos pode-se observar que há a necessidade de ser pedida e confirmada uma senha a quem estiver usando o painel de controle do administrador. Precisa-se ainda de uma maneira de entrar os valores T2 para a classe Corredor, T3 para a classe Sala e T4 e Iluminação Segura para a classe Sistema de Controle. É importante ressaltar que, exceto a operação relativa a T3, as outras operações **não** podem estar disponíveis no painel de controle das salas, sendo reservadas apenas aos painéis de controle usados pelo administrador. Por fim, precisa-se garantir que os esquemas de luzes entrados via painel de controle tenham uma luminosidade superior à definida em Iluminação Segura.

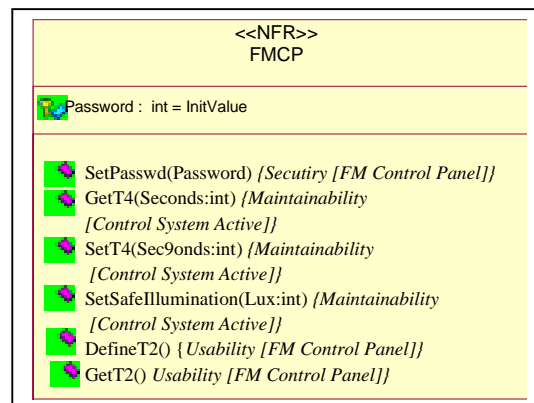


**Figura 8.29 – Detalhamento da Classe Painel de Controle**

A Figura 8.30 mostra o detalhamento da classe Painel de Controle Adm. (FMCP), com os atributos e operações decorrentes da satisfação das operacionalizações mostradas na Figura 8.28.

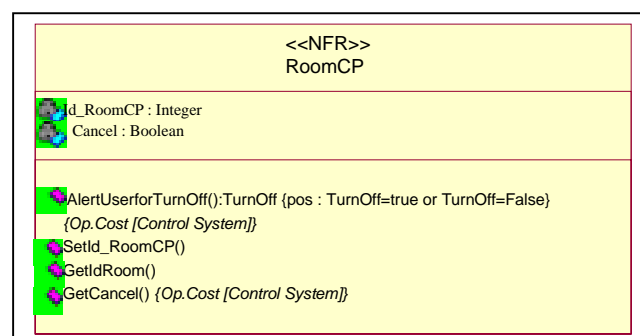
A classe Painel de Controle Sala (RoomCP), sendo uma classe nova, foi também avaliada junto aos grafos onde existiam referencias ao nome da classe. A Figura 8.18, mostrada anteriormente, mostra que uma submeta passa a ser parcialmente satisfeita

quando a confrontamos com o RNF de segurança. Esta submeta indicava que a classe Sala deveria desligar as luzes após estar desocupada por mais de T3 minutos.



**Figura 8.30 – Detalhamento da Classe Painel de Controle Administrador**

Para satisfazer a condição de que, em nenhuma hipótese um usuário possa estar numa sala com menos iluminação do que a segura, e considerando que possa haver falhas não detectáveis nos sensores que levem o sistema a chegar a uma conclusão errônea de que a sala estava desocupada, acrescentamos uma operacionalização dinâmica que determina que o painel de controle da sala deva avisar que as luzes serão desligadas esperando por um cancelamento dessa operação. Só no caso de não haver esse cancelamento é que as luzes serão desligadas. A Figura 8.31 mostra como ficou a classe Painel de Controle de Sala após as necessárias inclusões de atributos e operações para satisfazer esse RNF.



**Figura 8.31 – Detalhamento da Nova Classe Painel de Controle de Sala**

Quanto à Classe Painel de controle, adicionaremos a essa classe as mesmas operações acrescentadas na Figura 8.25 do estudo de caso I, de forma a garantir que



detectamos várias operacionalizações não presentes na classe Corredor, Figura 8.34. Em seguida detalharemos estas operacionalizações, cada qual marcada com uma letra entre parênteses para facilitar futura referência.

- Utilizar a leitura do sensor externo de luz para compor o esquema de luz de forma a economizar energia e por consequência ter de armazenar os valores de atenuação em luz além de valor percentual – Custo Operacional de Sistema de Controle - (A)
- Desligar luzes do corredor após este estar desocupado a T2 minutos - Custo Operacional de Sistema de Controle - (B)
- No caso de mau funcionamento do sensor de externo de luz e o corredor estando ocupado, os grupos de luzes devem ser ligados – RNF Seguro aplicado a Sistema de Controle e à Corredor - (C)
- No caso de mau funcionamento de algum sensor informar administrador – RNF Seguro aplicado a Corredor - (D)
- No caso de mau funcionamento do sensor de movimento colocar o corredor como ocupado – RNF Seguro aplicado a Corredor - (E)

A Figura 8.34 mostra a classe Corredor após adicionarmos todas as operações e atributos necessários para satisfazer as operacionalizações presentes nos grafos das Figuras 8.31 e 8.32. A Figura 8.36 mostra a classe Lugar que, como superclasse da classe corredor irá acolher algumas das operações que em princípio estariam na classe corredor.

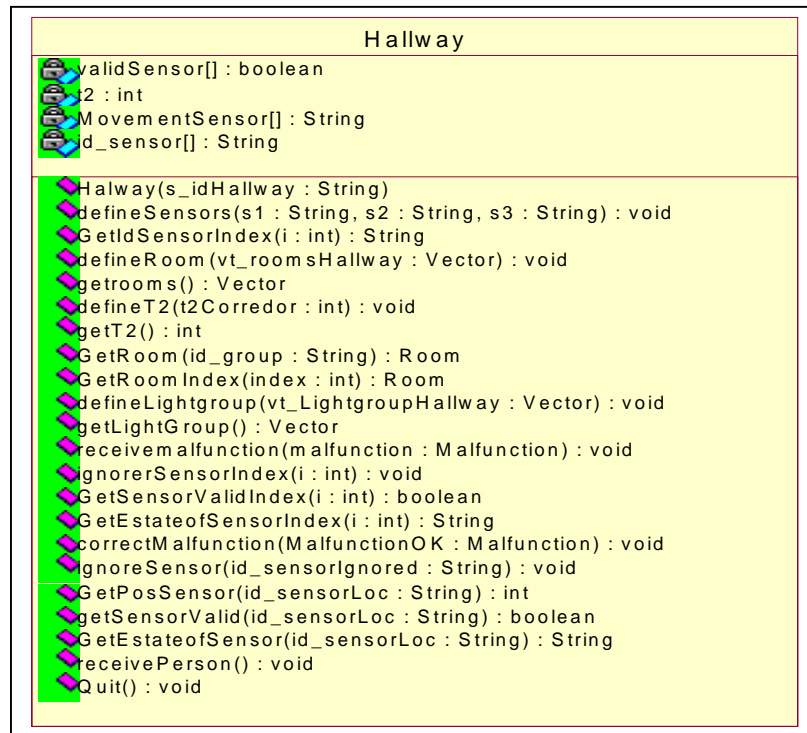


Figura 8.34 – Classe Corredor Antes dos RNFs

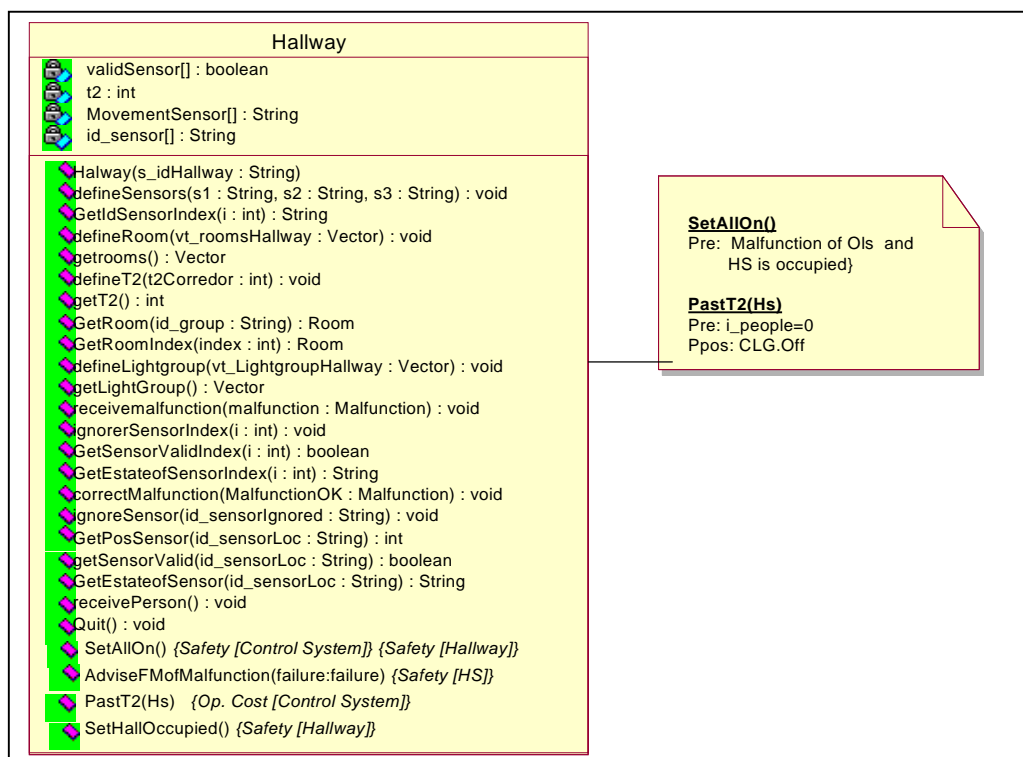
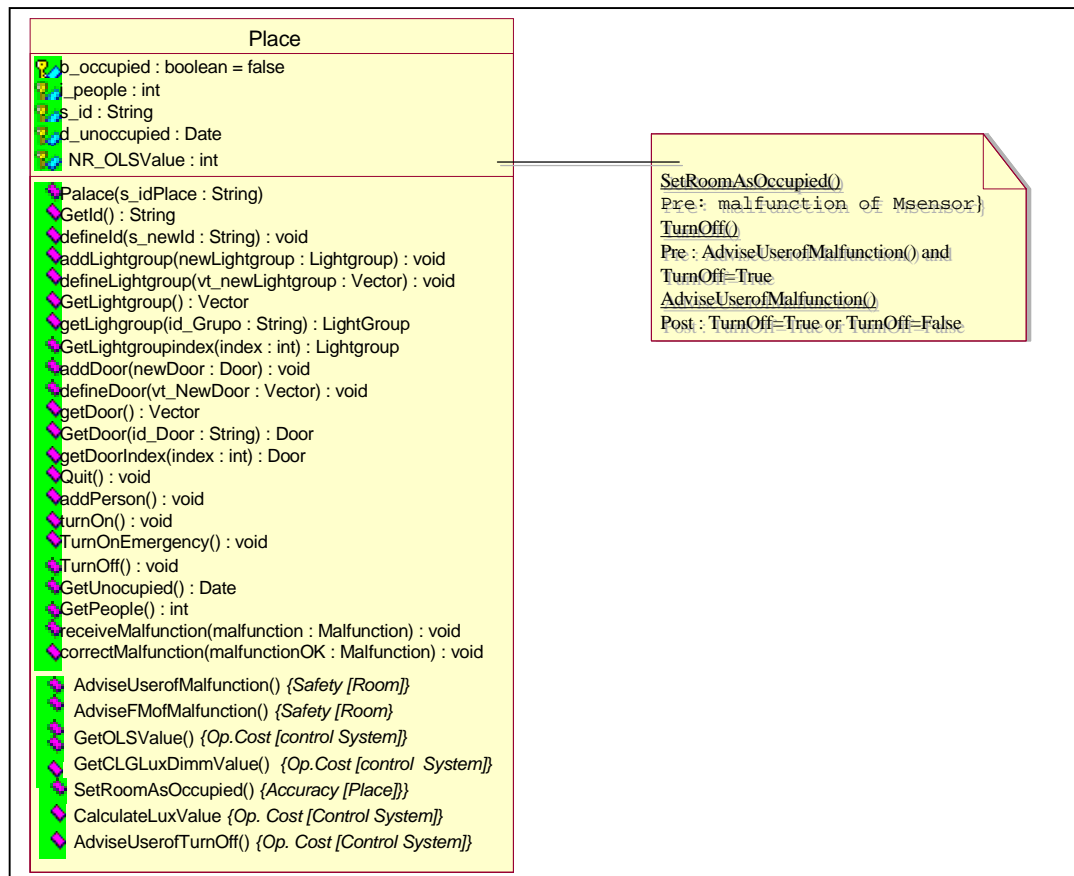


Figura 8.35 – Classe Corredor Após RNFs





**Figura 8.36 – Classe Lugar que é Superclasse de Corredor**

Podemos ver que a operacionalização descrita acima e marcada por (A), é satisfeita nessa classe Lugar através das operações “GetOLSValue” e “CalculateDimValues” que irão respectivamente obter o valor da quantidade de iluminação externa e compor este valor para chegar a um nível de atenuação suficiente para atingir o esquema de luz que estiver selecionado. O atributo “NR\_OLSValue” é adicionado para guardar o valor lido. Estas operações acabaram por ser colocadas na classe lugar uma vez que não apenas o corredor deverá utilizar luz externa na composição de sua iluminação, mas também as salas deverão fazê-lo.

A operacionalização descrita anteriormente e marcada por (B), será satisfeita pela operação “PastT2”, que tem como pré-condição para sua execução o corredor estar vazio e como resultado que os grupos de luzes do corredor estarão desligados.

A operacionalização marcada por (C) é satisfeita pela operação “SetAllOn”, sendo que esta tem como pré-condição para sua execução a ocorrência de um mau funcionamento do sensor de luz e o fato de o corredor estar ocupado.

Já a operacionalização marcada por (D) é satisfeita pela operação “AdviseFMofMalfunction”, colocada também na classe Lugar por serem necessárias também para as salas.

Finalmente, a operacionalização marcada por (E) é satisfeita pela operação “SetHallwayOccupied” da classe Corredor.

#### **8.2.1.2 - Diagrama de Seqüência e Colaboração**

Foram definidos somente quatro Diagramas de Seqüência e Colaboração neste estudo de caso. A dois destes se aplicavam alterações decorrentes de operações e pré-condições que foram introduzidas nas classes em decorrência da necessidade de satisfazer RNFs. A seção 7.3 já ilustra esses exemplos

Uma lista completa das classes, diagramas de seqüência e colaboração deste estudo de caso encontra-se no Apêndice B.

Não foi mostrado nenhum caso destes diagramas referentes ao estudo de caso I por estes não terem sido utilizados no estudo de caso realizado por Breitman.

### **8.3 - Sistema de Automação de Laboratórios de Análises Clínicas**

Este estudo de caso foi projetado utilizando-se novamente a idéia de dois times diferentes trabalhando em um mesmo projeto. No presente caso, um dos times era composto por uma equipe de profissionais alocadas ao desenvolvimento de um sistema de automação para laboratórios de análises clínicas e outro time era composto pelo autor desta tese.

O time composto pela equipe de profissionais encarregada de desenvolver o software, desenvolveu o modelo conceitual que espelhava as necessidades de seus clientes. Este modelo conceitual foi disponibilizado para o autor desta tese sob a forma de um diagrama de classes e alguns diagramas de seqüência e colaboração. Através da aplicação da estratégia proposta nessa tese, o autor desta gerou a visão não funcional representada por um conjunto de grafos de RNF e posteriormente realizou a integração da visão não funcional com o modelo apresentado pela outra equipe. Para gerar os modelos da visão não funcional o autor desta tese teve acesso a diversos usuários do futuro software e documentos internos do cliente como, por exemplo, Manuais da Qualidade referentes à certificação ISO 9000 do cliente.

Novamente, utilizamos como métrica de comparação o número de classes, operações e atributos encontrados pelo autor da tese em contraposição ao número de classes, operações e atributos existente antes da utilização da estratégia. Das 39 classes existentes, 19 (48%) sofreram algum tipo de alteração por força do uso da estratégia.

A Tabela 8.3 ilustra os dados obtidos ao final do estudo de caso.

	Classes	Operações	Atributos
Modelo conceitual original	39	213	105
Derivadas da aplicação da estratégia	9	98	41
Variação Percentual	23%	46,01%	39,04%

**Tabela 8.3 – Resultados Obtidos no Estudo de Caso III**

### 8.3.1 - Desenvolvendo a Visão Não Funcional

Para desenvolvermos a visão não funcional nesse estudo de caso primeiramente construímos o LAL do UdI, utilizando as técnicas de entrevista estruturada e observação e tendo como fontes de informação diversos usuários do futuro software e documentos internos do cliente como, por exemplo, Manuais da Qualidade referentes à certificação ISO 9000 do cliente.

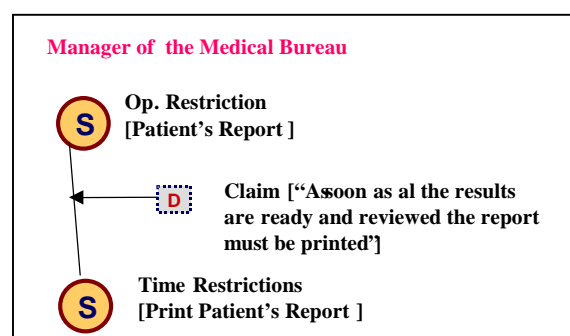
Aplicamos a parte da estratégia que trata da produção de visão não funcional para incorporarmos os RNFs ao LAL gerado, seguindo a estratégia detalhada na seção 5.1. A partir do resultado obtido partimos para a geração dos grafos de RNFs conforme as heurísticas descritas nas seções 5.2 e 5.3. Uma validação foi então feita com diversos atores do UdI para verificar se os grafos estavam completos e se não traziam nenhuma interdependência que não tivesse sido previamente identificada.

Um exemplo desse processo pode ser observado pelo símbolo do LAL *Laudo*. Ao avaliarmos esse símbolo utilizando a base de conhecimentos da ferramenta OORNF [Neto 00] (vide Figura 4, seção 4), notamos que o RNF *Restrições Operacionais* poderia vir a ser necessário a este símbolo. Recorremos à gerente da central médica, um dos usuários do futuro software envolvidos no processo de elicitação, para avaliarmos se isso realmente se aplicava e, caso afirmativo, como se aplicaria.

Descobrimos então que havia uma restrição de tempo associada à impressão do laudo, de forma que este deveria ser impresso assim que todos os exames estivessem

passados. Este RNF foi então acrescentado ao LAL e avaliamos algumas possíveis consequências da inclusão deste RNF, de onde avaliamos que seria necessário que o sistema pudesse imprimir o laudo e assiná-lo ao mesmo tempo. Esta necessidade foi então espelhada no símbolo Sistema de Automação Laboratorial.

Uma vez que os RNFs foram incorporados ao LAL, passamos a avaliar cada símbolo do LAL para montar os grafos de RNF. Utilizando o símbolo Laudo, verificamos que o RNF Restrições Operacionais constava das noções do símbolo e, portanto deveria dar origem a um grafo de RNF. A Figura 8.37 mostra o início da construção deste grafo onde identificamos a raiz do grafo e a primeira decomposição realizada, no caso uma decomposição por tipo.



**Figura 8.37 – Princípio da Composição de um Grafo de RNF para Laudo**

Uma vez que tínhamos feito essa decomposição partimos para um detalhamento das submetas e/ou operacionalizações que iriam satisfazer esse RNF. A Figura 8.38 mostra o grafo resultante.

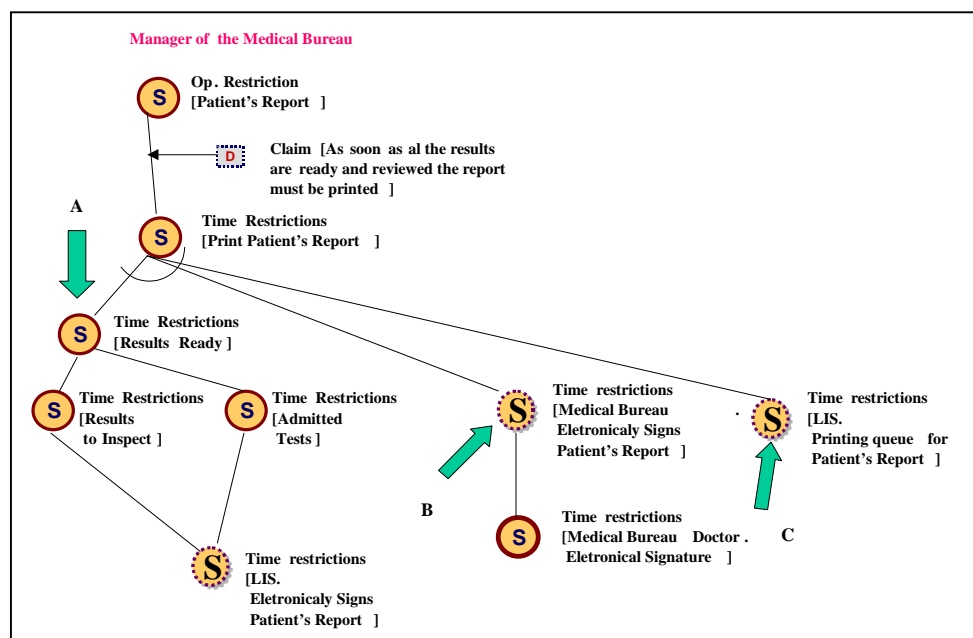
A parte identificada com a seta A mostra uma decomposição por tópico realizada no RNF. Uma vez que tínhamos que disponibilizar o laudo impresso o mais rápido possível, a primeira área que deveríamos observar era a da garantia que o laudo fosse impresso assim que os resultados estivessem prontos. Para isso introduzimos a noção de que o próprio sistema iria assinar os laudos eletronicamente, ou seja, iria imprimir

o laudo já contendo uma assinatura digitalizada da gerente da área de processamento de exames.

A parte identificada pela seta *B* mostra uma operacionalização dinâmica que estabelece que o médico que revisa o laudo deve poder fazê-lo em vídeo já o liberando para impressão com sua assinatura colocada de forma eletrônica (assinatura digitalizada). Anteriormente, o médico validava o laudo, mandava para impressão e posteriormente assinava manualmente.

A parte identificada pela seta *C* mostra uma operacionalização dinâmica que estabelece a necessidade de que o sistema (LIS) mantenha uma fila de impressão de laudos prontos e assinados de forma a não depender de um empregado para fazê-lo.

Avaliamos ao fim da decomposição que Assinatura eletrônica de laudo deveria ser um novo símbolo do LAL, e portanto, foi então detalhada no LAL.



**Figura 8.38 – Refinamento do Grafo da Figura 8.37**

Mais à frente no processo, ao validarmos os grafos de RNF com os usuários do futuro software, descobrimos que a gerente da área de processamento se opunha a

uma assinatura eletrônica de laudos. Após algumas negociações, foi estabelecido que por razões de confiabilidade do laudo, somente poderiam ser impressos pelo sistema os laudos nos quais **todos** os resultados estivessem dentro de uma faixa determinada, denominada de faixa de assinatura eletrônica. A Figura 8.39 mostra o Grafo resultante. Cabe ressaltar que a parte do grafo da Figura 8.38 que era anteriormente considerada satisfeita, passa a ser considerada parcialmente satisfeita.

O Apêndice C mostra uma extensiva lista de grafos de RNF que foram encontrados. Alguns dos grafos encontrados não foram representados atendendo a solicitação da empresa onde foi realizado o estudo de caso. Essa solicitação advém do fato de que a tese é uma publicação aberta ao público e muitos grafos continham detalhes que eles consideram estratégicos aos quais a concorrência não deve ter acesso.

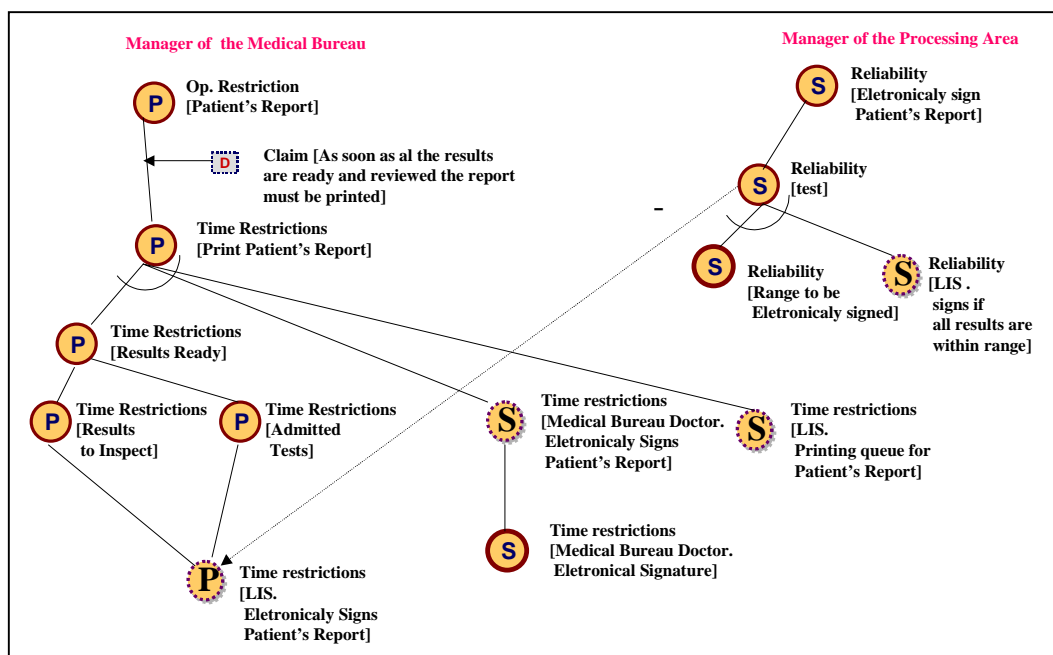
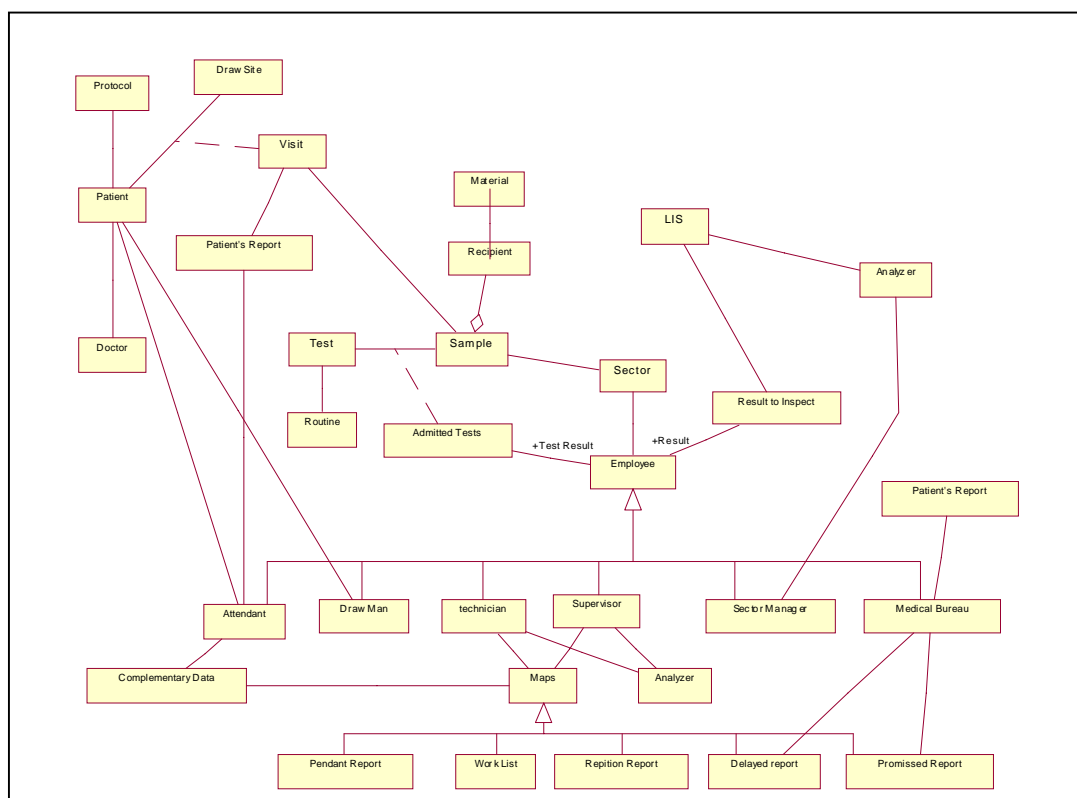


Figura 8.39 – Grafo Resultante Depois de validação com Cliente

### 8.3.2 - Integrando as Visões Funcional e Não Funcional

#### 8.3.2.1 Integrando os RNFs ao Diagrama de Classes

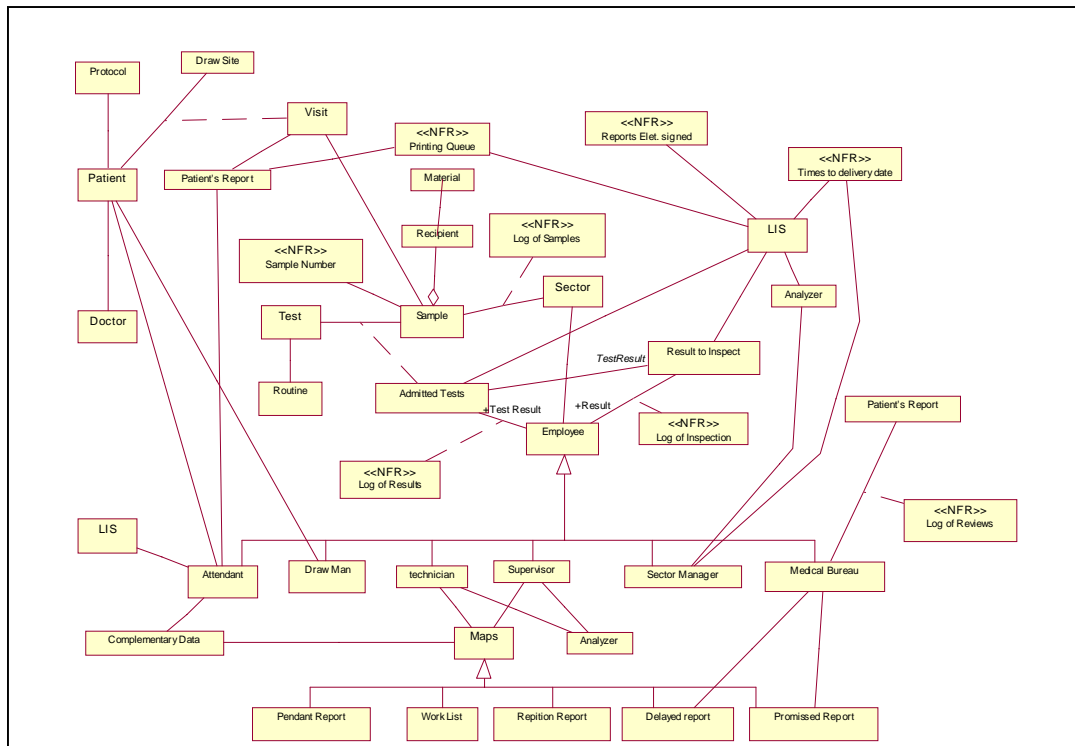
A integração dos RNFs encontrados na visão não funcional ao diagrama de classes, que basicamente representa a visão funcional dos desenvolvedores da empresa onde foi feito o estudo de caso, foi feita utilizando-se a estratégia detalhada na seção 7.2.



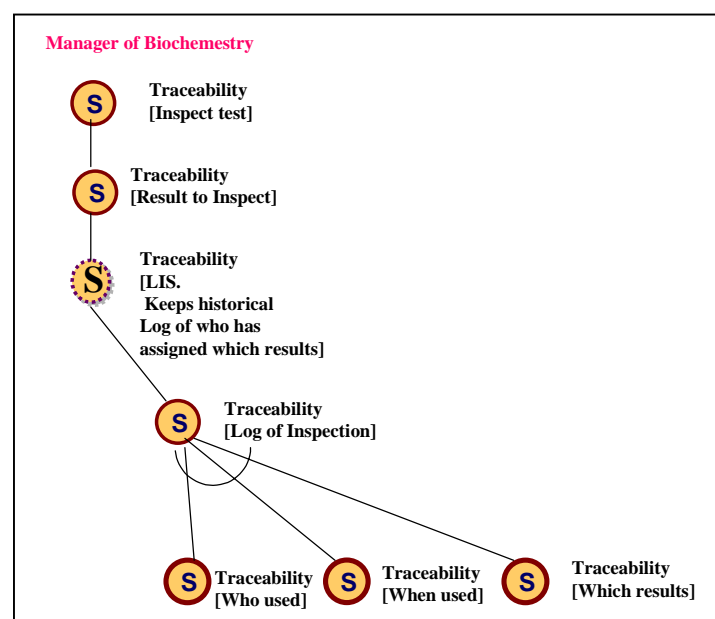
**Figura 8.40 – Diagrama de Classes Original do Sistema de Laboratórios**

Percorremos todas as classes do diagrama e para cada classe procuramos nos diversos grafos de RNF pela ocorrência do símbolo que nomeava a classe. Quando encontrávamos, avaliávamos se os atributos e operações da classe eram suficientes para implementar as operacionalizações estáticas e dinâmicas presentes no grafo. As Figuras 8.40 e 8.41 mostram, respectivamente, parte do diagrama de classes antes e depois da aplicação da estratégia proposta.





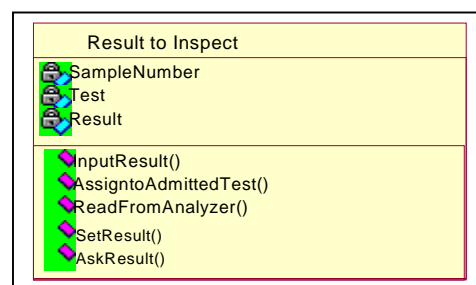
**Figura 8.41 – Parte do Diagrama de Classes Após a Aplicação da Estratégia**



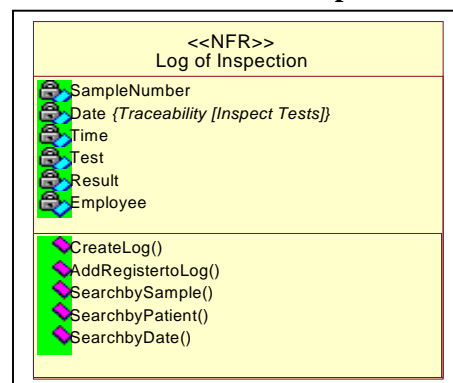
**Figura 8.42– Grafo de RNF Integrado à Classe Resultados a Inspeccionar**

A classe resultado a inspeccionar, por exemplo, aparece como um nó do grafo demonstrado na Figura 8.42 que exibe a necessidade de haver rastreabilidade dos resultados inspeccionados. Verificamos pelo grafo que um histórico de inspeção deve ser mantido com informações sobre quem inspeccionou, quando e quais resultados foram utilizados durante o processo de inspeção.

A Figura 8.43 mostra a Classe Resultado a Inspeccionar antes da integração dos RNFs. Podemos observar que ela não satisfazia a necessidade de se manter um histórico de resultados inspecionados. Como esse histórico demandaria uma série de operações de pesquisa, resolvemos criar uma nova classe Histórico de Resultados a Inspeccionar mostrada na Figura 8.44. Como essa classe não existia no LAL, por razão de consistência entre modelos acrescentamos esse símbolo ao LAL. Essa nova classe foi representada como uma classe de associação entre as classes empregado e resultados a inspecionar, visto que os atributos da nova classe não pertencem a nenhuma das duas classes sozinhas e sim, a uma instancia de associação de ambas.



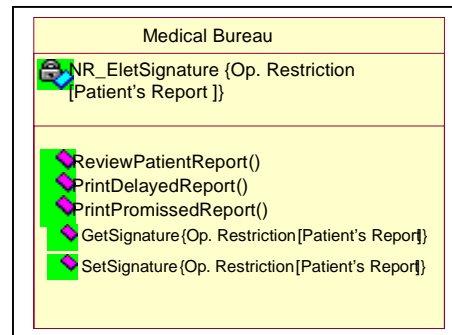
**Figura 8.43 – Classe Resultado a Inspeccionar Antes da Integração**



**Figura 8.44 – Classe Histórico de Inspeção Criada após a Integração das Visões**

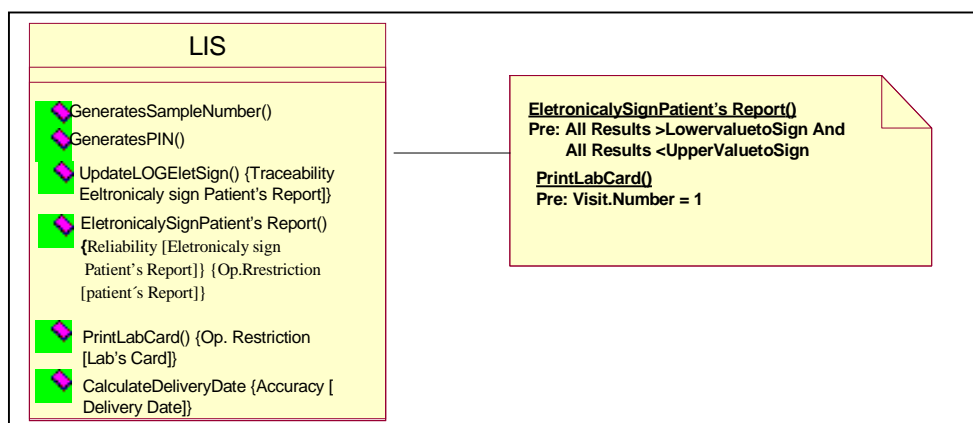
Num outro momento, ao revisarmos a classe Central Médica verificamos que a mesma aparecia no grafo mostrado na Figura 8.39 com uma operacionalização estática denotando a necessidade de termos armazenada a assinatura de cada funcionário da central médica, para que o sistema (denominado de LIS) pudesse imprimir o laudo assim que o médico terminasse a sua revisão, enviando-o para

impressão sem que o mesmo tivesse de voltar para ser assinado manualmente. A Figura 8.45 mostra a classe Central Médica com as mudanças advindas desse RNF.

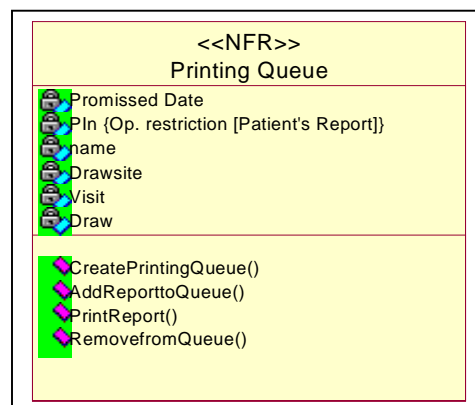


**Figura 8.45 – Classe Central Médica após Integração com RNFs**

É importante ressaltar que o procedimento definido antes da integração dos RNFs definia que o atendente iria imprimir o laudo para que depois o mesmo fosse submetido, em papel, à revisão e assinatura pela central médica. Uma vez que buscamos eventuais restrições operacionais aplicáveis ao laudo e desenhamos o grafo da Figura 8.39, passamos a nos aperceber que o procedimento que havia sido estabelecido preocupado apenas com as necessidades funcionais teria de ser revisto, resultando em alterações na classe LIS (sigla em inglês de sistema de automação laboratorial) e na criação de uma classe Fila de Impressão como pode ser visto na Figura 8.41. As Figuras 8.46 e 8.47 ilustram, respectivamente, as alterações realizadas na classe LIS e a classe Fila de Impressão criada como parte do novo desenho necessário ao sistema.



**Figura 8.46 – Classe LIS Depois da Integração com RNFs**



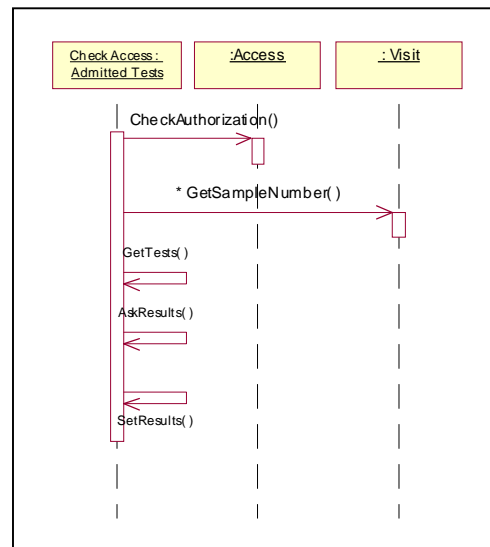
**Figura 8.47 – Classe Fila de Impressão Resultante da Integração de RNFs**

O Apêndice C mostra grande parte dos Grafos e Classes que foram utilizados nesse estudo de caso. Um pequeno número de classes e grafos foi mantido fora desse apêndice em respeito a restrições de confidencialidade de negócios impostas pela empresa onde foi efetuado o estudo de caso.

### 8.3.2.2 Integrando os RNFs aos Diagramas de Seqüência e Colaboração

Apesar de termos tido poucos diagramas para realizar estudos, estes poucos exemplos foram suficientes para que ficássemos convencidos da importância de se aplicar os RNFs a esses diagramas devido às inúmeras alterações encontradas.

Em seguida mostramos um exemplo do diagrama de seqüência que ilustra a entrada de resultados para exames de um paciente. As Figuras 8.48 e 8.50 ilustram, respectivamente, os diagramas de seqüência antes e depois dos RNFs serem considerados.

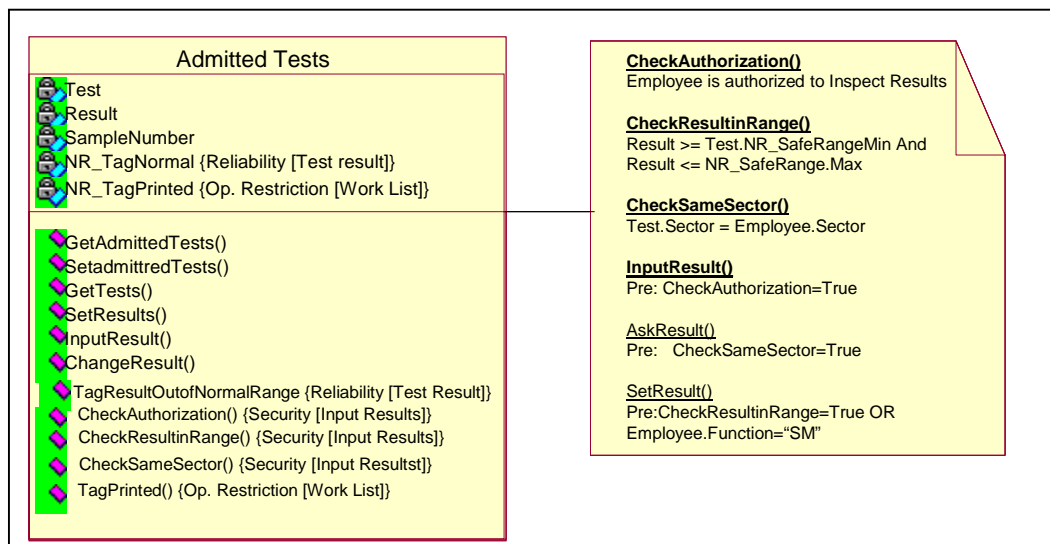


**Figura 8.48 – Diagrama de Seqüência para a Entrada de Resultados só com os Requisitos Funcionais**

Observando os dois diagramas, pode-se notar que as alterações se deram no nível de inclusão de novas classes colaborando com as já existentes, inclusão de novos procedimentos aplicados a classes já presentes e inclusão de condições e pré-condições de execução de operações.

Para realizarmos a integração, seguimos o especificado na seção 7.3. Percorremos todas as classes do diagrama de classes verificando todas as operações que tinham como origem RNFs. Para cada operação encontrada, procurávamos diagramas de seqüência e colaboração que tivessem essa classe como participantes e analisávamos se a inclusão desta operação àquele diagrama era necessária e em caso afirmativo procedíamos a essa inclusão.

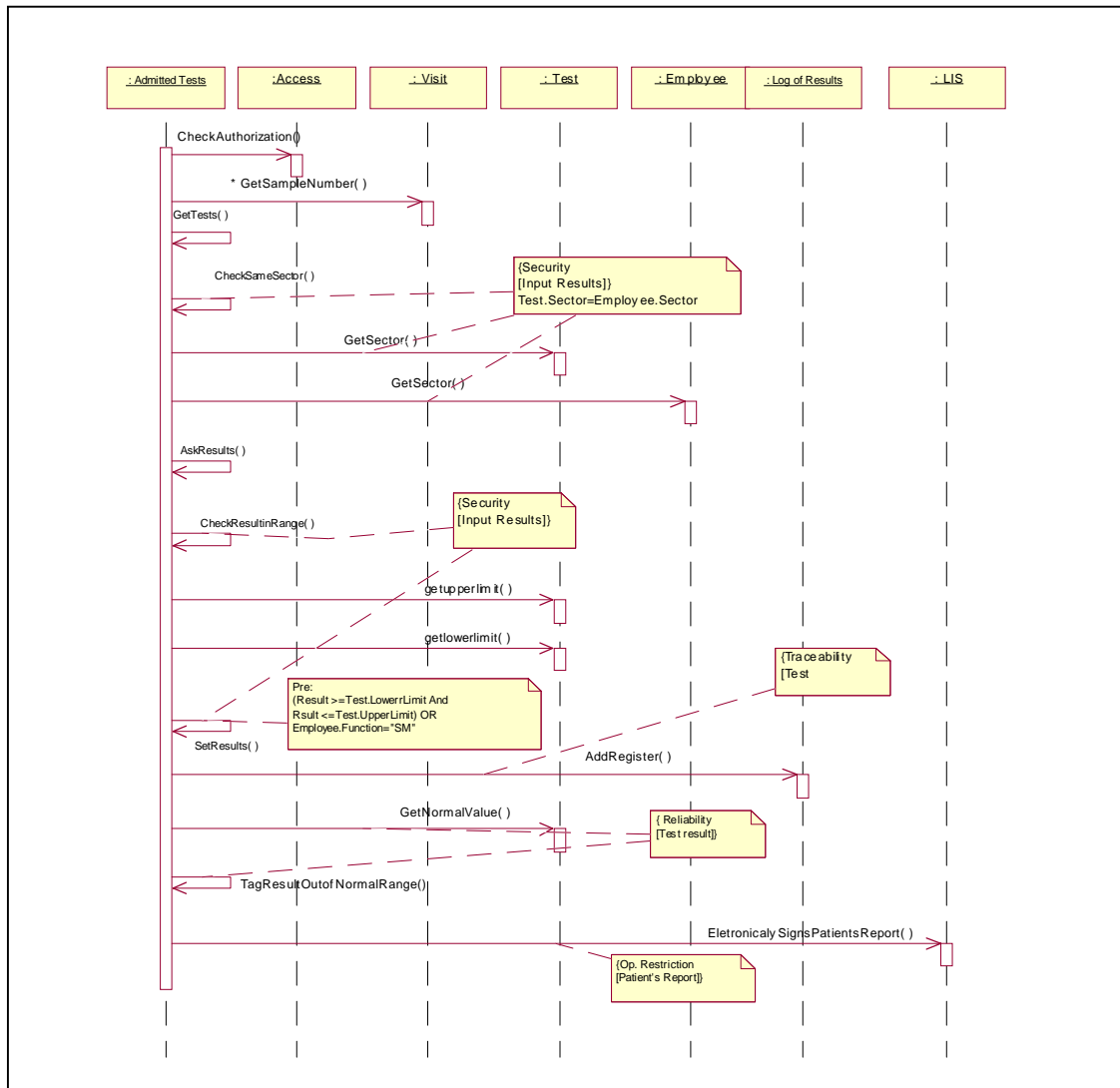
Ao observarmos as operações da classe Exames Admitidos (Figura 8.49), verificamos inicialmente que uma operação, “CheckAuthorization”, já era satisfeita na classe e no diagrama de seqüência mostrado na Figura 8.48 que foi gerado pelo time da empresa, não demandando assim nenhuma alteração.



**Figura 8.49 – Classe Exames Admitidos com RNFs**

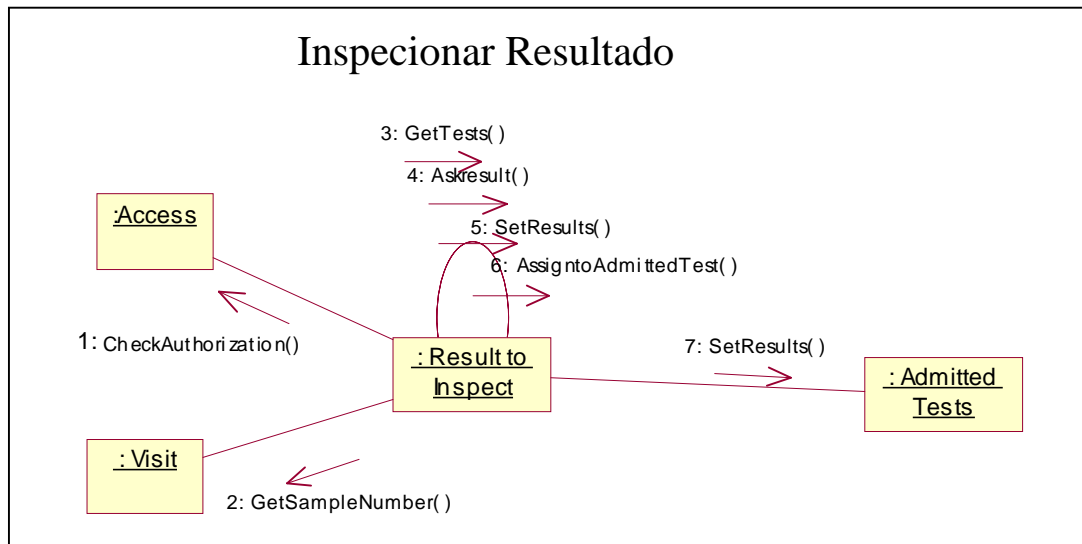
Entretanto, observamos que operações como “CheckSameSector”, entre outras, não eram satisfeitas. Uma vez que essas operações diziam respeito à entrada de resultados de exames, o diagrama de seqüência “entrar resultados” deveria passar a satisfazer essa necessidade. Para a realização dessa operação verificamos que seria necessária a execução da operação “GetSector” da classe Exame, e a operação “GetSector” da classe Empregado, uma vez que a condição de satisfação a ser cumprida era de que o setor do empregado entrando resultados teria de ser o mesmo setor de realização do exame. O comentário associado a essas operações demonstra justamente o fato de elas estarem ali para satisfazer o RNF segurança de entrada de resultados e a condição de teste necessária.

Podemos ainda observar na Figura 8.50 que em consonância com o que é definido na classe exames admitidos, Figura 8.49, a operação “SetResult” só irá ser realizada se o resultado entrado for menor que o valor máximo e maior que o valor mínimo ou se o empregado que esteja entrando o resultado seja um gerente de setor, função “SM”.

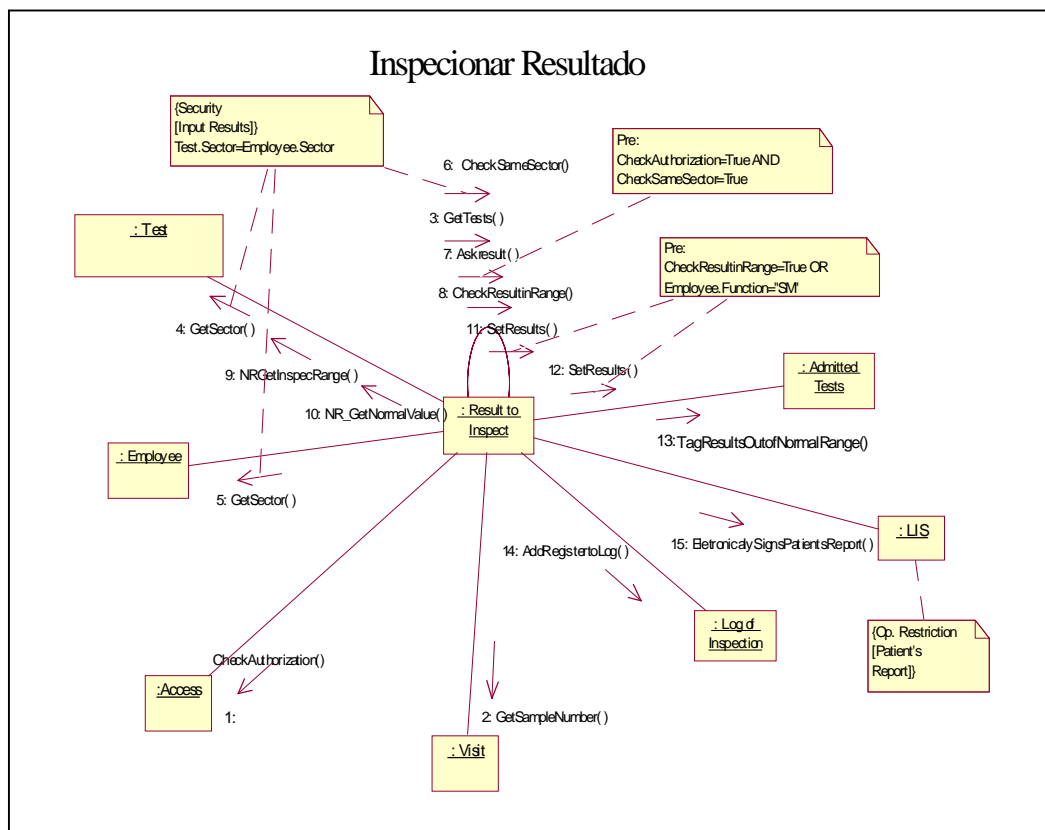


**Figura 8.50 – Diagrama de Seqüência para Entrada de Resultados com RNFs**

Ao avaliarmos a classe resultados a inspecionar, Figura 8.51, encontramos uma situação semelhante à encontrada no exemplo acima. A única diferença é que aqui ao invés de avaliar se o resultado está entre os limites mínimo e máximo do exame, deveremos avaliar se o resultado está dentro do intervalo seguro também definido na classe exame. As figuras 8.52 e 8.53 mostram, respectivamente, os diagramas de colaboração para inspecionar resultados antes e depois dos RNFs serem integrados.



**Figura 8.51- Diagrama de Colaboração Original para Inspeccionar Resultado**



**Figura 8.52 – Diagrama de Colaboração para Inspeccionar Resultados com RNFs**

Podemos ver que aqui também utilizamos comentários ligados às operações introduzidas por RNFs para demonstrar a origem da necessidade dessas operações e as condições a elas associadas.



O Apêndice C mostrar uma lista dos diagramas de sequência e colaboração utilizados durante esse estudo de caso

## 8.4 – Consolidando os Estudos de Caso

Os três estudos de caso realizados apresentaram quantidades expressivas de alterações nos modelos conceituais analisados conforme pode ser visto pela tabela 8.1. Em média, nos três estudos de caso 46% das classes existentes sofreram algum tipo de alteração para satisfazer RNFs, sendo que os modelos resultantes apresentaram um número 45% maior de operações e um montante de 37% a mais de atributos, denotando que se a estratégia tivesse sido utilizada durante o desenvolvimento dos softwares avaliados, poderia ter levado a um software com uma maior qualidade intrínseca, pelo conceito de completeza de um software, bem como possivelmente a uma menor demanda por alterações no software tanto na fase de aceitação do sistema quanto depois do mesmo implantado.

É importante ressaltar que num projeto típico de software, RNFs só começam a ser detectados como necessários durante os testes de aceitação pelo usuário, ou mais fortemente ainda, depois de terem sido entregues, uma vez que somente nessa hora o sistema será realmente exercitado com dados reais em fluxos reais [Cysneiros 01]. Portanto, o custo de corrigir RNFs que não tenham sido elicitados ou tenham sido tratados de maneira inadequada tende a ser bastante elevado.

	Classes Existentes	Classes Afetadas	%	Operações Existentes	Operações Achadas	%	Atributos Existentes	Atributos Achados	%
<b>Estudo de Caso I</b>	8	6	75	105	50	48	22	10	45
<b>Estudo de Caso II</b>	15	4	27	115	45	39	32	8	25
<b>Estudo de Caso III</b>	39	19	48	213	98	46	105	41	39

**Tabela 8.1 – Resumo dos três Estudos de Caso**

Com o objetivo de contrapor os benefícios advindos do uso da estratégia com os eventuais acréscimos no custo do desenvolvimento do software advindos do uso da mesma, medimos a quantidade de tempo gasta com a produção da visão não funcional e sua integração a visão funcional, bem como a quantidade de tempo gasto para a produção da visão funcional de forma a podermos verificar o overhead causado pelo uso da estratégia proposta (Percentagem da não funcional em relação a funcional). Durante o terceiro estudo de caso a produção da visão funcional por parte da equipe da empresa consumiu 1728 horas/homem de trabalho, enquanto a construção da visão não funcional e sua integração aos modelos conceituais da visão funcional consumiram 121 horas/homem, ou seja, aproximadamente 7% do tempo como overhead provocado pelo uso da estratégia.

No estudo de caso I a produção da visão não funcional e sua integração a visão funcional consumiram 45 horas/homem, enquanto no estudo de caso II apenas a integração das duas visões consumiu 21 horas/homem. O estudo de caso II apresenta apenas o tempo de integração devido a termos utilizado nesse estudo de caso os modelos da visão não funcional gerados no estudo de caso I. Assim procedemos por terem ambos os estudos de caso utilizado a mesma fonte de conhecimento.

Infelizmente os tempos gastos na produção das visões funcionais desses dois estudos de caso não se encontram disponíveis, ficando impossível de precisar um overhead para esses dois estudos de caso. Entretanto considerando-se o tempo gasto com a visão não funcional e sua integração a visão funcional e ainda a diferença de complexidade entre esses dois estudos de caso e o estudo de caso III acreditamos que o overhead teria se situado próximo ao verificado nesse último.

Enfatizamos ainda que um overhead de 7%, apesar de obtido com apenas um estudo de caso, é bem próximo do encontrado durante um estudo de caso realizado

anteriormente, 10% [Cysneiros 01] no qual utilizamos um embrião da estratégia aqui proposta. A diferença de valores pode ser atribuída tanto a imprecisão do processo de medição quanto a possíveis ganhos de eficiência decorrentes do aperfeiçoamento da estratégia.

## 9 – Conclusão

Nos últimos anos temos observado um substancial desenvolvimento da engenharia de requisitos. Como parte desse desenvolvimento foi mostrado que uma correta elicitação de requisitos é vital para obtermos softwares de qualidade. Dentro do processo de elicitação de requisitos, tem sido dada uma grande atenção fundamentalmente para os requisitos funcionais. Atualmente a engenharia de requisitos apresenta várias propostas para consistentemente lidar com os requisitos funcionais de um software, enquanto os requisitos não funcionais têm recebido pouca atenção. Entretanto, alguns autores vêm destacando os RNFs como sendo de crucial importância para a obtenção de softwares que atendam as expectativas dos clientes [Chung 93][Boehm 96] [Dardenne 93][Cysneiros 99].

Igualmente importante é estabelecer como os requisitos elicitados irão guiar o resto do processo de desenvolvimento de software. Grande parte dos aspectos de qualidade do software é estabelecida durante o desenho do software[Yu 2000]. Enorme parte dos aspectos de qualidade de um sistema é expressa por RNFs que, muitas vezes, são também denominados “atributos de qualidade de um software” [Boehm, 1978] [Bowen, 1985]. Da mesma forma que os requisitos funcionais os RNFs estão em constante evolução e é importante não apenas sermos capazes de detectar essa evolução, mas também de avaliar suas consequências sobre o software em questão.

Poucos trabalhos tentam mostrar como levar os RNFs até o desenho do software. Cysneiros [Cysneiros 97] apresenta uma proposta para levar os RNFs ao modelo Entidade Relacionamento (MER) com o apoio do uso do framework de RNFs proposto por Chung [Chung 93]. Essa proposta, entretanto era pouco sistematizada e

possuía problemas na sua integração ao MER, tendo sido melhorada em [Cysneiros 01]. Em relação à proposta apresentada em [Cysneiros 01], uma mudança importante ocorreu na representação dos RNF nos modelos conceituais que foi alterada para facilitar a rastreabilidade para a visão não funcional, principalmente nos casos em que uma classe tinha impactos de mais de um RNF. Introduzimos também a integração aos cenários, diagramas de sequência e colaboração que não estavam presentes anteriormente. Neto [Neto 00] propõe uma estratégia para representar os RNFs no LAL em toda sua extensão e, utilizando-se das heurísticas de derivação propostas por Hadad [Hadad 97] e Leonardi [Leonardi 97], chegar até um modelo de classes que contemple os RNFs. Entretanto, a proposta de Neto não favorece a manipulação de RNFs durante a elicitação de requisitos, principalmente no aspecto de conflitos, e é dependente da utilização do processo de desenvolvimento utilizado na proposta.

Mostramos durante a tese uma estratégia que estabelece como se elicitar e tratar RNFs desde o início do processo de desenvolvimento de software. Propomos ainda um processo sistematizado de integração destes RNFs ao desenho do sistema, independentemente do método utilizado no desenvolvimento do software.

É importante ressaltar que o uso do LAL como ancora não impede o uso da estratégia aqui proposta em softwares que tenham sido construídos com outros tipos de métodos para desenvolvimento de software. Nesse caso deveremos gerar o LAL e adaptar os modelos conceituais às restrições que utilizam o LAL. Nos três estudos de caso que fizemos procedemos dessa maneira e o número de classes que já utilizavam símbolos do LAL foi da ordem de 97%, indicando que o uso de nomes que serão símbolos do LAL parece ser um procedimento natural mesmo quando se utiliza outros métodos de desenvolvimento.

Dentro dessa estratégia apresentamos uma proposta de como produzir o que chamamos de visão não funcional do software, a qual representa o ciclo evolutivos dos RNFs apresentando heurísticas para a elicitação dos RNFs, sua representação em modelos não funcionais e avaliação de possíveis interdependências tanto positivas quanto negativas entre RNFs. Mostramos então como esses RNFs irão restringir o modelo conceitual do software de forma a obtermos ao final um modelo conceitual mais completo que expresse não apenas os requisitos funcionais, mas também os não funcionais.

Para validar essa estratégia elaboramos três estudos de caso baseados no uso da replicação de projetos proposta por Basili [Basili 86]. Com os estudos de caso pretendemos ser capazes de avaliar se a utilização da estratégia proposta nessa tese teria como resultado modelos conceituais mais completos e que melhor espelhassem as necessidades do cliente.

Durante estes estudos de caso, avaliamos três diferentes modelos conceituais gerados por times independentes utilizando métodos de desenvolvimento distintos. Fizemos isto contrapondo o modelo obtido por estas outras equipes, que, de uma maneira geral, espelhavam o que chamamos de visão funcional do software, com a visão não funcional por nós gerada. Desta forma, aplicando as heurísticas de integração, fomos capazes de determinar diversas classes, operações e atributos que não estavam presentes nos modelos originais e que deveriam ter sido implementadas de forma a satisfazer as necessidades não funcionais do software. A tabela 9.1 mostra uma síntese do que foi encontrado nestes estudos de caso.

Estes dados corroboram dados anteriormente obtidos durante outros estudos de caso com RNFs que foram realizados pelo autor [Cysneiros 99] [Cysneiros 01]. Esses dados são sumarizados na tabela 9.2. Podemos ver nessa tabela resultados obtidos de

um estudo de caso anterior que mostra três diferentes times onde apenas o time da área de processamento utilizou uma estratégia para lidar com RNFs.

Na tabela 9.2 é mostrado ainda, o número de alterações provocadas no software para atender a satisfação de RNFs em dois momentos distintos. O primeiro momento refere-se às mudanças realizadas no software antes da entrega do software, o que inclui basicamente alterações realizadas durante as fases de testes e de aceitação pelo cliente. O segundo momento diz respeito a alterações realizadas no software para satisfazer RNFs reclamados pelos clientes nos seis primeiros meses após a efetiva colocação do software em produção.

Podemos ver, pelos dados apresentados na tabela 9.2, que o uso de uma estratégia que considerava RNFs levou-nos, nesse estudo de caso, não apenas a um menor número de alterações no software para satisfazer RNFs como, também, como era de se esperar, a um menor tempo para fazê-lo. É possível ver que a diferença entre as equipes é acentuada depois da entrada do software em operação, o que pode ser explicado pelo fato de ser nesse momento que os clientes efetivamente exercitam o sistema e, em consequência, se apercebem de que o software não apresenta comportamento adequado às suas necessidades.

	Classes Existentes	Classes Afetadas	%	Operações Existentes	Operações Achadas	%	Atributos Existentes	Atributos Achados	%
<b>Estudo de Caso I</b>	8	6	75	105	50	48	22	10	45
<b>Estudo de Caso II</b>	15	4	27	115	45	39	32	8	25
<b>Estudo de Caso III</b>	39	19	48	213	98	46	105	41	39

**Tabela 9.1 – Síntese dos Resultados dos Estudos de Caso**

Podemos observar na tabela 9.1 que o montante de alterações foi significativo em todos os três estudos de caso realizados nessa tese. Em média, nos três estudos de caso, 46% das classes existentes sofreram algum tipo de alteração para satisfazer

RNFs, sendo que, em média, os modelos resultantes apresentaram um número 45% maior de operações e um montante de 37% a mais de atributos, denotando que a estratégia, se aplicada desde o início do processo de desenvolvimento de software, poderia ter levado a um software com uma maior qualidade intrínseca, sob a ótica da completeza do software, bem como possivelmente a uma menor demanda por alterações no software, tanto na fase de aceitação do sistema quanto depois do mesmo implementado.

<b>Time</b>	Alterações antes da entrega do software	Tempo gasto para implementar as alterações	Alterações após a entrega do software	Tempo gasto para implementar as alterações
<b>Área de Processamento</b>	2	84 h.	1	56 h.
<b>Área de Atendimento</b>	4	212 h.	6	632 h.
<b>Área Administrativa</b>	3	150 h.	10	822 h.

**Tabela 9.2 – Alterações Devido a RNFs**

Esses dados são condizentes com os mostrados na Tabela 9.2, sugerindo que a utilização da estratégia proporcionaria obtermos um software mais completo com os RNFs já incluídos no software desde seus primeiros desenhos e com conseqüente economia de custos, por minimizar o número de alterações necessárias de se realizar no software tanto na fase de aceitação do software quanto durante a evolução do mesmo.

Outro ponto interessante de se observar na tabela 9.1, é que em todos os casos, o percentual de operações incluídas foi superior ao de atributos incluídos. Isto já era esperado devido a dois fatores. Primeiramente, para cada atributo incluído, freqüentemente temos que incluir pelo menos uma operação para lidar com ele de forma a não ferir o encapsulamento de dados. Segundo, porque muitos dos RNfs se referem a aspectos puramente dinâmicos de como o sistema deve se comportar, refletindo, assim, na necessidade da existência de operações que implementem esses aspectos.



Conforme mencionado na seção 8.4, o overhead estimado provocado pelo uso desta estratégia situa-se na casa dos 7%, e embora tendo sido medido em apenas um dos estudos de caso, é bastante próximo ao overhead, 10%, verificado durante um estudo de caso anterior que resultou nos dados apresentados na Tabela 9.2 [Cysneiros 01]. Apesar de não podermos quantificar em termos de horas/homem o que teria sido gasto para corrigir as alterações que iriam advir do não tratamento dos RNFs, acreditamos que um overhead de 7%, se confrontado com o número médio de alterações encontradas nos estudos de caso, será compensado com folga tanto pelo custo economizado em não necessitarmos realizar essas alterações, quanto pela maior satisfação do cliente com a qualidade do software. Note-se que na Tabela 9.2 o tempo médio para a correção de um erro por não conformidade do software com um RNF foi de 49 horas. Se considerarmos que em média eram usados dois profissionais para atuar em cada uma dessas alterações temos então que cada alteração custava em torno de 25 horas/homem. Portanto, se durante o estudo de caso III tivermos evitado a ocorrência de pelo menos 5 mudanças no software as 121 horas gastas para a aplicação da estratégia já terão sido compensadas. Como 19 das 39 classes foram de alguma forma afetadas pela inclusão de atributos ou operações advindas da integração dos RNFs, parece seguro afirmar que o uso da estratégia foi largamente compensador.

É importante ressaltar que reconquistar a confiança de um cliente em um software que lhe é apresentado com erros é sempre uma tarefa difícil e cara, quando não impossível.

## **9.1 - Contribuições**

A principal contribuição dessa tese aborda aspectos pouco tratados na literatura existente e reside na proposição de uma estratégia para lidar com requisitos não funcionais desde as primeiras etapas do processo de desenvolvimento de software,

que possibilite não apenas a elicitação dos RNFs, mas também a plena integração destes aos modelos conceituais que representam o software. Acreditamos que, procedendo dessa maneira possamos obter softwares que melhor atendam as expectativas de qualidade do cliente e do desenvolvedor e ainda obter menores custos de desenvolvimento.

Poucos trabalhos propõem um tratamento explícito para RNFs sob a ótica do processo de desenvolvimento de software. Dentre os trabalhos existentes, destacamos os seguintes:

- Boehm [Boehm 96] monta uma base de conhecimento com RNFs priorizados pela visão do cliente, mas que entretanto só trata RNFs primários, ficando assim num nível de abstração ainda muito alto para identificação de conflitos entre RNFs ou com requisitos funcionais.
- Kirner [Kirner 96] descreve as propriedades de 6 RNFs no domínio de sistemas de tempo real, performance, confiabilidade, seguro, segurança, manutenibilidade e usabilidade. O trabalho de Kirner descreve como medir estes RNFs e propõe algumas heurísticas de como tratar com estes requisitos durante o desenvolvimento deste tipo de software.
- O Framework proposto por Chung [Chung 93] [Chung 95] [Chung 00] é a abordagem mais completa até o momento e procura tratar de todos os tipos de requisitos não funcionais desde as primeiras etapas do processo de desenvolvimento de software. Apesar de ser um framework bastante completo e eficaz para lidar com requisitos não funcionais durante a elicitação de requisitos, este framework não favorece a integração destes requisitos nas etapas seguintes do desenvolvimento de software, ficando, assim, uma lacuna aberta para que

conflitos entre requisitos funcionais e não funcionais passem despercebidos durante o projeto do software.

- Neto [Neto 00] propõe uma estratégia apoiada pela ferramenta OORNF para o desenvolvimento de software, baseada no léxico ampliado da linguagem estendido para tratar RNFs. A partir do léxico, são gerados os cenários, cartões CRC e um modelo conceitual orientado a objetos. Apesar desta proposta ter apresentado bons resultados nos estudos de casos controlados a que foi submetida, por tratar de RNFs apenas no LAL, ela não favorece o tratamento de interdependências entre RNFs. Outro problema é que o tratamento de RNFs até o desenho do software só é efetivo se utilizarmos a estratégia de derivação de cenários proposta por Hadad [Hadad 97] e a estratégia de derivação de cartões CRCs e diagramas de classes proposta por Leonardi [Leonardi 97].

Essa tese visa a ajudar no preenchimento de algumas das lacunas deixadas por esses trabalhos, propondo uma estratégia que enxerga o processo de desenvolvimento de software como sendo composto de duas visões independentes, uma que irá espelhar os aspectos funcionais do software e outra que irá espelhar os aspectos não funcionais.

A proposição da construção da visão não funcional apresentada nessa tese visa a demonstrar um processo detalhado de elicitação e representação de RNFs.

Como auxiliar no objetivo de facilitar a elicitação de RNFs, procuramos dar suporte ao aspecto evolutivo dos RNFs estendendo o LAL, para que este possa demonstrar a ligação entre noções e impactos, que são derivados da necessidade de satisfazer um RNF, e o RNF que originou estas noções e impactos. A ferramenta OORNF foi alterada de forma a refletir estas alterações, através da inclusão de mecanismos que possibilitam conectar uma noção/impacto de um símbolo que tenha

sido originada para satisfazer um RNF com a noção do símbolo onde este RNF é explicitado.

É necessário ressaltar que esse suporte ferramental é o único disponível para apoiar a estratégia. Nenhum auxílio automatizado é provido para a integração dos RNFs aos modelos conceituais.

Mostramos ainda como elicitar RNFs a partir do uso do LAL e com o uso de uma base de conhecimentos de RNFs, gerando os grafos de RNF que fazem parte da visão não funcional.

Quanto ao aspecto das interdependências entre grafos, definimos nessa tese uma série de heurísticas para realizar-se uma análise sistematizada dos grafos de RNFs de forma a auxiliar na detecção de interdependências, tanto positivas quanto negativas entre RNFs.

Os trabalhos anteriormente citados também não mostram como podemos sistematicamente utilizar nos modelos conceituais que representam o software a ser construído, os RNFs elicitados e representados utilizando-se a proposta desses trabalhos. Apesar de elicitar RNFs ser um grande passo na direção de se produzir softwares que atendam melhor os desejos dos clientes, é necessário que tenhamos uma forma sistemática de trazer esses RNFs para dentro do modelo conceitual que irá guiar a codificação do software. A parte da estratégia apresentada nessa tese que lida com a integração das visões funcionais e não funcionais procura justamente preencher esse vazio existente nos métodos existentes para se lidar com RNFs.

Parte dessa estratégia de integração reside na proposta de uma extensão à UML em seus diagramas de classe, seqüências e colaboração para tratar com requisitos não funcionais. Essa extensão pretende representar nesses diagramas as conseqüências das operacionalizações dos RNFs encontrados na visão não funcional, em conjunto com

uma ligação para os modelos dessa visão que possibilite a rastreabilidade desses RNFs. Essa rastreabilidade é de grande auxílio no tratamento do aspecto evolutivo do software, uma vez que podemos nos reportar às fontes que originaram a inclusão nos modelos conceituais de uma determinada classe, operação, atributo ou mesmo mensagem oriunda da operacionalização de um RNF.

Os estudos de caso realizados sugerem que o uso da estratégia pode resultar em um software que atenda melhor as expectativas gerais do cliente, ao invés de se ter um software apenas funcionalmente correto. Desses resultados podemos ainda inferir que a não detecção da necessidade de diversas operações e atributos durante a geração do modelo conceitual do software, possivelmente teria demandado um razoável número de alterações a serem realizadas, na melhor das hipóteses durante a aceitação do software e na pior após a entrega do mesmo, o que traria um aumento do custo de produção do software em relação ao custo que se teria tido se a estratégia tivesse sido utilizada inicialmente, principalmente se considerarmos que erros detectados nessas fases são os mais caros de se corrigir.

Finalmente, a estratégia proposta mostra-se bastante interessante como um mecanismo de validação de desenhos de software em andamento ou mesmo consolidados, permitindo assim que a avaliação de um projeto utilizando-se a ótica de RNFs seja realizada mesmo com o projeto em andamento, ou mesmo em sistemas legados que precisam evoluir.

## **9.2 - Trabalhos Futuros**

Futuros trabalhos irão abordar a implementação de uma ferramenta que gerencie a comparação de grafos de RNFs também é importante no sentido de facilitar o trabalho de identificação de interdependências e evitar que grafos não sejam comparados

devido a um eventual grande número de grafos com o conseqüente aumento da dificuldade de se acompanhar quais os grafos já foram comparados.

Outra linha de trabalho deverá incluir a ampliação da estratégia para tratar com outros diagramas da UML como Casos de Uso, Diagramas de Estado e Diagrama de Atividade.

Uma terceira linha de pesquisa deverá abordar a influência que os RNFs exercem na atividade de determinar qual a arquitetura de software que deverá ser utilizada, visto que muitos RNFs afetam diretamente o tipo de arquitetura que deverá ser utilizada caso eles venham a ser satisfeitos.

Uma quarta linha de atuação residirá na definição de um processo automatizado para apoiar, mesmo que de forma parcial, o processo de integração dos RNFs aos modelos conceituais, investigando que possíveis técnicas de inteligência artificial poderiam ser utilizadas para que esse processo seja o mais sistematizado e independente da atuação do engenheiro de software possível.

Finalmente, deverão ser realizados novos estudos de casos, onde não haja a participação do autor da tese para verificar o grau de facilidade de aplicação da estratégia por outras pessoas.

## 10 – Bibliografia

- [AI84] *Artificial Intelligence*, An International Journal, Special Volume on Qualitative Reasoning about Physical Systems, Vol. 24, No. 1-3. Dec. 1984.
- [Basili 84] Basili, V.R. and Perricon, B. "*Software Errors and Complexity An Empirical Investigation*" ACM Communication 27 No 1 45-52, 1984.
- [Basili 86] Basili, V.R. Selby, R.W., Hutchens, D.H. "*Experimentation in Software Engineering*" IEEE transactions on Software Engineering Vol. SE-12 No. 7 July 1986 pp733-742.
- [Basili 88] Basili, V.R. and Rombach,H.D. "*The TAME Project : Towards Improvement-Oriented Software Environments*" IEEE Trans. On Soft. Eng. No 6 758-773, 1988.
- [Basili 91] Basili, V.R. and Musa, J. "*The Future Engineering of Software a Management Perspective*", IEEE Computer 24 1991 pp:90-96
- [Basili 92] Basili, V.R. "*Software Modeling and Measurment : The Goal/Question/Metric Paradigm*" University of Maryland Technical Report UMIACS-TR-92-96, 1992
- [Basili 97] Mashiko, Y. And Basili, V.R. "*Using the GQM Paradigm to Investigate Influential Factors for Software Process Improvement*" J. System Software 1997 pp: 17-32.
- [Berry 98] Berry, Daniel e Lawrence, Brian. *Requirements Engineering*. IEEE Software, March/April 1998, pp. 26-29
- [Boehm 76] Boehm, B., Brown, J.R. and Lipow, M. "*Quantitative Evaluation of Software Quality*" in Proc. of 2nd International Conference on Soft. Eng. San Francisco, oct 1976, pp:592-605.
- [Boehm78] Boehm, B. "*Characteristics of Software Quality*" North Holland Press, 1978.

- [Boehm 84] B.W. Boehm, “*Verifying and validating software requirements and design specification.*” IEEE software Vol 1 pp-94-103, Jan 1984.
- [Boehm96] Boehm, Barry e In, Hoh. “*Identifying Quality-Requirement Conflict*”s. IEEE Software, March 1996, pp. 25-35
- [Booch 91] Booch, G. “*Object Oriented Design with Applications*” Benjamin Cummings, Redwood City, CA, 1991.
- [Bowen 85] T. P. Bowen *et al*, “Specification of software quality attributes”, Rep. RADC-TR-85-37, Rome Air Development Center, Griffiss Air Force Base, NY, Feb 1985
- [Brackett 90] Brackett, J.W. “*Software Requirements*” SEI-CM-19-1.2, Software Engineering Institute, January 1990.
- [Breitman 98] Breitman, K.K., Leite J.C.S.P, “*A Framework for Scenario Evolution*” Proc. of the Third IEE inter. Conference on Req. Eng. IEEE Computer Society Press, Los Alamitos, CA, 1998, pp:214-221.
- [Breitman 00] Breitman, K.K. “*Evolução de Cenários*” Tese de Doutorado submetida na PUC-Rio em Maio de 2000.
- [Brooks 87] Brooks Jr.,F.P. “*No Silver Bullet: Essences and Accidents of Software Engineering*” IEEE Computer Apr 1987, No 4 pp:10-19, 1987.
- [Carrol 95] Carrol, J. “*Introduction: The Scenario Perspective on System development*” Carrol J(ed.). Scenario-Based design: Envisioning Work and technology in System Development. Wiley, New York, 1995, pp 1-18.
- [Chung 93] Chung L., “*Representing and Using Non-Functional Requirements: A Process Oriented Approach*” Ph.D. Thesis, Dept. of Comp.. Science. University of Toronto, June 1993. Also tech. Rep. DKBS-TR-91-1.
- [Chung 95] Chung, L., Nixon, B. “*Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach*” Proc. 17th Int. Con. on Software Eng. Seattle, Washington, April pp: 24-28, 1995.



- [Chung 00] Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. “*Non-Functional Requirements in Software Engineering*” Kluwer Academic Publishers 1999.
- [Cysneiros97] Cysneiros, L.M. and Leite, J.C.S.P. “*Integrando Requisitos Não Funcionais ao Processo de Desenvolvimento de Software*” – XI Simpósio Brasileiro de Engenharia de Software – Fortaleza – Out/1997
- [Cysneiros 98] Cysneiros, L.M. and Leite, J.C.S.P. “*Requisitos Não funcionais como validadores de modelos de dados*” – 1º Workshop de Engenharia de Requisitos – Maringá – Out/1998
- [Cysneiros 99] Cysneiros, L.M. and Leite, J.C.S.P. “*Integrating Non-Functional Requirements into data model*” 4<sup>th</sup> International Symposium on Requirements Engineering – Ireland June 1999.
- [Cysneiros 99b] Cysneiros, L.M. and Leite, J.C.S.P. “*Integrando Requisitos Não Funcionais na Modelagem Orientada a Objetos*” – Proc. do 2º Workshop Ibero-Americano em Engenharia de Requisitos – **Buenos Aires** – set/1999
- [Cysneiros 01] Cysneiros, L.M., Leite, J.C.S.P. and Neto, J.S.M. “*A Framework for Integrating Non-Functional Requirements into Conceptual Models*” Requirements Engineering Journal – In press, Oct –2000.
- [Dagstuhl 99] Börger E., Gotzhein R. “*Requirements Engineering Case Study “Light Control”*” <http://rn.informatik.uni-kl.de/~recs>.
- [Davis 93] Davis, A. “*Software Requirements: Objects Functions and States*” Prentice Hall, 1993.
- [Dardenne 93] Dardenne, A., van Lamsweerde A, Fickas, S.. “*Goal Directed Requirements Acquisition*”. Science of Computer Programming, Vol. 20 pp: 3-50, Apr. 1993.
- [Dobson 91] Dobson, J.E. “*On Non-Functional Requirements*” PDCS Technical Report Series University of Lancaster No 65 May 1991.
- [EAGLE 95] Evaluation of Natural Language Processing Systems, <http://www.issco.unige.ch/ewg95> 1995.

- [Ebert97] Ebert, C. "*Dealing with Nonfunctional in Large Software System*"s. Annals of Software Engineering, 3, 1997, pp. 367-395.
- [Fenton 97] Fenton, N.E. and Pfleeger, S.L. "*Software Metrics: A Rigorous and Practical Approach*" 2<sup>nd</sup> ed., International Thomson Computer Press, 1997
- [Finkelstein 96] Finkelstein, A. and Dowell J. "A comedy of Errors: The London Ambulance Service Case Study" Proceedings of the Eighth International Workshop on Software Specification and Design, IEEE Computer Society Press pp 2-5 1996.
- [Fowler 97] Fowler, M. and Kendall Scott "*UML Destilled*" Addison-Wesley Inc, 1997
- [Franco 92] Franco, Ana Paula Moreira. "*Métodos e Representação de Suporte à Aquisição de Linguagens de Aplicação*" Dissertação de Mestrado, Departamento de Informática, PUC-Rio, 1992.
- [Gamma 94] Gamma, E., Helm, R., Johnson,R. and Vlissides, J. "*Design Patterns: Elements of reusable Object-Oriented Software*", Reading, MA: Addinson-Wesley, 1994
- [Greenspan 84] Greenspan S., "*Requirements Modelling: A Knowledge representation approach*" Ph.D Thesis, Computer Research Group, University of Toronto, 1984.
- [Hadam 97] Hadad, Graciela et. al. "*Construcción de Escenarios a partir del Léxico Extendido del Language*" JAIIO'97, Buenos Aires, 1997, pp. 65-77.
- [Haddad 99] Hadad, Graciela et. al. "*Enfoque Middle-out en la Construcción e Integración de Escenario*". Anais do Workshop de Engenharia de Requisitos, Buenos Aires, Argentina, 1999, pp. 79-94.
- [Hauser 88] Hauser, J.R. and Hsiao, K. "*The House of Quality*" Harvard Business review, May 1998 pp:63-73.

- [ISO 9126] ISO9126 “*Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use*” International Organization for Standardization, Geneva, 1992.
- [Jacobson 92] Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. “*Object-Oriented Software engineering: A Use Case Driven Approach*” Addison-Wesley, Reading, MA/ACM Press, New York, 1992
- [Jacobson 99] Jacobson, I., Booch, G., Rumbaugh, J. “*The Unified Software Development Process*” Addison-Wesley Inc, 1999.
- [Keller 90] Keller, S.E. et al “*Specifying Software Quality Requirements with Metrics*” in Tutorial System and Software Requirements Engineering IEEE Computer Society Press 1990 pp:145-163
- [Kirner 96] Kirner T.G. , Davis A .M. , “*Nonfunctional Requirements of Real-Time Systems*”, Advances in Computers, Vol 42 pp 1-37 1996.
- [Kotonya 95] Kotonya G. , Sommerville I. , “*Requirements Engineering With Viewpoints*”, Cooperative Systems Engineering Group Technical Report CSEG/10/1995.
- [Kotonya 98] Kotonya, Gerald e Sommerville, Ian.”*Requirements Engineering: Processes and Techniques*” John Willey & Sons, 1998.
- [Lindstrom93] Lindstrom, D.R. “*Five Ways to Destroy a Development Project*” IEEE Software, September 1993, pp. 55-58.
- [Leite 92] Leite, Julio C.S.P. “*Enhancing the Semantics of Requirements Statements*” Proceedings of the XII International Conference of the Sociedad Chilena de Ciencia de la Computacion, Santiago, 1992, pp. 281-297.
- [Leite 93] Leite J.C.S.P. and Franco, A.P.M. “*A Strategy for Conceptual Model Acquisition* ” in Proceedings of the First IEEE International Symposium on Requirements Engineering, SanDiego, Ca, IEEE Computer Society Press, pp 243-246 1993.

- [Leite 94] Leite J.C.S.P. , “*Engenharia de Requisitos- Notas de Aula*”, 1994.
- [Leite 95] Leite J.C.S.P., Oliveira, A.P.A., “*A Client Oriented Requirements Baseline*”, Proc of the 2<sup>nd</sup> IEEE International Conference on Requirements Engineering, 1995.
- [Leite 97] Leite, J.C.S.P. et.al. ” *Enhancing a Requirements Baseline with Scenarios.*” Requirements Engineering Journal, 2(4):184-198, 1997.
- [Leite 00] Leite, J.C.S.P. et al “*A Scenario Construction Process*” Requirements Engineering Journal (2000) 5:38-61, 2000.
- [Leonardi 97] Leonardi, Carmen. et. al. “*Una Estrategia de Análisis Orientada a Objetos basada en Escenarios*” Actas II Jornadas de Ingeniaria de Software JIS97, Donstia, San Sebastian, España, Set. 1997.
- [Lyu 96] Lyu, M.R. (ed.) “*Handbook of Software Reliability Engineering*” McGraw-Hill, 1996.
- [Loucopoulos 95] Loucopoulos, and Karakostas, “*System Requirements Engineering*”, McGraw Hill International Series in Software Engineering, 1995.
- [Mamani 99] Mamani, Nestor A. “*Integrando requisitos não funcionais aos requisitos baseados em ações concertas*”, Dissertação de mestrado, Departamento de informática, PUC-RIO, Maio, 1999.
- [Meyer 85] Meyer, B. “*On Formalism in Specifications*”, IEEE Software, January 1985.
- [Mostow 85] Mostow, J. “*Towards Better Models of Design Process*” AI Magazine, Vol. 6 No. 1 1989, pp:44-57 Spring 85.
- [Musa 87] Musa, J., Lannino, A. and Okumoto, K. “*Software Reliability: Measurment, Prediciton, Application*” New York, McGraw-Hill, 1987

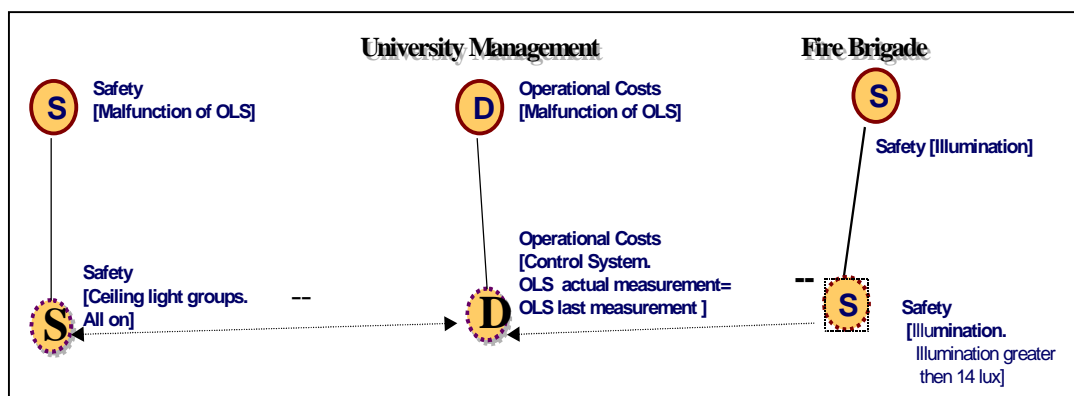
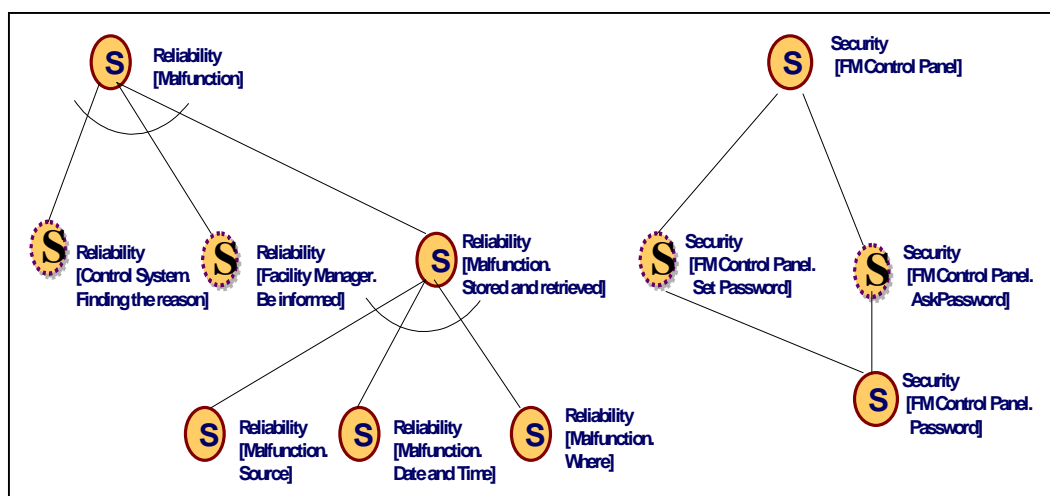
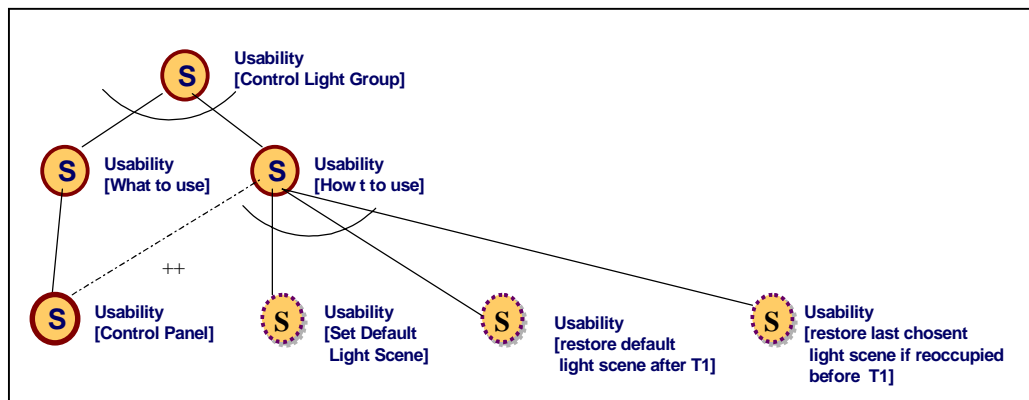
- [Mylopoulos 92] Mylopoulos, J., Chung, L., Yu, E. and Nixon, B., “*Representing and Using Non-functional Requirements: A Process-Oriented Approach*”, IEEE Trans. on Software Eng, 18(6), pp:483-497, June 1992.
- [Mylopoulos 95] Mylopoulos, J. “*Capturing Intentions in Requirements engineering: A modeling Perspective*” Invited Talk at the IFIP WG 2.9 meeting, Bramshill, UK, March 1995.
- [Neto 00] Neto, J.S.M. “*Integrando Requisitos Não Funcionais ao Modelo de Objetos*” Dissertação de Mestrado da PUC-Rio, Mar/2000.
- [Pfleeger 95] Pfleeger, S.L. “*Experimental Design and Analysis in Software Engineering*” ACM SIGSOFT Software Engineering Notes Vol. 20 No. 1 Jan. 1995 pp: 22-25.
- [Potts 94] Potts, Colin et. al. “*Inquiry-based Requirements Analysis*”, IEEE Software, March 1994, pp. 21-32.
- [Rational 97] Rational et al, “*Object Constraint Language Specification*” 1997.  
[Http://www.rational.com](http://www.rational.com)
- [Reubestein 91] Reubenstein, H.B and Waters, R.C. , “*Requirements apprentice - assistance for requirements acquisition*”, IEEE Transaction on Soft. Eng. vol 17 No 3, pp-226-247, 1991.
- [Roman 85] Roma, G. “*A Taxonomy of Current Issues in Requirements Engineering*”, IEEE Computer Vol. 18, No. 4 Apr. 1985, pp:14-23
- [Rumbaugh 99] Rumbaugh, J., Jacobson, I. and Booch, G. “*The Unified Modeling Language Reference Manual*”, Addison-Wesley, 1999.
- [Sommerville 92] Sommerville, I. “*Software Engineering*” fourth edition, Addison-Wesley, 1992.
- [Sommerville 98] Sommerville, I. and Sawyer, P. “*Requirements Engineering – A Good Practice Guide*”. John Wiley & Sons, 1997.

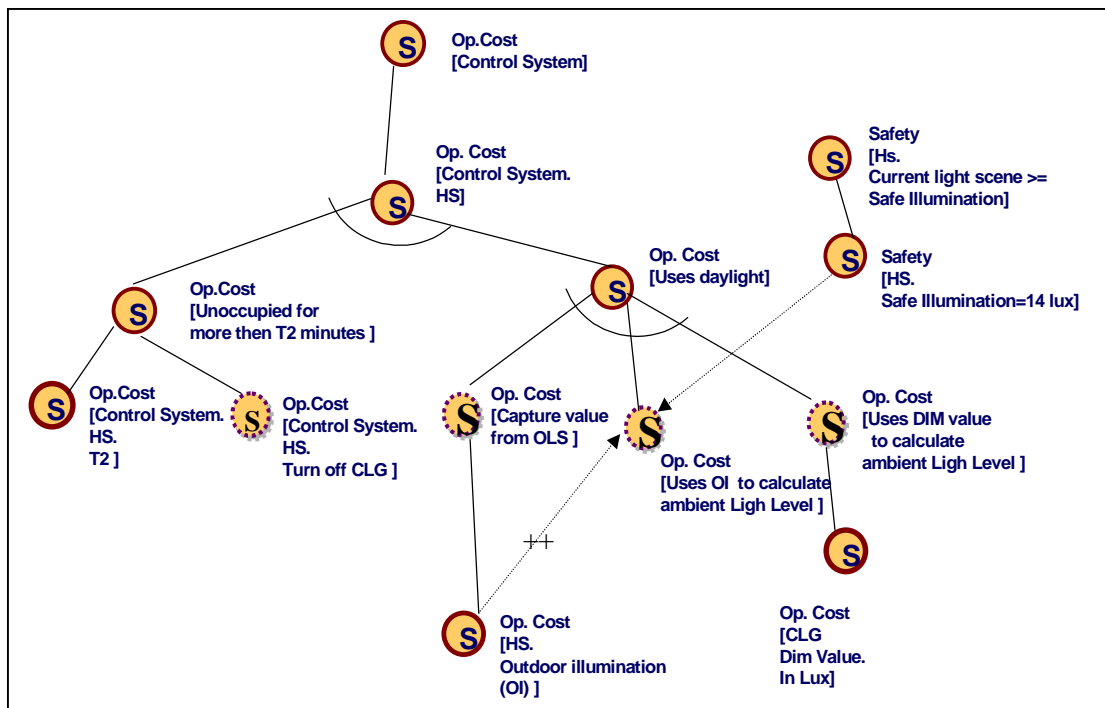
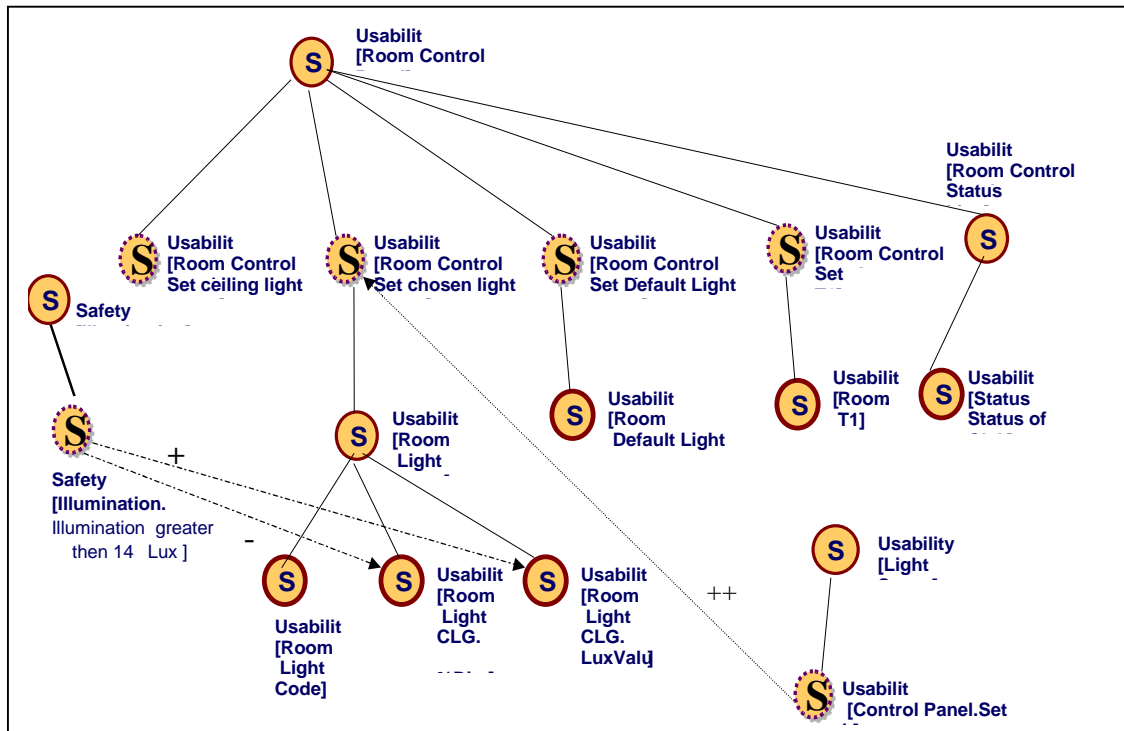
- [Toro 99] Toro, D. et. al. “*A Requirements Elicitation Approach Based in Templates and Patterns*”, Anais do Workshop de Engenharia de Requisitos, Buenos Aires, Argentina, 1999. pp. 17-29.
- [Rumbaugh 99] Rumbaugh, J., Jacobson, I. and Booch, G. “*The Unified Modeling Language Reference Manual*” Addison-Wesley, 1999.
- [Yeh 84] Yeh, R. et. al. “*Software Requirements: New Directions and Perspectives*” Handbook of Software Engineering, 1984, pp. 519-543
- [Yu 00] D. Gross, E. Yu “*From Non-Functional Requirements to Design through Patterns*” Proceedings of the 6th International Workshop on Requirements Engineering: Foundations for Software Quality (June 5-6, 2000), Stockholm, Sweden. submitted version.
- [Zorman 95] Zorman L. “*Requirements Envisaging by Utilizing Scenarios (Rebus)*”. Ph.D. dissertation, University of Southern California, 1995.

## Apêndice A – Estudo de Caso I

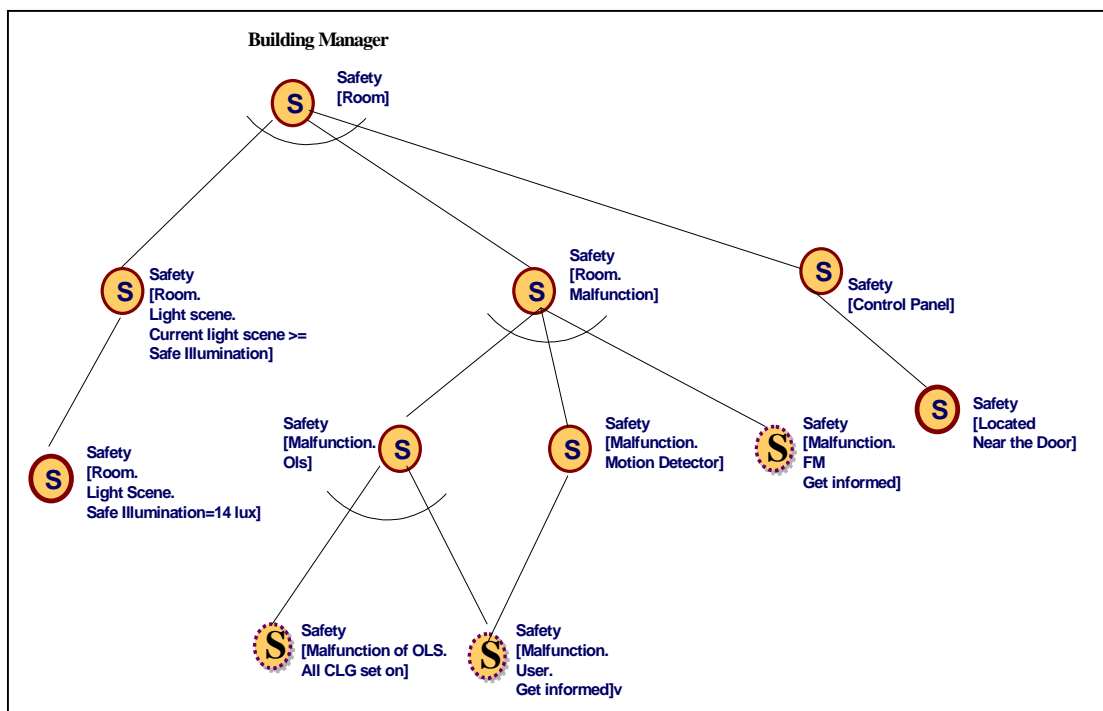
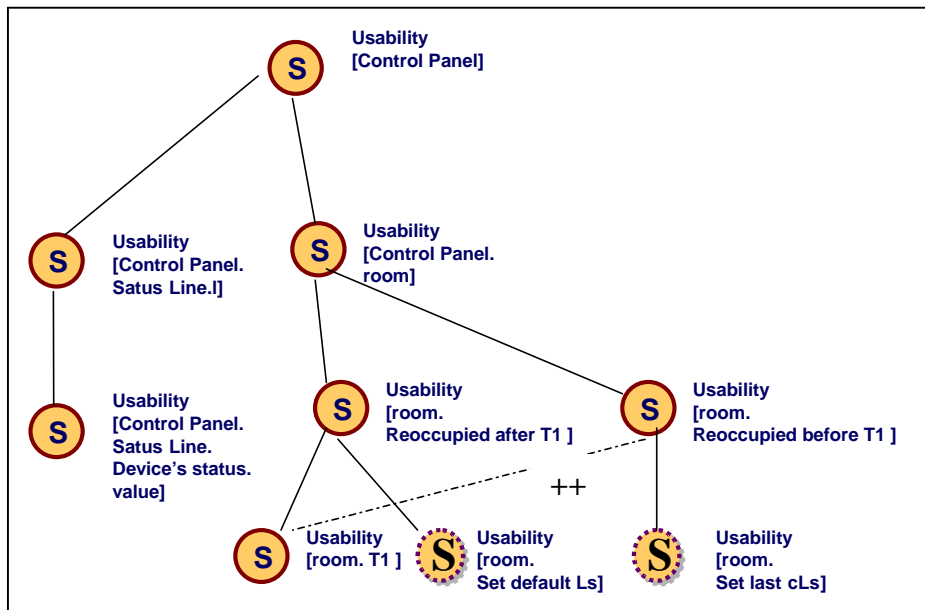
Este apêndice exibe todos os grafos de RNF achados para o estudo de caso I e o detalhamento das classes que sofreram alterações por conta de satisfazer RNFs.

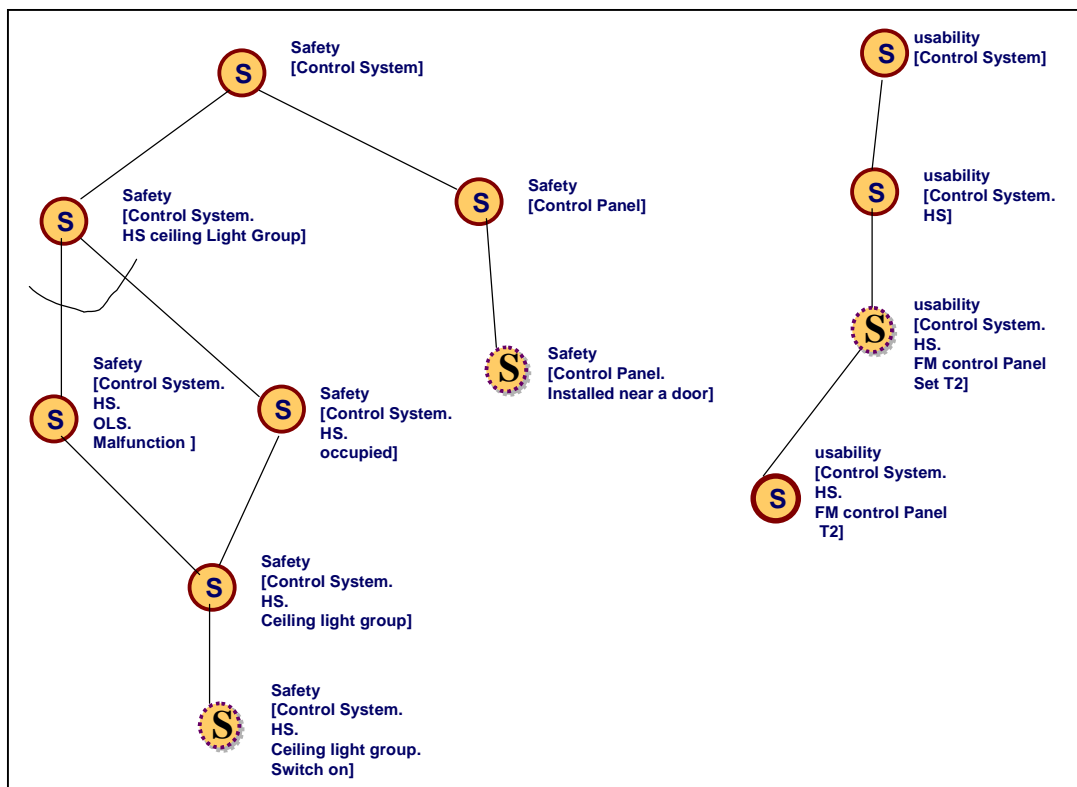
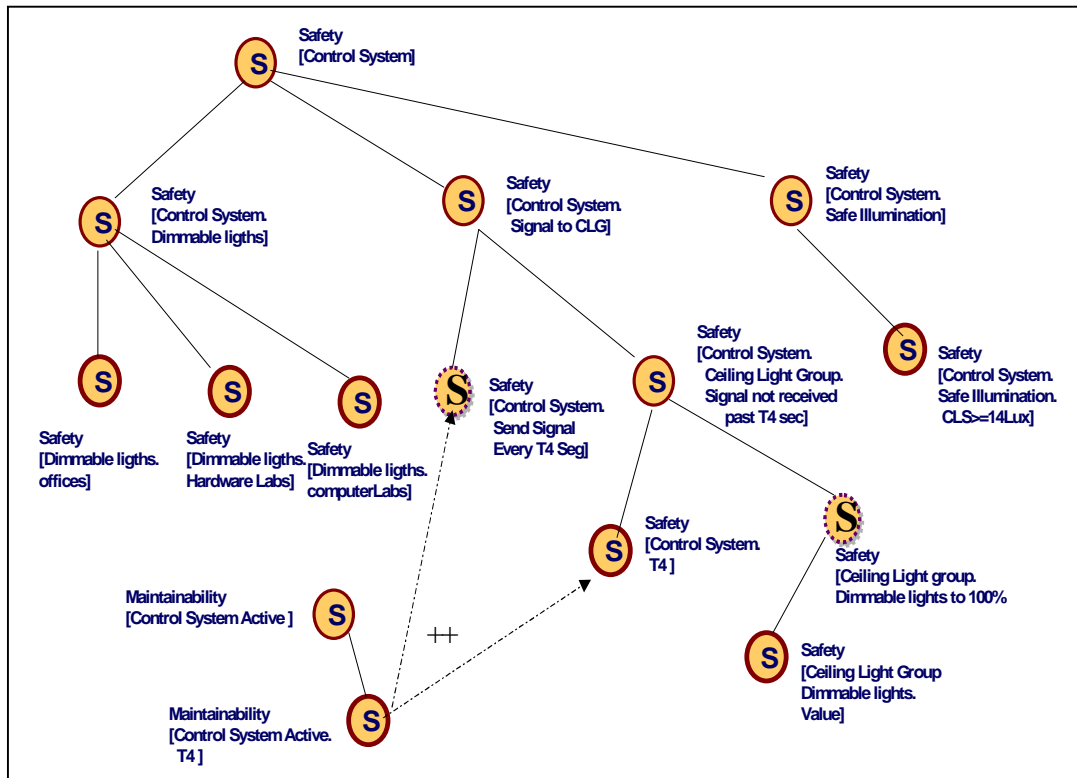
### Grafos de RNF

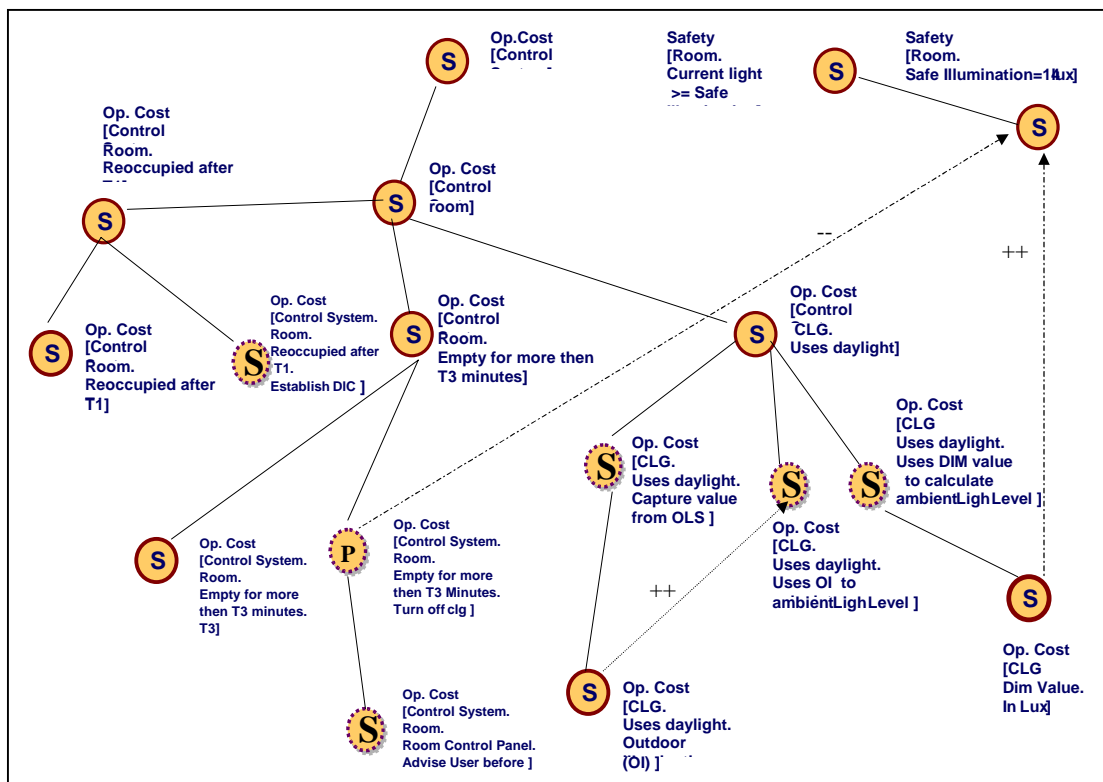
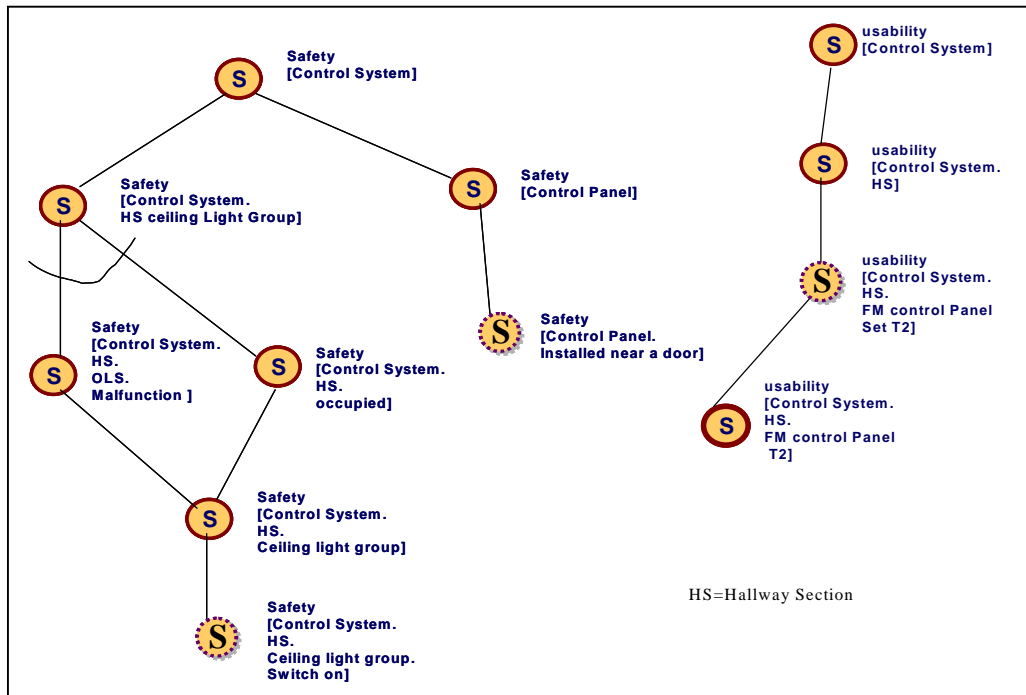


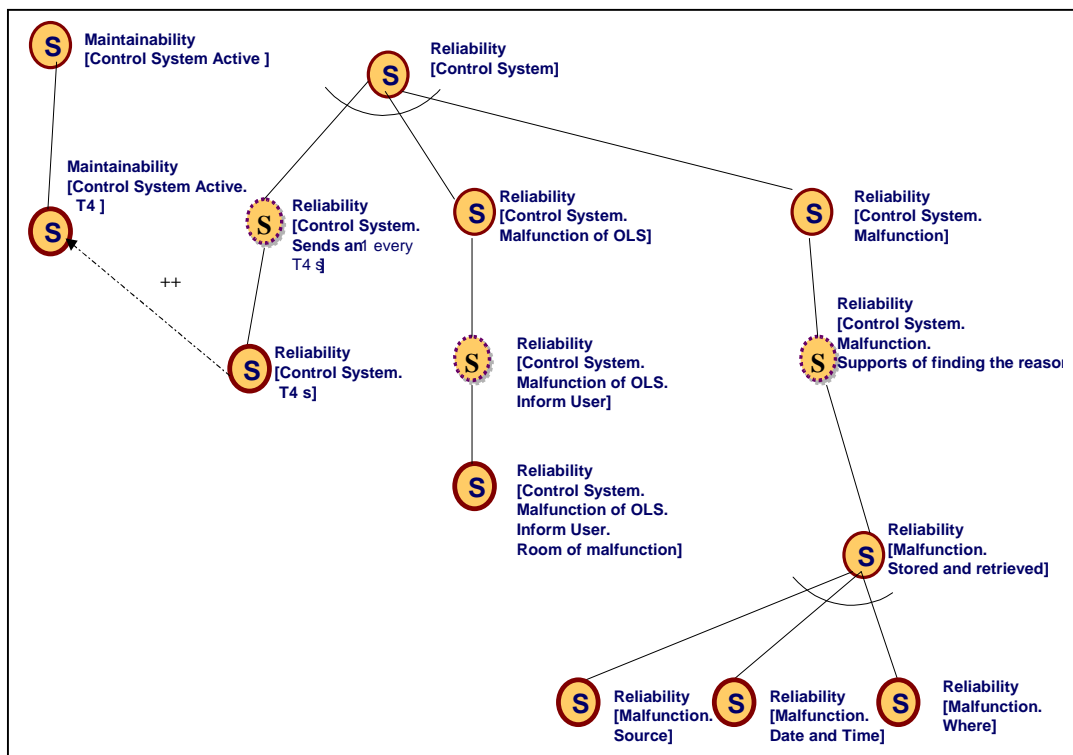
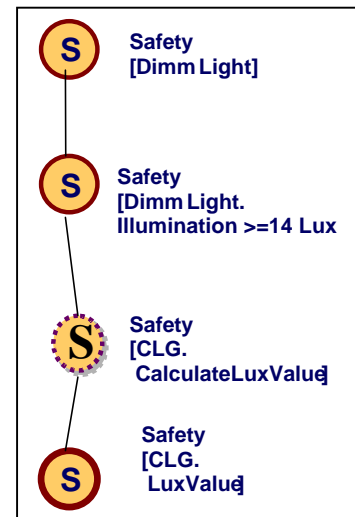


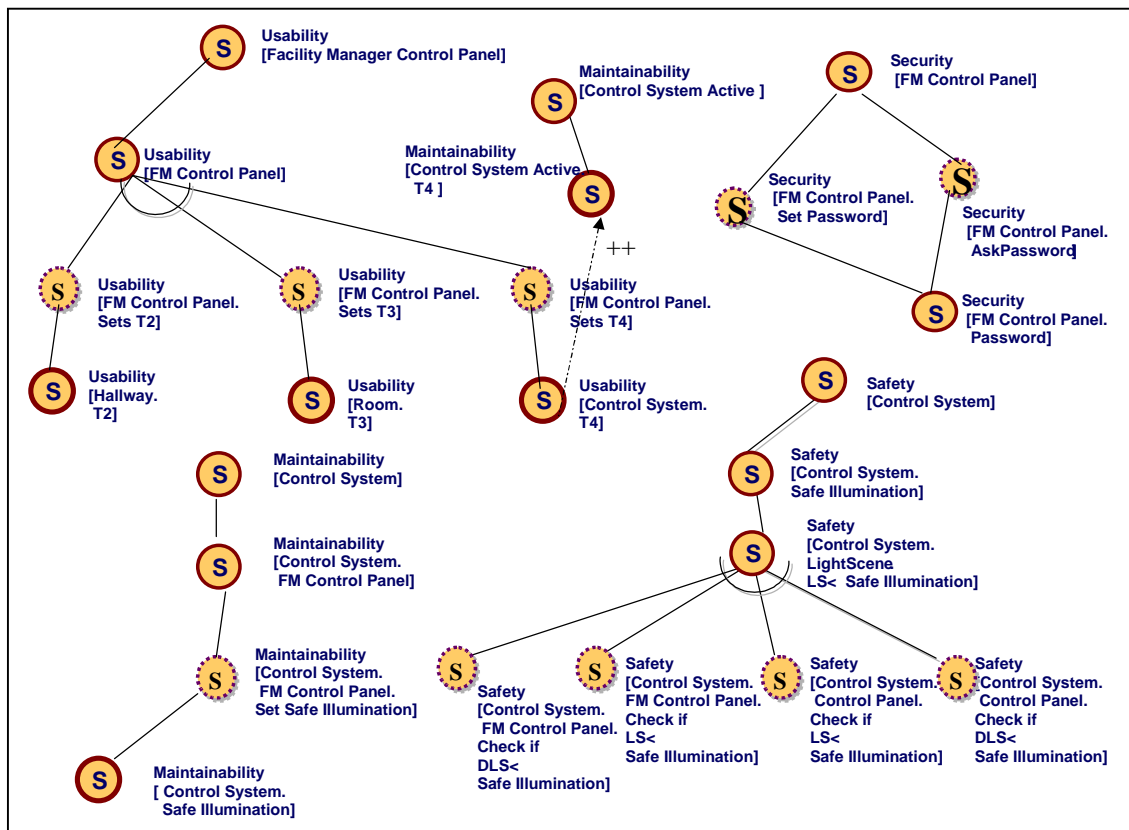




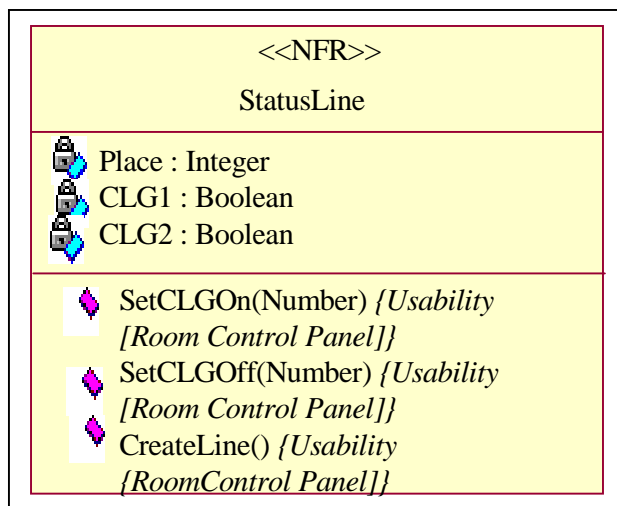
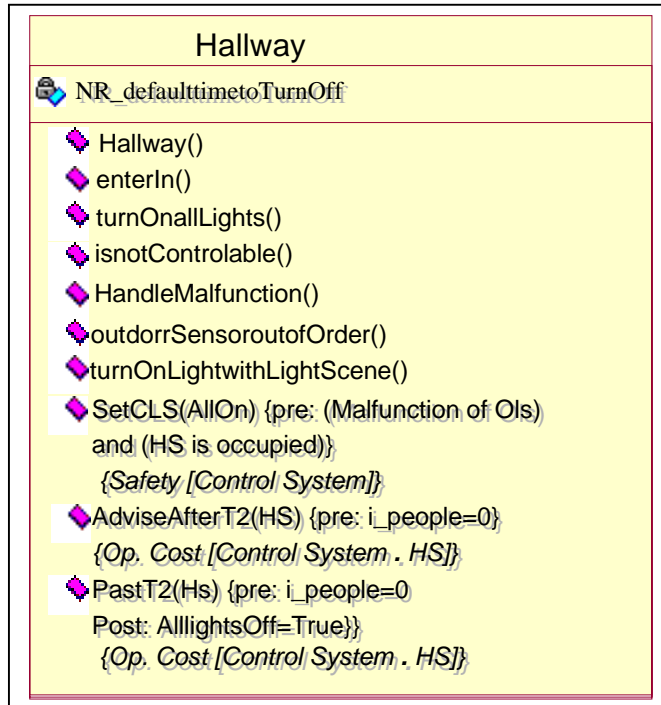


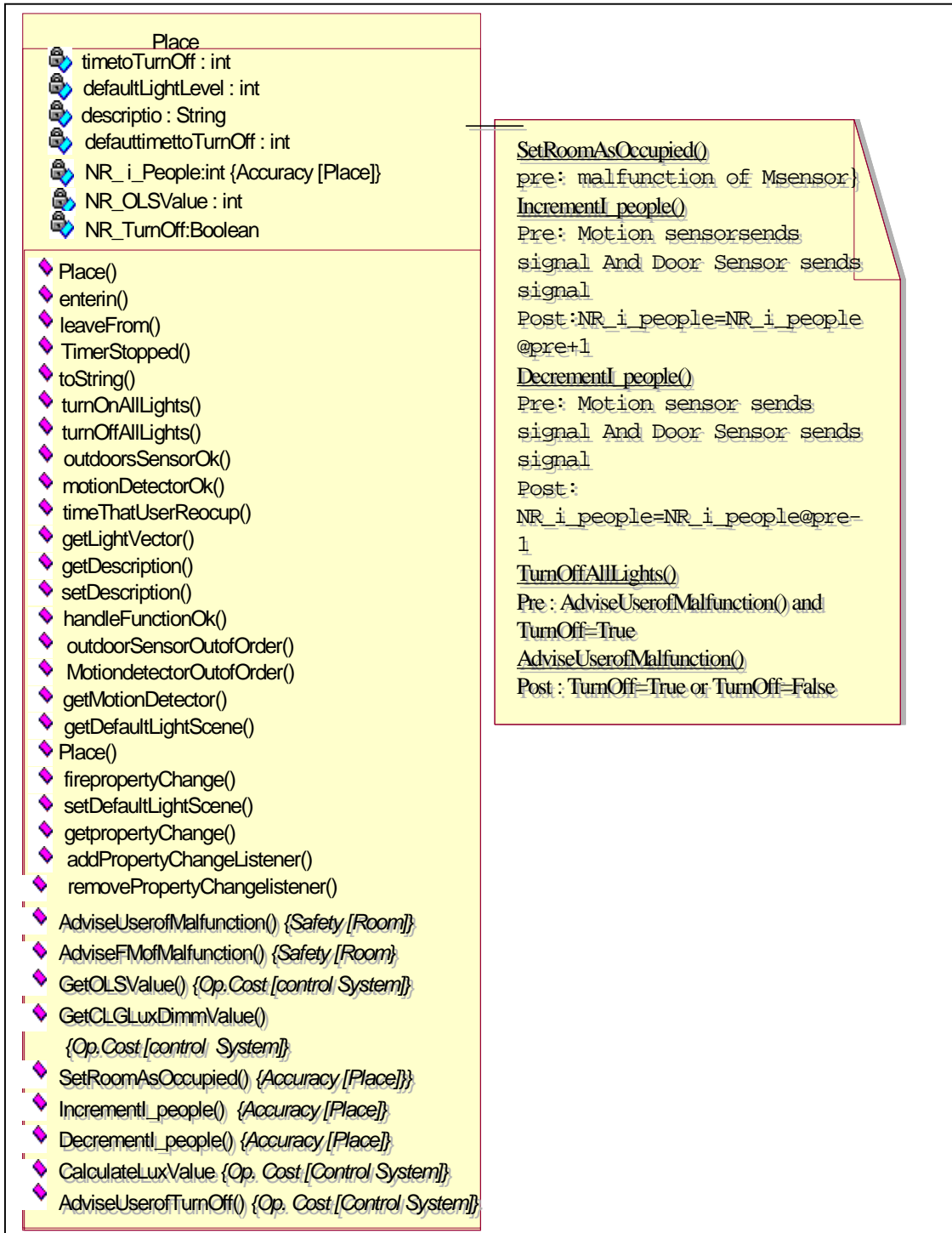


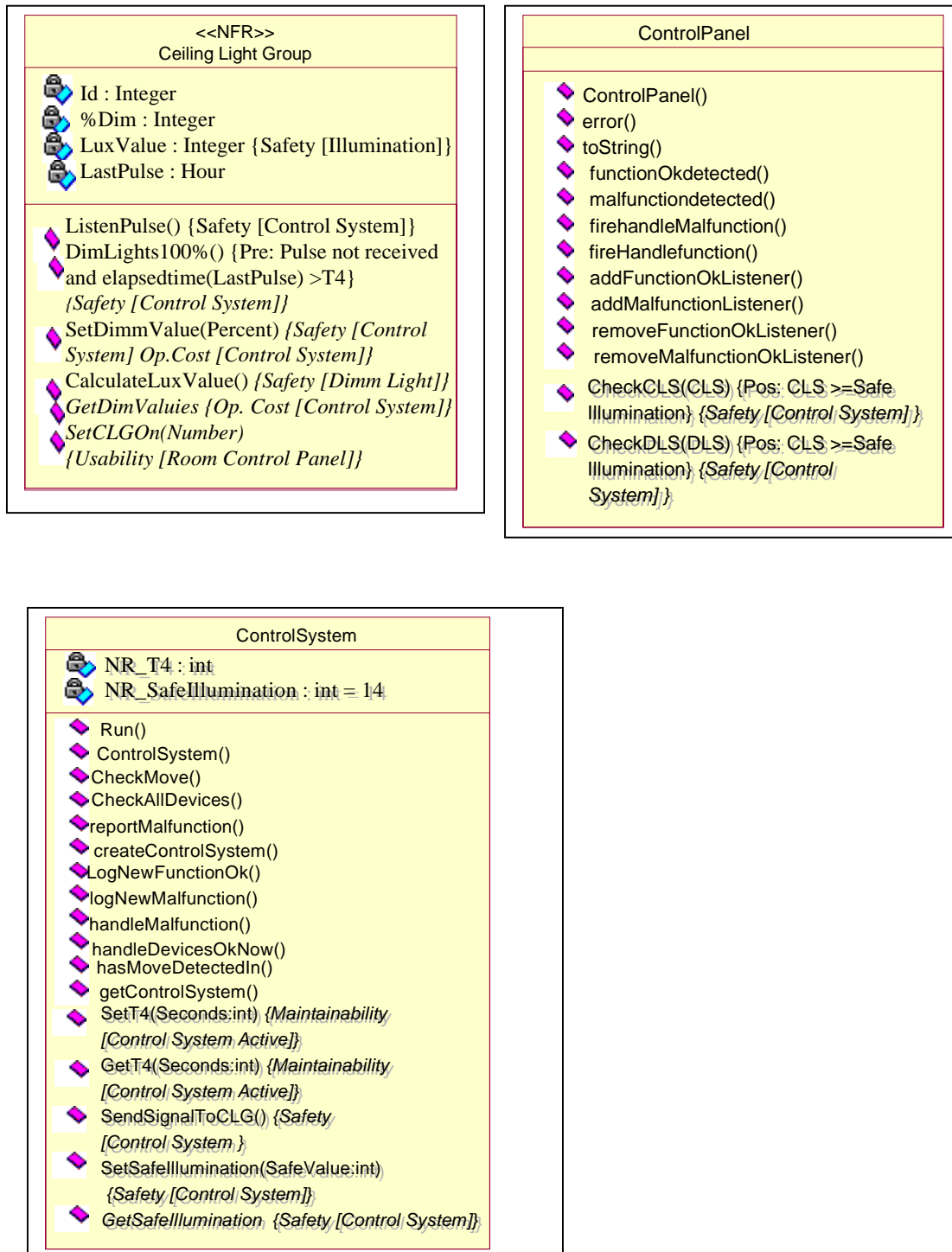




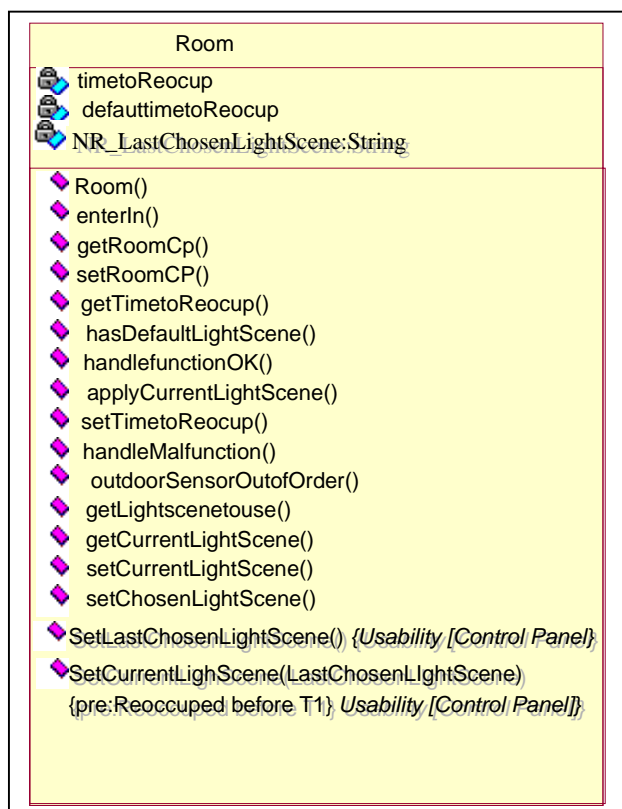
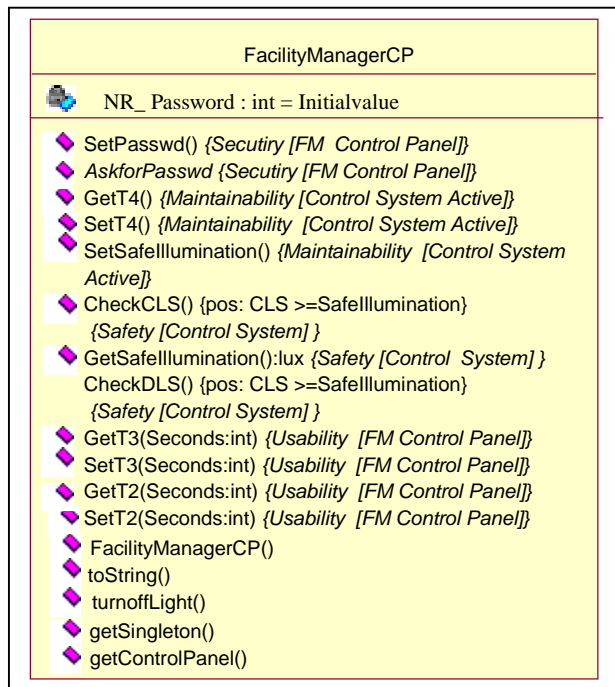
## Classes







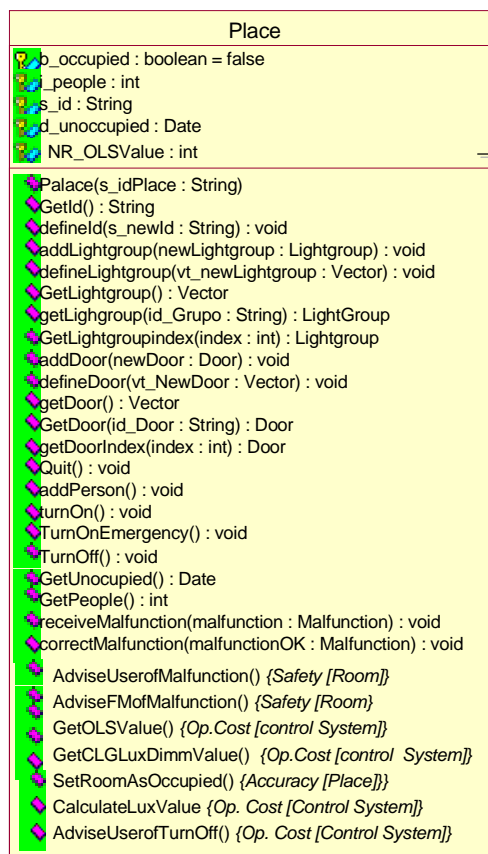
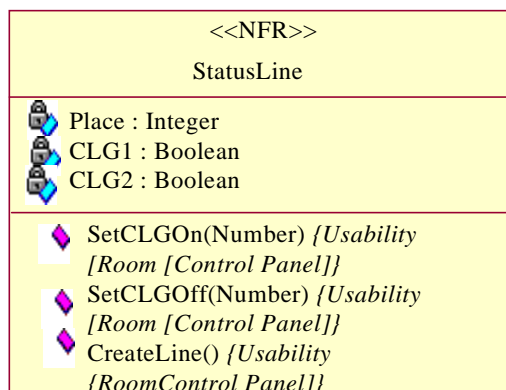




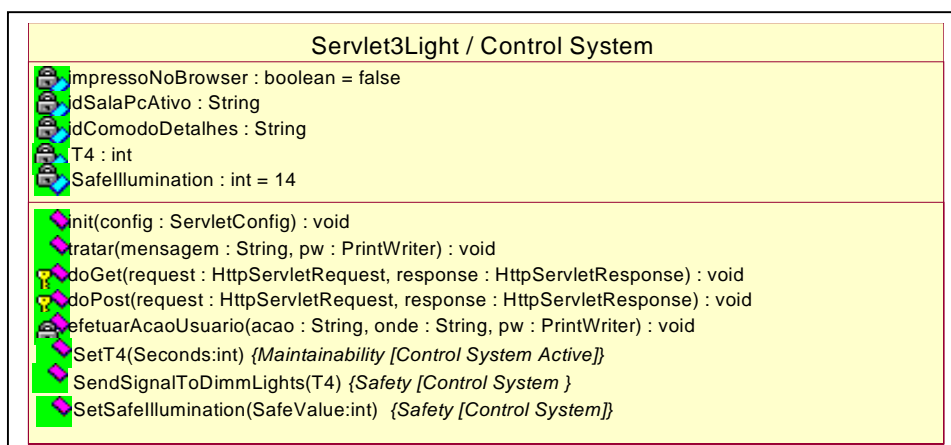
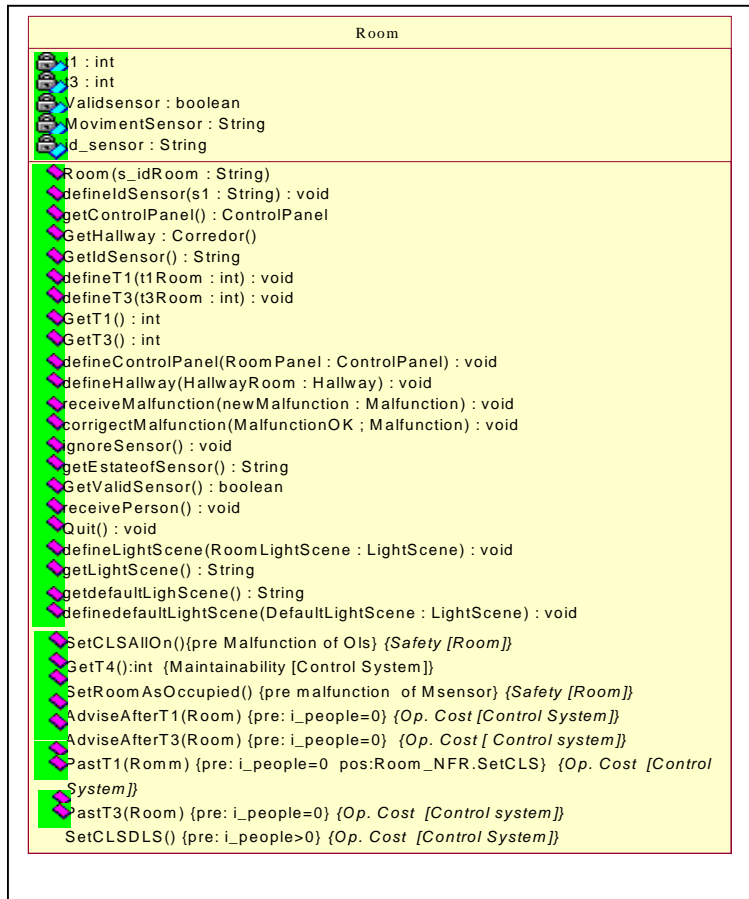
## Apêndice B – Estudo de Caso II

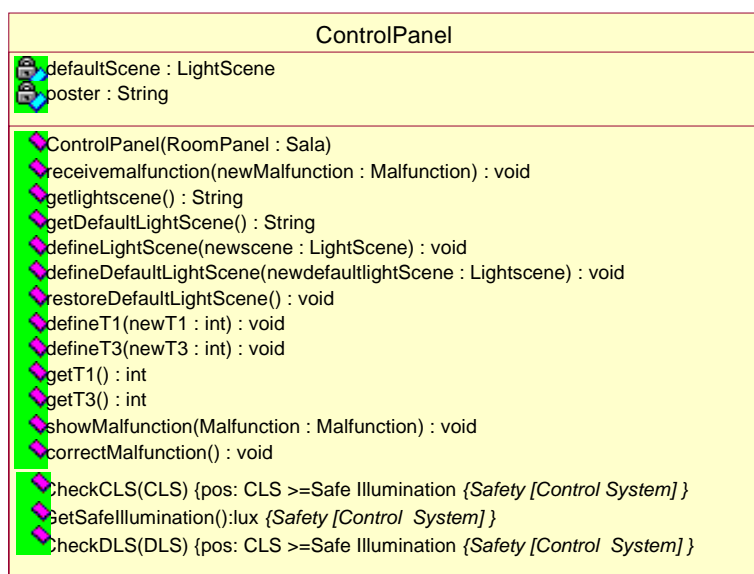
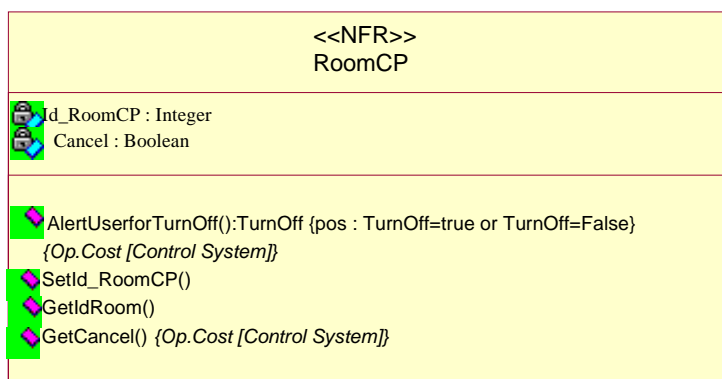
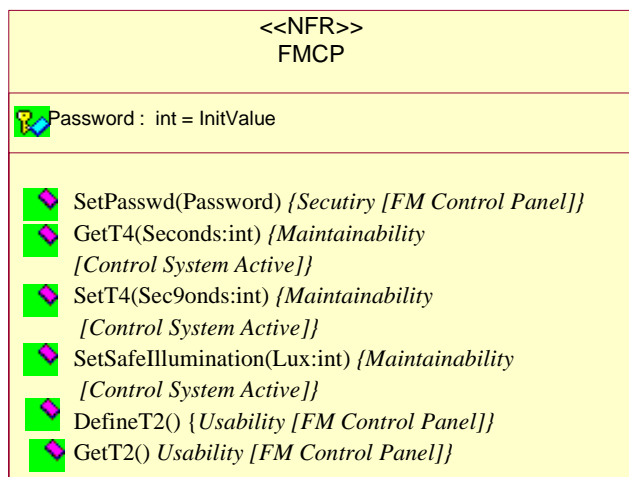
Este apêndice exibe todas as classes que sofreram alterações por conta de satisfazer RNFs. Os grafos de RNF utilizados são os mesmos utilizados no Estudo de Caso I e mostrados no apêndice A.























### Classes



SetRoomAsOccupied()  
 Pre: malfunction of Msensor}  
TurnOff()  
 Pre: AdviseUserofMalfunction() and  
 TurnOff=True  
AdviseUserofMalfunction()  
 Post : TurnOff=True or TurnOff=False

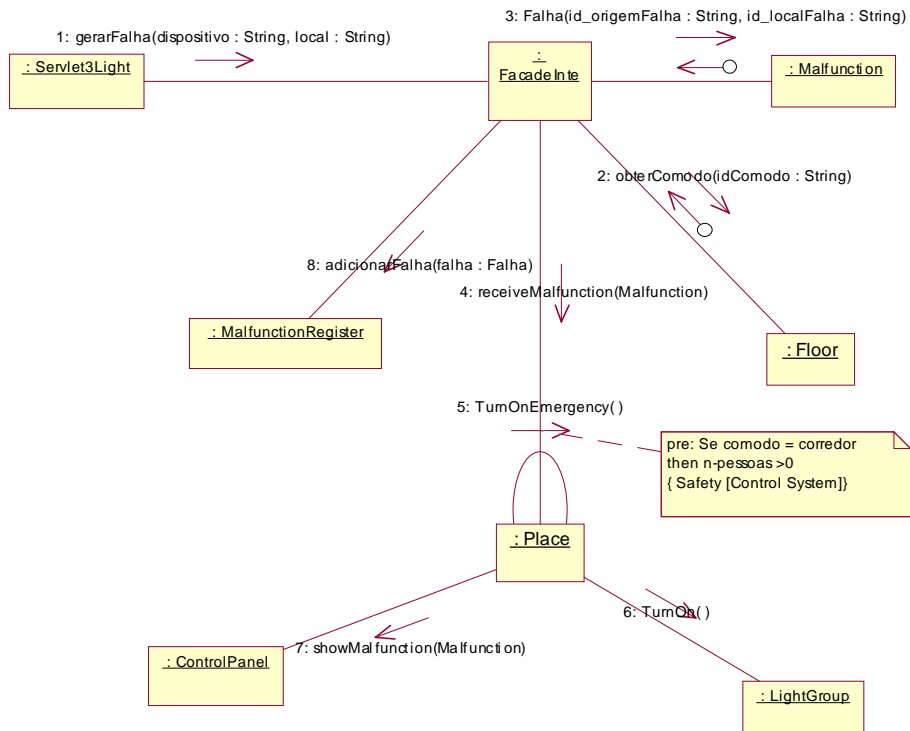




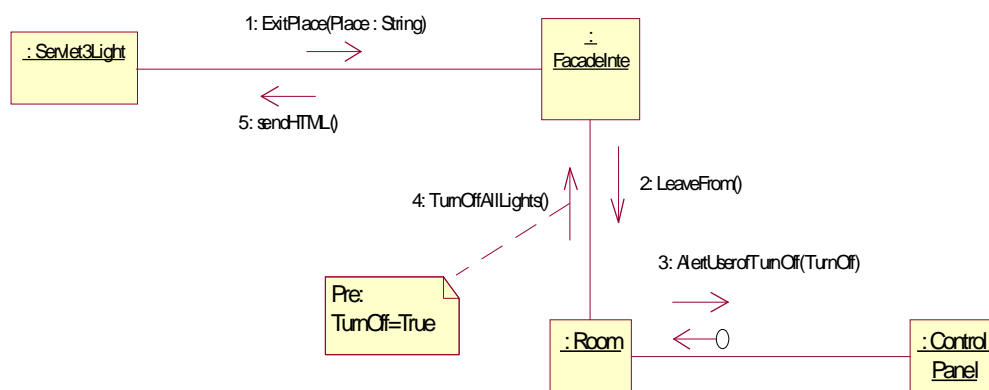
LightGroup	
	_intensity : int
	_lightsOn : int
	valid : boolean
	s_id : String
	s_status : String
	%Dim : Integer
	LuxValue : Integer {Safety [Illumination]}
	LastPulse : Hour
	LightGroup(s_idGrupo : String)
	GetId() : String
	TurnOn() : void
	ChangeIntensity(i_novaIntensidade : int) : void
	correctMalfunction() : void
	TurnOff() : void
	GetStatus() : String
	receivemalfunction() : void
	ListenPulse() {Safety [Control System]}
	DimLights100%() {Pre: Pulse not received and elapsedtime>LastPulse) >T4} {Safety [Control System]}
	SetDimmValue(Percent) {Safety [Control System] Op.Cost [Control System]}
	CalculateLuxValue() {Safety [Dimm Light]}
	GetDimValuies() {Op. Cost [Control System]}
	SetCLGOn(Number) {Usability [Room Control Panel]}

## Diagramas de Colaboração

### Ocorre uma Falha

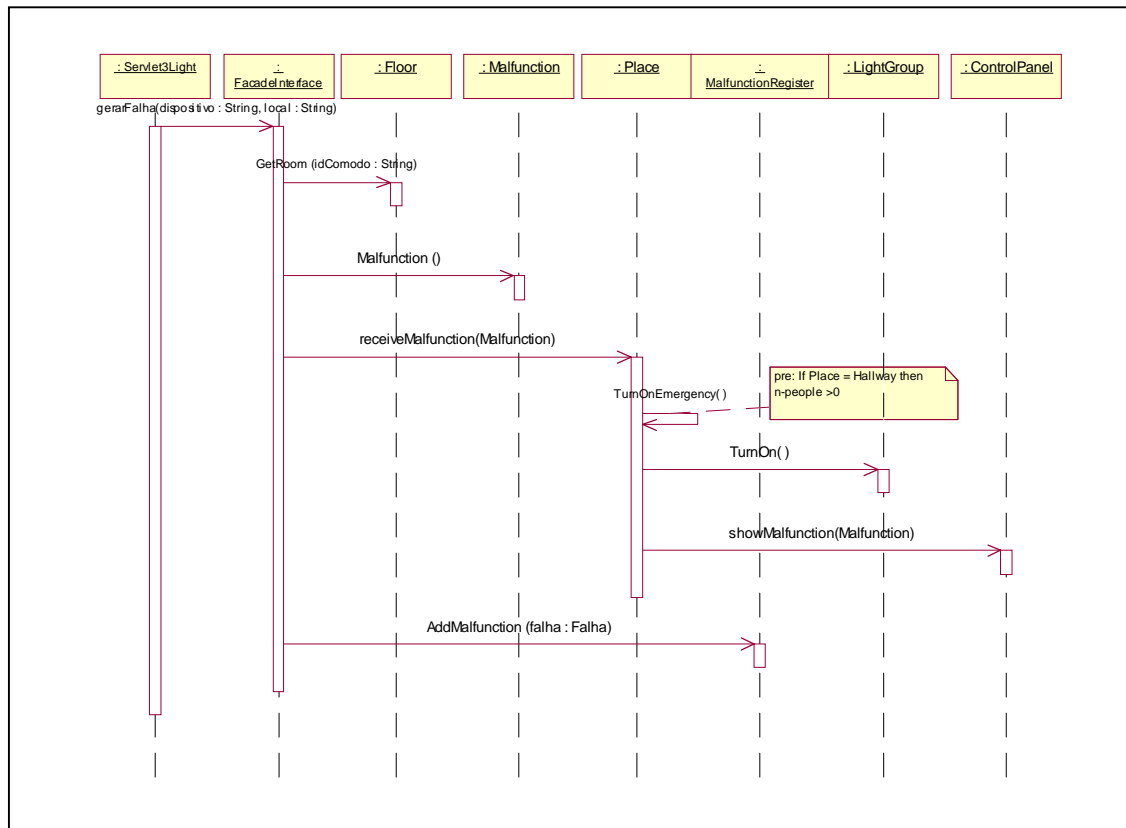


### Usuário sai de sala

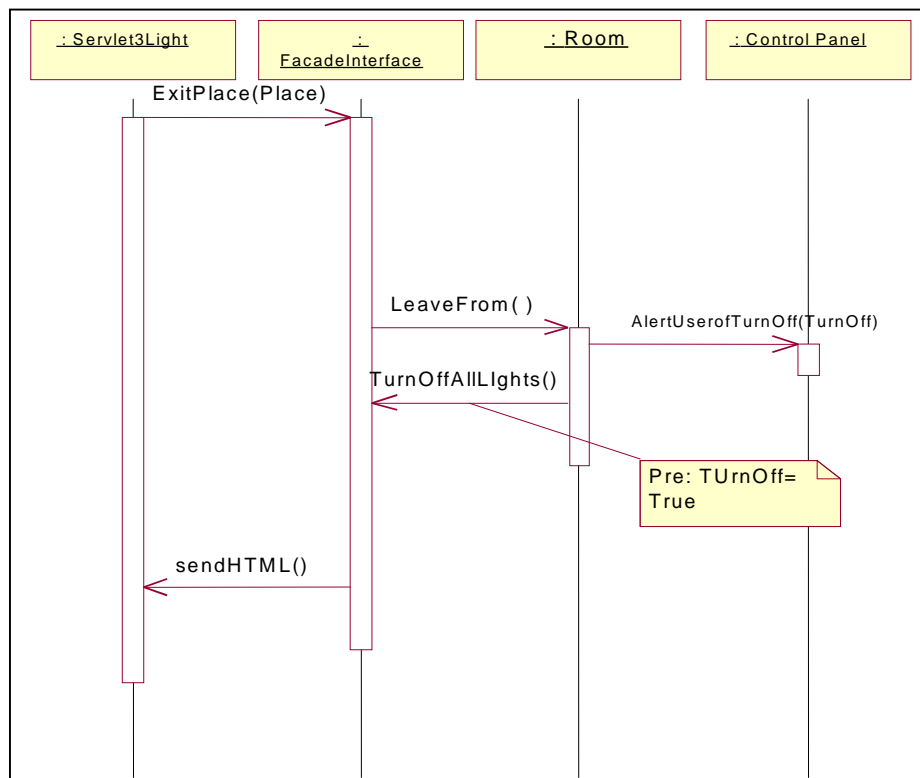


## Diagramas de Seqüência

### Ocorre uma Falha



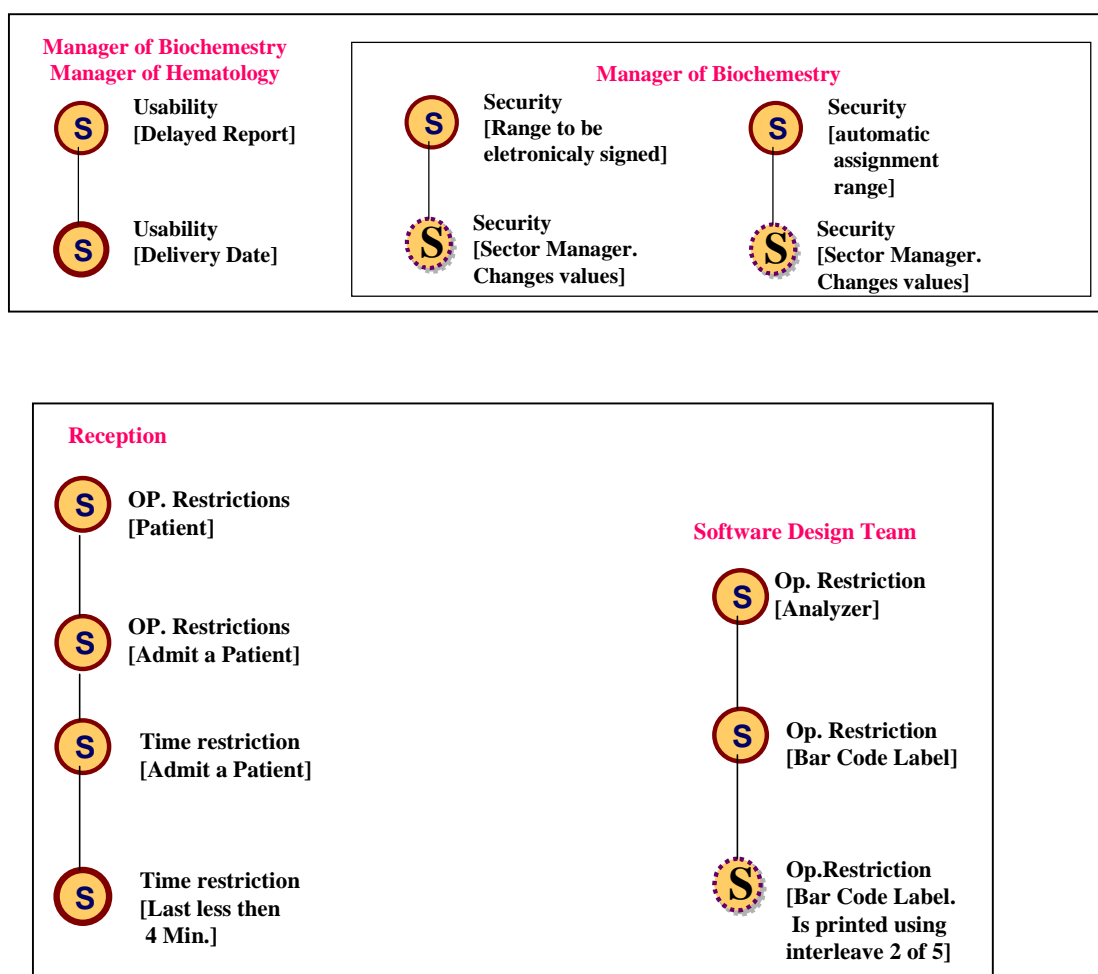
### Usuário sai de sala



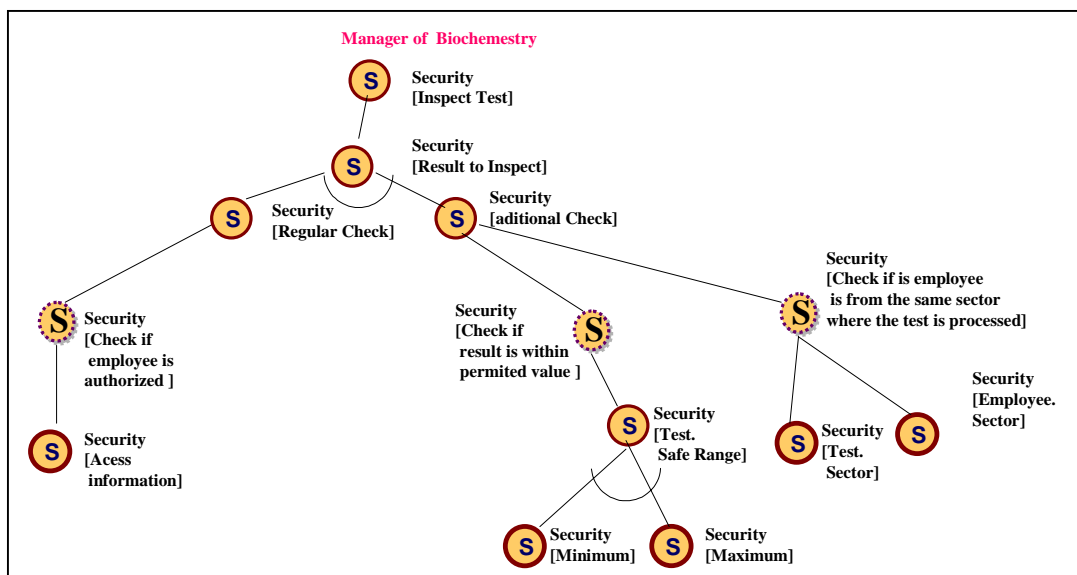
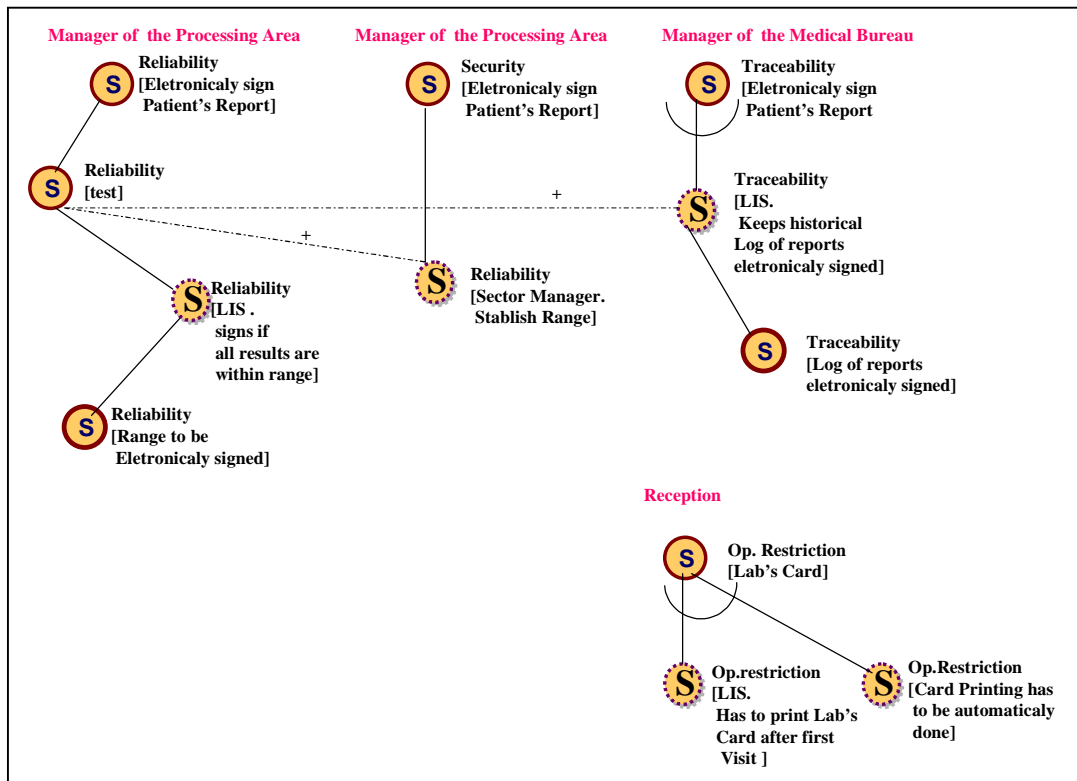
## Apêndice C – Estudo de Caso III

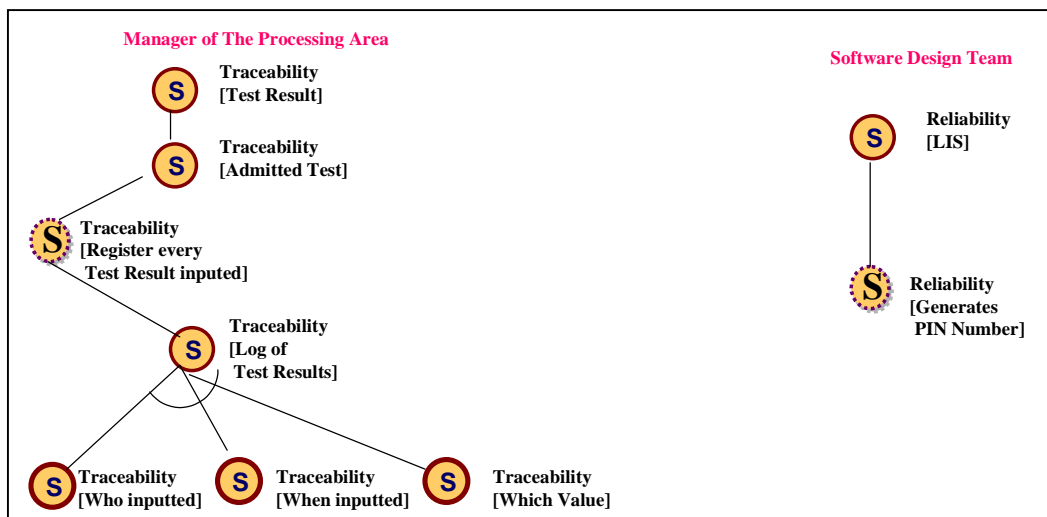
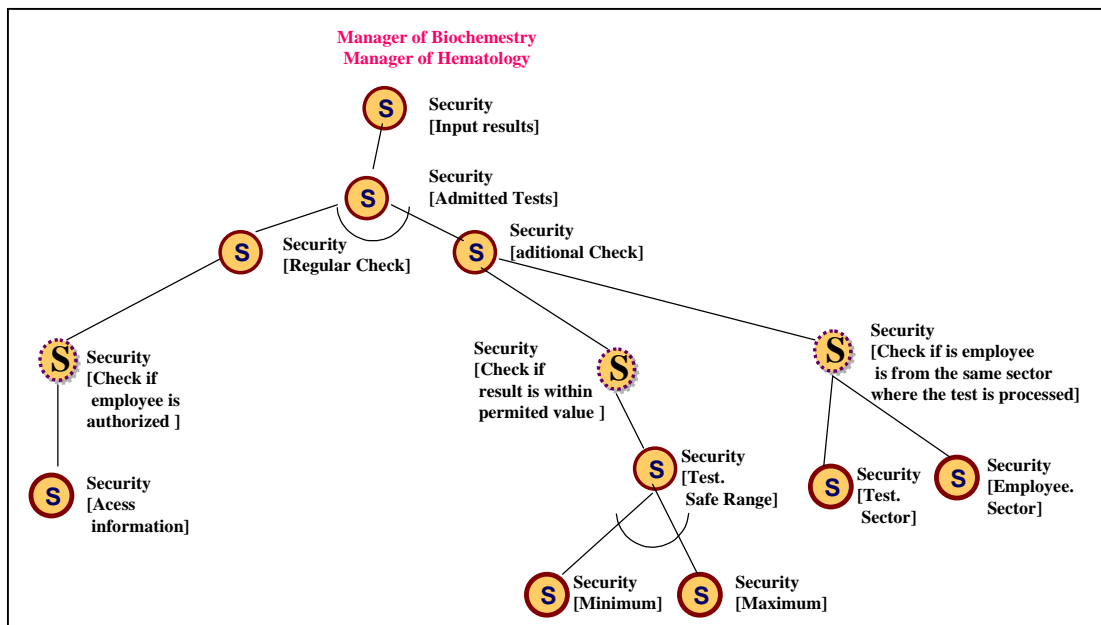
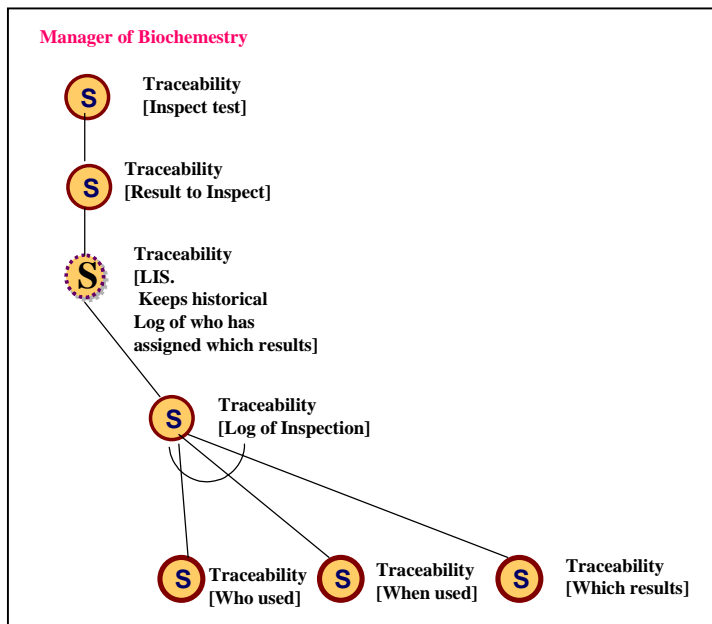
Este apêndice exhibe todos os grafos de RNF achados para o estudo de caso III e o detalhamento de grande parte das classes que sofreram alterações por conta de satisfazer RNFs, bem como os diagramas de sequência e colaboração alterados. Algumas classes foram omitidas desta publicação em respeito à restrições de confidencialidade de negócios impostas pela empresa onde foi efetuado o estudo de caso.

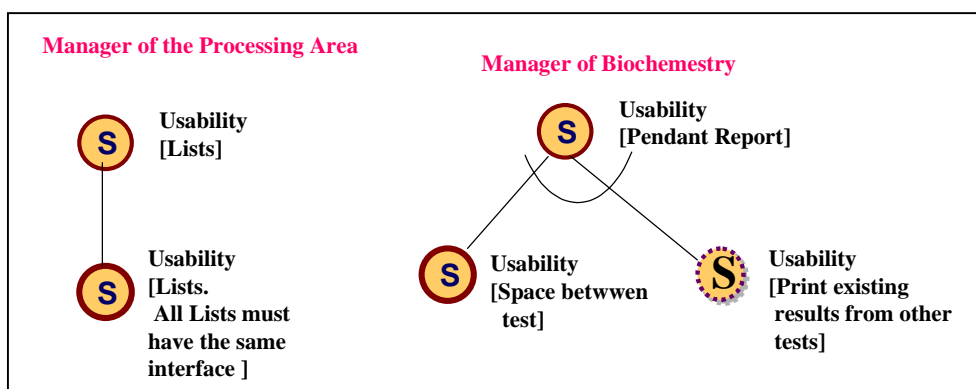
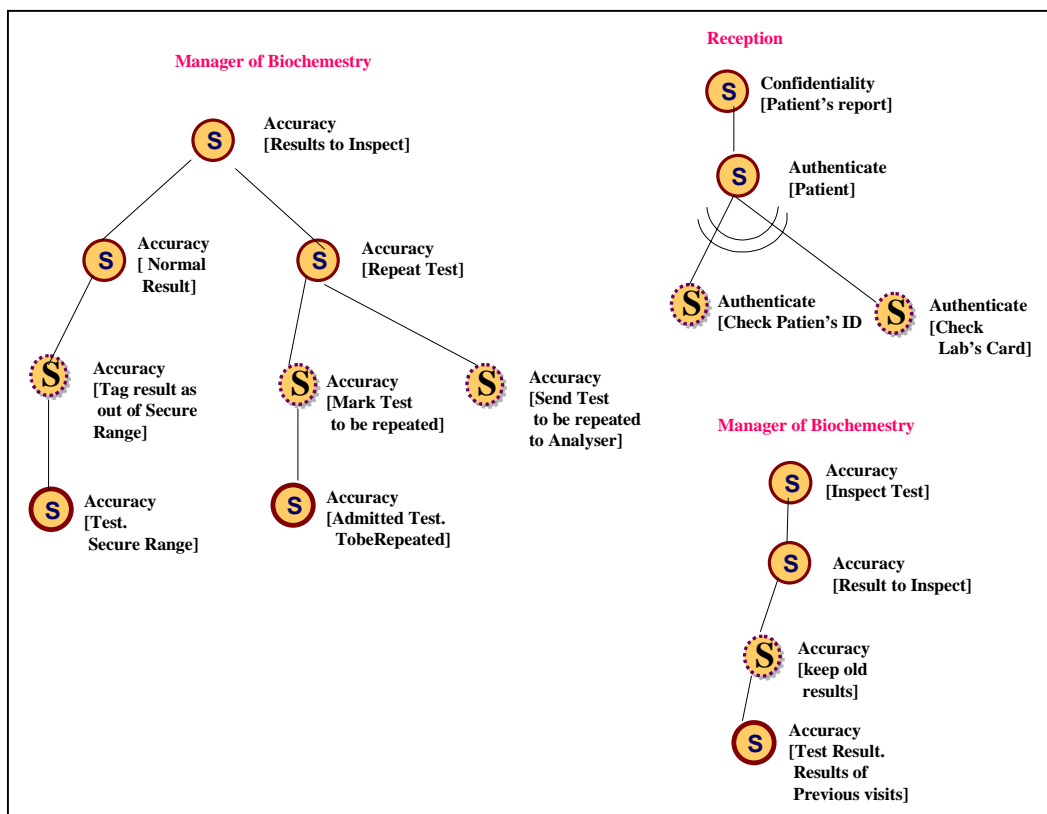
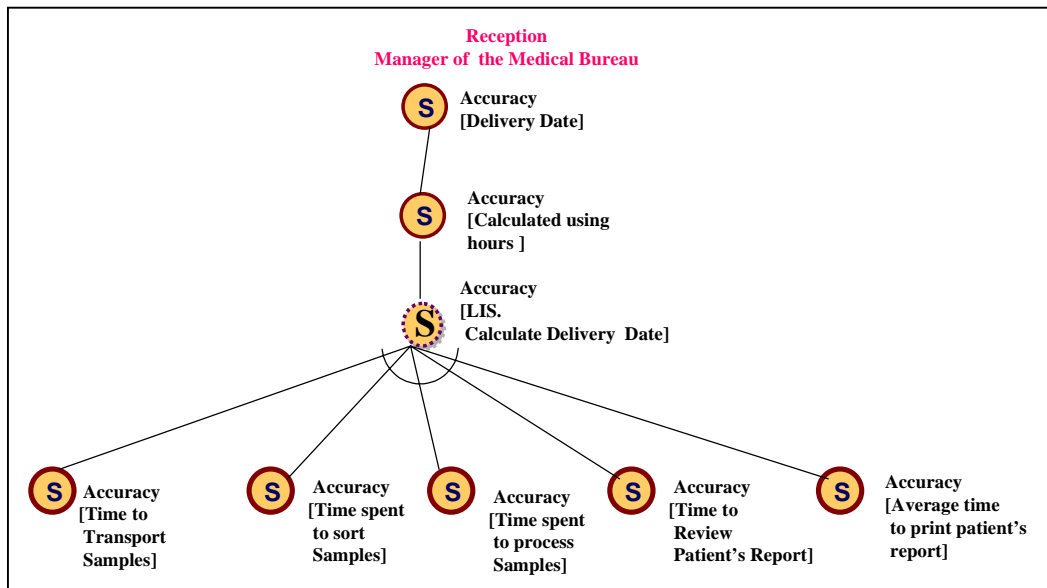
### Grafos de RNF

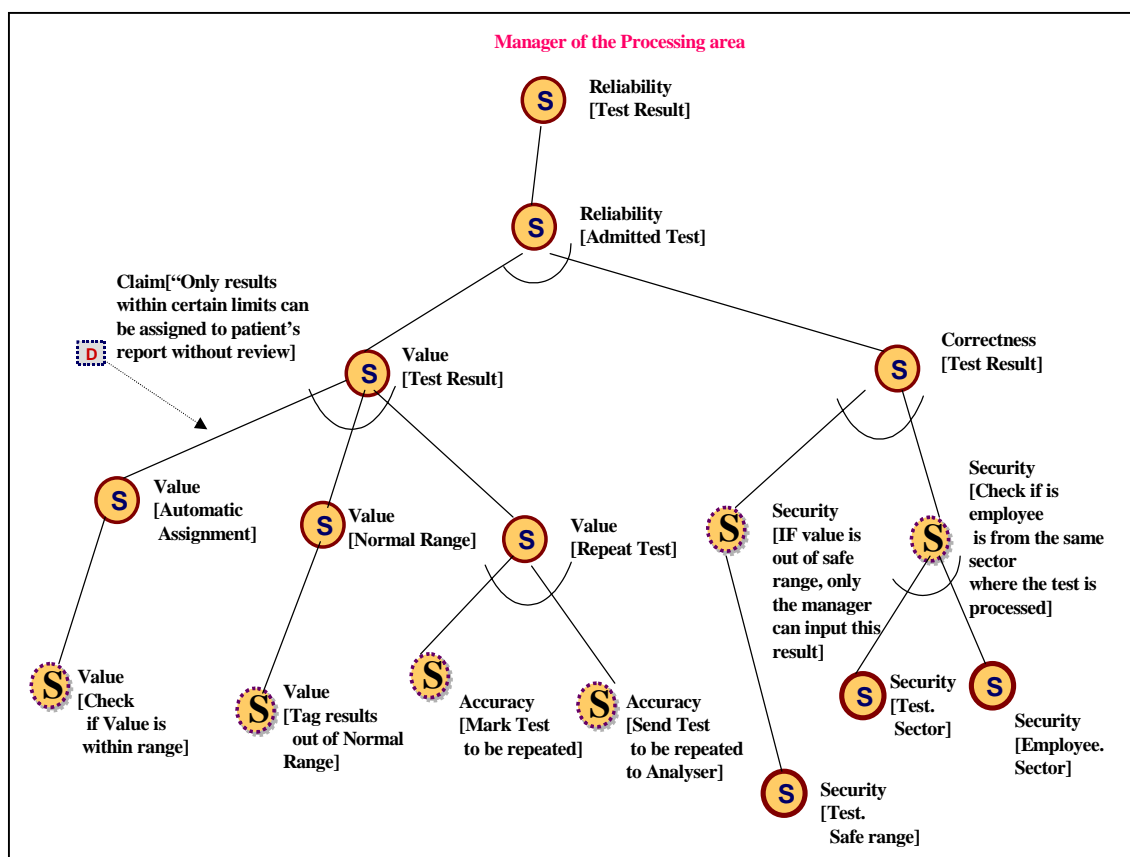
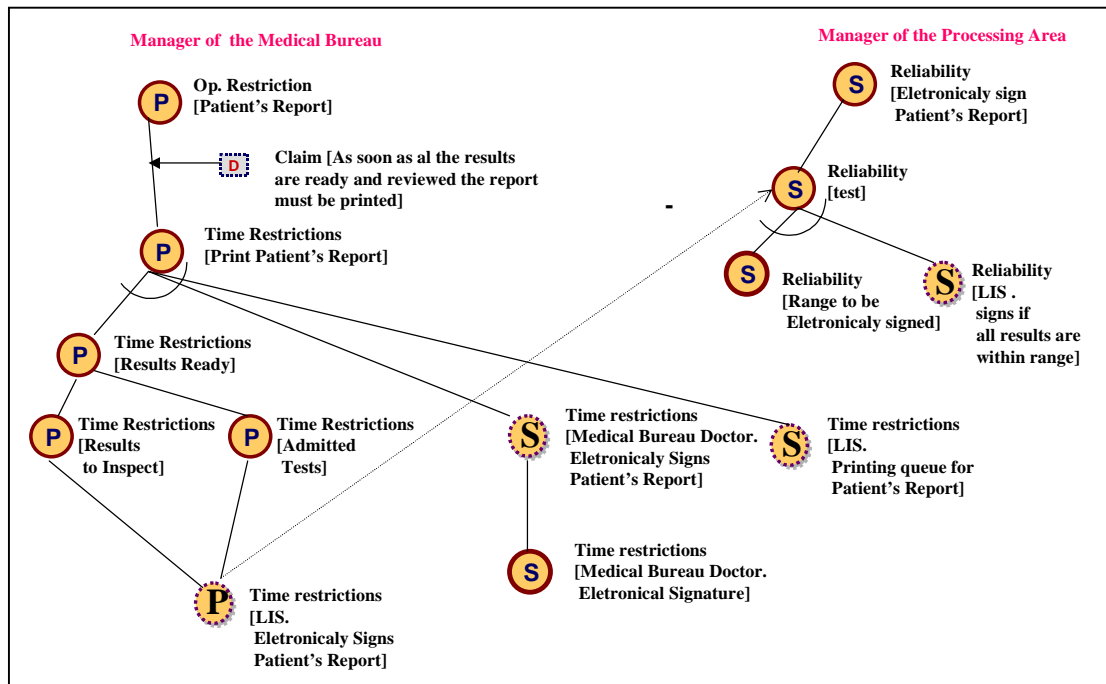


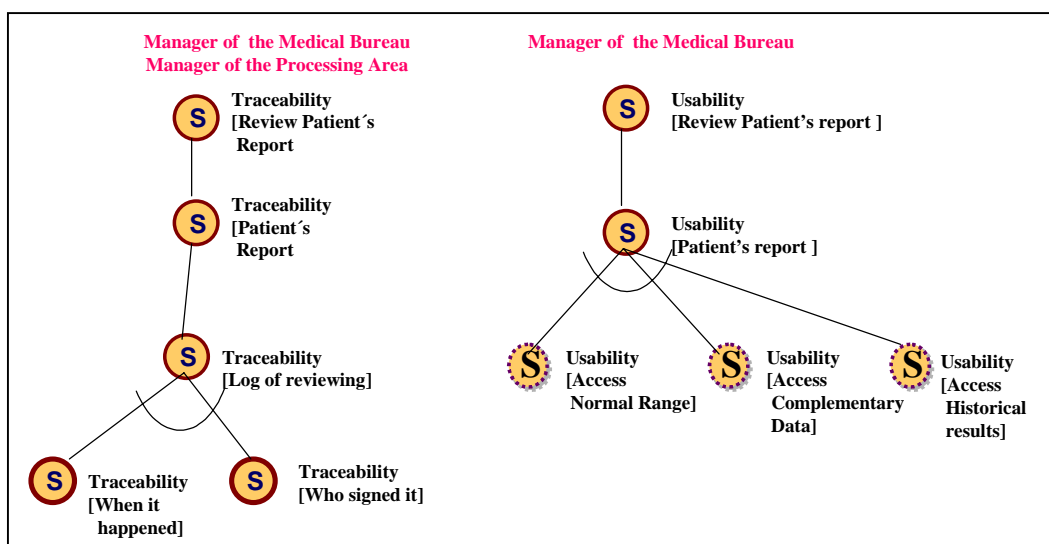
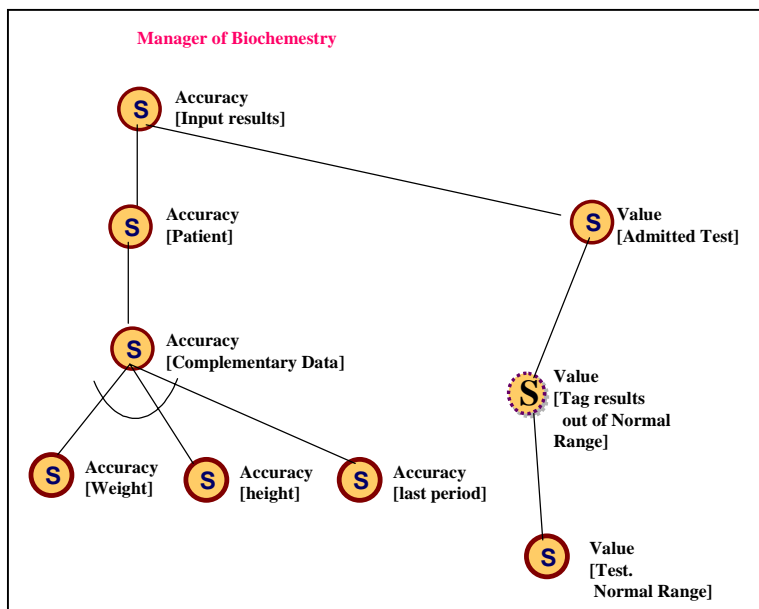
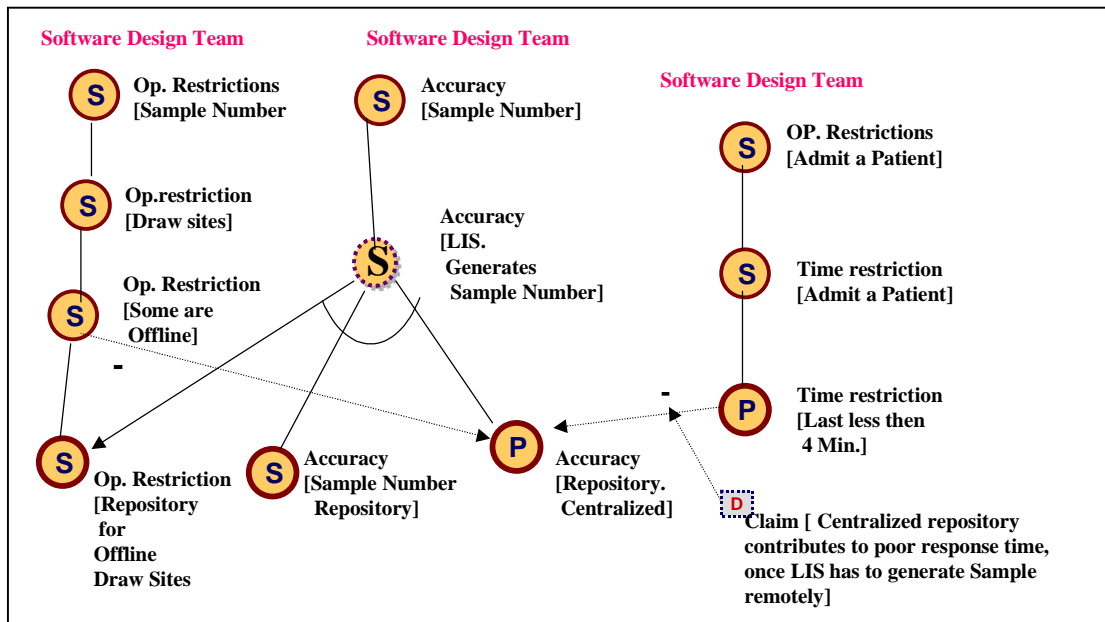


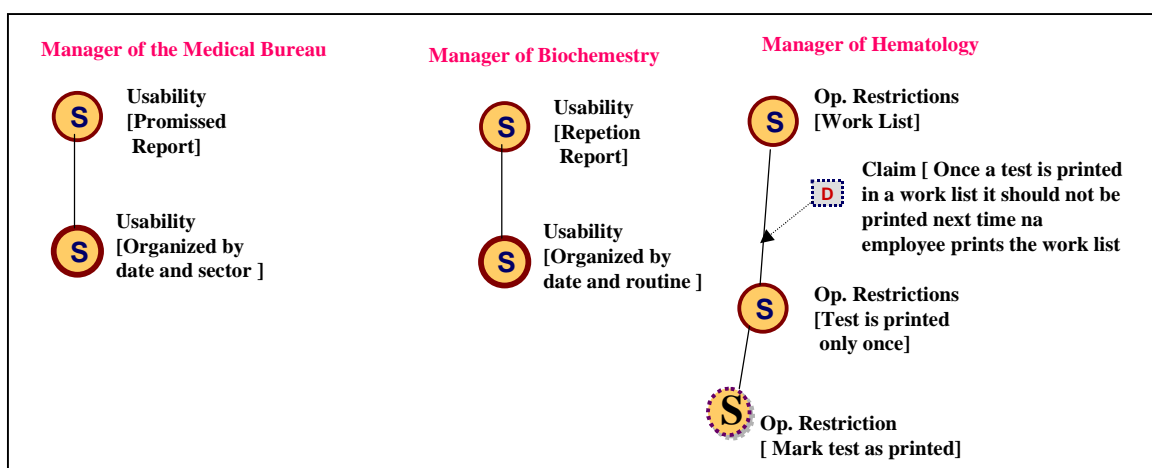
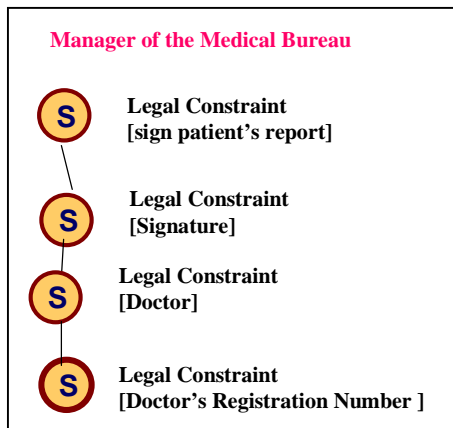
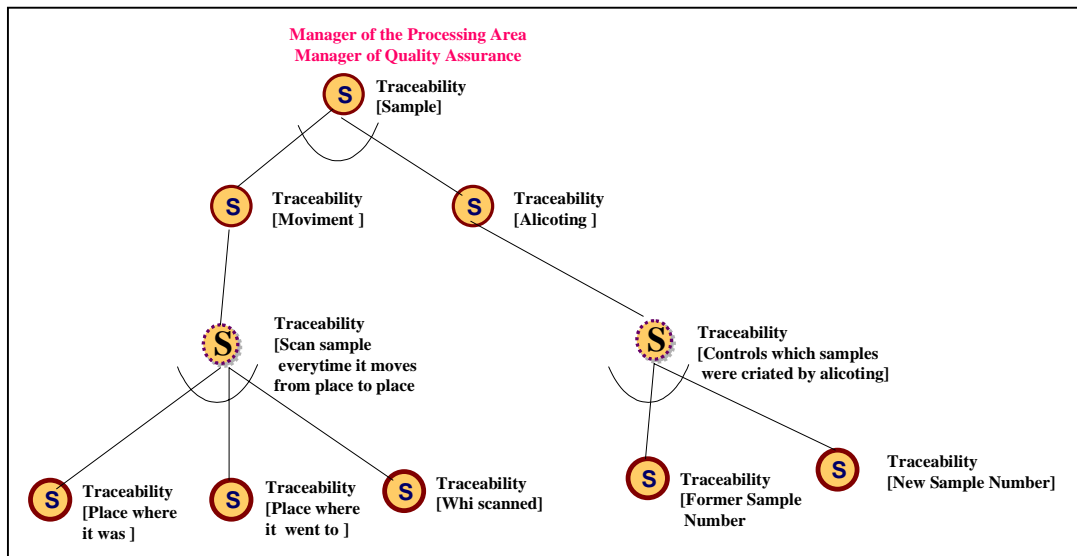




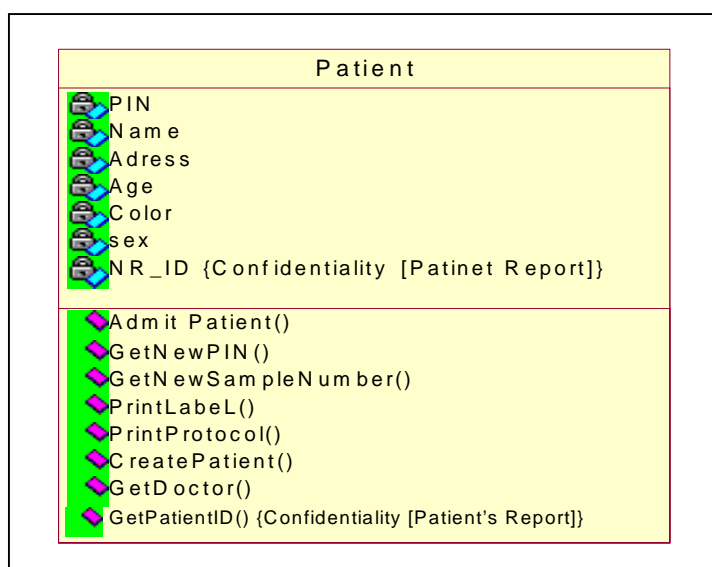
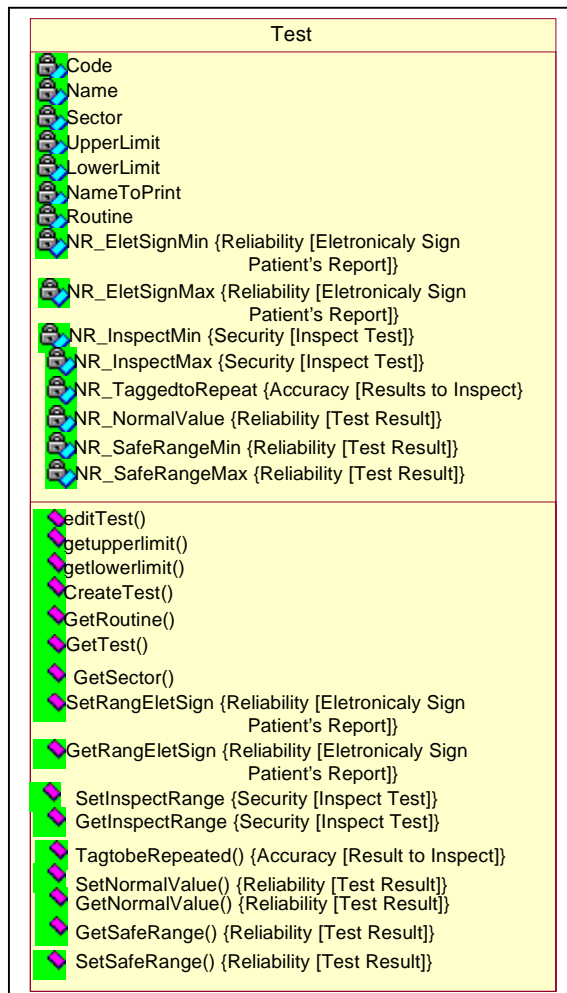


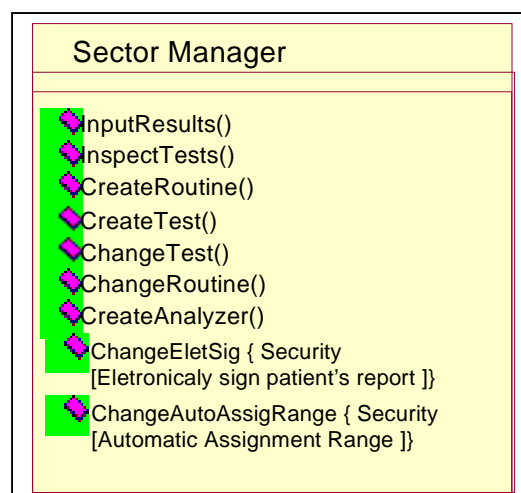
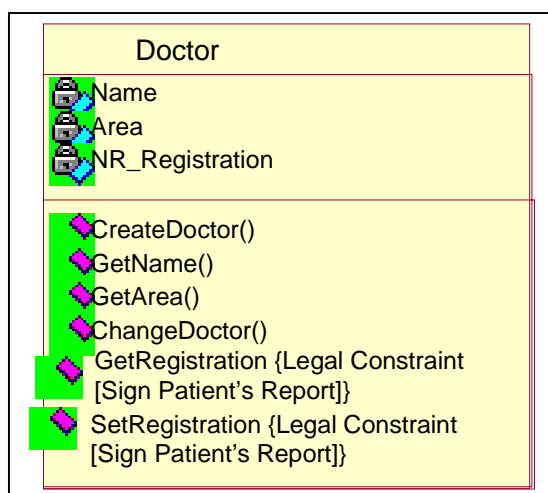
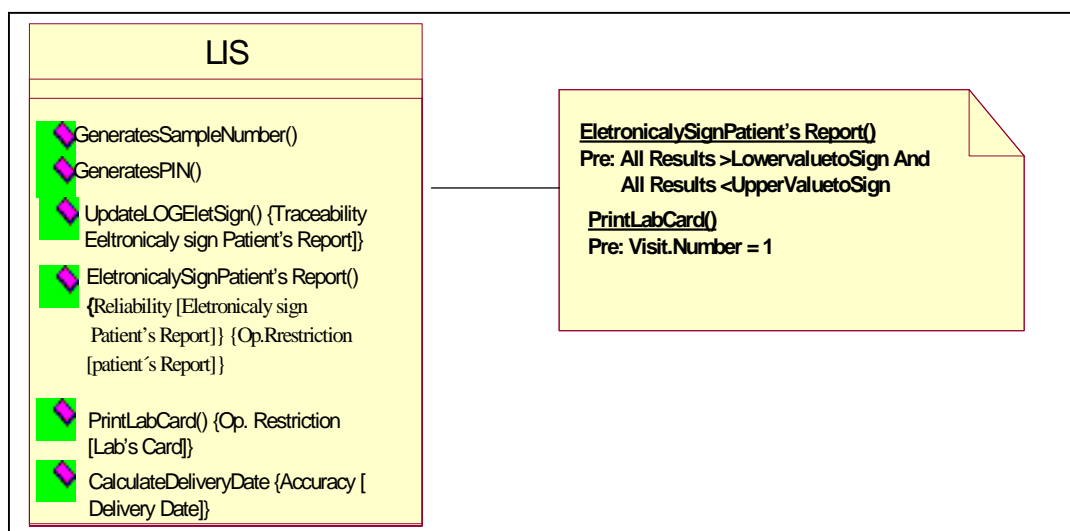
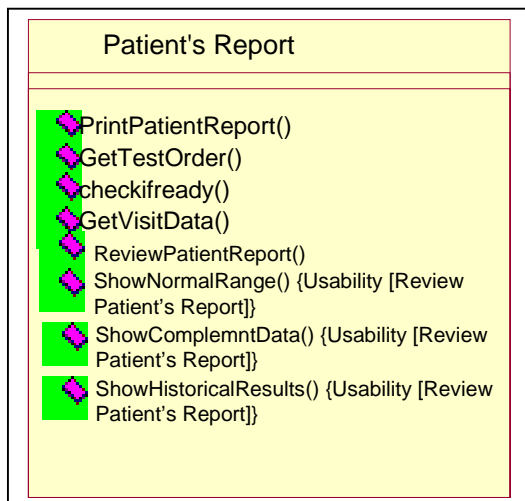




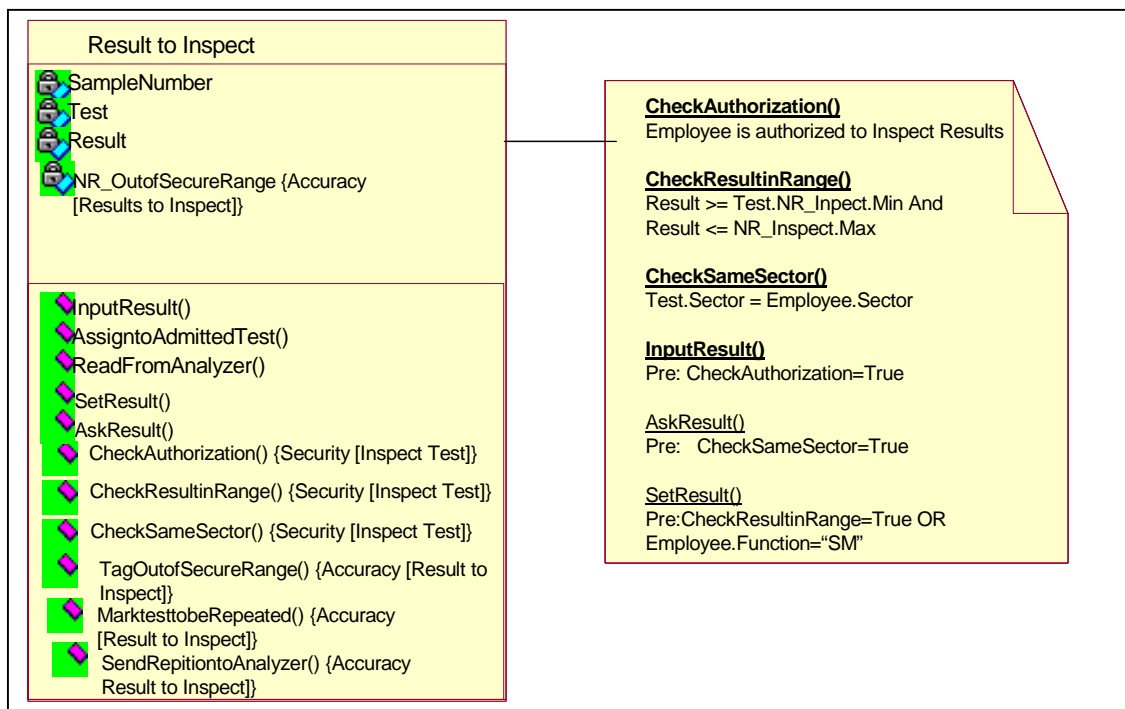
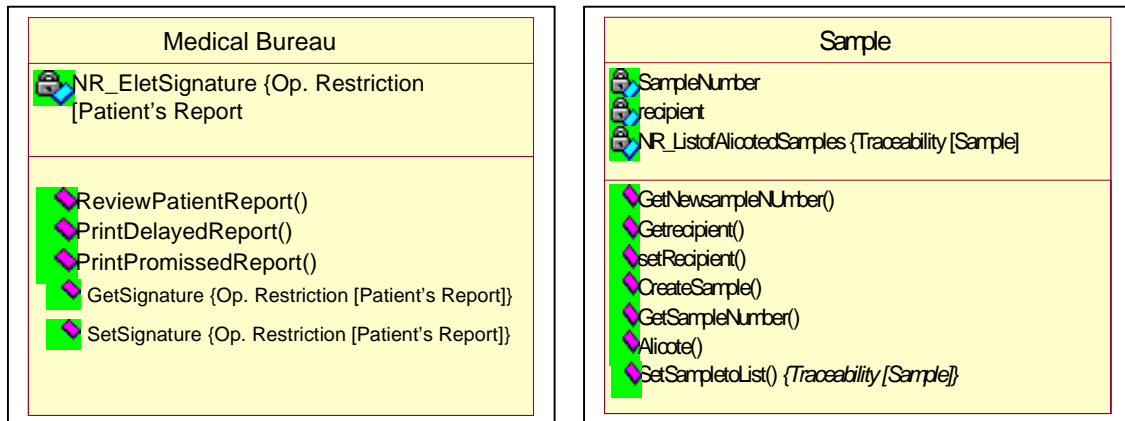


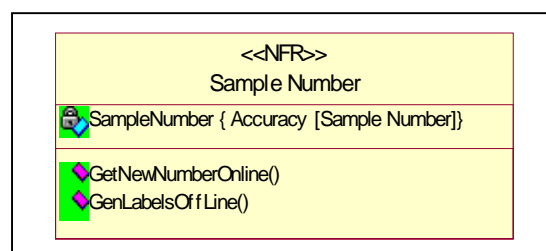
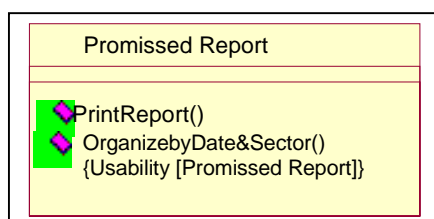
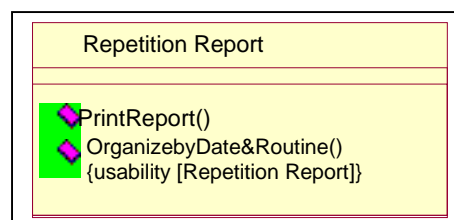
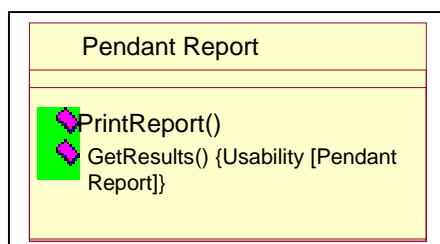
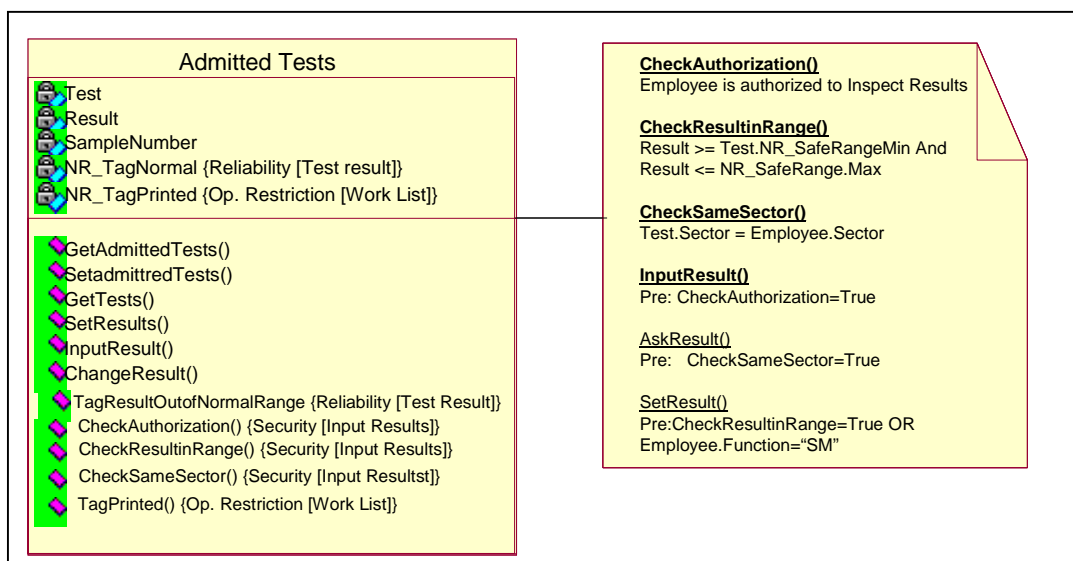
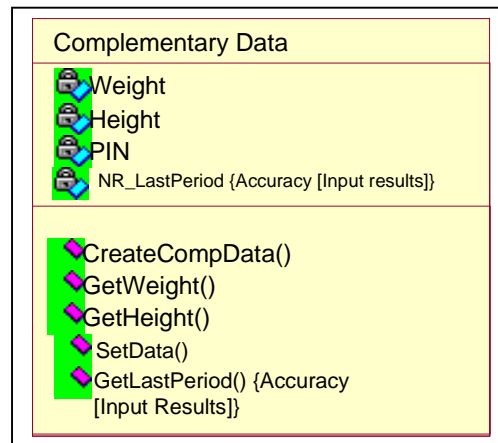
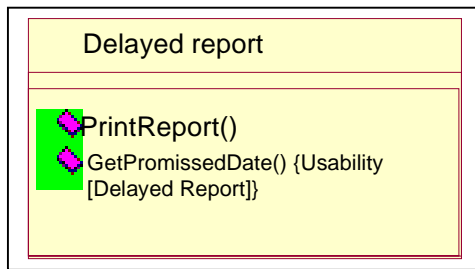
## Classes

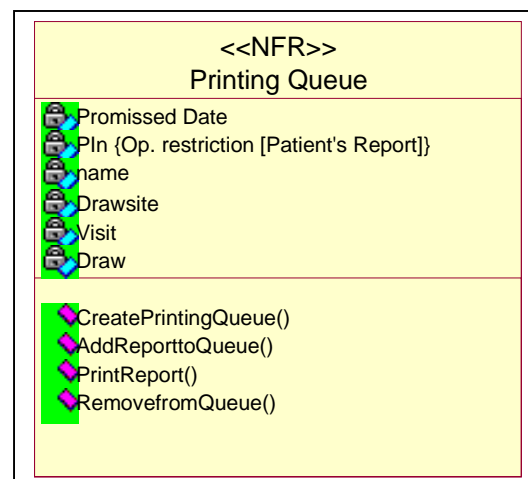
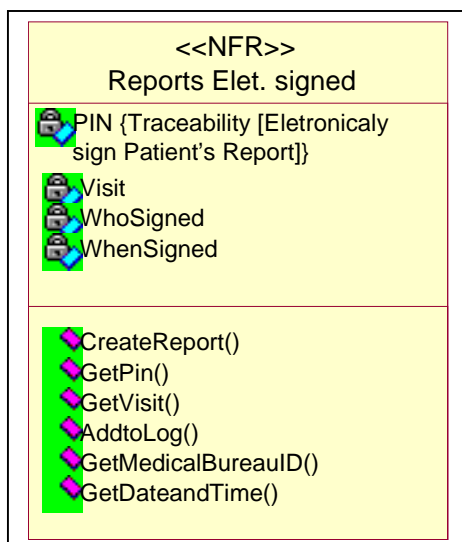
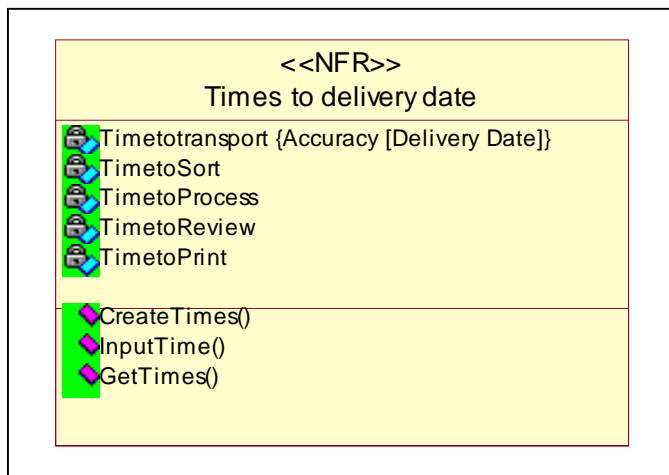
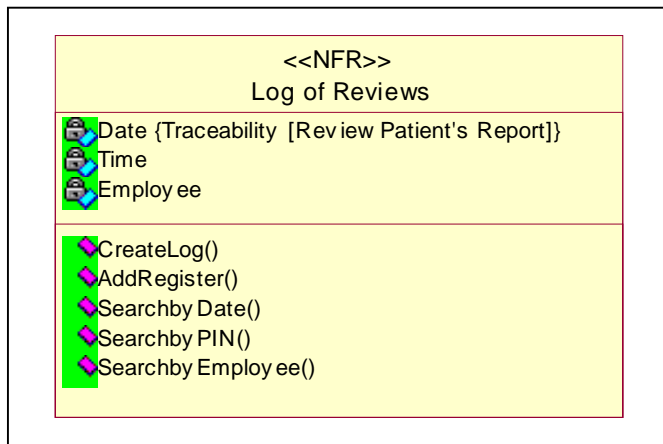


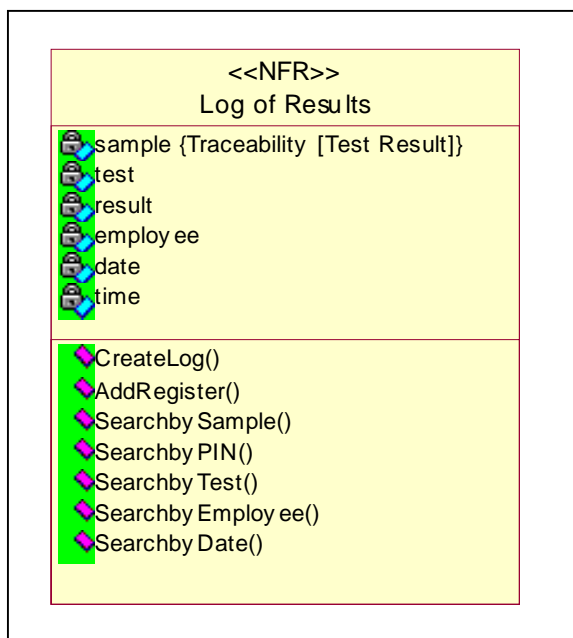
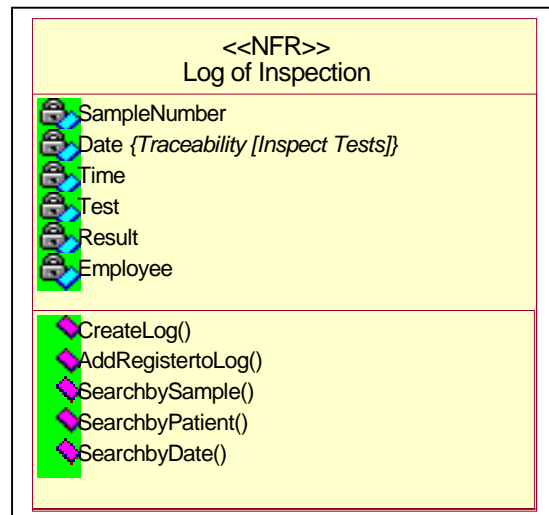
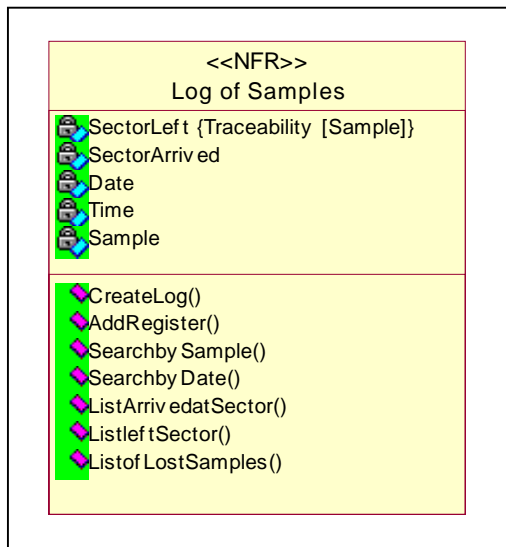




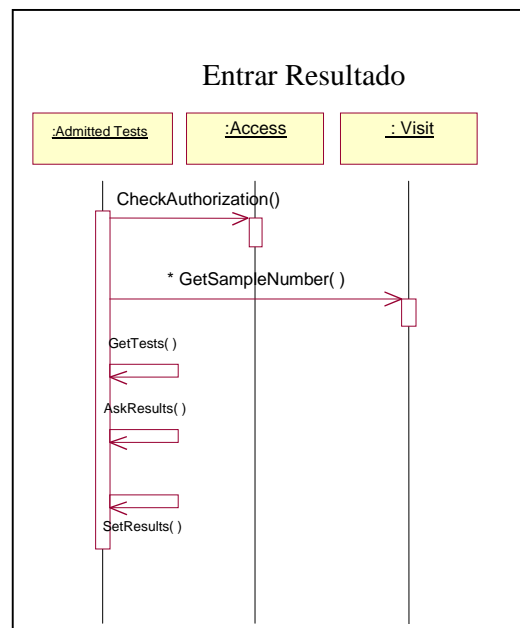
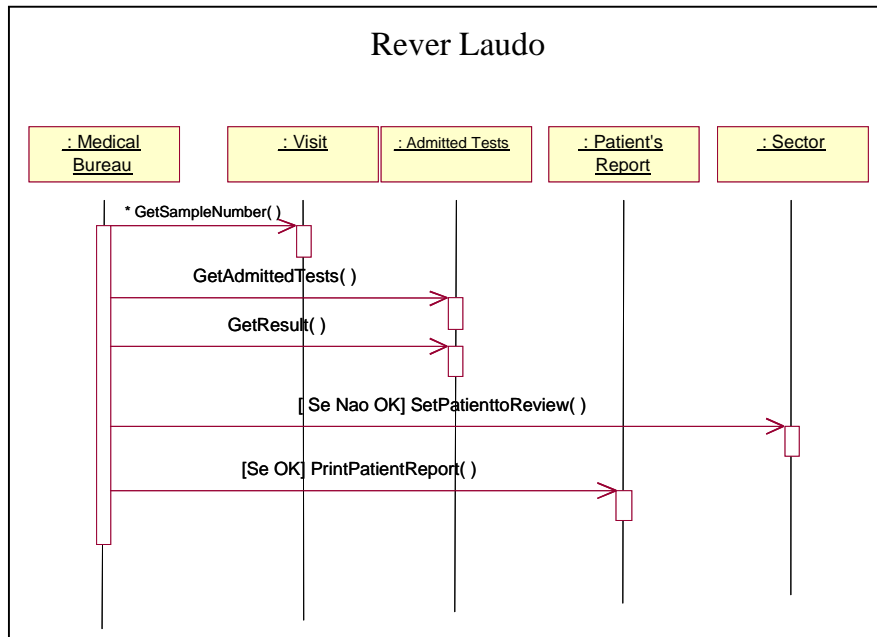


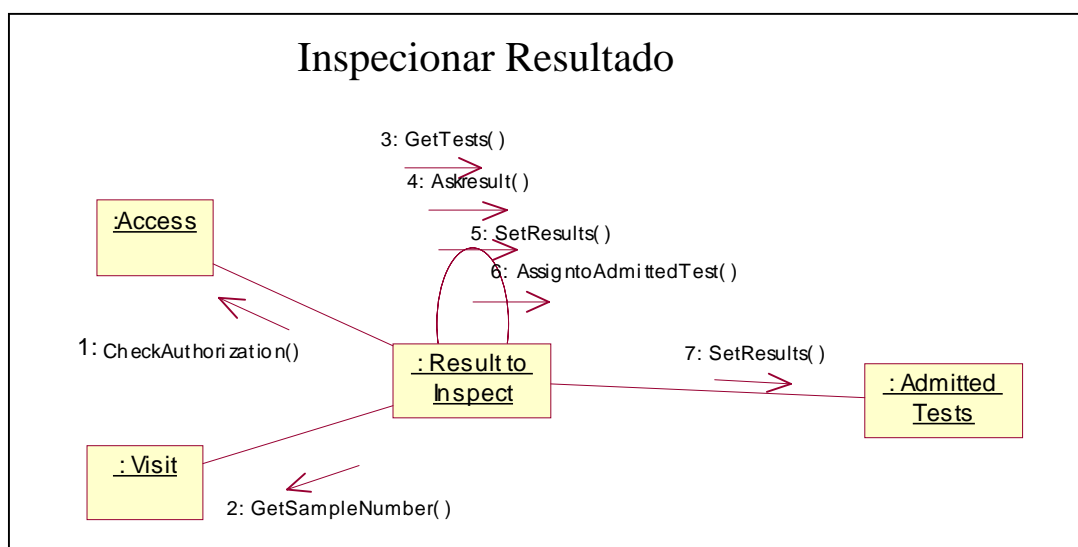
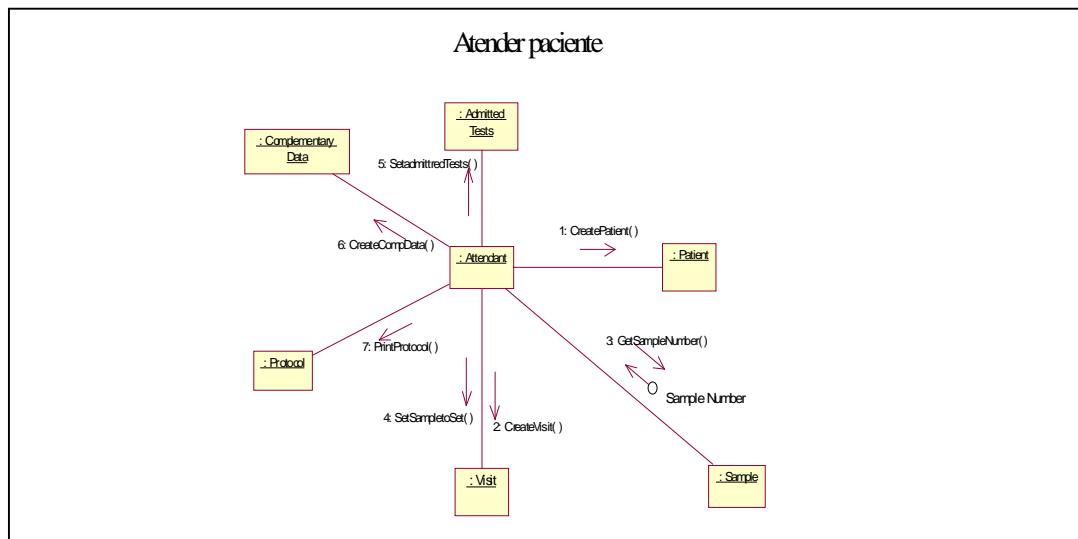






## Diagramas de Seqüência e Colaboração Antes dos RNFs





## Diagramas de Seqüência e Colaboração Após dos RNFs

