



Pós-Graduação em Ciência da Computação

**“Uma Abordagem Orientada a Objetivos para as Fases
de Requisitos de Linhas de Produtos de Software”**

Por

Clarissa César Borba

Dissertação de Mestrado



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, AGOSTO/2009



Universidade Federal de Pernambuco

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Clarissa César Borba

“Uma Abordagem Orientada a Objetivos para as Fases de Requisitos de Linhas de Produtos de Software”

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR(A): Prof. Dr. Jaelson Freire Brelaz de Castro

CO-ORIENTADOR(A): Prof. Dra. Carla Taciana Lima Lourenço Silva

RECIFE, AGOSTO/2009

Dedico este trabalho à minha amada família:
Paulo, Pêu, Maricota, painho, mainha e Marcelo.

Agradecimentos

A Deus por permitir mais esta vitória pessoal e profissional.

Ao meu marido, Paulo, pelo seu apoio emocional, dedicação, companheirismo e amor. Aos meus filhos, Pedro e Mariana, por serem minha alegria de vida e a razão dessa nova conquista. Eu amo vocês!

Aos meus pais, Niltinho e Lurdinha, por toda minha educação, meus valores e princípios. Pela presença constante e dedicação em todos os momentos de minha vida. Essa conquista é também de vocês!

À tia Eva e Dona Graça, por muitas vezes fazerem o papel de mãe na minha vida.

À minha co-orientadora, Carla, meus sinceros agradecimentos por todas as reuniões, idéias trocadas e revisões. Ao longo deste período você se tornou uma amiga e sua ajuda foi fundamental.

Ao meu orientador, Jaelson, pela confiança, incentivo, apoio e orientação no desenvolvimento deste trabalho. Muito obrigada!

Aos meninos do grupo SPG, em especial a Rodrigo, Leopoldo e Márcio, pela disponibilidade, paciência e ricos comentários. A ajuda de vocês foi de grande valia!

Aos meus amigos do TRE, Júnior, Gracinha e George, por toda a flexibilidade, motivação e confiança. Além disso, agradecimento especial a George pela dedicação e presteza no dia da minha defesa. Sem o apoio de vocês tudo teria sido mais difícil!

Ao meu amigo Akio, pela convivência diária no TRE e pelo constante incentivo, principalmente, nos momentos mais difíceis do mestrado.

À minha amigona Carla Dias, pela amizade de tantos anos e por ser minha melhor confidente.

A todos os colegas do Grupo LER que direta ou indiretamente participaram deste trabalho.

Aos professores Sérgio Soares e Fernanda Alencar que participaram da banca examinadora deste trabalho e que contribuíram com ricas sugestões de melhorias.

Resumo

Uma Linha de Produtos de Software (LPS) é um grupo de produtos de software com características comuns e variáveis, que também pode ser chamada de família de produtos. As características de uma LPS precisam ser documentadas explicitamente para possibilitar o reuso estratégico dos seus artefatos. Na atividade de Engenharia de Requisitos, isto significa que além de capturar as suas variabilidades, também é preciso relacionar os diferentes tipos de requisitos, tais como organizacionais, não-funcionais e funcionais, além de manter o rastreamento entre eles. Atualmente a captura desta informação é feita usando os modelos de *features*, mas esses não capturam requisitos não-funcionais explicitamente e nem a influência positiva/negativa destes requisitos para alcançar configurações alternativas de uma aplicação na LPS. Esta influência pode ajudar na escolha de uma configuração específica para uma aplicação alcançar os objetivos da organização. Um objetivo é um estado do mundo que os *stakeholders* desejam alcançar enquanto que uma *feature* é uma característica que o sistema deve apresentar. Partindo destas definições, abordagens orientadas a objetivos podem ser usadas como uma forma efetiva para descobrir requisitos variáveis e comuns de uma LPS, bem como para reduzir os custos associados à configuração de um produto específico na família de produtos. Uma abordagem de requisitos orientada a objetivos que tem sido usada para o desenvolvimento de sistemas complexos é o *framework* *i**. O *i** fornece uma maneira natural de identificar e especificar tanto os interesses dos *stakeholders* como as características do sistema pretendido. Este trabalho propõe uma extensão da linguagem de modelagem do *i**, chamada **i*-c** (*i* with cardinality*), que adiciona cardinalidade nos elementos de modelos intencionais e assim, permite a identificação e modelagem de *features* a partir de modelos orientados a objetivos. Para guiar a configuração de uma aplicação específica em uma LPS foi definida a abordagem **G2FM** (*Goal to Feature Model*). Ela propõe um processo de identificação e modelagem de *features* comuns e variáveis de uma LPS em modelos *i** com cardinalidade e, a partir destes, produz um modelo de *features* equivalente.

Palavras-chave: Linhas de Produto de Software, Modelo de *Features*, Engenharia de Requisitos Orientada a Objetivos, Modelos *i**.

Abstract

A Software Product Line (SPL) is a group of similar software products with commonalities and variabilities, and can be also called product family. The variability of a product line has to be documented explicitly to enable a strategic reuse of requirements artifacts. In Requirements Engineering, this means it should relate different types of requirements, such as organizational, non-functional and functional, and keep trace among them. Feature modeling is used to capture commonalities and variabilities in system families, but does not capture explicitly non-functional requirements and the positive/negative influence from these requirements to the system configuration alternatives elicited during the development of the core assets. This influence could help to choose a specific configuration for an application to meet organizational goals. Goals are states of affairs in the world that the stakeholders want to attain while features are characteristics the system must present. In this light, it seems clear that goal-oriented approaches can be used as an effective way for discovering SPL common and variable requirements, as well as reducing time and costs associated with the configuration of an individual product in a product family. The i^* framework is a goal-oriented approach that has been used to develop complex systems and to identify and specify both the interests of stakeholders and the characteristics of the desired system. In this work, we propose a new extension to the i^* framework, called **i^* -c** (i^* with cardinality), which add cardinality in the elements of intentional models and then, allows the identification and modeling features from goal-oriented approaches. In order to guide the configuration of a specific application in a SPL, we propose an approach, called **G2FM** (Goal to Feature Model). It includes a process for identification and modeling of common and variable features in i^* models with cardinality, and based on them, generating an equivalent feature model.

Keywords: Software Product Line, Feature Model, Goal-Oriented Requirements Engineering, i^* Framework.

Conteúdo

Agradecimentos.....	iv
Resumo.....	iv
Abstract	ivi
Lista de Figuras.....	x
Lista de Tabelas	xv
Lista de Abreviaturas	xvii
1. Introdução	1
1.1. Contextualização	2
1.2. Motivação	3
1.3. Objetivos.....	4
1.4. Metodologia.....	4
1.5. Organização da dissertação	6
2. Engenharia de Linhas de Produtos de Software e Engenharia de Requisitos	
Orientada a Objetivos: Estado da Arte	7
2.1 Engenharia de Linhas de Produto de Software.....	8
2.1.1. Engenharia de Domínio	11
2.1.2. Engenharia de Aplicação	12
2.1.3. Modelo de <i>Features</i>	13
2.1.3.1. Conceito do Modelo de <i>Features</i>	14
2.2. Engenharia de Requisitos Orientada a Objetivos	18
2.2.1. Engenharia de Requisitos – Uma visão Geral.....	19
2.2.2. Processo de Engenharia de Requisitos.....	21
2.2.3. Abordagens Orientada a Objetivos	21
2.2.3.1. <i>Framework</i> NFR	23
2.2.3.2. Modelos de Objetivos	25

2.2.3.3. Framework i*	26
2.3. Considerações Finais	36
3. Adaptando modelos i* para suportar cardinalidade.....	38
3.1. Comparação entre abordagens orientadas a objetivos para a modelagem de variabilidade 39	
3.1.1. Modelo de Objetivos	41
3.1.2. Modelo PL-AOVGraph.....	43
3.1.3. Modelos i* Aspectual.....	46
3.1.4. Modelos i*.....	48
3.2. Extensão do modelo i* para inclusão de cardinalidade.....	51
3.3. Considerações Finais	59
4. Processo para extrair <i>features</i> a partir do Modelo i*	60
4.1. Visão geral do processo	61
4.2. Construção do modelo SR (opcional).....	63
4.3. Identificação de elementos com possibilidade de serem <i>features</i>	75
4.4. Reengenharia do Modelo SR.....	76
4.5. Elaboração do Modelo de <i>Features</i>	79
4.6. Reorganização do Modelo de <i>Features</i> (opcional).....	86
4.7. Configuração do produto no modelo SR	87
4.8. Elaboração do Modelo de Configuração do Produto	89
4.9. Resumo das Heurísticas.....	90
4.10. Discussão	92
4.11. Considerações Finais	94
5. Estudo de Caso: Mobile Media.....	95
5.1. Construção do modelo SR (opcional).....	97
5.1.1. Caso de Uso: Adicionar Foto	97
5.1.2. Caso de Uso: Apagar Foto	98
5.1.3. Caso de Uso: Visualizar Conteúdo	101
5.1.4. Caso de Uso: Adicionar Álbum	105
5.1.5. Caso de Uso: Apagar Álbum	108
5.1.6. Caso de Uso: Fornecer Legenda	112
5.2. Identificação de elementos com possibilidade de serem <i>features</i>	115

5.3.	Reengenharia do Modelo SR.....	119
5.4.	Elaboração do Modelo de <i>Features</i>	123
5.4.1.	Caso de Uso: Adicionar Foto	125
5.4.2.	Caso de Uso: Apagar Foto	125
5.4.3.	Caso de Uso: Visualizar Conteúdo	126
5.4.4.	Caso de Uso: Adicionar Álbum	127
5.4.5.	Caso de Uso: Apagar Álbum	128
5.4.6.	Caso de Uso: Fornecer Legenda	129
5.5.	Reorganização do Modelo de <i>Features</i> (opcional).....	132
5.6.	Configuração do produto no modelo SR.....	140
5.6.1.	Caso de Uso: Adicionar Foto	141
5.6.2.	Caso de Uso: Apagar Foto	141
5.6.3.	Caso de Uso: Visualizar Conteúdo	142
5.6.4.	Caso de Uso: Adicionar Álbum	142
5.6.5.	Caso de Uso: Apagar Álbum	143
5.6.6.	Caso de Uso: Fornecer Legenda	144
5.7.	Elaboração do Modelo de Configuração do Produto	145
5.8.	Discussão	148
5.9.	Considerações Finais	149
6.	Conclusão.....	150
6.1.	Contribuições.....	151
6.2.	Trabalhos Futuros	152
	Referências	154

Lista de Figuras

Figura 2.1 - Atividades Essenciais do Desenvolvimento de uma Linha de Produtos (figura retirada de [Clements, P. and Northrop, L., 2002])	9
Figura 2.2 – <i>Framework</i> de Linha de Produtos de Software (figura retirada de [Pohl, K. et al., 2005]).....	11
Figura 2.3 – Notação das <i>features</i> utilizadas no método FODA [Kang, K. et al., 1990]	15
Figura 2.4 – Notação das <i>features</i> utilizadas nesta dissertação, baseada em [Czarnecki, K. et al., 2005a]	15
Figura 2.5 – Exemplo de Modelo de <i>Features</i> (figura retirada de [Czarnecki et al., 2005b])	18
Figura 2.6 - Visão Geral do <i>framework</i> NFR	24
Figura 2.7 – Exemplo da decomposição do NFR Segurança	25
Figura 2.8 - Modelo de Objetivos (figura retirada de [Yu, Y. et al., 2008]).....	26
Figura 2.9 - Atores e relacionamentos entre atores [Grau, G. et al., 2009]	27
Figura 2.10 - Dependência entre atores	28
Figura 2.11 – Graus de Dependência em i^*	31
Figura 2.12 - Exemplo de Modelo de Dependência Estratégica para Agendamento de Reuniões (figura retirada de [Santander, V. F. A., 2002])	32
Figura 2.13 - Fronteira do Ator.....	33
Figura 2.14 - Ligação Meio-Fim	33
Figura 2.15 - Ligação de Decomposição de Tarefas	34
Figura 2.16 - Ligações de Composição	35
Figura 2.17 - Exemplo de Modelo de Razão Estratégica para o ator Iniciador de Reunião do sistema Agendamento de Reuniões (figura retirada de [Santander, V. F. A., 2002]).	36

Figura 3.1 - Modelo de <i>features</i> do sistema de comércio eletrônico.....	40
Figura 3.2 - Modelo de objetivos do sistema de comércio eletrônico	43
Figura 3.3 - Modelo PL-AOVGraph do sistema de comércio eletrônico.....	44
Figura 3.4 - Modelo i* aspectual do sistema de comércio eletrônico	48
Figura 3.5 - Modelo i* do sistema de comércio eletrônico	49
Figura 3.6 - Modelo i*-c do sistema de comércio eletrônico	54
Figura 3.7 – Metamodelo da linguagem i*-c (adaptado a partir de [Lucena, M. et al., 2008])	55
Figura 4.1 - Processo para extrair <i>features</i> a partir do modelo SR do i* em BPMN ...	62
Figura 4.2 – Especificação do caso de uso Adicionar Foto do sistema MobileMedia .	63
Figura 4.3 - Atores e seus principais objetivos.....	64
Figura 4.4 - Tarefa Adicionar Foto.....	65
Figura 4.5 - Tarefas do caso de uso Adicionar Foto.....	66
Figura 4.6 – Dependência de recurso entre as tarefas da tarefa Adicionar Foto	67
Figura 4.7 – Tarefas alternativas do caso de uso Adicionar Foto.....	69
Figura 4.8 - Modelagem de pré-condições, eventos de <i>trigger</i> da tarefa Adicionar Foto	70
Figura 4.9 – Modelagem do relacionamento <i>includes</i> no caso de uso Adicionar Foto	71
Figura 4.10 – Inclusão de objetivo entre a tarefa Adicionar Foto e o objetivo Gerenciamento de Mídia do ator sistema	73
Figura 4.11 – Modelo SR final do caso de uso Adicionar Foto	74
Figura 4.12 – Modelo SR do caso de uso Adicionar Foto com possíveis elementos de serem <i>features</i>	76
Figura 4.13 – Modelo SR com cardinalidade do caso de uso Adicionar Foto	79
Figura 4.14 – Modelo de <i>Features</i> com as <i>features</i> Mobile Media e Adicionar Foto.	84
Figura 4.15 – Modelo de <i>Features</i> com as <i>sub-features</i> Armazenar Foto e Fornecer Legenda	84

Figura 4.16 – Modelo de <i>Features</i> com as <i>sub-features</i> Salvar Automaticamente, Salvar Pelo Usuário, Foto, Local e Nome	85
Figura 4.17 – Modelo de <i>features</i> para o caso de uso Adicionar Foto	85
Figura 4.18 – Modelo de <i>Features</i> reorganizado do caso de uso Adicionar Foto	87
Figura 4.19 – Modelo SR do caso de uso Adicionar Foto com elementos que estarão presentes no modelo de configuração do produto	89
Figura 4.20 – Modelo de Configuração do caso de uso Adicionar Foto	90
Figura 5.1 – Tela do sistema MobileMedia da funcionalidade de adicionar foto	96
Figura 5.2 – Primeira atividade do processo G2FM: Construção do modelo SR.....	97
Figura 5.3 – Especificação original do caso de uso Apagar Foto.....	98
Figura 5.4 – Modelo SR do caso de uso Apagar Foto	99
Figura 5.5 – Especificação original do caso de uso Visualizar Conteúdo.....	101
Figura 5.6 – Modelo SR do caso de uso Visualizar Conteúdo	103
Figura 5.7 – Especificação original do caso de uso Adicionar Álbum.....	105
Figura 5.8 – Modelo SR do caso de uso Adicionar Álbum.....	108
Figura 5.9 – Especificação original do caso de uso Apagar Álbum.....	109
Figura 5.10 – Modelo SR do caso de uso Apagar Álbum	110
Figura 5.11 – Especificação original do caso de uso Fornecer Legenda.....	112
Figura 5.12 – Modelo SR do caso de uso Fornecer Legenda	114
Figura 5.13 – Segunda atividade do processo G2FM: Identificação de elementos com possibilidade de serem <i>features</i>	115
Figura 5.14 – Modelo SR do caso de uso Apagar Foto com possíveis elementos de serem <i>features</i>	116
Figura 5.15 – Modelo SR do caso de uso Visualizar Conteúdo com possíveis elementos de serem <i>features</i>	117
Figura 5.16 – Modelo SR do caso de uso Adicionar Álbum com possíveis elementos de serem <i>features</i>	117

Figura 5.17 – Modelo SR do caso de uso Apagar Album com possíveis elementos de serem <i>features</i>	118
Figura 5.18 – Modelo SR do caso de uso Fornecer Legenda com possíveis elementos de serem <i>features</i>	118
Figura 5.19 – Terceira atividade do processo G2FM: Reengenharia do Modelo SR.	119
Figura 5.20 – Modelo SR com cardinalidade do caso de uso Apagar Foto.....	119
Figura 5.21 – Modelo SR com cardinalidade do caso de uso Visualizar Conteúdo...	121
Figura 5.22 – Modelo SR com cardinalidade do caso de uso Adicionar Álbum.....	121
Figura 5.23 – Modelo SR com cardinalidade do caso de uso Apagar Álbum.....	122
Figura 5.24 – Modelo SR com cardinalidade do caso de uso Fornecer Legenda.....	122
Figura 5.25 – Quarta atividade do processo G2FM: Elaboração do Modelo de <i>Features</i>	123
Figura 5.26 - Modelo de <i>Features</i> do caso de uso Apagar Foto.....	126
Figura 5.27 – Modelo de <i>Features</i> do caso de uso Visualizar Conteúdo.....	127
Figura 5.28 – Modelo de <i>Features</i> do caso de uso Adicionar Álbum.....	128
Figura 5.29 – Modelo de <i>Features</i> do caso de uso Apagar Álbum	129
Figura 5.30 – Modelo de <i>Features</i> do caso de uso Fornecer Legenda	130
Figura 5.31 – Modelo de <i>Features</i> do sistema Mobile Media.....	131
Figura 5.32 – Quinta atividade do processo G2FM: Reorganização do Modelo de <i>Features</i>	132
Figura 5.33 – Operações com Foto e Álbum destacadas.....	133
Figura 5.34 – Primeira reorganização do Modelo de <i>Features</i> do sistema Mobile Media.....	134
Figura 5.35 – Modelo de <i>Features</i> com a <i>feature</i> Fornecer Legenda destacada várias vezes	136
Figura 5.36 – Segunda reorganização do modelo de <i>features</i> do sistema Mobile Media	137

Figura 5.37 – Modelo de <i>Features</i> com <i>features</i> com mesmo significado destacadas	138
Figura 5.38 – Modelo de <i>Features</i> reorganizado do sistema Mobile Media.....	139
Figura 5.39 – Sexta atividade do processo G2FM: Configuração do produto no modelo SR	140
Figura 5.40 - Modelo SR do caso de uso Apagar Foto com elementos que estarão presentes na configuração do produto	141
Figura 5.41 - Modelo SR do caso de uso Visualizar Conteúdo com elementos que estarão presentes na configuração do produto	142
Figura 5.42 - Modelo SR do caso de uso Adicionar Álbum com elementos que estarão presentes na configuração do produto	143
Figura 5.43 - Modelo SR do caso de uso Apagar Álbum com elementos que estarão presentes na configuração do produto	144
Figura 5.44 - Modelo SR do caso de uso Fornecer Legenda com elementos que estarão presentes na configuração do produto	145
Figura 5.45 – Última atividade do processo G2FM: Elaboração do Modelo de Configuração do produto	146
Figura 5.46 – Modelo de Configuração de um produto do MobileMedia.....	147
Figura 5.47 – Modelo de <i>Features</i> da <i>release</i> 1 do sistema MobileMedia original...	149

Lista de Tabelas

Tabela 2.1 – Conceito do Modelo de <i>Features</i> utilizadas nesta dissertação, baseado em [Czarnecki, K. et al., 2005b].....	17
Tabela 2.2 – Tipo de dependência entre atores.....	29
Tabela 3.1 – Mapeamento entre modelo de <i>features</i> e modelo de objetivos.....	42
Tabela 3.2 – Mapeamento entre modelo de <i>features</i> e modelo PL-AOVGraph.....	45
Tabela 3.3 – Mapeamento entre modelo de <i>features</i> e, modelo i* aspectual e i*	47
Tabela 3.4 – Equivalência entre modelo de <i>features</i> , modelo de objetivos, PL-AOVGraph, i* Aspectual e i*.....	50
Tabela 3.5 – Relação entre construtores do modelo de <i>features</i> e da linguagem i*-c..	52
Tabela 3.6 – Restrições em OCL dos elementos adaptados do metamodelo	57
Tabela 3.7 – Restrições em OCL para utilização da versão i* de [Grau, G. et al., 2009]	59
Tabela 4.1 - Cardinalidade para os modelos i*.....	78 77
Tabela 4.2 – Mapeamento entre a cardinalidade nos modelos i* e nos modelos de <i>features</i>	80 79
Tabela 4.3 – Template de tabela a ser preenchida com informações para o modelo de <i>features</i>	82 81
Tabela 4.4 – Informações dos elementos do caso de uso Adicionar Foto que estarão presentes no modelo de <i>features</i>	83
Tabela 4.5 – Resumo das heurísticas	91
Tabela 5.1 – Resumo das heurísticas H1 e H2	115
Tabela 5.2 – Resumo das heurísticas H3, H4, H5, H6 e H7.....	120
Tabela 5.3 – Resumo das heurísticas H8, H9, H10, H11 e H12.....	123
Tabela 5.4 – Mapeamento entre a cardinalidade nos modelos i* e nos modelos de <i>features</i>	124

Tabela 5.5 – Informações dos elementos do caso de uso Apagar Foto que estarão presentes no modelo de <i>features</i>	125
Tabela 5.6 – Informações dos elementos do caso de uso Visualizar Conteúdo que estarão presentes no modelo de <i>features</i>	126
Tabela 5.7 – Informações dos elementos do caso de uso Adicionar Álbum que estarão presentes no modelo de <i>features</i>	127
Tabela 5.8 – Informações dos elementos do caso de uso Apagar Álbum que estarão presentes no modelo de <i>features</i>	128
Tabela 5.9 – Informações dos elementos do caso de uso Fornecer Legenda que estarão presentes no modelo de <i>features</i>	129
Tabela 5.10 – Resumo das heurísticas H13 e H14	140

Lista de Abreviaturas

BPMN	<i>Business Process Modeling Notation</i>
DIS	<i>Detailed Interaction Script</i>
ELPS	Engenharia de Linhas de Produto de Software
ER	Engenharia de Requisitos
FODA	<i>Feature-Oriented Domain Analysis</i>
G2FM	<i>Goal To Feature Model</i>
GORE	Engenharia de Requisitos Orientada a Objetivos
I*-C	<i>I* with cardinality</i>
LPS	Linhas de Produtos de Software
NFR	Requisitos não-funcionais
OCL	<i>Object Constraint Language</i>
PRiM	<i>Process Reengineering i* Method</i>
SD	Modelo de Dependência Estratégica
SIG	<i>Softgoal Interdependency Graph</i>
SR	Modelo de Razão Estratégica
VP	Ponto de Variação

Capítulo

1.Introdução

Este capítulo apresenta as principais motivações e objetivos para a realização deste trabalho, além de apresentar a estrutura do documento.

1.1. Contextualização

A Engenharia de Requisitos (ER) é a fase do desenvolvimento de sistemas de software responsável pela identificação dos objetivos do sistema pretendido; pela operacionalização de tais objetivos em serviços e restrições; e pela atribuição da responsabilidade dos requisitos resultantes a agentes, tais como humanos, hardware e software [Lamsweerde, A., 2000]. A Engenharia de Requisitos Orientada a Objetivos (do inglês, *Goal-Oriented Requirements Engineering* ou GORE) tem crescido como uma forma promissora de descrever sistemas de software. Ela visa preencher as lacunas existentes no desenvolvimento de software, baseando-se nos objetivos dos *stakeholders*, de modo que o software a ser desenvolvido corresponda ao que realmente os *stakeholders* desejam [Lamsweerde, A., 2001]. Uma abordagem de requisitos orientada a objetivos que tem sido usada para o desenvolvimento de sistemas complexos é o *framework* i* [Yu, E., 1995]. I* fornece uma maneira natural de identificar e especificar tanto os interesses dos *stakeholders* como as características do sistema pretendido.

A ER tanto pode ser aplicada para desenvolver produtos de software únicos, como para desenvolver Linhas de Produtos de Software (LPS) [Pohl, K. et al., 2005]. Uma Linha de Produtos de Software é um conjunto de sistemas de software que compartilham um conjunto comum e gerenciado de *features* para satisfazer necessidades específicas de um segmento particular de mercado [Clements, P. and Northrop, L., 2002]. Uma *feature* pode ser vista como uma propriedade de sistema ou funcionalidade que é relevante para alguns *stakeholders* e usada para capturar características comuns e variáveis entre produtos de uma mesma família [Czarnecki, K. and Eisenecker, U., 2000].

Na Engenharia de Requisitos para Linhas de Produtos de Software, os modelos de *features* são utilizados para capturar similaridades e variabilidades de famílias de produtos [Kang, K. et al., 1990]. Entretanto, é um grande desafio estabelecer um relacionamento entre as *features* de uma aplicação e os objetivos dos *stakeholders*. Isto dificulta justificar porque uma determinada configuração de *features* é requerida para um sistema específico. Já as abordagens orientadas a objetivos podem ser usadas como uma forma efetiva para descobrir requisitos variáveis e comuns de uma LPS, além de reduzir os custos associados à configuração de um produto específico na família de produtos. Como resultado, algumas abordagens orientada a objetivos para modelar a variabilidade de requisitos têm sido propostas.

Esta dissertação concentra-se na Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software, mais especificamente o *framework* i*, permitindo estruturar os requisitos na fase inicial do desenvolvimento dos sistemas de acordo com as intenções dos *stakeholders* para uma família de produtos.

1.2. Motivação

As características de uma Linha de Produtos de Software precisam ser documentadas explicitamente para possibilitar o reuso estratégico dos seus artefatos. Na atividade de Engenharia de Requisitos, isto significa que além de capturar as variabilidades de requisitos, também é preciso relacionar os diferentes tipos de requisitos, tais como organizacionais, não-funcionais e funcionais, além de manter o rastreamento entre eles. Na atividade de LPS, a captura desta informação é feita usando os modelos de *features*, mas estes não capturam requisitos não-funcionais explicitamente e nem a influência positiva/negativa destes requisitos para alcançar configurações alternativas de uma aplicação específica. Isto motivou a utilização de uma abordagem GORE, especificamente a abordagem i*, para construção de modelos de *features*.

Apesar de abordagens orientadas a objetivos serem usadas: (i) para capturar *features* utilizando modelos mais significativos, (ii) para manter o rastreamento entre as *features* do sistema e suas motivações e (iii) para justificar a configuração de alternativas do sistema para realização dos objetivos dos *stakeholders*, a atividade de requisitos não é especializada para LPS e por isso, vários estudos vêm sendo realizados nesta área. Nos últimos anos, algumas abordagens orientadas a objetivos para requisitos em LPS vêm sendo propostas. Contudo, as abordagens pesquisadas (modelos de objetivos [Yu, Y. et al., 2008], PL-AOVGraph [Batista, T. et al., 2008], i* Aspectual [Alencar, F. et al., 2008] e i* [Yu, E., 1995]) focam em capturar, com a mesma expressividade, a variabilidade capturada pelos modelos de *features*. Porém, observamos que a expressividade destas propostas é limitada [Borba, C. and Silva, C., 2009] e que não foram encontrados processos bem definidos que ofereçam diretrizes para expressar a variabilidade capturada pelos modelos de *features*. Esse fato foi a principal motivação para o desenvolvimento da proposta da linguagem i*-c (*i* with cardinality*) e da abordagem G2FM (em inglês, *Goal To Feature Model*), contribuições principais desta dissertação de mestrado.

1.3. Objetivos

Motivados pelo problema exposto na seção anterior, as contribuições que este trabalho pretende oferecer foram divididas em objetivos gerais e específicos. O objetivo geral sintetiza o que pretendemos alcançar. Os objetivos específicos tornam o objetivo geral operacional e indica o que será realizado especificamente na pesquisa.

O objetivo geral desta dissertação é apresentar uma abordagem que permita a identificação e modelagem de *features* comuns e variáveis em LPS e a configuração de aplicações específicas, a partir de modelos orientados a objetivos, mais especificamente modelos *i**.

A partir desse objetivo geral, definimos os seguintes objetivos específicos:

- Estender o *framework i** para Linhas de Produtos de Software, adicionando cardinalidade aos modelos *i** para permitir a identificação e modelagem de *features* a partir de modelos orientados a objetivos. O *framework i** fornece um meio de identificar a variabilidade na fase inicial de requisitos, sendo possível contribuir com maior expressividade na Engenharia de Requisitos de LPS. Ele também guia na escolha das características do sistema pretendido, através dos relacionamentos de contribuição dos requisitos funcionais com relação à satisfação de requisitos não-funcionais;
- Apresentar uma abordagem que permita tanto a identificação quanto modelagem de *features* comuns e variáveis em Linhas de Produtos de Software;
- Utilizar requisitos não-funcionais para justificar a configuração de aplicações específicas de uma família de software, a partir de modelos *i**. A influência positiva/negativa dos requisitos não-funcionais permite alcançar configurações alternativas de uma aplicação na linha de produtos para realização dos objetivos dos *stakeholders*.

1.4. Metodologia

A pesquisa em Engenharia de Software pode ser conduzida através de quatro métodos: o científico, o de engenharia, o experimental e o analítico [Wohlin, C. et al., 2000] [Travassos, G. et al., 2002]. Este trabalho de pesquisa seguiu o método de engenharia que é

baseado na observação de soluções existentes, na identificação de problemas nessas soluções e na sugestão de abordagens para melhorar as soluções analisadas.

Inicialmente foi feito um estudo sobre as duas principais subáreas envolvidas neste trabalho: a Engenharia de Linhas de Produto de Software (ELPS) e a Engenharia de Requisitos Orientada a Objetivos (ou *Goal-Oriented Requirements Engineering* – GORE). Na área de Linhas de Produtos de Software [Clements, P. and Northrop, L., 2002] [Pohl, K. et al., 2005] foram apresentadas as principais atividades e o modelo de *features* [Kang, K. et al., 1990], artefato central usado nestas atividades e ao longo desta dissertação. Na área de Engenharia de Requisitos Orientada a Objetivos foram mostradas algumas abordagens existentes (*framework* NFR [Chung, L. K. et al., 2000], modelo de objetivos [Yu, Y. et al., 2008] e *framework* *i** [Yu, E., 1995]), com maior foco no *framework* *i** por ser a abordagem principal deste trabalho.

O passo seguinte foi fazer uma comparação da expressividade entre abordagens orientadas a objetivos para LPS (modelos de objetivos [Yu, Y. et al., 2008], PL-AOVGraph [Batista, T. et al., 2008], *i** Aspectual [Alencar, F. et al., 2008] e *i** [Yu, E., 1995]) e um modelo de *features* [Czarnecki, K. et al., 2005a]. Tendo feito isso, concluiu-se que essas três abordagens têm a expressividade limitada, pois não são capazes de representar a cardinalidade do modelo de *features*. O resultado desta comparação será publicado em [Borba, C. and Silva, C., 2009] e nos levou a estender o *framework* *i** para suportar cardinalidade e assim, permitir a identificação e modelagem de *features* a partir de modelos orientados a objetivos. Esta nova extensão do *i** foi denominada de ***i*-c*** (*i* with cardinality*).

Considerando os conhecimentos adquiridos e os resultados preliminares alcançados, definimos uma abordagem chamada **G2FM** (*Goal To Feature Model*) que permite capturar *features* comuns e variáveis em uma LPS a partir da linguagem ***i*-c***. Esta abordagem é a principal contribuição desta dissertação.

Em seguida, apresentamos uma comparação entre a abordagem **G2FM** e outra abordagem baseada em modelos *i** para Linhas de Produtos de Software (IStarLPS), ressaltando as suas principais diferenças.

Por fim, com o objetivo de avaliar o trabalho, realizamos um estudo de caso no qual aplicamos a nossa abordagem em uma LPS, chamada **MobileMedia** [Figueiredo, E. et al., 2008].

1.5. Organização da dissertação

Este trabalho encontra-se estruturado da seguinte forma:

Capítulo 2 – Engenharia de Linhas de Produtos de Software e Engenharia de Requisitos Orientada a Objetivos: Estado da Arte - apresenta o estado da arte das duas principais subáreas envolvidas neste trabalho, focando principalmente no modelo de *features* e no framework *i**.

Capítulo 3 – Adaptando modelos *i para suportar cardinalidade** - apresenta uma comparação entre trabalhos relacionados para modelar variabilidade em LPS com relação à expressividade fornecida pelos modelos de *features*. A partir desta comparação, é proposta uma extensão do *i**, chamada ***i*-c*** (*i* with cardinality*), para representar cardinalidade. Algumas restrições associadas a esta extensão são apresentadas, bem como o metamodelo desta nova linguagem.

Capítulo 4 – Processo para extrair *features* a partir do modelo *i** - apresenta em detalhes a abordagem **G2FM** (*Goal To Feature Model*), que permite capturar *features* comuns e variáveis em uma LPS a partir de uma extensão do modelo *i** (***i*-c***).

Capítulo 5 – Estudo de Caso: Mobile Media - apresenta a utilização da abordagem **G2FM** com um sistema de Linha de Produtos de Software, o **Mobile Media**.

Capítulo 6 – Conclusão - apresenta as conclusões finais acerca do trabalho apresentado e considerações para o desenvolvimento de trabalhos futuros.

Referências – são apresentadas as referências bibliográficas utilizadas para produção deste trabalho.

Capítulo

2. Engenharia de Linhas de Produtos de Software e Engenharia de Requisitos Orientada a Objetivos: Estado da Arte

Este capítulo discute o estado da arte das duas principais subáreas envolvidas neste trabalho: a Engenharia de Linhas de Produto de Software (ELPS) e a Engenharia de Requisitos Orientada a Objetivos (ou *Goal-Oriented Requirements Engineering* – GORE). Serão apresentadas as principais atividades realizadas para a ELPS e o modelo de *features*, artefato central usado nestas atividades. Em seguida mostraremos algumas abordagens GORE existentes, focando principalmente no *framework* i*. Por último, apresentamos as considerações finais.

2.1 Engenharia de Linhas de Produto de Software

À medida que as indústrias de software aumentam a sua capacidade de produção, cresce também a demanda por sistemas mais complexos e com mais qualidade [Kotonya, G. and Sommerville, I., 1998]. Desta forma, um diferencial competitivo entre as empresas de software é a habilidade para reagir rapidamente às mudanças de mercado, com agilidade e custos reduzidos, sem comprometer a qualidade. Isto tem se tornado possível através da Engenharia de Linhas de Produtos de Software (em inglês, *Software Product Line*) que se baseia na reutilização de software, construindo uma família de aplicações em vez de desenvolver produtos separadamente.

Uma Linha de Produtos de Software (ou simplesmente LPS) é um conjunto de sistemas de software que compartilham um conjunto comum e gerenciado de *features* para satisfazer necessidades específicas de um segmento particular de mercado [Clements, P. and Northrop, L., 2002]. A área de LPS emergiu como uma abordagem de desenvolvimento de software importante, viável e cujo objetivo não é apenas o reuso de código ou a integração de novos componentes [Clements, P. and Northrop, L., 2002], mas também a reconfiguração de arquiteturas, o reuso de requisitos e de outros artefatos. Segundo [Pohl, K. et al., 2005], os principais objetivos deste paradigma são:

- i. **Redução dos custos de desenvolvimento:** a reutilização de artefatos em diferentes tipos de sistemas implica em uma redução de custo para cada sistema, já que não é necessário desenvolver componentes do zero;
- ii. **Aumento da qualidade:** os *core assets* (ou artefatos de software) de uma LPS são reusados em muitos produtos, significando que foram testados e revisados várias vezes. Dessa forma, há uma chance maior de detecção de falhas e a correção das mesmas aumenta a qualidade dos produtos;
- iii. **Redução do tempo de entrega dos produtos:** inicialmente, o tempo de desenvolvimento de uma LPS é grande, pois, os *core assets* precisam ser desenvolvidos. Depois, este tempo é reduzido, já que muitos componentes previamente desenvolvidos podem ser reusados para cada novo produto.

Basicamente, a engenharia de uma linha de produtos de software define dois processos. O primeiro, também conhecido como **Engenharia de Domínio**, engloba a fase de desenvolvimento de *core assets*, onde são estabelecidas as similaridades e variabilidades da LPS. O segundo processo é responsável por derivar aplicações de uma linha de produtos baseados nos *core assets* pré-definidos. Este processo de desenvolvimento do produto é também conhecido como **Engenharia de Aplicação**. Segundo [Clements, P. and Northrop, L., 2002], estes dois processos são supervisionados pela atividade de gestão técnica e organizacional, também conhecida como **Gerenciamento**, conforme podemos ver na Figura 2.1.

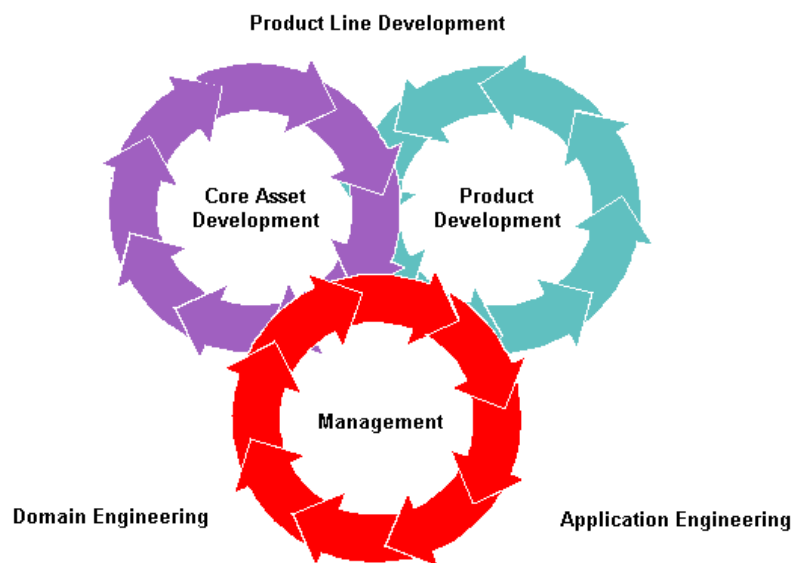


Figura 2.1 - Atividades Essenciais do Desenvolvimento de uma Linha de Produtos (figura retirada de [Clements, P. and Northrop, L., 2002])

Cada círculo da figura representa uma das atividades essenciais do desenvolvimento de uma linha de produtos. Todas elas estão relacionadas e indicam um constante movimento, mostrando que são altamente iterativas e podem ocorrer em qualquer ordem.

- **Desenvolvimento dos *core assets*:** essa atividade pertence à Engenharia de Domínio. Exemplos de *core assets* são: arquiteturas, componentes, planos de testes, casos de testes, documentação de projeto, especificação de requisitos, modelos de domínio, etc. Essa atividade é descrita em mais detalhes a seguir, na Seção 2.1.1.
- **Desenvolvimento dos produtos:** essa atividade pertence à Engenharia de Aplicação e depende tanto dos artefatos gerados na atividade anterior como dos requisitos do produto a ser desenvolvido. Essa atividade é descrita em mais detalhes a seguir, na Seção 2.1.2.
- **Gerenciamento:** essa atividade tem um papel fundamental no sucesso do desenvolvimento de uma linha de produtos. O gerenciamento pode ser tanto técnico como organizacional. O gerenciamento técnico engloba o desenvolvimento dos *core assets* e do produto; já o gerenciamento organizacional trata da própria organização.

Muitas empresas iniciam o desenvolvimento de uma linha de produtos a partir dos *core assets* (abordagem pró-ativa). Outras desenvolvem um ou mais produtos para então, extrair os *core assets* (abordagem reativa). Ambas as abordagens compartilham um dos grandes benefícios da linha de produtos: todos os membros da linha de produtos são baseados nos mesmos *core assets*.

De acordo com [Clements, P. and Northrop, L., 2002], a Linha de Produtos de Software permite que a organização ganhe vantagem competitiva devido ao aumento da produtividade em larga escala, aumento da qualidade, redução dos custos, redução do tempo de entrega e minimização de riscos dos produtos. Contudo, iniciar uma linha de produtos de software é uma decisão de negócio delicada, pois a organização deve estabelecer objetivos sólidos e específicos para adotar essa estratégia de reuso. Inicialmente, os custos da linha de produtos são um pouco maiores do que os benefícios, pois muitos dos custos estão associados com a sua implantação. Os benefícios, por outro lado, aumentam a cada entrega de produto. Uma vez estabelecida a abordagem, a produtividade da organização acelera e, então, os benefícios superam os custos.

Na Figura 2.2, encontra-se representada a abordagem de LPS segundo [Pohl, K. et al., 2005]. Os subprocessos de engenharia de domínio e de aplicação como também as suas atividades estão descritos em detalhes nas subseções seguintes.

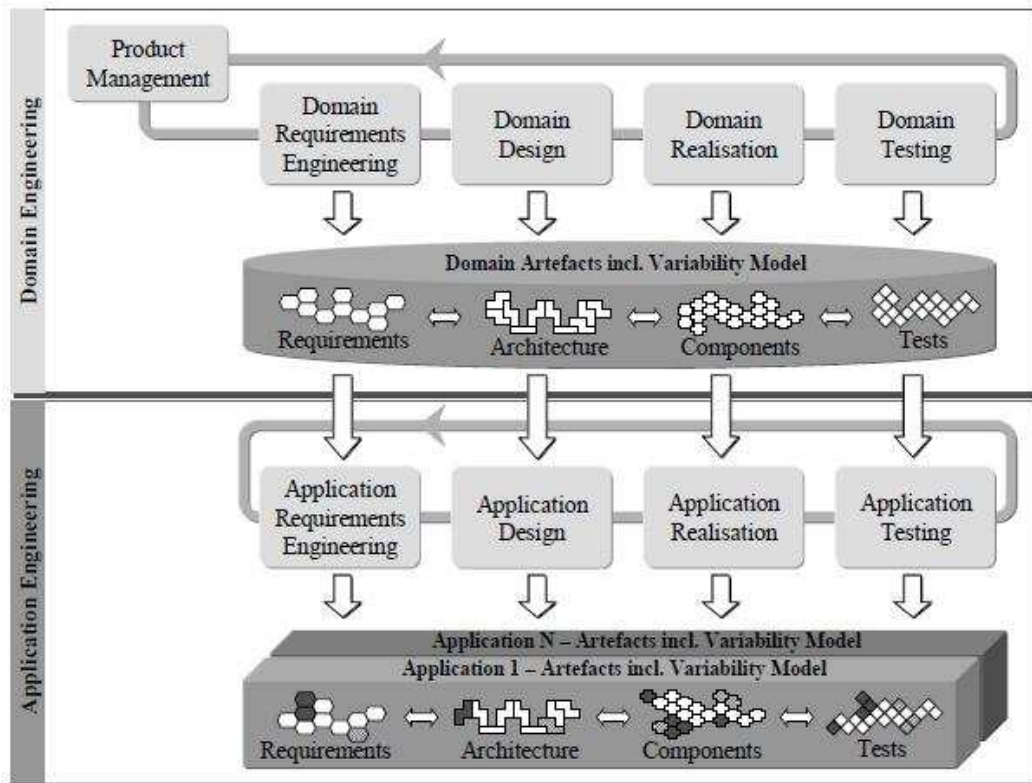


Figura 2.2 – Framework de Linha de Produtos de Software (figura retirada de [Pohl, K. et al., 2005])

2.1.1. Engenharia de Domínio

De acordo com [Pohl, K. et al., 2005], a Engenharia de Domínio é o processo de LPS onde os pontos comuns e variáveis da linha de produtos são definidos. Essas semelhanças e diferenças dos produtos de uma LPS são importantes para a análise do domínio. Os subprocessos da Engenharia de Domínio são descritos resumidamente a seguir, de acordo com a Figura 2.2

- **Gerenciamento de Produto (*Product Management*):** tem como principal objetivo a integração do desenvolvimento, da produção e do *marketing* de produtos para maximizar o investimento e satisfazer às necessidades dos clientes. Esta atividade utiliza técnicas objetivas para definir o que faz parte e o que não faz do escopo de uma linha de produtos;
- **Engenharia de Requisitos de Domínio (*Domain Requirements Engineering*):** nesta atividade pretende-se identificar os requisitos e para isso é feita a análise de pontos comuns e variáveis entre produtos da LPS;

- **Projeto de Domínio (*Domain Design*):** é a arquitetura de referência da linha de produtos que contém os pontos de variação, a plataforma de suporte e a produção em larga escala de bens adequados às necessidades individuais dos clientes. Essa arquitetura de referência fornece uma estrutura de alto nível, comum a todas as aplicações da linha de produtos;
- **Realização de Domínio (*Domain Realisation*):** nesta atividade são tratados os detalhes do projeto e da implementação dos componentes reutilizáveis de software;
- **Testes de Domínio (*Domain Testing*):** consiste em validar e verificar os componentes reutilizáveis, evitando que os erros sejam propagados para o processo de testes de aplicação.

Os resultados destes cinco subprocessos supracitados são importantes para a eficiente realização da Engenharia de Aplicação.

2.1.2. Engenharia de Aplicação

Segundo [Pohl, K. et al., 2005], a Engenharia de Aplicação é o processo de LPS onde as aplicações são construídas através da reutilização dos artefatos de domínio e pela exploração da variabilidade da linha de produtos. Esses artefatos de domínio são os resultados dos vários subprocessos da Engenharia de Domínio, vistos anteriormente. Os subprocessos da Engenharia de Aplicação são descritos resumidamente a seguir, de acordo com a Figura 2.2.

- **Engenharia de Requisitos de Aplicação (*Application Requirements Engineering*):** tem como objetivo a identificação dos requisitos dos *stakeholders*¹ para a aplicação. É necessário fazer um mapeamento entre os requisitos da aplicação e os requisitos resultantes do processo de Engenharia de Domínio. A principal preocupação desta atividade é detectar diferenças entre requisitos de aplicação e os produzidos na engenharia de requisitos de domínio, para que se possa reutilizar o máximo possível dos requisitos de domínio. É feita a documentação que contém o modelo de variabilidade de domínio com os dados correspondentes aos pontos que a aplicação reutiliza;

¹ São pessoas ou organizações que serão afetadas pelo sistema e tem influência, direta ou indireta, sobre os requisitos do sistema – usuários finais, gerentes e outros envolvidos no processo organizacional influenciados pelo sistema, engenheiros

- **Projeto de Aplicação (*Application Design*):** este subprocesso contém as atividades de produção da arquitetura da aplicação que é semelhante ao projeto de software de um sistema único. A arquitetura da aplicação determina toda a estrutura de uma aplicação específica e tem que satisfazer os requisitos dessa aplicação.
- **Realização de Aplicação (*Application Realisation*):** são tratados todos os detalhes de projeto de implementação e configuração de componentes. É produzida a aplicação desejada de acordo com a seleção e configuração dos componentes de software reutilizáveis e as características específicas da aplicação. Após a junção de todos os componentes, a aplicação pode ser testada;
- **Testes de Aplicação (*Application Testing*):** consistem em atividades necessárias para validar e verificar uma aplicação de acordo com a sua especificação. É necessário definir um conjunto de testes que se apliquem adequadamente à aplicação a ser testada. Também é útil ter uma documentação dos testes para que se possa repetir o processo de teste da aplicação.

Para produzir uma linha de produtos é importante analisar as similaridades e variabilidades de forma a produzir produtos com as mesmas características base, mas com diferenças que ajudam a dar maior resposta às necessidades individuais dos clientes. As similaridades e variabilidades podem ser capturadas com modelos de *features* [Kang, K. et al., 1990]. Esses modelos são apresentados mais detalhadamente na próxima seção.

2.1.3. Modelo de *Features*

Modelos de *features* representam as similaridades e variabilidades numa linha de produtos de software [Kang, K. et al., 1990] e também podem ser usados para definir os *core assets* dos produtos para usuários individuais.

Os modelos de *features* são desenvolvidos no subprocesso de Engenharia de Requisitos de Domínio, quando se faz a identificação e análise dos requisitos da LPS e, também são utilizados no subprocesso de Engenharia de Requisitos de Aplicação, quando se faz a seleção das *features* para uma aplicação específica.

Originalmente, o modelo de *features* foi proposto como parte do método FODA (do inglês, *Feature-Oriented Domain Analysis*) [Kang, K. et al., 1990]. Esse método propõe um modelo de *features* para linhas de produtos e desde a sua publicação em 1990, vem sendo adotado e estendido em outras abordagens. Por exemplo, algumas organizações definem variações deste método fornecendo mais detalhes na análise de domínio (ex. FORM [Kang, K. et al., 1998]), enquanto que outras adicionam cardinalidade ao modelo de *features* (do inglês, *cardinality-based feature modeling*) [Czarnecki, K. et al., 2005a].

2.1.3.1. Conceito do Modelo de *Features*

Uma *feature* pode ser vista como uma propriedade de sistema ou funcionalidade que é relevante para alguns *stakeholders* e usada para capturar características comuns e variáveis entre produtos de uma mesma família [Czarnecki, K. and Eisenecker, U., 2000]. As *features* estão organizadas num diagrama, geralmente na forma de uma árvore com a raiz, representando um conceito (e.g. um sistema de software), que pode ser refinada em *features* e estas em *sub-features*. Um modelo de *features* é constituído por um ou mais diagramas de *features* que as organiza hierarquicamente.

Existem diversas notações que representam os modelos de *features*. Em FODA [Kang, K. et al., 1990], uma *feature* do tipo opcional é representada com o círculo aberto, enquanto uma *feature* do tipo obrigatória não tem círculos. As decomposições deste método podem ser: *and* (todas as *sub-features* devem estar presentes em um produto concreto) e *xor* (ou-exclusivo), representadas com traços e arcos, respectivamente. A Figura 2.3 apresenta a notação de *features* de acordo com o método FODA [Kang, K. et al., 1990].

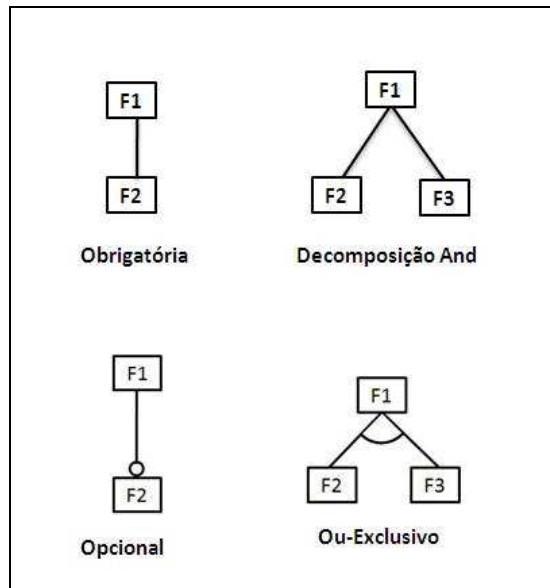


Figura 2.3 – Notação das *features* utilizadas no método FODA [Kang, K. et al., 1990]

Nessa dissertação é utilizada a notação de *features* baseada em cardinalidade [Czarnecki, K. et al., 2005a] onde uma *feature* do tipo obrigatória é representada com círculo fechado, uma *feature* do tipo opcional é representada com círculo aberto e *features* agrupadas são representadas com arcos. No caso de ser agrupada, pode-se ainda fazer a separação entre *or* (ou) ou *xor* (ou-exclusivo), de acordo com a Figura 2.4.

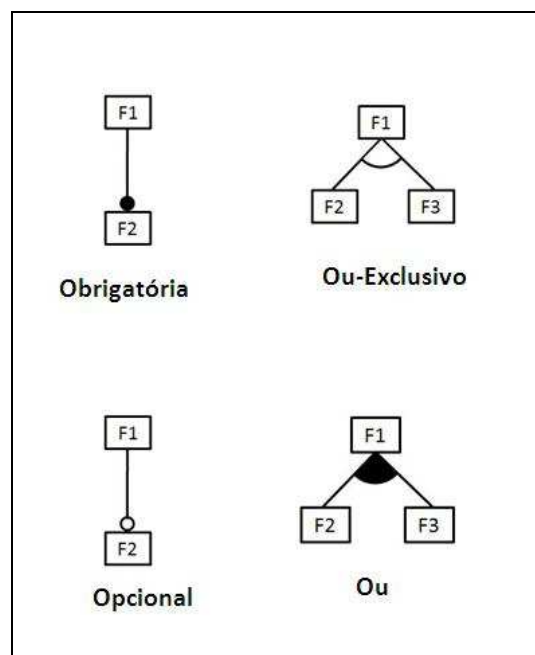
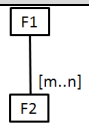
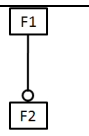
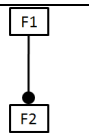
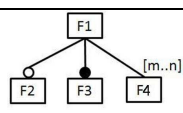
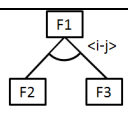
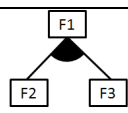
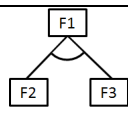


Figura 2.4 – Notação das *features* utilizadas nesta dissertação, baseada em [Czarnecki, K. et al., 2005a]

Segundo [Czarnecki, K. et al., 2005b], *features* podem ser representadas com cardinalidade, i.e., [1..*] ou [3..3], onde cardinalidade de uma *feature* significa um intervalo que denota a quantidade de vezes que uma *feature*, com as suas *sub-features*, podem estar presentes em uma determinada aplicação. *Features* obrigatórias e opcionais são consideradas casos especiais de *features* com cardinalidades [1..1] e [0..1], respectivamente. A cardinalidade de um grupo de *features* significa um intervalo de quantas *features* podem ser selecionadas do grupo e são representadas da seguinte forma: <i..j>, onde i é a quantidade mínima de *features* selecionadas e j, a quantidade máxima. As *features* agrupadas do tipo *or* ou *xor* são representadas com cardinalidade <1..k> e <1..1>, respectivamente. A Tabela 2.1 apresenta a descrição e a representação para cada um dos tipos de *features*.

A Figura 2.5 apresenta um modelo de *features* com cardinalidade de um perfil de segurança de um sistema operacional. A *feature* raiz (**SecurityProfile**) tem duas *sub-features*: **passwordPolicy** e **permissionSet(String)**. O símbolo com círculo preenchido indica que a *sub-feature* **passwordPolicy** é obrigatória (cardinalidade [1..1]), significando que a *feature* deve existir uma vez e apenas uma vez em cada aplicação. Esta *feature* tem duas *sub-features* também obrigatórias, **expiration** e **chars**. O símbolo da ramificação vazia sob a *feature* **expiration** indica que o grupo de *features* **inDays(Int)** e **never** tem cardinalidade <1-1>, significando que apenas uma dessas *sub-features* agrupadas estará presente em um determinado produto. Já o símbolo da ramificação vazia com a cardinalidade <2-4> sob a *feature* **chars** mostra um grupo de *features* (**lowerCase**, **upperCase**, **digit** e **specialChar**), onde no mínimo 2 e no máximo 4 *sub-features* existirão na configuração de um produto. A cardinalidade da *feature* **permissionSet(String)** é [0..*], significando que pode existir nenhuma ou muitas ocorrências desta *feature* no produto. Esta *feature* tem todas as suas *sub-features* obrigatórias: **fileIO**, **fileDialog** e **environmentVariables**. O símbolo da ramificação vazia sob a *feature* **fileIO** indica que o grupo de *features* **unrestricted** e **restricted** tem cardinalidade <1-1>. A *feature* **fileDialog** tem duas *sub-features*: **open** e **close**. O símbolo com círculo vazio indica que **open** e **close** são *features* opcionais (cardinalidade [0..1]), i.e., elas podem estar presentes ou não na aplicação. A *feature* **restricted** tem uma *sub-feature* **filepath(String)** com cardinalidade [0..*].

Tabela 2.1 – Conceito do Modelo de *Features* utilizadas nesta dissertação, baseado em [Czarnecki, K. et al., 2005b]

Conceito	Significado	Explicação
	Feature com cardinalidade [m..n]	Cardinalidade quando inserida em <i>features</i> tem a notação de colchetes; por exemplo, uma cardinalidade [m..n] indica que no mínimo m e no máximo n ocorrências de F2 devem estar presentes em um produto concreto.
	Feature com cardinalidade [0..1] (opcional)	Uma <i>feature</i> opcional pode ou não estar presente em um produto concreto. A cardinalidade é [0..1].
	Feature com cardinalidade [1..1] (obrigatória)	Uma <i>feature</i> obrigatória deve estar presente exatamente uma vez em um produto concreto. A cardinalidade é [1..1].
	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005]	Neste tipo de relação, as <i>sub-features</i> possuem diferentes tipos de relações.
	Features agrupadas com cardinalidade <i-j>	Uma cardinalidade de grupo é um intervalo inserido em um grupo de <i>features</i> que indica quantas <i>features</i> podem ser selecionadas do grupo em um produto concreto.
	Features agrupadas com cardinalidade <1-k>, onde k= tamanho do grupo (ou-inclusivo)	Uma cardinalidade de grupo <1..k> indica que pelo menos uma e no máximo k <i>features</i> podem ser selecionadas do grupo e que k é o número total de <i>features</i> no grupo.
	Features agrupadas com cardinalidade <1-1> (ou-exclusivo)	Uma cardinalidade de grupo <1..1> indica que exatamente uma <i>feature</i> pode ser selecionada do grupo.

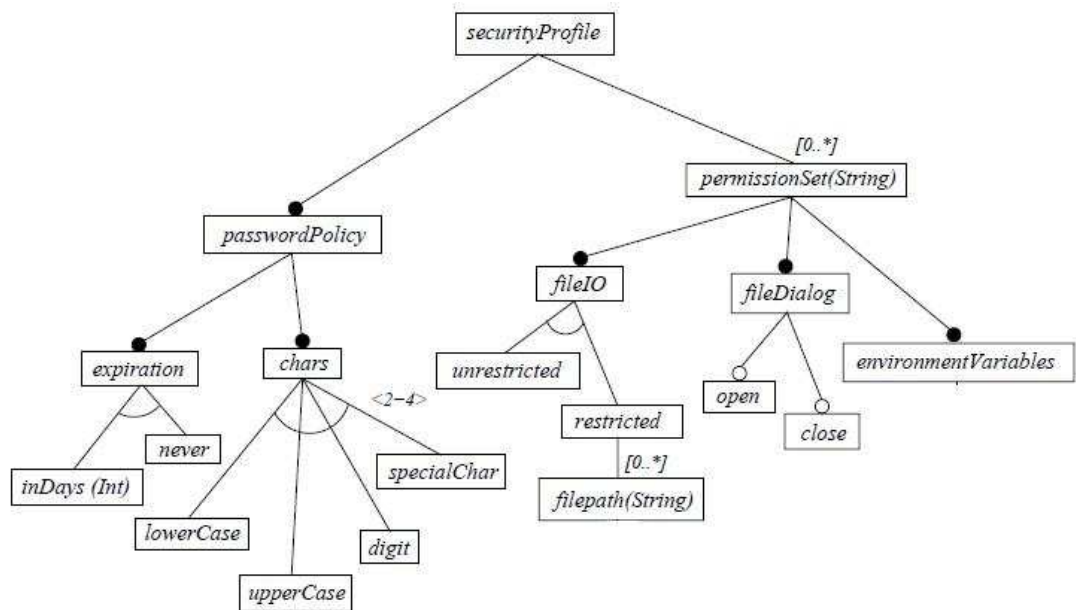


Figura 2.5 – Exemplo de Modelo de *Features* (figura retirada de [Czarnecki et al., 2005b])

Modelos de *features* não capturam requisitos não-funcionais explicitamente e nem a influência positiva/negativa destes requisitos para alcançar configurações alternativas de uma aplicação na LPS. Essa influência pode ajudar na escolha de uma configuração específica para uma aplicação alcançar os objetivos da organização. A captura dessa informação se torna possível com a utilização de abordagens orientadas a objetivos em LPS, pois, essas abordagens ajudam a melhorar a semântica dos modelos de *features* e a justificar a configuração de um determinado produto.

Nesta dissertação utilizamos modelos de *features* para representar as *features* comuns e variáveis de uma LPS identificadas a partir de modelos orientados a objetivos, mais especificamente modelos *i**, apresentados na próxima seção.

2.2. Engenharia de Requisitos Orientada a Objetivos

Nos primeiros estudos empíricos na área de Engenharia de Requisitos, foi observado que existem muitos requisitos inadequados, inconsistentes, incompletos ou ambíguos, e que eles têm um grande impacto na qualidade do software final [Bell, T. E. and Thayer, T. A., 1976].

Em [Boehm, B., 1981] foi estimada que a correção de erros de requisitos, depois do software implementado, pode custar 200 vezes mais do que correções realizadas durante a Engenharia de Requisitos. Estudos recentes comprovam que os problemas relacionados a não adoção de um processo de Engenharia de Requisitos acarreta em falhas, atrasos e altos custos de desenvolvimento de software [STA, 1995]. De acordo com um estudo realizado na Europa [ESI, 1996] com mais de 3800 organizações de 17 países, 50% dos problemas de desenvolvimento de software estão na área de especificação de requisitos e os outros 50% no gerenciamento de requisitos. Portanto, pode-se considerar que a Engenharia de Requisitos é uma das fases mais importantes do processo de Engenharia de Software [Lamsweerde, A., 2000]. Nos últimos anos, ela também tem sido explorada no contexto de LPS.

2.2.1. Engenharia de Requisitos – Uma visão Geral

A fim de melhorar o entendimento sobre requisitos de software, inicialmente é necessário definir o que é a Engenharia de Requisitos. Segundo [Lamsweerde, A., 2000], a Engenharia de Requisitos é a fase do desenvolvimento de sistemas de software responsável pela identificação dos objetivos do sistema pretendido, pela operacionalização de tais objetivos em serviços e restrições, e pela atribuição da responsabilidade dos requisitos resultantes para agentes como humanos, hardware e software.

De acordo com [Kotonya, G. and Sommerville, I., 1998], Engenharia de Requisitos trata-se de um termo para cobrir todas as atividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos, num sistema baseado em computador. Nesta definição, o termo engenharia implica em técnicas sistemáticas e repetitivas que são usadas para assegurar que os requisitos do sistema sejam completos, consistentes e relevantes.

Atualmente, uma das definições para Engenharia de Requisitos mais aceita é a encontrada em [Zave, P., 1997]. Segundo ela, a Engenharia de Requisitos é o ramo da engenharia de software preocupada com objetivos do mundo real e como funcionalidades e restrições em sistemas de software devem ser projetadas e implementadas para satisfazer esses objetivos. Ela também está preocupada com o relacionamento destes fatores para uma especificação precisa do comportamento do software e sua evolução ao longo do tempo e através de famílias de software. Esta será a definição utilizada neste trabalho.

Outra definição necessária para o correto entendimento de nosso trabalho é a de requisitos. Várias definições têm sido usadas para requisitos de software. Elas representam pers-

pectivas diferentes e graus variados de detalhes e precisão. Segundo a IEEE [IEEE-610, 1994], um requisito pode ser tratado como:

- i. Uma condição ou capacidade necessária para um usuário resolver um problema;
- ii. Uma condição ou capacidade que deve ser atendida por um sistema ou um componente de sistema para satisfazer um contrato, padrão ou especificação, ou outro documento formalmente imposto;
- iii. Uma representação documentada de uma condição ou capacidade, conforme os itens (i) e (ii).

A definição da IEEE cobre a visão do usuário sobre requisitos (comportamento externo do sistema), a visão do desenvolvedor (ii) e o conceito fundamental que requisitos devem ser documentados (iii).

Outra definição sugere que um requisito é uma necessidade do usuário ou uma característica, função ou atributo necessário do sistema que pode ser percebido de uma posição externa daquele sistema [Davis, A. M., 1993]. Essa definição enfatiza “o quê” o sistema deve conter.

A definição adotada neste trabalho será a encontrada em [Kotonya, G. and Sommerville, I., 1998] que descreve um requisito como:

- Uma facilidade no nível do usuário;
- Uma propriedade muito geral do sistema;
- Uma restrição específica no sistema;
- Uma restrição no desenvolvimento do sistema.

Para compreender e modelar requisitos é necessário estabelecer os tipos dos requisitos. Em [Kotonya, G. and Sommerville, I., 1998], os requisitos podem ser classificados como funcionais, não-funcionais ou organizacionais.

Requisitos funcionais definem o que o sistema deve fazer. São aqueles requisitos que capturam o comportamento pretendido pelo sistema. Esse comportamento pode ser expresso como serviços, tarefas ou funções que o sistema precisa executar.

Requisitos não-funcionais (NFRs) são as restrições nas funções oferecidas pelo sistema. Incluem restrições de tempo, restrições no processo de desenvolvimento, padrões e quali-

dades de um software: como custo, desempenho, confiabilidade, manutenibilidade, portabilidade, usabilidade, dentre outras.

Requisitos organizacionais dizem respeito às metas da empresa, às suas políticas estratégicas, aos relacionamentos entre os seus atores junto com seus respectivos objetivos.

2.2.2. Processo de Engenharia de Requisitos

Um processo de Engenharia de Requisitos é um conjunto estruturado de atividades que são seguidas para entregar, validar e manter os documentos de requisitos de um sistema. Atividades deste processo incluem elicitação de requisitos, análise e negociação de requisitos, documentação de requisitos, validação de requisitos e gerenciamento de requisitos [Kotonya, G. and Sommerville, I., 1998]. A descrição completa de um processo deve incluir quais atividades são realizadas, a estruturação e agendamento das atividades, quem serão os responsáveis por cada atividade, as entradas e saídas das atividades, e ferramentas a serem usadas. Essas atividades não são necessariamente sequenciais. Tipicamente o processo começa com a elicitação de requisitos, passando à análise e negociação. Posteriormente, os requisitos são documentados e validados. O gerenciamento de requisitos ocorre ao longo de todas as outras atividades e tem como objetivo controlar a evolução e as mudanças nos requisitos.

Segundo [Kotonya, G. and Sommerville, I., 1998], um documento de requisitos é uma declaração oficial dos requisitos do sistema para clientes, usuários finais e desenvolvedores do sistema. Os documentos de requisitos podem conter desde descrições em linguagem natural até modelos do sistema. Os modelos são uma representação abstrata de objetos reais onde detalhes são simplificados para facilitar o entendimento [Bezivin, J., 2005].

2.2.3. Abordagens Orientada a Objetivos

O conceito de *goal* (em português, significa objetivo ou meta) vem sendo utilizado em muitas áreas da Ciência da Computação: em Inteligência Artificial, é utilizado no planejamento para descrever os estados desejáveis do mundo; em Engenharia de Software, é usado para modelar requisitos funcionais e não-funcionais. Neste trabalho adotaremos a palavra objetivo como sinônimo de *goal*.

A Engenharia de Requisitos Orientada a Objetivos (do inglês, *Goal-Oriented Requirements Engineering* ou GORE) visa preencher as lacunas existentes no desenvolvimento de

software, baseando-se nos objetivos dos *stakeholders*, de modo que o software a ser desenvolvido corresponda ao que realmente os *stakeholders* desejam. Segundo [Lamsweerde, A., 2001], um objetivo é aquilo que o sistema deverá atingir.

As técnicas de GORE destacam-se devido ao fato de cobrirem tanto os Requisitos Funcionais, como também os Requisitos Não-Funcionais (do inglês, *Non-Functional Requirements* – NFRs) [Chung, L. K. et al., 2000]. Requisitos Funcionais são as funcionalidades do sistema e os requisitos não-funcionais são requisitos qualitativos (como segurança, confiabilidade, disponibilidade, tempo de resposta), ou seja, é desejável que o sistema tenha, mas não é especificado como esses requisitos serão satisfeitos. Às vezes, conflitos são gerados com a escolha de dois ou mais requisitos não-funcionais incompatíveis na sua satisfação simultânea (por exemplo, deseja-se que o sistema seja seguro como também tenha um tempo de resposta reduzido).

A principal preocupação da GORE é conhecer as propriedades do sistema que os *stakeholders* desejam, de modo a obter resultados mais adequados às suas necessidades. Nesta abordagem, o maior interesse está no “Por que” da existência das tarefas e a razão pela qual uma informação é mais estável, em vez do “Como” uma tarefa pode ser executada.

Segundo [Lapouchnian, A., 2005], um modelo de objetivos captura a variabilidade no domínio do problema através do refinamento de objetivos alternativos e atribuições de responsabilidades alternativas. Tendo em conta que as abordagens orientadas a objetivos fornecem um modo natural de identificar a variabilidade na fase inicial de requisitos, é possível encontrar variações de um mesmo sistema e beneficiar-se dessas diferenças pontuais. Dessa forma, essas abordagens podem ser usadas como uma forma efetiva de: (i) descobrir os requisitos comuns e variáveis de uma LPS e, (ii) reduzir tempo e custos associados com a configuração de um produto específico numa família de produtos, características essenciais para o desenvolvimento desse trabalho e para a escolha de uma abordagem GORE para LPS.

À medida que o uso de objetivos cresceu na Engenharia de Requisitos, diversas abordagens foram propostas, como por exemplo, KAOS [Lamsweerde, A., 2003], V-graph [Yu, Y. et al., 2004], *framework* NFR [Chung, L. K. et al., 2000], modelo de objetivos [Yu, Y. et al., 2008] e *framework* i* [Yu, E., 1995]. Neste trabalho, serão apresentadas as três últimas com maior destaque no *framework* i* que fará parte da abordagem a ser proposta.

2.2.3.1. *Framework* NFR

O *framework* NFR (*Non-Functional Requirements Framework*) foi proposto por [Chung, L. K. et al., 2000] e é baseado em objetivos usados para modelar e analisar requisitos não-funcionais. Os principais conceitos envolvidos nesta abordagem são:

- *Softgoal*² – é o conceito central do *framework*, e representa um objetivo que não possui uma definição clara nem precisa de como é satisfeito. São classificados como satisfeitos ou rejeitados, e essa decisão é avaliada de acordo com a satisfação e contribuição dos seus descendentes. Os *softgoals* são usados para representar os requisitos não-funcionais e são interdependentes uns dos outros. As relações de dependência são úteis para entender como um *softgoal* pode ser satisfeito.
- Catálogos – é um artefato que é tanto consumido quanto produzido como saída pelo uso do *framework*. Um catálogo armazena conhecimento acumulado em experiências prévias e é uma fonte de informação sobre NFRs.
- SIG (*Softgoal Interdependency Graph*) – é um grafo que representa o inter-relacionamento entre os requisitos não-funcionais. O grafo é inspirado na estrutura de árvore E/OU para solução de problemas. Os *softgoals* são conectados por links de interdependência (positivas ou negativas), onde um *softgoal* é refinado em outros *softgoals*. À medida que os *softgoals* são refinados, deve-se decidir aceitar ou não as possíveis operacionalizações obtidas no grafo, ou seja, decidir se os *softgoals* estão suficientemente detalhados a ponto de se tomar decisões sobre o projeto.

A Figura 2.6 mostra a visão geral do *framework* NFR:

² Não existe tradução aceitável para português, por esse motivo o termo será mantido do original

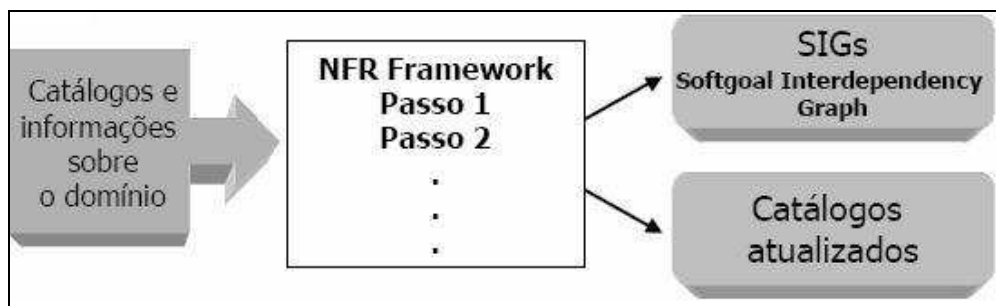


Figura 2.6 - Visão Geral do *framework* NFR

A utilização do *framework* NFR [Chung, L. K. et al., 2000] consiste em uma série de passos, através dos quais os requisitos não-funcionais são identificados, refinados, correlacionados com outros requisitos e operacionalizados. Esses passos ajudam a decompor os requisitos não funcionais, além de tratar ambigüidades e outros conflitos de especificação. Considera-se que o *framework* possui três fases:

- i. A criação prévia de catálogos que visam expressar o conhecimento sobre requisitos não-funcionais;
- ii. A definição dos gráficos relacionados com o problema em questão, a saída é um grafo SIG;
- iii. A seleção de alternativas e avaliação do impacto das decisões no problema sendo tratado, avaliando as contribuições e operacionalizações.

[Chung, L. K. et al., 2000] divide a decomposição dos NFRs 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 NFR. No caso do NFR **Segurança**, por exemplo, **Integridade**, **Confidencialidade** e **Disponibilidade** podem ser possíveis decomposições deste objetivo. A Figura 2.7, extraída de [Chung, L. K. et al., 2000], mostra um exemplo da decomposição do NFR **Segurança** aplicado à conta corrente como parte de um estudo de caso de automação bancária. Neste exemplo, além das possíveis decomposições **Integridade**, **Confidencialidade** e **Disponibilidade**, percebe-se ainda, que o *softgoal* **Integridade** é decomposto em **Compleitude** e **Precisão**. Os outros *softgoals* não são aqui decompostos para tornar o grafo simples de ser visualizado ao nível de exemplo. Em uma abordagem completa todas seriam decompostas.

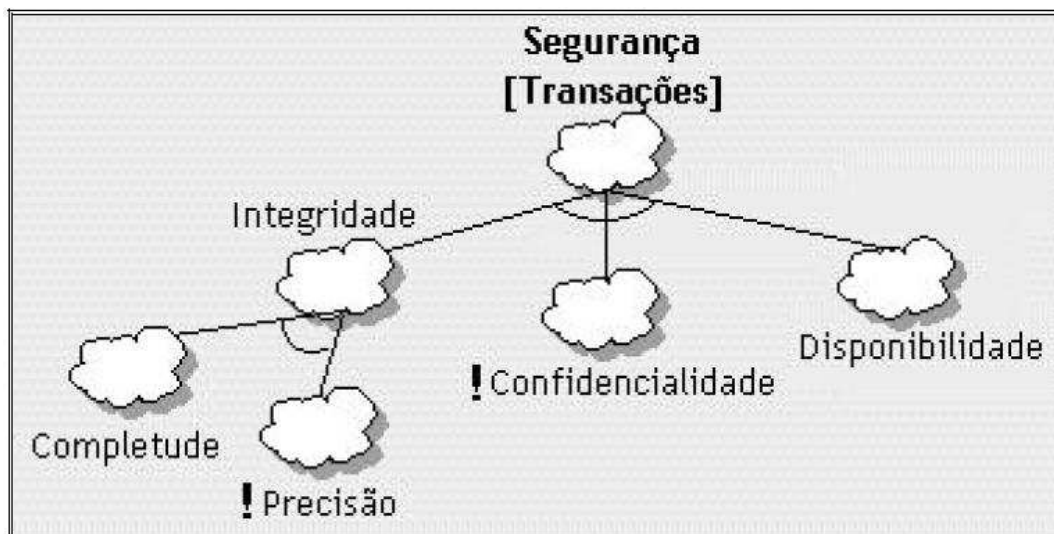


Figura 2.7 – Exemplo da decomposição do NFR Segurança

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

O *framework* não mostra exatamente como são identificados os requisitos não-funcionais, apenas sugere que eles devem ser obtidos através do conhecimento sobre o domínio para o qual o sistema será desenvolvido. O *framework* visa clarificar o significado dos requisitos não-funcionais e fornecer alternativas para satisfazê-los ao nível mais alto.

2.2.3.2. Modelos de Objetivos

A análise tradicional de objetivo consiste na decomposição de objetivos em sub-objetivos através de árvores de decomposição AND/OR. Objetivos podem ser satisfeitos ou não satisfeitos. Quando um objetivo G possui uma decomposição AND em G1, G2,..., Gn, então, G é satisfeita se, e somente se, Gi é satisfeita para todos os i's. Se G é uma decomposição OR, então G é satisfeita se, e somente se, existe um i tal que Gi seja satisfeito.

No modelo de objetivos apresentando em [Yu, Y. et al., 2008], um objetivo é refinado em um número de sub-objetivos (que também podem ser *softgoals*) através de decomposições AND/OR e as relações (“+”, “-”, “++” e “--”) são regras de contribuição entre objetivos e *softgoals*. Os requisitos funcionais são modelados por objetivos e os atributos de qualidade (ou requisitos não-funcionais) são modelados como *softgoals*. As alternativas aparecem na forma de decomposições OR e são chamadas de pontos de variação (em inglês, *variation po-*

int ou VP) (ver Figura 2.8). Por exemplo, de acordo com o ponto de variação VP1 da Figura 2.8, o objetivo **Collect timetables** tem duas soluções alternativas: ou é realizada **By Person** ou **By System**. O mesmo raciocínio é aplicado para os outros pontos de variação.

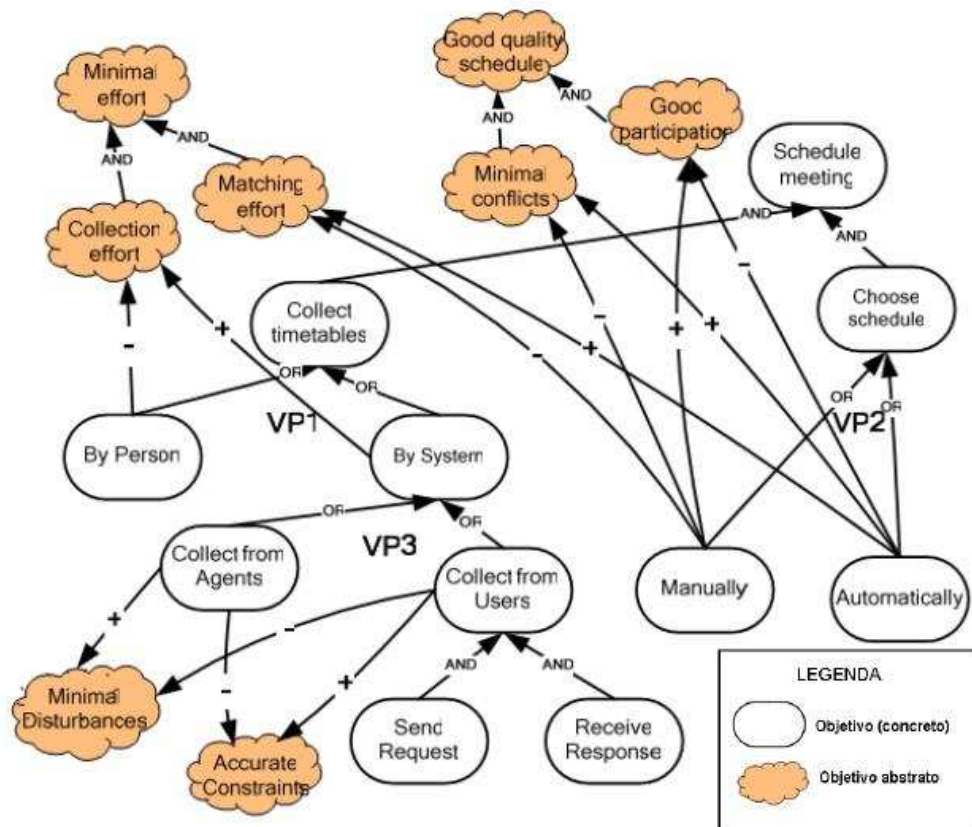


Figura 2.8 - Modelo de Objetivos (figura retirada de [Yu, Y. et al., 2008])

2.2.3.3. Framework i*

O *framework i** [Yu, E., 1995] é uma abordagem de Engenharia de Requisitos orientada a objetivos, desenvolvido para modelar e analisar, sob uma visão estratégica e intencional, processos que envolvem vários participantes. É uma abordagem centrada nos *stakeholders* do sistema e nas dependências entre eles. Os *stakeholders* são representados como atores que dependem uns dos outros para alcançarem seus objetivos, realizarem tarefas e fornecerem recursos [Yu, E., 1997]. Cada objetivo é analisado do ponto de vista do seu ator, resultando em um conjunto de dependências entre pares de atores. Desta forma, o ambiente do sistema e o sistema em si são vistos como atores organizacionais, que dependem da ajuda uns dos outros para que seus objetivos sejam cumpridos.

O i* apresenta diversas variantes como apresentado em [Lucena, M. et al., 2008] onde cada variante pode apresentar diferenças sintáticas e semânticas. Para evitar problemas com o entendimento, a versão do i* utilizada nesse trabalho é a apresentada em [Grau, G. et al., 2009]. Essa versão é uma evolução da versão original apresentada em [Yu, E., 1995] e conserva os principais elementos do i* original.

O i* permite a construção de dois tipos de modelos, o modelo de Dependência Estratégica (*Strategic Dependency model* – SD) e o modelo de Razão Estratégica (*Strategic Rationale model* - SR). O modelo SD fornece uma descrição das relações de dependências externas entre os atores da organização. O modelo SR, por sua vez, fornece uma análise de como os objetivos podem ser cumpridos através das contribuições dos demais atores. Esses dois modelos serão descritos com mais detalhes nesta seção.

A estrutura conceitual do *framework* i* é utilizada para: (i) obter uma compreensão mais apurada sobre os relacionamentos da organização, de acordo com os diversos atores do sistema; (ii) compreender as razões envolvidas nos processos de decisão; e (iii) ilustrar as várias características de modelagem que podem ser apropriadas à Engenharia de Requisitos.

Conceitos Básicos

De acordo com o *framework* i* [Yu, E., 1995], um ator é considerado uma entidade ativa que realiza ações para atingir objetivos. O termo ator é utilizado para referenciar genericamente qualquer unidade para a qual dependências intencionais possam ser atribuídas. Os atores complexos podem ser diferenciados em três tipos (Figura 2.9): papel (*role*), posição (*position*) ou agente (*agent*).

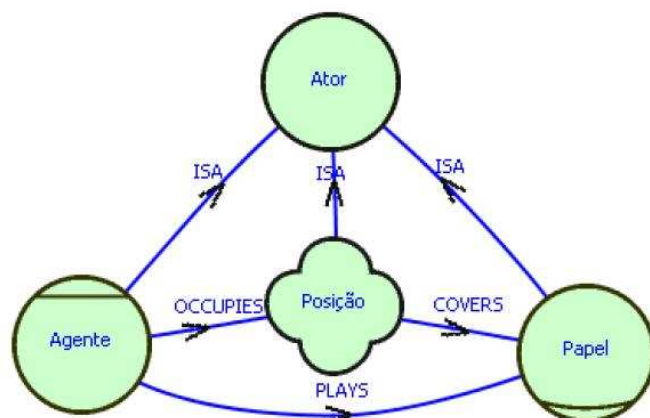


Figura 2.9 - Atores e relacionamentos entre atores [Grau, G. et al., 2009]

A Figura 2.9 ilustra a representação gráfica e as possíveis associações entre agentes, papéis e posições. Dizemos que um agente executa um determinado papel e ocupa uma determinada posição, que cobre um papel. A seguir são definidos os tipos de atores [Grau, G. et al., 2009]:

- Agente: é um ator que possui manifestações físicas concretas. Tanto pode referir-se a humanos quanto a agentes artificiais de software ou hardware;
- Papel: representa a caracterização abstrata do comportamento de um ator social dentro de determinados contextos sociais ou domínios de informação. Representa as funções que podem ser exercidas por um agente dentro da organização;
- Posição: representa uma entidade intermediária entre um agente e um papel. É o conjunto de papéis tipicamente ocupados por um agente, ou seja, representa uma posição dentro da organização onde o agente pode desempenhar várias funções.

Os atores possuem relacionamentos que permitem organizar os mapeamentos desejáveis entre eles. Os relacionamentos são de especialização (*Is-a*), agregação (*Is-partOf*), cobertura (*Covers*), atuação (*Plays*) e ocupação (*Occupies*). Quando essa distinção é feita, a análise torna-se mais detalhada e exige uma atenção maior.

Os atores dependem uns dos outros para conseguirem atingir os seus objetivos. Conforme [Yu, E., 1995], existe o ator dependente (*dependor*), o ator de quem se depende (*dependee*) e o objeto central dessa dependência (*dependum*), de modo que as dependências fiquem com o formato *dependor* → *dependum* → *dependee* (Figura 2.10).

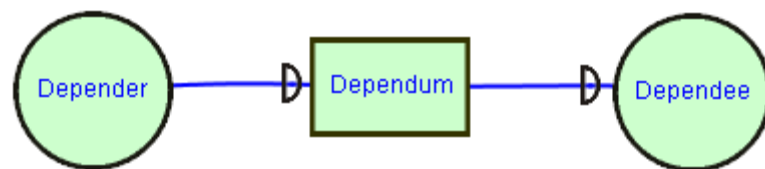
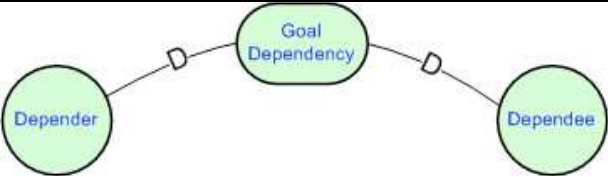
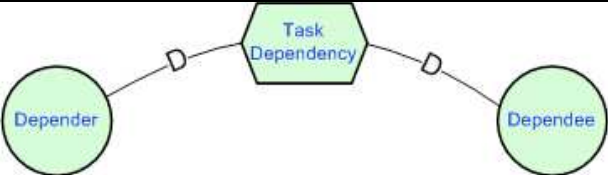
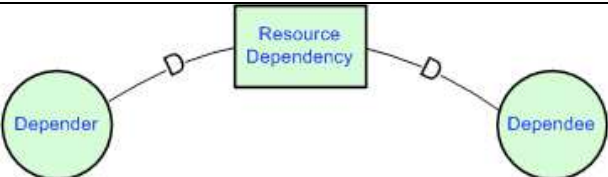
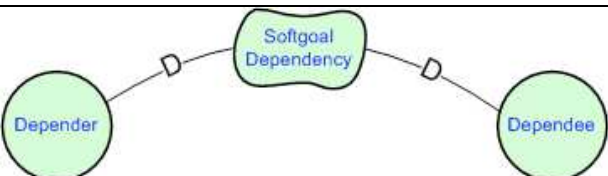


Figura 2.10 - Dependência entre atores

O *dependor* é o ator que depende de outro ator (*dependee*) para que um acordo (*dependum*) possa ser realizado. Os relacionamentos de dependência usados em i* descrevem a natureza do acordo e podem ser de quatro tipos: tarefas, recursos, objetivo ou *softgoal*.

Quando o ator *dependee*, numa relação de dependência, disponibilizar um recurso necessário, executar uma tarefa de sua responsabilidade, atender a um objetivo do ator dependente, ou realizar alguma tarefa para alcançar um *softgoal*, teremos a relação de dependência satisfeita pelo *dependee* responsável por ela. A Tabela 2.2 apresenta os tipos de ligações de dependência do *framework* i*.

Tabela 2.2 – Tipo de dependência entre atores

Dependência	Representação
Objetivo (Goal)	
Tarefa (Task)	
Recurso (Resource)	
Softgoal (Softgoal)	

A seguir descrevemos os quatro tipos de dependências propostos em i*:

- Dependência de **tarefa**: o *depender* depende do *dependee* para que este execute uma atividade. No entanto, não é fornecido ao *dependee* “o porquê” da realização da tarefa. O *depender* torna-se vulnerável, pois o *dependee* pode falhar na realização da tarefa por não seguir os passos indicados para realizar a tarefa ou pode estabelecer outras prioridades antes de realizá-la.
- Dependência de **recurso**: o *depender* depende do *dependee* em relação à disponibilidade de um recurso, seja ele físico ou de informação. Isto significa que o *dependee* deve fornecer um recurso que o *depender* necessita para realizar outras atividades no

ambiente organizacional. O *dependor* torna-se vulnerável, pois o recurso pode não ser disponibilizado pelo *dependee*. Neste tipo de dependência o que está em questão é a disponibilidade ou não de um recurso.

- Dependência de **objetivo**: o *dependor* depende do *dependee* para modificar um estado do mundo. O *dependee* tem como papel satisfazer/alcançar um objetivo para o *dependor*. O *dependee* pode atender a este objetivo e modificar o estado atual do mundo escolhendo a forma como isto será feito. No entanto, o *dependor* torna-se vulnerável, pois o *dependee* pode falhar em atingir este objetivo.
- Dependência de **softgoal**: define-se *softgoal* como um objetivo cuja avaliação de realização é bastante subjetiva e o seu significado não é claramente conhecido. Assim não se pode afirmar objetivamente se o mesmo foi satisfeito. Geralmente, o atendimento e a avaliação do *softgoal* acontecem ao longo do processo da realização de tarefas associadas com o *softgoal*. Estes *softgoals* são também referenciados como requisitos não-funcionais no contexto da Engenharia de Requisitos. O *dependor* depende do *dependee* esperando que o mesmo realize alguma tarefa que satisfaça o *softgoal*. O *dependor* torna-se vulnerável, pois o *dependee* pode falhar na obtenção das condições que satisfaçam ao *softgoal*.

Modelo de Dependência Estratégica

O Modelo de Dependência Estratégica (modelo SD) é composto por nós e ligações. Os nós representam os atores no ambiente e as ligações são as dependências entre os atores. Este modelo fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores relevantes, ou seja, as relações de dependências externas entre os atores da organização.

No modelo SD, além dos tipos de dependências entre atores supracitadas, pode-se identificar ainda diferentes graus de dependências ou vulnerabilidades [Grau, G. et al., 2008], apresentados graficamente na Figura 2.11:

- Aberta (open): é usada quando, mesmo com falha na relação de dependência, as intenções não são afetadas. A dependência aberta é graficamente representada por “O”.

- Compromissada (committed): é usada quando, na falha da relação de dependência, as intenções são afetadas, mas sem consequências sérias. Graficamente não possui sinal relacionado.
- Crítica (critical): é usada quando, na falha da relação de dependência, as intenções são afetadas seriamente levando-se a uma preocupação quanto à viabilidade de toda a rede de relacionamentos e dependências. A dependência crítica é graficamente representada por “X”.

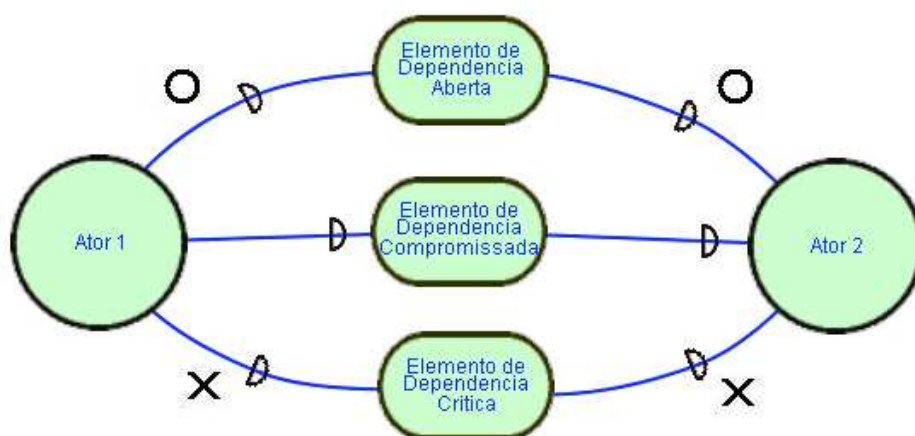


Figura 2.11 – Graus de Dependência em i*

O exemplo apresentado na Figura 2.12, retirado de [Santander, V. F. A., 2002], modela as dependências estratégicas envolvidas no processo de agendamento de reuniões em uma organização. Este processo é apoiado por um sistema computacional denominado **Agendador de Reunião**. O **Iniciador de Reunião** depende do **Participante** em relação ao seu comparecimento à reunião e delega boa parte do trabalho de agendamento de reuniões ao sistema **Agendador de Reunião**. O **Agendador de Reunião** determina quais são as datas possíveis para o agendamento de uma reunião com base na informação de disponibilidade (dependência do tipo tarefa “Entra Datas Disponíveis (m)”) fornecida por cada **Participante**. O **Iniciador de Reunião** não interfere na forma em que o **Agendador de Reunião** determina as datas de agendamento aceitáveis, desde que as mesmas sejam encontradas e definidas. Isto se reflete na dependência do tipo objetivo “Reunião Agendada” do **Iniciador** para o **Agendador**. Por outro lado, para se chegar a uma data de consenso para o agendamento de uma reunião, **Participantes** dependem que o **Agendador de Reunião** forneça a proposta de data (dependência do tipo recurso “Data Proposta(m)”). Uma vez proposta uma data de agendamento, o sistema **Agendador de Reunião** depende que os **Participantes** concordem com a mesma (dependência do tipo recurso “Concordância (m,p)”). Para **Participantes Importantes**, o **Iniciador de Reunião** depende criticamente do comparecimento des-

tes à reunião (dependência do tipo objetivo “Comparecer Reunião(ip,m)”), bem como desejam garantir que estes **Participantes Importantes** compareçam à reunião (dependência do tipo *soft-goal* “Assegurado(ComparecerReunião (ip,m))”). Finalmente, o **Agendador de Reunião** depende que o **Iniciador de Reunião** forneça um intervalo de datas, dentro do qual uma reunião deve ser agendada (dependência do tipo tarefa “EntrarIntervaloDatas(m)”).

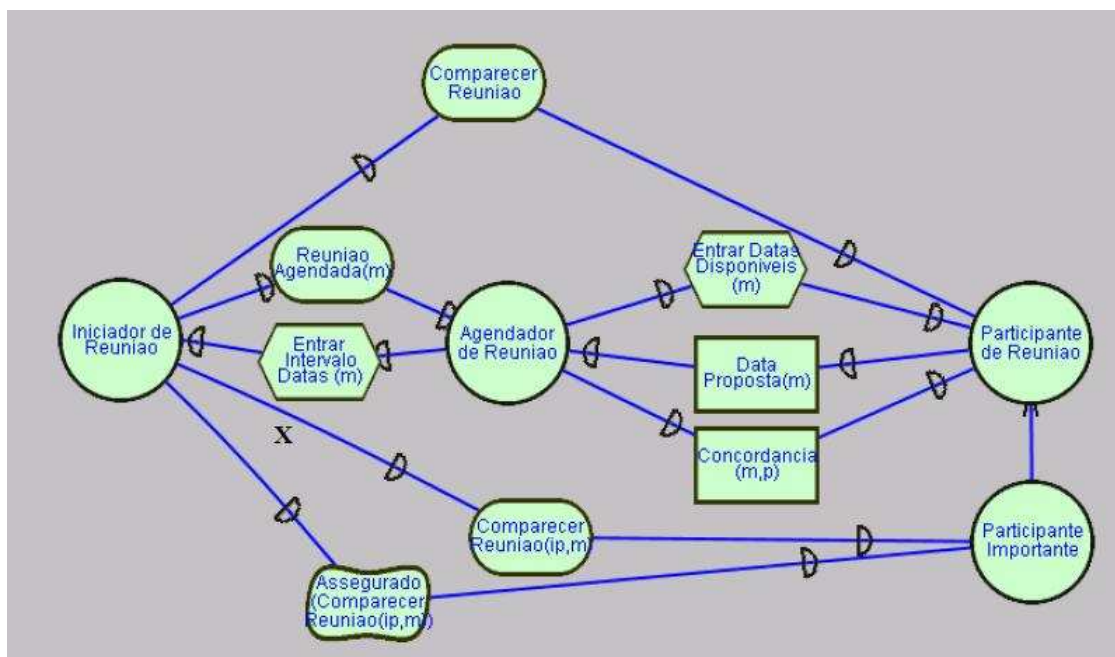


Figura 2.12 - Exemplo de Modelo de Dependência Estratégica para Agendamento de Reuniões (figura retirada de [Santander, V. F. A., 2002])

Modelo de Razão Estratégica

O modelo SR representa as intenções dos *stakeholders*, demonstrando o modo como serão satisfeitas as várias dependências de cada ator, além de apresentar, mais aprofundadamente, as razões da necessidade de existirem essas dependências.

Enquanto o modelo SD trata apenas dos relacionamentos externos entre os atores, o modelo SR é utilizado para descrever os relacionamentos internos: interesses, preocupações e motivações dos atores participantes de um processo. Ele possibilita a avaliação das possíveis alternativas de definição do processo, investigando mais detalhadamente as razões ou intencionalidades existentes, por trás das dependências entre os atores. As intencionalidades de cada ator são identificadas e representadas dentro da fronteira de cada ator (Figura 2.13).

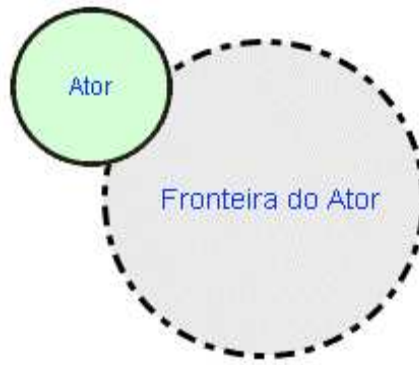


Figura 2.13 - Fronteira do Ator

Assim como o SD, o modelo SR também é composto por ligações de dependência. Além de utilizar os quatro tipos de dependências apresentados no modelo SD - recurso, tarefa, objetivo e *softgoal*, o modelo SR dispõe de outros três tipos de relacionamentos: as ligações de meio-fim, as ligações de decomposição de tarefas e as de contribuição.

As ligações de meio-fim indicam um relacionamento entre um nó fim e um meio para atingi-lo. Este tipo de ligação sugere alternativas existentes para se alcançar um determinado fim, onde os meios são expressos em forma de tarefas e os fins, geralmente, expressos em forma de objetivos [Grau, G. et al., 2009]. A representação deste tipo de ligação pode ser visualizada na Figura 2.14

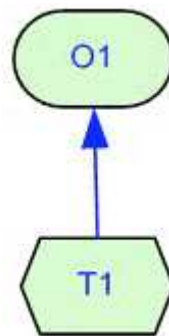


Figura 2.14 - Ligação Meio-Fim

Já as ligações de decomposição de tarefas ligam um nó de tarefa a seus nós componentes, que segundo [Yu, E., 1995] podem ser outras tarefas, objetivos, recursos ou *softgoals*. Uma tarefa decomposta só pode ser considerada completa quando todos os seus nós componentes forem realizados ou satisfeitos. A representação deste tipo de ligação pode ser visualizada na Figura 2.15.

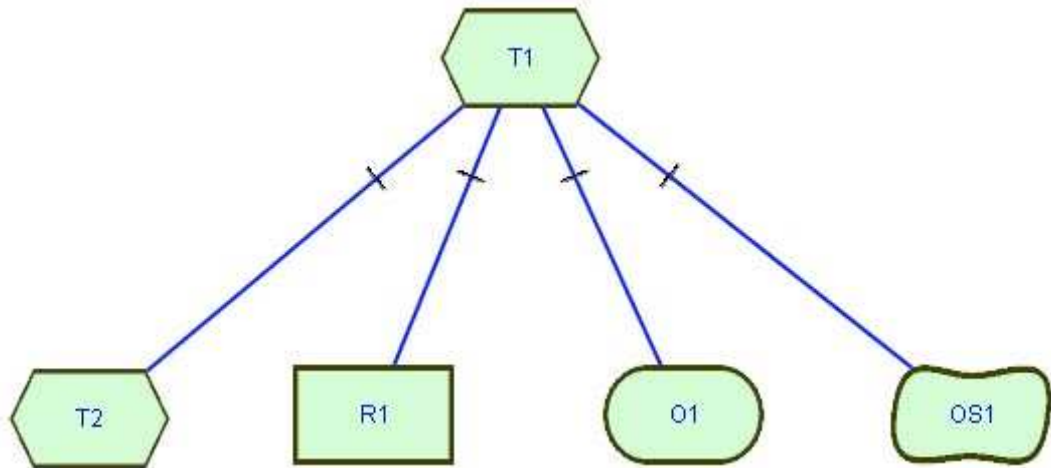


Figura 2.15 - Ligação de Decomposição de Tarefas

Os links de contribuição servem para modelar a forma como os elementos podem contribuir para a satisfação ou cumprimento de um *softgoal*. Qualquer um dos links de contribuição apresentados na Figura 2.16 pode ser usado para ligar qualquer elemento a um *softgoal*.

Relativamente às ligações de contribuição, segundo Yu [Yu, E., 1995], originalmente apenas existiam as ligações que contribuía positivamente (*Help*) ou negativamente (*Hurt*) para a satisfação de uma dependência. Atualmente, existem outras ligações de contribuição que realizam a dependência (*Make*), as que não se conhece qual será a sua influência na dependência (*Unknown*) e as que não permitem que a dependência seja satisfeita (*Break*). Estes atributos indicam o tipo de contribuição relacionada com os objetivos das dependências, podendo ser [Grau, G. et al., 2008]:

- **Make:** é uma contribuição positiva, suficientemente forte para satisfazer o *softgoal*.
- **Some +:** é uma contribuição positiva, mas cuja força (influência) é desconhecida.
- **Help:** é uma contribuição positiva parcial, porém não é suficiente para que ela sozinha satisfaça o *softgoal*.
- **Unknown:** é uma contribuição cuja influência é desconhecida.
- **Break:** é uma contribuição negativa, suficientemente forte para recusar a satisfação do *softgoal*.
- **Some -:** é uma contribuição negativa, mas a força de sua influência é desconhecida.

- **Hurt**: é uma contribuição negativa parcial, porém não é suficiente para que ela sozinha recuse a satisfação de um *softgoal*.
- **Or**: é uma contribuição cujo destino é satisfeito se algum dos descendentes for satisfeitos.
- **And**: é uma contribuição cujo destino é satisfeito se todos os descendentes forem satisfeitos.

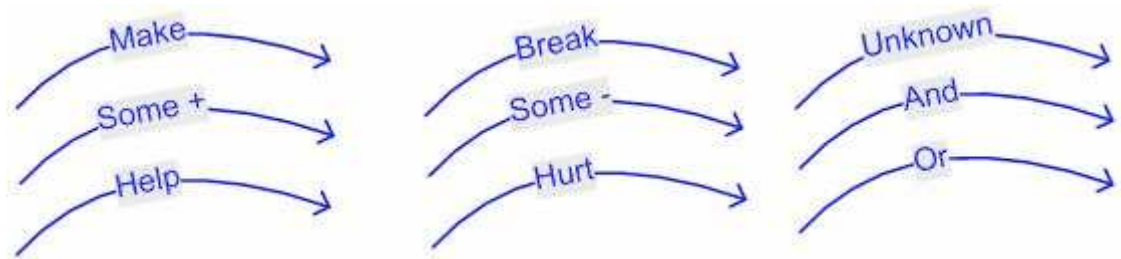


Figura 2.16 - Ligações de Composição

Na Figura 2.17 é apresentada uma parte do modelo de razão estratégica para o problema de agendamento de reuniões, já mostrado na Figura 2.12.

A título de exemplo, a Figura 2.17 descreve a razão estratégica associada com o ator **Iniciador de Reunião**. De acordo com esta figura, o fato de a reunião ser agendada é indicado pelo objetivo **Reunião Agendada** associado com a tarefa **Organizar Reunião**. Neste ponto, outros objetivos poderiam ser incluídos, tais como: equipamentos a serem reservados para a reunião e memorandos a serem enviados, visando lembrar a reunião. Também é observado que a organização da reunião deve ser feita de forma rápida e com menor esforço possível. Estes desejos são representados pelos *softgoals* **Rápido** e **Pouco Esforço**. O **Iniciador de Reunião** pode agendar uma reunião manualmente (tarefa **Agendar Reunião**) ou utilizar o ator/sistema **Agendador de Reunião** (tarefa **Deixar Agendador Agendar Reunião**). Desta forma, grande parte do trabalho de agendar reunião pode ser atribuído ao ator **Agendador de Reunião**.

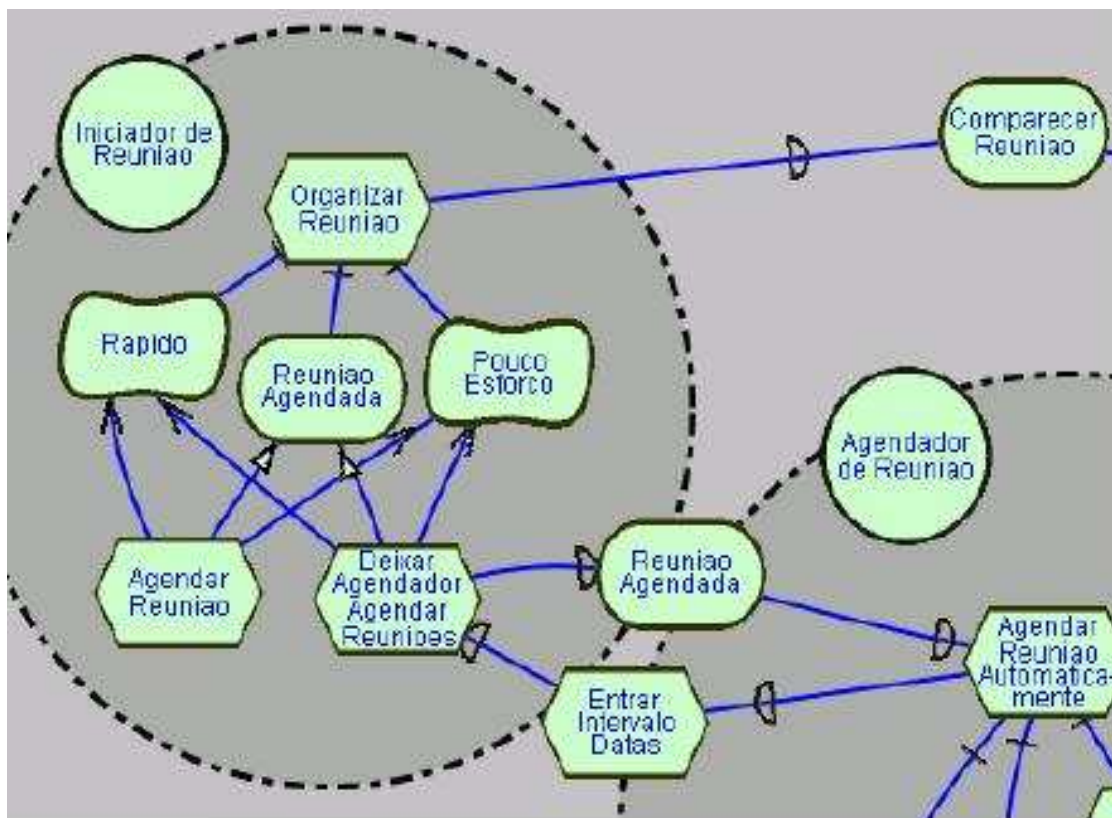


Figura 2.17 - Exemplo de Modelo de Razão Estratégica para o ator Iniciador de Reunião do sistema Agendamento de Reuniões (figura retirada de [Santander, V. F. A., 2002]).

Nesta dissertação, utilizamos modelos i^* para identificar *features* comuns e variáveis de uma LPS, como também para configurar uma aplicação da LPS, pois esses modelos permitem expressar os interesses dos *stakeholders* que podem guiar na escolha das características do sistema pretendido. Porém, os modelos i^* não são capazes de modelar a representação de cardinalidade. Portanto, no próximo capítulo, propomos uma extensão da linguagem de modelagem do i^* , chamada i^*-c , para suportar cardinalidade e assim, conseguir representar a expressividade dos modelos de *features*.

2.3. Considerações Finais

Este capítulo discute o estado da arte das duas principais subáreas envolvidas neste trabalho: a Engenharia de Linhas de Produto de Software (ELPS) e a Engenharia de Requisitos Orientada a Objetivos (ou *Goal-Oriented Requirements Engineering* – GORE).

Primeiramente foram apresentadas as principais atividades realizadas para a ELPS e o modelo de *features*, artefato central usado nestas atividades e ao longo desta dissertação. Es-

ses modelos não capturam requisitos não-funcionais explicitamente e nem a influência positiva/negativa destes requisitos para justificar configurações alternativas de uma aplicação na LPS. A captura destas informações se torna possível com a utilização de abordagens orientadas a objetivos em LPS, justificando a integração dessas abordagens com modelos de *features*.

Em seguida, foram mostradas algumas abordagens GORE existentes (*framework* NFR, modelos de objetivos e *framework* i^*), com maior foco no *framework* i^* que por ter uma maneira natural de representar conceitos, foi escolhida como abordagem principal desta dissertação. Porém, esta abordagem precisa ser estendida para suportar cardinalidade e assim, se adequar ao uso no contexto de LPS.

O capítulo seguinte apresentará uma comparação da expressividade de abordagens orientadas a objetivos em relação a modelos de *features* e a definição de uma nova linguagem i^* , chamada i^*-c (i^* with cardinality), que é uma extensão do metamodelo do i^* com cardinalidade.

Capítulo

3. Adaptando modelos i^* para suportar cardinalidade

Inicialmente, este capítulo apresenta uma comparação entre modelos orientados a objetivos para modelagem da variabilidade fornecida pelos modelos de *features*. Em seguida, é proposta uma extensão do i^* , chamada $i^*\text{-c}$ (*i^* with cardinality*), para representar cardinalidade. O metamodelo estendido para esta nova linguagem de modelagem é apresentado, bem como algumas restrições associadas a esta extensão. Por último, encontram-se as considerações finais.

A variabilidade de uma linha de produtos deve ser documentada explicitamente para permitir o reuso estratégico de artefatos de requisitos. Isto significa relacionar diferentes tipos de requisitos (organizacionais, funcionais e não-funcionais) e manter o rastreamento entre eles. Como vimos no capítulo anterior, modelos de *features* são utilizados para representar as variabilidades e similaridades de características de produtos em uma LPS [Kang, K. et al., 1990]. Porém, modelos de *features* não capturam explicitamente requisitos não-funcionais e a influência positiva/negativa desses requisitos com relação a outros requisitos do sistema durante o desenvolvimento dos *core assets*. Esta influência pode ajudar na escolha de uma determinada configuração de um produto para cumprir objetivos organizacionais.

Um objetivo é um estado do mundo que os *stakeholders* desejam atingir, enquanto *features* são características que o sistema deve apresentar. Partindo destas definições, abordagens orientadas a objetivos podem ser usadas para capturar *features* utilizando modelos mais significativos, para manter o rastreamento entre as *features* do sistema e suas motivações, bem como para justificar a configuração de alternativas do sistema para realização dos objetivos dos *stakeholders*.

Algumas abordagens orientadas a objetivos para requisitos em LPS têm sido propostas para capturar as semânticas de modelos de *features* (modelos de objetivos [Yu, Y. et al., 2008], PL-AOVGraph [Batista, T. et al., 2008], i* Aspectual [Alencar, F. et al., 2008] e i* [Yu, E., 1995]). Em [Borba, C. and Silva, C., 2009] apresentamos uma comparação entre essas abordagens levando em consideração a expressividade relacionada à modelagem da variabilidade fornecida pelos modelos de *features*. Esta comparação é apresentada na seção seguinte que tem como resultado a motivação para a abordagem **G2FM** (em inglês, *Goal To Feature Model*).

3.1. Comparação entre abordagens orientadas a objetivos para a modelagem de variabilidade

A expressividade é fator determinante na escolha de uma técnica de modelagem e está relacionada aos conceitos oferecidos pela técnica [Estrada, H., 2008]. Para avaliar esta propriedade, analisamos em [Borba, C. and Silva, C., 2009] como várias técnicas orientadas a objetivos, para a modelagem da variabilidade em LPS, expressam cada conceito presente no modelo de *features* proposto por [Czarnecki, K. et al., 2005a] e [Czarnecki, K. et al., 2005b].

Em [Borba, C. and Silva, C., 2009], inicialmente foi ilustrado um modelo de *features* baseado em cardinalidade [Czarnecki, K. et al., 2005a] de um sistema de comércio eletrônico.

O modelo da Figura 3.1 contém um diagrama de *features* com **ECommerce** como sua *feature* raiz.

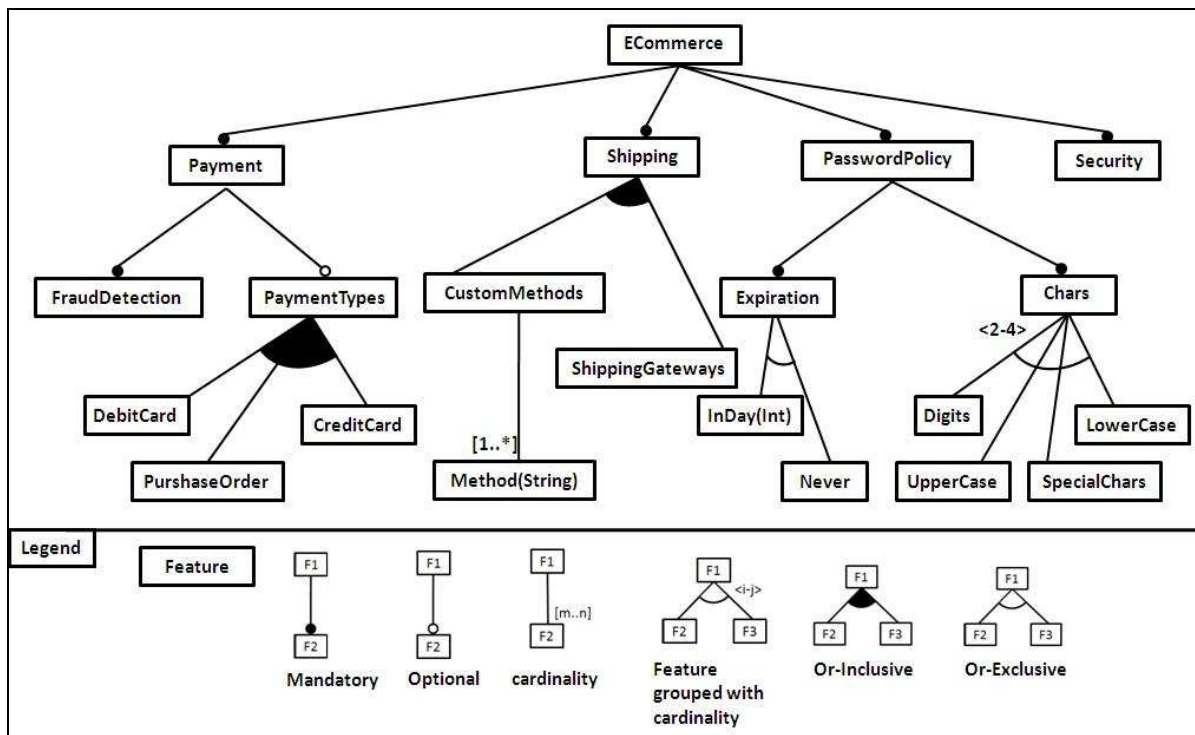


Figura 3.1 - Modelo de *features* do sistema de comércio eletrônico

A *feature* raiz tem quatro *sub-features*: **Payment**, **Shipping**, **PasswordPolicy** e **Security**. O símbolo com círculo preenchido indica que estas *sub-features* obrigatórias tem a cardinalidade [1..1], significando que a *feature* deve existir uma vez e apenas uma vez. Por outro lado, o símbolo com círculo vazio indica que **PaymentTypes** é uma *feature* opcional com cardinalidade [0..1]. A cardinalidade da *feature* **Method(String)** é [1..*], significando que existe pelo menos uma ocorrência desta *feature* no produto. Caso a *feature* **PaymentTypes** seja válida no produto, neste caso **DebitCard**, **PurshaseOrder** e **CreditCard** são membros de um grupo de *features*. O símbolo da ramificação preenchida indica um grupo com cardinalidade <1-k>, onde k é o tamanho do grupo (ou-inclusivo). Assim, **PaymentTypes** pode ser qualquer subconjunto não-vazio dos três tipos de pagamento. O símbolo da ramificação vazia na *feature* **Expiration** indica um grupo com cardinalidade <1-1> (ou-exclusivo), significando que só pode existir um tipo de expiração para a senha. Já o símbolo da ramificação vazia com

a cardinalidade $\langle 2-4 \rangle$ sob a *feature* **Chars** mostra um grupo que no mínimo 2 e no máximo 4 *sub-features* podem ser escolhidas na configuração do produto.

Na seqüência, serão apresentadas modelagens do mesmo sistema de comércio eletrônico utilizando diferentes abordagens orientadas a objetivos.

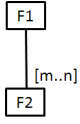
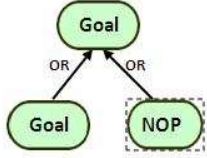
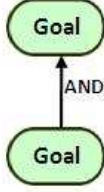
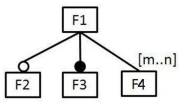
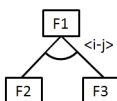
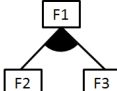
3.1.1. Modelo de Objetivos

Em [Yu, Y. et al., 2008], modelos de objetivos são utilizados para representar a forma como o sistema e o seu ambiente operacional podem atingir os objetivos dos *stakeholders*. A partir desses modelos, são identificados os objetivos que pertencem ao sistema e os que pertencem ao seu ambiente. Estes últimos são substituídos por objetivos NOP (nenhuma operação), significando que não devem ser mapeados para *features*, pois estão sob responsabilidade do ambiente. Os objetivos restantes (os que pertencem ao sistema) são mapeados para *features* de acordo com o mapeamento apresentado na Tabela 3.1.

Os autores reconhecem que decomposições AND/OR não podem ser mapeadas diretamente para *features* do tipo alternativas e opcionais. Os modelos de *features* produzidos por esta abordagem são limitados a: (i) *features* decompostas em *sub-features* de um único tipo (obrigatório, opcional) e (ii) *features* que não podem ser agrupadas utilizando cardinalidade como $\langle i,j \rangle$. A Figura 3.2 mostra o mapeamento do modelo de *features* apresentado na Figura 3.1 para o modelo orientado a objetivos apresentado em [Yu, Y. et al., 2008].

Nesta representação foi acrescentada a influência de algumas alternativas em relação à satisfação do *softgoal* **Security**. Observa-se que o objetivo **Expiration** pode ser alcançado através de duas formas diferentes: a senha pode expirar em dias ou nunca expirar. Cada alternativa contribui de maneira diferente para satisfazer o *softgoal* **Security**. Os círculos destacados na Figura 3.2 indicam situações onde os conceitos (A) e (E) apresentados na Tabela 3.1 deveriam ter sido utilizados. Infelizmente, a abordagem apresentada em [Yu, Y. et al., 2008] não suporta cardinalidade e, por isso, a expressividade se torna limitada. Neste exemplo, como uma possível solução, poderíamos ter usado a decomposição AND para refletir o conceito (A) e a decomposição OR para refletir (E). No entanto, em ambos os casos, eles não conseguiriam captar o mesmo significado da Figura 3.1.

Tabela 3.1 – Mapeamento entre modelo de *features* e modelo de objetivos

	Modelo de <i>Features</i>	Modelo de Ojetivos
(A)	<i>Feature</i> com cardinalidade [m..n] 	Não suporta
(B)	<i>Feature</i> com cardinalidade [0..1] (opcional) 	Em uma decomposição OR com no mínimo um sub-objetivo NOP, os objetivos não NOP são mapeados para <i>features</i> opcionais. 
(C)	<i>Feature</i> com cardinalidade [1..1] (obrigatória) 	Decomposição AND onde objetivos são mapeados para <i>features</i> obrigatórias. 
(D)	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005] 	Não suporta
(E)	<i>Features</i> agrupadas com cardinalidade <i-j> 	Não suporta
(F)	<i>Features</i> agrupadas com cardinalidade <1-k>, onde k é tamanho do grupo (or-inclusive) 	Decomposição OR-inclusive

(G)	Features agrupadas com cardinalidade <1-1> (or-exclusive)	Decomposição OR com o símbolo “ ” significando que esta decomposição é agora OR-exclusive.

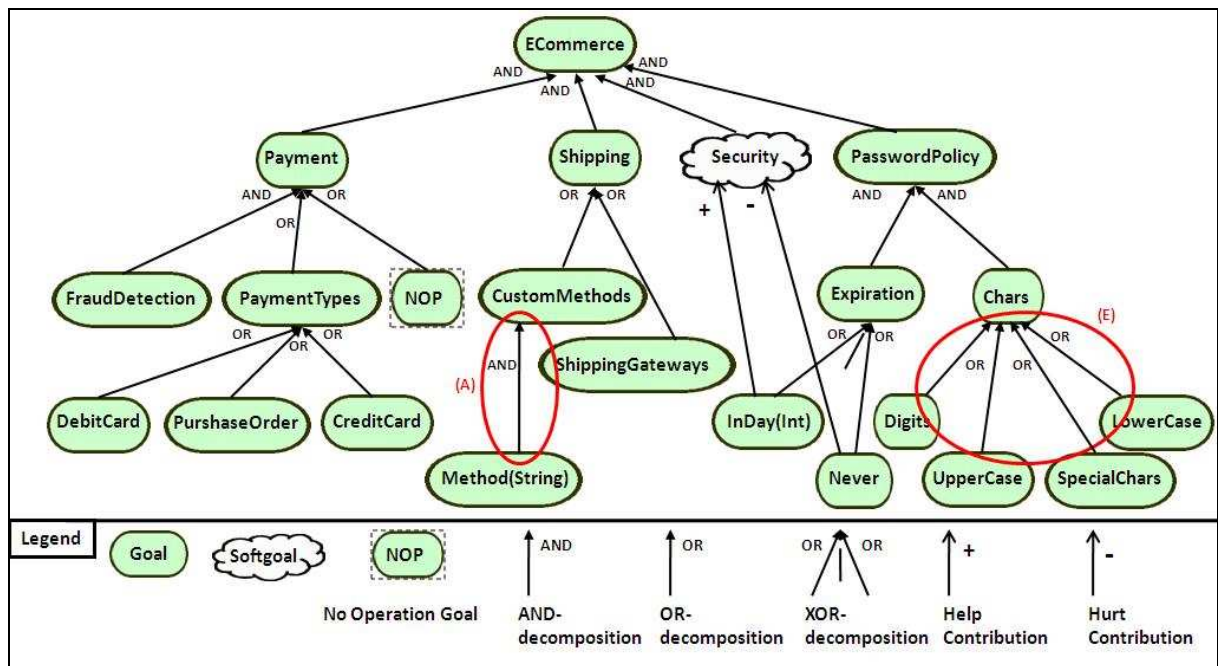


Figura 3.2 - Modelo de objetivos do sistema de comércio eletrônico

3.1.2. Modelo PL-AOVGraph

O paradigma orientado a aspectos fornece meios para a identificação, modularização, representação e composição de preocupações transversais (preocupações que atravessam as

fronteiras de outras preocupações) durante todas as fases do ciclo de desenvolvimento de software [Rashid, A. et al., 2003].

AOV-graph é um modelo intencional orientado a aspectos, representado por decomposições AND/OR [Silva, L. and Leite, J., 2005]. Seus relacionamentos mapeiam tanto os conflitos positivos e negativos entre requisitos (objetivos, *softgoals* e tarefas), como também representam escolhas de opções de como um determinado requisito pode ser alcançado, sendo uma escolha razoável para representar variabilidade. Em [Batista, T. et al., 2008], o modelo AOV-graph é estendido para representar variabilidade em LPS, resultando no modelo PL-AOVGraph. A Tabela 3.2 resume os construtores mapeados entre modelo de *features* e o modelo PL-AOVGraph.

A Figura 3.3 ilustra a representação da variabilidade do sistema de comércio eletrônico representado no modelo de *features* da Figura 3.1 no modelo PL-AOVGraph. Os círculos destacadas na Figura 3.3 indicam limitações da expressividade do modelo PL-AOVGraph, pois não suporta cardinalidade correspondente aos conceitos (A) e (E) apresentados na Tabela 3.2. Semelhante ao caso anterior, como uma possível solução, pode-se utilizar a relação AND para refletir (A) e a relação OR-inclusive para refletir (E). Novamente, em ambos os casos, eles não representam o mesmo significado da Figura 3.1.

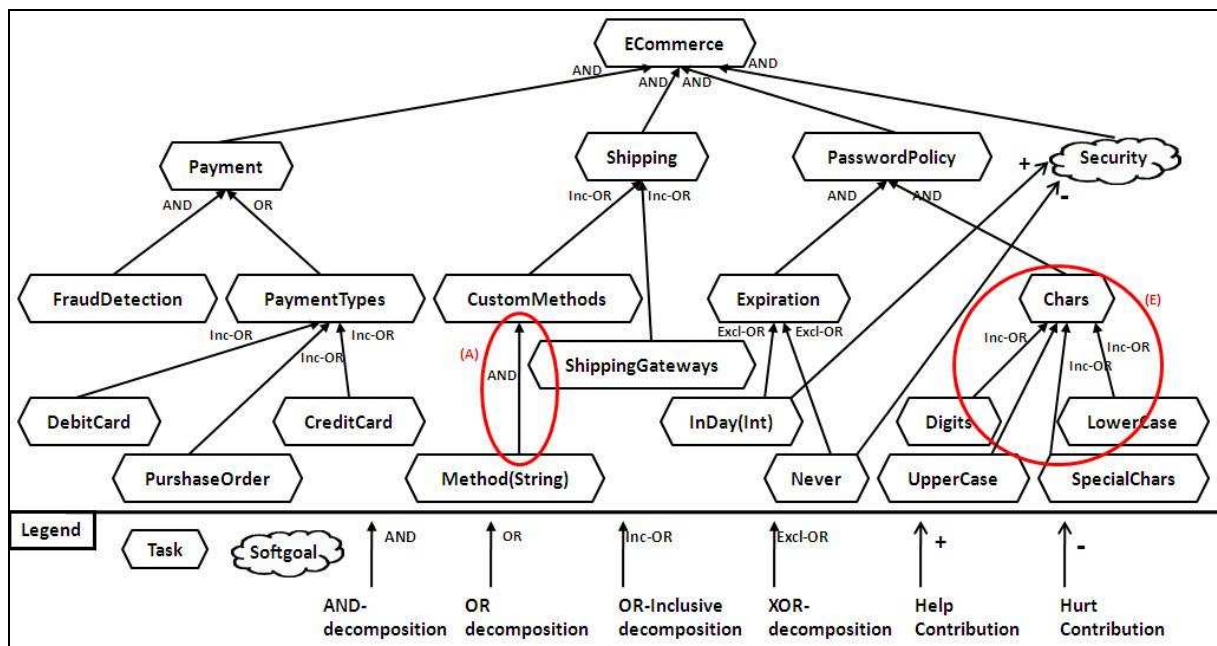
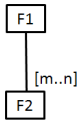
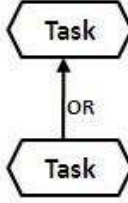
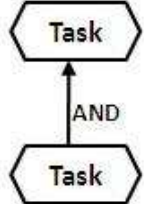
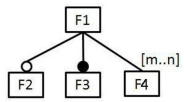
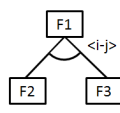
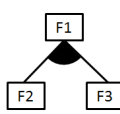
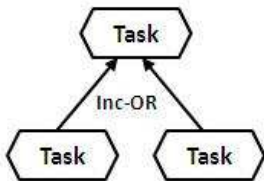
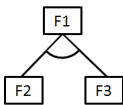
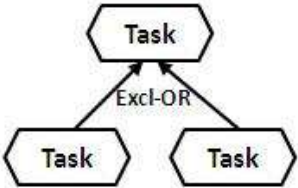


Figura 3.3 - Modelo PL-AOVGraph do sistema de comércio eletrônico

Tabela 3.2 – Mapeamento entre modelo de *features* e modelo PL-AOVGraph

	Modelo de <i>Features</i>	PL-AOVGraph
(A)	Feature com cardinalidade [m..n] 	Não suporta
(B)	Feature com cardinalidade [0..1] (opcional) 	Relação OR 
(C)	Feature com cardinalidade [1..1] (obrigatória) 	Relação AND 
(D)	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005] 	Não suporta
(E)	Features agrupadas com cardinalidade <i-j> 	Não suporta
(F)	Features agrupadas com cardinalidade <1-k>, onde k é tamanho do grupo (or-inclusive) 	Relação OR-inclusive 

(G)	<i>Features</i> agrupadas com cardinalidade <1-1> (or-exclusive) 	Relação OR-exclusive ³ . 
-----	---	---

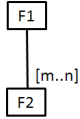
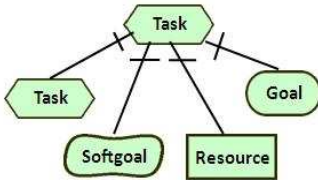
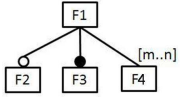
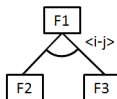
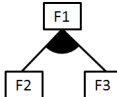
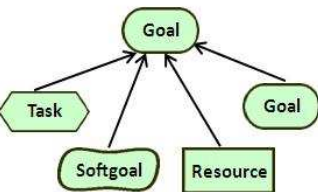
3.1.3. Modelos i* Aspectual

Em [Silva, C. et al., 2008] é apresentada a forma como a linguagem de modelagem i* aspectual pode ser usada para descrever *features* tanto na engenharia de domínio quanto na engenharia de aplicação. São apresentadas heurísticas para permitir fazer o mapeamento para o modelo i* aspectual através do modelo de *features*. A abordagem de i* aspectual [Alencar, F. et al., 2008] é utilizada para modularizar variabilidades e para compor *features* variáveis com *features* comuns, pois, os autores acreditam que representar essas *features* como aspectos pode reduzir tempo e custo durante a atividade de configuração. Assim, as *features* variáveis podem ser modularizadas em aspectos e depois, compostas com as *features* comuns na engenharia de aplicação. Para facilitar ainda mais a atividade de configuração, os modelos i* aspectuais também permitem representar a influência de alternativas para satisfazer os *softgoals*. O mapeamento entre os construtores do modelo de *features* e do modelo i* aspectual podem ser encontrados na Tabela 3.3.

A Figura 3.4 apresenta a representação do i* aspectual do sistema de comércio eletrônico modelado com *features* na Figura 3.1. Os círculos destacados na Figura 3.4 indicam as limitações da abordagem, pois a mesma não suporta cardinalidade, as relações opcionais e as relações alternativas/agrupadas (os conceitos A, B, E e G da Tabela 3.3). Foi decidido então, substituir essas relações por outros tipos (relacionamento *Task-Decomposition* no lugar de B, relacionamento *Crosscut Means-End* em vez de E e G, e a ligação *Crosscut Task-Decomposition* no lugar de A) que não capturam exatamente a mesma semântica. A difícil compreensão desta figura deve-se ao fato do problema de legibilidade dos modelos i*.

³ Não existe exemplo em [Batista, T. et al., 2008] ilustrando esta relação, então, para manter a consistência, nós adotamos o uso da relação OR-Exclusive (Exc-Or) porque os autores mencionaram a sua existência (mas não ilustraram).

Tabela 3.3 – Mapeamento entre modelo de *features* e, modelo i* aspectual e i*

	Modelo de <i>Features</i>	i* Aspectual e i*
(A)	<i>Feature</i> com cardinalidade [m..n] 	Não suporta
(B)	<i>Feature</i> com cardinalidade [0..1] (opcional) 	Não suporta
(C)	<i>Feature</i> com cardinalidade [1..1] (obrigatória) 	Relacionamento de decomposição de tarefas (<i>Task-Decomposition</i>) 
(D)	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005] 	Não suporta
(E)	<i>Features</i> agrupadas com cardinalidade <i-j> 	Não suporta
(F)	<i>Features</i> agrupadas com cardinalidade <1-k>, onde k é tamanho do grupo (or-inclusive) 	Relacionamento meio-fim (<i>Means-End</i>) 
(G)	<i>Features</i> agrupadas com cardinalidade <1-1> (or-exclusive)	Não suporta

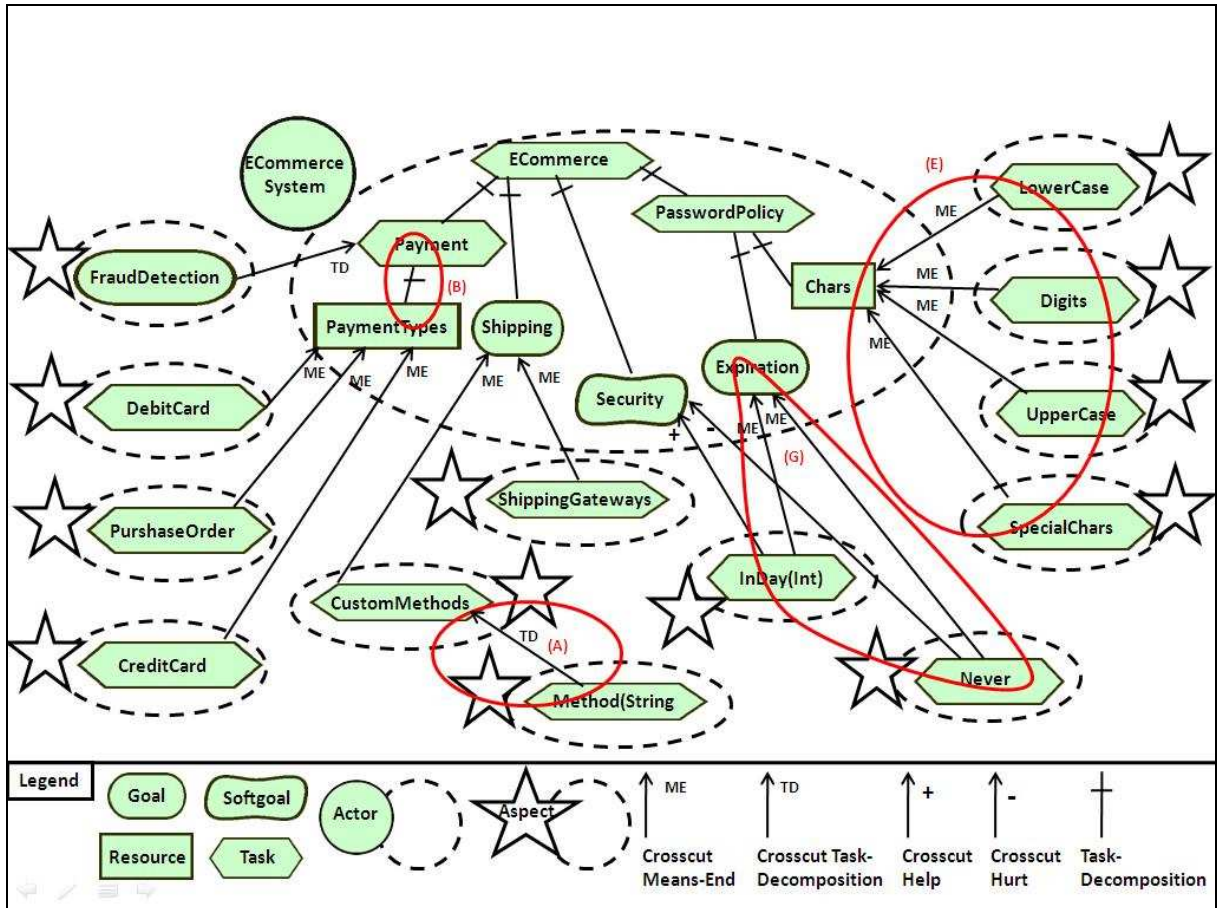


Figura 3.4 - Modelo i* aspectual do sistema de comércio eletrônico

3.1.4. Modelos i*

De forma similar, a Figura 3.5 apresenta a representação do i* original [Yu, E., 1995] do sistema de comércio eletrônico modelado com *features* da Figura 3.1. Os círculos destacados na Figura 3.5 também indicam as limitações da abordagem, pois a mesma não suporta cardinalidade, nem as relações opcionais e nem as relações alternativas/agrupadas (os conceitos A, B, E e G da Tabela 3.3). O mapeamento entre os construtores do modelo de *features* e

do modelo i* original também podem ser encontrados na Tabela 3.3, juntamente com o i* aspectual.

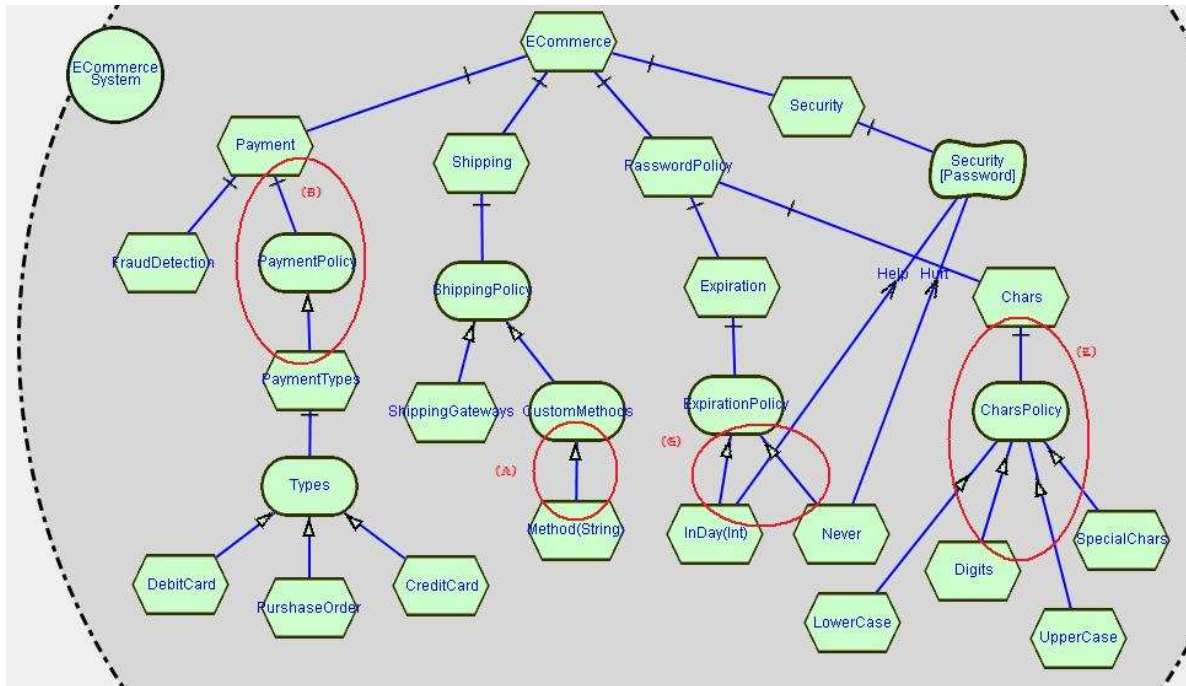


Figura 3.5 - Modelo i* do sistema de comércio eletrônico

A Tabela 3.4 mostra o resumo da comparação dos modelos de objetivos, PL-AOVGraph, i* Aspectual e i*. Observamos que modelos de objetivos e modelos PL-AOVGraph têm a expressividade limitada, pois não são capazes de modelar a representação da cardinalidade do modelo de *features*. Os modelos i* Aspectual e i* não conseguem representar cardinalidade nem alternativas do modelo de *features*.

Com a finalidade de solucionar as limitações supracitadas, na próxima seção nós propomos uma abordagem que tem o objetivo de preencher as lacunas do mapeamento entre modelo de *features* e modelos orientados a objetivos mencionados acima, adicionando cardinalidade nos modelos i*. Para que a linguagem i* suporte este tipo de cardinalidade, foi preciso definir uma extensão do i*. A notação gráfica, bem como o seu metamodelo e um exemplo do seu uso serão apresentados em detalhes na seção a seguir.

Tabela 3.4 – Equivalência entre modelo de *features*, modelo de objetivos, PL-AOVGraph, i* Aspectual e i*

	Modelo de <i>Features</i>	Modelo de Objetivos	PL-AOVGraph	i* Aspectual e i*
(A)	<i>Feature</i> com cardinalidade [m..n]	Não suporta	Não suporta	Não suporta
(B)	<i>Feature</i> com cardinalidade [0..1] (opcional)	Em uma decomposição OR com no mínimo um sub-objetivo NOP, os objetivos não NOP são mapeados para <i>features</i> opcionais.	Relação OR	Não suporta
(C)	<i>Feature</i> com cardinalidade [1..1] (obrigatória)	Decomposição AND onde objetivos são mapeados para <i>features</i> obrigatórias	Relação AND	Relacionamento de decomposição de tarefas (<i>Task-Decomposition</i>)
(D)	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005]	Não suporta	Não suporta	Não suporta
(E)	<i>Features</i> agrupadas com cardinalidade <i-j>	Não suporta	Não suporta	Não suporta
(F)	<i>Features</i> agrupadas com cardinalidade <1-k>, onde k é tamanho do grupo (or-inclusive)	Decomposição OR-inclusive	Relação OR-inclusive	Relacionamento meio-fim (<i>Means-End</i>)

(G)	<i>Features</i> agrupadas com cardinalidade <1-1> (or-exclusive) <pre> graph TD F1[F1] --- J(()) J --- F2[F2] J --- F3[F3] </pre>	Decomposição OR com o símbolo “ ” significando que esta decomposição é agora OR-exclusive	Relação OR-exclusive	Não suporta
-----	---	---	----------------------	-------------

3.2. Extensão do modelo *i** para inclusão de cardinalidade

Modelos *i** não foram desenvolvidos com a intenção de modelar LPS, portanto faz-se necessário ajustar este modelo para que possa ter informação adicional e assim, ajudar no desenvolvimento de uma linha de produtos. Dessa forma, propomos uma extensão da linguagem *i**, chamada ***i*-c*** (*i* with cardinality*), onde foram adicionadas cardinalidade, permitindo capturar mais informação sobre a variabilidade de uma LPS, de forma semelhante aos modelos de *features* [Czarnecki, K. et al., 2005a].

Vimos anteriormente que uma *feature* pode indicar qualquer característica ou funcionalidade do sistema e é usada para capturar similaridades ou variabilidades nos produtos de uma LPS [Czarnecki, K. and Eisenecker, U, 2000]. Partindo desta definição, na linguagem ***i*-c***, os elementos do tipo tarefa e recurso são considerados como os elementos intencionais do modelo SR a terem cardinalidade, já que determinam funcionalidades e características do sistema, respectivamente (vide conceitos **A**, **B**, **C** e **D** da Tabela 3.5).

A ligação meio-fim nos modelos SR significa que um objetivo (fim) pode ser alcançando através do sub-elemento de onde parte a ligação (ou seja, o meio) e representa uma ligação *or-inclusive*. Como desejamos modelar uma LPS, adicionamos cardinalidade nas ligações meio-fim da linguagem ***i*-c***. As *features* obtidas através dos sub-elementos dessa ligação estarão agrupadas no modelo de *features* e a cardinalidade pertencerá ao relacionamento (vide conceitos **E**, **F** e **G** da Tabela 3.5). Porém, se a quantidade de sub-elementos dessa ligação for exatamente 1, então a cardinalidade pertencerá ao sub-elemento e não ao relacionamento.

A ligação de decomposição de tarefas nos modelos SR denota obrigatoriedade, significando que seus sub-elementos estarão presentes na aplicação. Assim, a cardinalidade pertencerá aos sub-elementos (desde que sejam tarefas ou recursos) e não à ligação. Dessa forma, a ligação de decomposição de tarefas da linguagem ***i*-c*** não terá cardinalidade.

Os *links* de contribuição servem para modelar a forma como os elementos podem contribuir para a satisfação ou cumprimento de um *softgoal*. Esses elementos se forem do tipo tarefa ou recurso poderão ter cardinalidade, a depender da aplicação. Dessa forma, a ligação de contribuição da linguagem **i*-c** não terá cardinalidade.

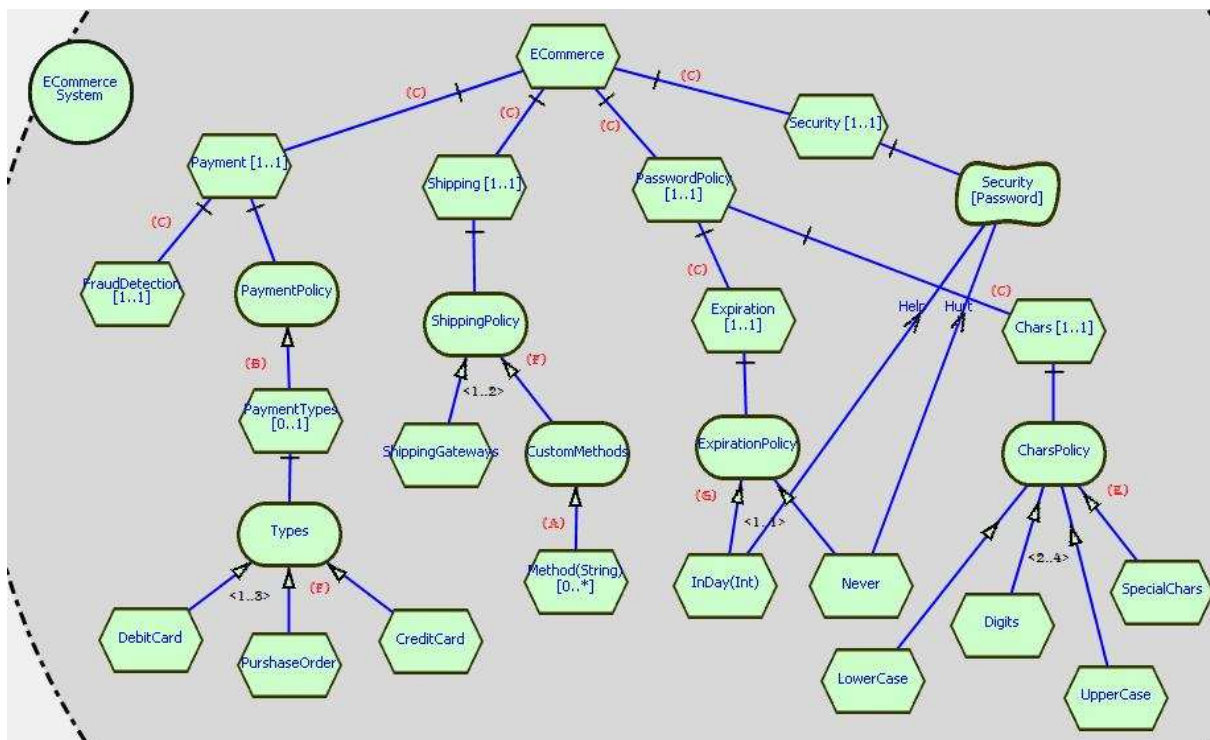
A seguir, apresentamos a notação do **i*-c** (Tabela 3.5), um exemplo (Figura 3.6) e o metamodelo desta linguagem (Figura 3.7). A Tabela 3.5 apresenta como os construtores da notação da linguagem **i*-c** representam os construtores do modelo de *features*.

Tabela 3.5 – Relação entre construtores do modelo de *features* e da linguagem **i*-c**

Modelo de <i>Features</i>		Linguagem i*-c
(A)	Feature com cardinalidade [m..n] <pre> graph TD F1[F1] --- "[m..n]" F2[F2] </pre>	Elementos do tipo recurso e tarefa com cardinalidade [m..n]
(B)	Feature com cardinalidade [0..1] (opcional) <pre> graph TD F1[F1] --- "[0..1]" F2[F2] </pre>	Elementos do tipo recurso e tarefa com cardinalidade [0..1]
(C)	Feature com cardinalidade [1..1] (obrigatória) <pre> graph TD F1[F1] --- "[1..1]" F2[F2] </pre>	Elementos do tipo recurso e tarefa com cardinalidade [1..1]
(D)	Relações que incluem <i>features</i> opcionais, obrigatórias e com cardinalidade [Benavides, D. et al., 2005] <pre> graph TD F1[F1] --- "[0..1]" F2[F2] F1[F1] --- "[1..1]" F3[F3] F1[F1] --- "[m..n]" F4[F4] </pre>	Elementos do tipo tarefa ou recurso com cardinalidade [m..n], [0..1] e [1..1]

(E)	Features agrupadas com cardinalidade $\langle i..j \rangle$	Ligação meio-fim com cardinalidade $\langle i..j \rangle$
(F)	Features agrupadas com cardinalidade $\langle 1..k \rangle$, onde k é tamanho do grupo (or-inclusive)	Ligação meio-fim com cardinalidade $\langle 1..k \rangle$
(G)	Features agrupadas com cardinalidade $\langle 1..1 \rangle$ (or-exclusive)	Ligação meio-fim com cardinalidade $\langle 1..1 \rangle$

A Figura 3.6 ilustra a representação da variabilidade do sistema de comércio eletrônico representado no modelo de *features* da Figura 3.1 com a notação da linguagem **i*-c**.



A Figura 3.7 apresenta o metamodelo da linguagem **i*-c** que foi adaptado a partir de [Lucena, M. et al., 2008] para suportar cardinalidade. Um modelo **i*** (**IstarModel**) é composto por pelo menos um nó (**Node**) que poderá ser um **Dependum** ou um nó dependente (**DependableNode**). Um **DependableNode** pode depender através de uma ligação (**DependerLink**) de um **Dependum**, e a satisfação deste último pode ser da responsabilidade de outro **DependableNode** através de uma ligação **DependeeLink**. Um **Dependum** será definido por um tipo (**DependumKind**). No caso do **DependableNode**, este é especializado num ator (**Actor**) ou num elemento intencional (**InternalElement**).

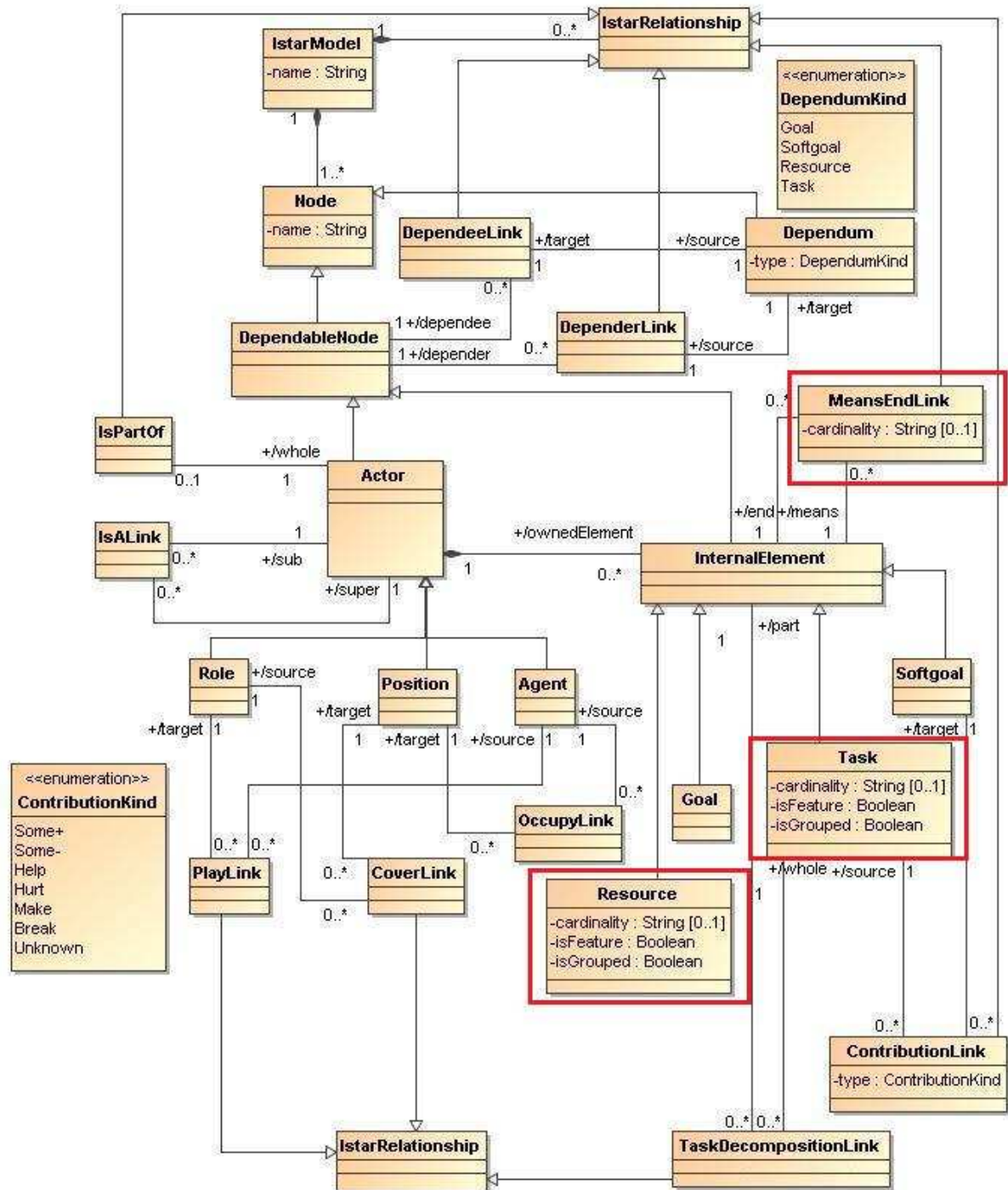


Figura 3.7 – Metamodelo da linguagem i*-c (adaptado a partir de [Lucena, M. et al., 2008])

Um **InternalElement** pode ser especializado em recurso (**Resource**), objetivo (**Goal**), **Softgoal** ou tarefa (**Task**). Um **InternalElement** pode também ser um meio (**means**) para atingir outro **InternalElement** (**end**) através de uma ligação de meio-fim (**MeansEndLink**). Um **Softgoal** pode ter contribuições de outros **InternalElement** através da ligação de contri-

buição (**ContributionLink**). Um **ContributionLink** será definido por um tipo (**ContributionKind**). Uma **Task** poderá ainda ser decomposta em outras tarefas, através da ligação de decomposição **TaskDecompositionLink**.

Um ator (**Actor**) também pode conter zero ou mais **InternalElement**, ser um subtipo de outro ator através da ligação **IsALink**, fazer parte de outros atores através da ligação **IsPartOf**, ou ainda, pode ser especializado em papel (**Role**), posição (**Position**) ou agente (**Agent**). Sendo que um **Agent** pode desempenhar papéis (**Role**) através da ligação **PlayLink**, e pode ocupar uma posição (**Position**) através da ligação **OccupyLink**. Entretanto, um **Role** pode cobrir posições (**Position**) através da ligação **CoverLink**.

O **IstarModel** também pode ser uma agregação de relacionamentos do i* (**IstarRelationship**), sendo que este último é especializado pelas várias ligações existentes no modelo.

Os elementos **Resource**, **Task** e **MeansEndLink** (destacados no metamodelo) foram adaptados para conter a informação de cardinalidade que ajudarão a obter as *features* na abordagem **G2FM**, apresentada em detalhes no capítulo seguinte. Existem algumas restrições quanto ao uso destes elementos na abordagem **G2FM** e por isso foi necessária a construção de algumas regras escritas como expressões em OCL (*Object Constraint Language*) [OMG, 2003].

Essas restrições se referem quanto à utilização da cardinalidade ou-exclusiva dos elementos e dos relacionamentos. Quando a cardinalidade pertencer ao elemento (**Task** ou **Resource**), então, este elemento não é agrupado e a cardinalidade não poderá pertencer ao relacionamento. O inverso também é verdade: quando a cardinalidade pertencer ao relacionamento (**MeansEndLink**), então, os sub-elementos desta ligação estão agrupados e a cardinalidade não poderá pertencer a esses sub-elementos. O atributo que determina se um elemento é agrupado ou não, é **isGrouped**. A Tabela 3.6 apresenta essas restrições como regras OCL dos elementos **Resource**, **Task** e **MeansEndLink** e sua tradução em linguagem natural.

Tabela 3.6 – Restrições em OCL dos elementos adaptados do metamodelo

Regra	Elemento no Metamodelo	Restrição em OCL	Texto Gerado em Linguagem Natural
OCL1	Resource	<pre> context Resource inv: if isFeature and isGrouped then self.cardinality[0] -> MeansEndLink.cardinality[1] else if isFeature then self.cardinality[1] -> MeansEndLink.cardinality[0] else self.isGrouped = false self.cardinality[0] end if </pre>	Regra para um <i>Resource</i>: Se <i>Resource</i> é <i>feature</i> e é agrupado, então não existe cardinalidade no elemento e sim no seu relacionamento <i>MeansEndLink</i>. Se <i>Resource</i> é <i>feature</i> e não é agrupado, então existe cardinalidade no elemento e não no seu relacionamento <i>MeansEndLink</i>. Se <i>Resource</i> não é <i>feature</i>, então <i>isGrouped</i> é falso e não existe cardinalidade
OCL2	Task	<pre> context Task inv: if isFeature and isGrouped then self.cardinality[0] -> MeansEndLink.cardinality[1] else if isFeature then self.cardinality[1] -> MeansEndLink.cardinality[0] else self.isGrouped = false self.cardinality[0] end if </pre>	Regra para uma <i>Task</i>: Se <i>Task</i> é <i>feature</i> e é agrupado, então não existe cardinalidade no elemento e sim no seu relacionamento <i>MeansEndLink</i>. Se <i>Task</i> é <i>feature</i> e não é agrupado, então existe cardinalidade no elemento e não no seu relacionamento <i>MeansEndLink</i>. Se <i>Task</i> não é <i>feature</i>, então <i>isGrouped</i> é falso e não existe cardinalidade
OCL3	MeansEndLink	<pre> context MeansEndLink inv: if self.cardinality[1] then self.means -> size>1 Resource.cardinality[0] and Task.cardinality[0] else self.means -> size=1 Resource.cardinality[1] or </pre>	Regra para um <i>MeansEndLink</i>: Se existir cardinalidade no relacionamento <i>MeansEndLink</i> então, esse relacionamento possui mais de um sub-elemento e a cardinalidade não pertencerá aos seus sub-elementos. Se não existir cardinalidade no relacionamento <i>MeansEndLink</i> então, esse relacionamento possui apenas um sub-elemento e a cardinalidade

		<i>Task.cardinality[1]</i> <i>end if</i>	pertencerá ao seus sub-elemento.
--	--	---	----------------------------------

A regra OCL1 é referente ao elemento **Resource** e diz o seguinte:

- i. Se os atributos **isFeature** e **isGrouped** são verdadeiros, então a cardinalidade não existe no elemento e sim no elemento **MeansEndLink**. Isso significa que se o elemento Recurso é *feature* e é agrupada, então a cardinalidade é do agrupamento e não do elemento.
- ii. Se o atributo **isFeature** é verdadeiro e o atributo **isGrouped** é falso, então a cardinalidade existe no próprio elemento e não existe no elemento **MeansEndLink**. Isso significa que se o elemento Recurso é *feature* e não é agrupada, então a cardinalidade é do elemento e não do agrupamento.
- iii. Se o atributo **isFeature** é falso, então o atributo **isGrouped** é falso e não existe cardinalidade. Isso quer dizer que se o elemento Recurso não é *feature*, então, não pode ser agrupada nem ter cardinalidade.

A regra OCL2 é análoga à OCL1, porém com o elemento **Task**.

A regra OCL3 é referente ao relacionamento **MeansEndLink** e diz o seguinte:

- i. Se existir cardinalidade no relacionamento **MeansEndLink** então, esse relacionamento possui mais de um sub-elemento e a cardinalidade não pertencerá aos seus sub-elementos.
- ii. Se não existir cardinalidade no relacionamento **MeansEndLink** então, esse relacionamento possui apenas um sub-elemento e a cardinalidade pertencerá a esse sub-elemento.

As regras OCL1, OCL2 e OCL3 são necessárias para a utilização da abordagem **G2FM** apresentada no próximo capítulo. Como o metamodelo da Figura 3.7 é bastante genérico, faz-se necessária a utilização de regras para que este metamodelo corresponda à versão do i* adotada nesse trabalho [Grau, G. et al., 2009]. Dessa forma, duas regras foram adaptadas a partir de [Lucena, M. et al., 2008]. A Tabela 3.7 apresenta essas duas novas regras.

Tabela 3.7 – Restrições em OCL para utilização da versão i* de [Grau, G. et al., 2009]

Restrição em OCL	Texto Gerado em Linguagem Natural
<i>context MeansEndLink inv:</i> <i>self.means.ocllsTypeOf(Task)</i> <i>self.end.ocllsTypeOf(Goal)</i> <i>self.means.ocllsTypeOf(Goal)</i> <i>self.end.ocllsTypeOf(Goal)</i> <i>implies</i> <i>or</i> <i>implies</i>	Regra para um <i>MeansEndLink</i>: O elemento meio (<i>means</i>) pode ser do tipo tarefa (<i>Task</i>) ou objetivo (<i>Goal</i>) e o elemento fim (end) é sempre do tipo objetivo (<i>Goal</i>).
<i>context ContributionLink inv</i> <i>self.target.ocllsTypeOf(SoftGoal)</i> <i>self.source.ocllsTypeOf(SoftGoal)</i> <i>self.source.ocllsTypeOf(Goal)</i> <i>self.source.ocllsTypeOf(Resource)</i> <i>self.source.ocllsTypeOf(Task)</i> <i>implies</i> <i>or</i> <i>or</i> <i>or</i>	Regra para um <i>ContributionLink</i>: Os elementos <i>Softgoal</i>, <i>Goal</i>, <i>Resource</i> e <i>Task</i> podem contribuir para a satisfação de um <i>soft-goal</i>.

3.3. Considerações Finais

Neste capítulo foi apresentada uma comparação da expressividade entre abordagens orientadas a objetivos (modelos de objetivos, PL-AOVGraph, i* Aspectual e i*) e um modelo de *features*. Com o resultado desta comparação, conclui-se que essas três abordagens têm a expressividade limitada, pois não são capazes de modelar a representação da cardinalidade do modelo de *features*. Essa limitação nos levou a estender o i* para suportar cardinalidade e assim, solucionar o problema de mapeamento entre modelo de *features* e modelos orientados a objetivos mencionados acima. Apresentamos a notação gráfica adotada e a utilização desta notação em um exemplo. Por fim, definimos o metamodelo da nova linguagem **i*-c** e as restrições associadas a ele.

O capítulo seguinte apresentará em detalhes a abordagem **G2FM**, objeto principal desta dissertação de mestrado.

Capítulo

4. Processo para extrair *features* a partir do Modelo i*

Este capítulo apresenta em detalhes a abordagem **G2FM** (*Goal To Feature Model*) que permite capturar *features* comuns e variáveis em uma linha de produto de software a partir de uma extensão da linguagem de modelagem do i*, a linguagem **i*-c**. Para exemplificar a utilização desta abordagem, um pequeno exemplo de uma linha de produtos de software é utilizado. Por último, são apresentadas as considerações finais.

Neste capítulo é apresentado o processo que descreve nossa abordagem, chamada de **G2FM** (*Goal To Feature Model*), que tem como propósito possibilitar a identificação de *features* nos modelos SR do i* para a construção do modelo de *features* de um determinado produto da LPS. Para alcançar este propósito, definimos algumas atividades e heurísticas, formando um processo bem definido que será ilustrado com um caso de uso que faz parte de uma linha de produtos de software, o **MobileMedia** [Figueiredo, E. et al., 2008]. O principal objetivo do **MobileMedia** é manipular fotos, música e vídeo em dispositivos móveis como, por exemplo, telefones celulares.

Para ilustrar a abordagem **G2FM**, neste capítulo focaremos apenas na modelagem do caso de uso **Adicionar Foto** do sistema **MobileMedia**. A modelagem completa desse sistema será apresentada apenas no capítulo seguinte, como parte do nosso estudo de caso.

4.1. Visão geral do processo

A Figura 4.1 apresenta uma visão geral do processo que descreve a abordagem **G2FM**. Este processo foi modelado na notação BPMN (*Business Process Modeling Notation*) [BPMN, 2008] com auxílio de uma ferramenta [BizAgi, 2009].

O processo é composto por sete atividades, sendo duas opcionais: construção do modelo SR, identificação de elementos com possibilidade de serem *features*, reengenharia do modelo SR, elaboração do modelo de *features*, reorganização do modelo de *features*, configuração do produto no modelo SR, e por fim, elaboração do modelo de configuração do produto. O responsável pelas cinco primeiras atividades, geralmente tem o papel de engenheiro do domínio. Já para as atividades de configuração do produto e elaboração do modelo de configuração, o responsável tem outro papel, geralmente de engenheiro de configuração.

Nas subseções seguintes, detalhamos e justificamos cada uma das fases, utilizando como exemplo um fragmento do sistema **MobileMedia**. À medida que cada atividade do processo for apresentada, mostraremos como ela é aplicada no exemplo escolhido.

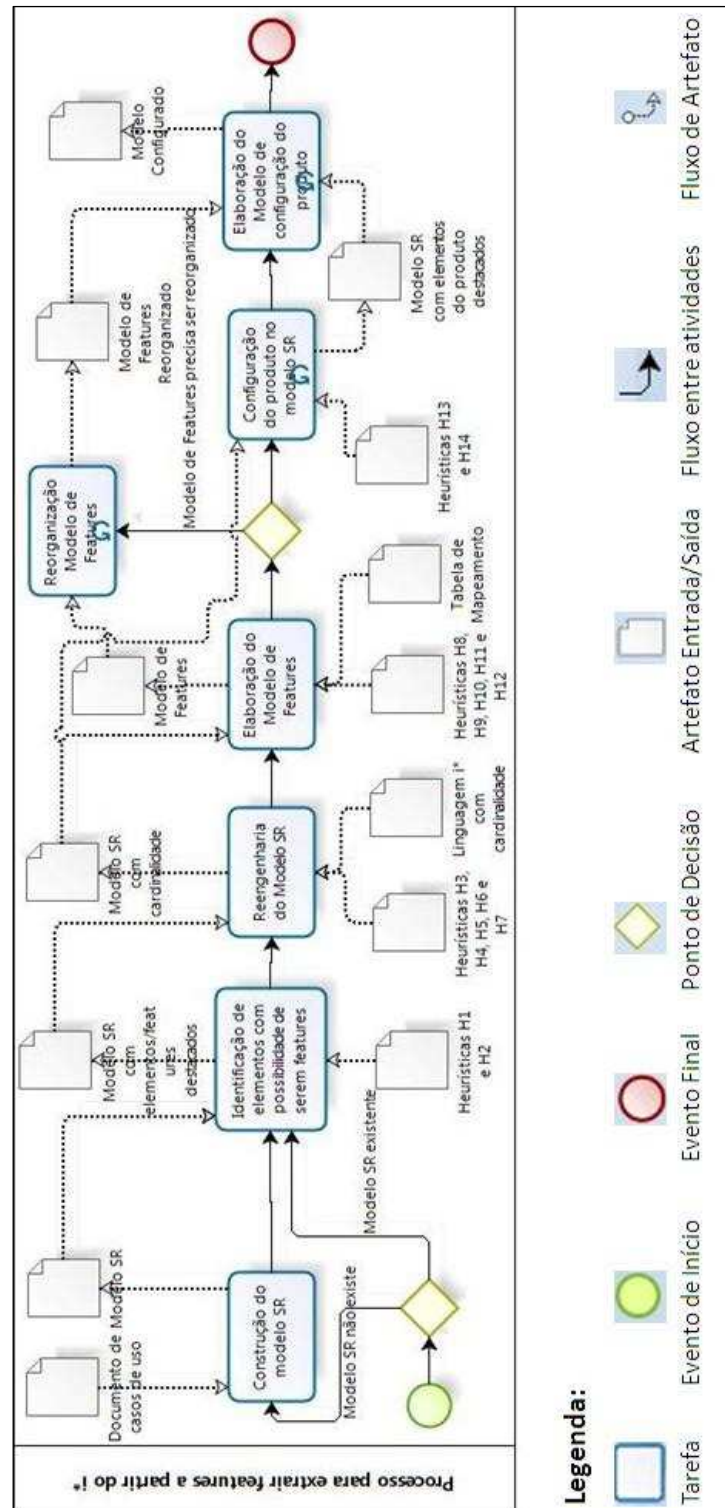


Figura 4.1 - Processo para extrair *features* a partir do modelo SR do i* em BPMN

4.2. Construção do modelo SR (opcional)

A primeira atividade do processo é a construção do modelo SR do *framework i**, que tem como entrada o documento de casos de uso do sistema a ser modelado. Esta atividade pode ser considerada opcional, desde que já exista um modelo SR do sistema em desenvolvimento.

Para a realização desta atividade, baseamo-nos no PRiM (*Process Reengineering i* Method*) [Grau, G. et al., 2008] que é um método aplicado na reengenharia de processos de negócio. Adaptamos algumas atividades da fase 2 desse método, pois esta se refere à construção do modelo SR do *i**. Para a construção desse modelo, o PRiM utiliza uma notação própria para cenários de processo, chamada de “*Detailed Interaction Script*” (DIS). Esta notação é bastante parecida com a descrição de casos de uso, podendo ser adaptada para a sua utilização no processo **G2FM**.

O sistema **MobileMedia** é baseado em casos de uso, então para mostrarmos como esta atividade é aplicada nesse sistema, selecionamos o caso de uso **Adicionar Foto**. A especificação original deste caso de uso encontra-se na Figura 4.2, porém, durante a demonstração do exemplo, sentimos a necessidade de adicionar mais informações no caso de uso para conseguirmos um rico modelo *i**.

Usecase: Add Photo
Actor: Mobile Phone (system) and User
Description: The user can store (add) an image in an album available in the device
Precondition: Application must be launched; There must be an album available.
Trigger: User chooses to add an image
Scenario: 1. (select) The user selects an album to store the image. 2. (add) The user selects the option to add image. 3. (path) User provides the path from where the image is uploaded. 4. (label) User assigns a label to the image. 5. (saved) The new image is stored in the selected album. 6. (update) The list is updated with the modified information
Extensions: 3a. (path) Invalid Information: 3a1. Incorrect path is provided 3a2. The image at the specified path is inaccessible 4a. (label) The label already exists: 4a1. options to either replace the existing image 4a2. save the image with other name 5a. (saved) Problem occurs with storing image in specified album and/or device memory 6a. (update) Unable to update the modified information
Includes: Provide Label Usecase

Figura 4.2 – Especificação do caso de uso Adicionar Foto do sistema MobileMedia

A fase 2 do PRiM é composta por vários passos e regras que foram simplificados para a sua utilização no nosso processo: identificação e modelagem dos atores e objetivos principais, construção dos modelos i* que o PRiM divide em operacional e intencional. Para construir o modelo i* operacional é necessário executar os seguintes passos: modelagem de atividades, modelagem de tarefas, modelagem de tarefas que produzem/fornecem recursos, modelagem de tarefas que consomem recursos, modelagem de tarefas alternativas, modelagem de pré-condições, pós-condições e eventos de *trigger*. Para construção do modelo i* intencional é preciso executar a análise dos elementos intencionais e a análise de atributos de qualidade. A Regra 7 que trata da modelagem dos relacionamentos do tipo *includes* e *extends* não foi extraída do PRiM, sendo uma contribuição deste trabalho. Abaixo, detalhamos esses passos e regras apenas como guia para a construção do modelo SR.

- **Passo 1.1 - Identificação e modelagem dos atores e objetivos principais:** o primeiro passo da modelagem do i* é a identificação dos atores e seus principais objetivos a partir da documentação do sistema a ser modelado. Geralmente, os atores já vêm identificados na própria documentação do sistema, enquanto para identificar os objetivos é necessária uma análise da documentação do sistema.

Aplicação no exemplo: De acordo com a especificação do caso de uso **Adicionar Foto**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente (vide Figura 4.3).

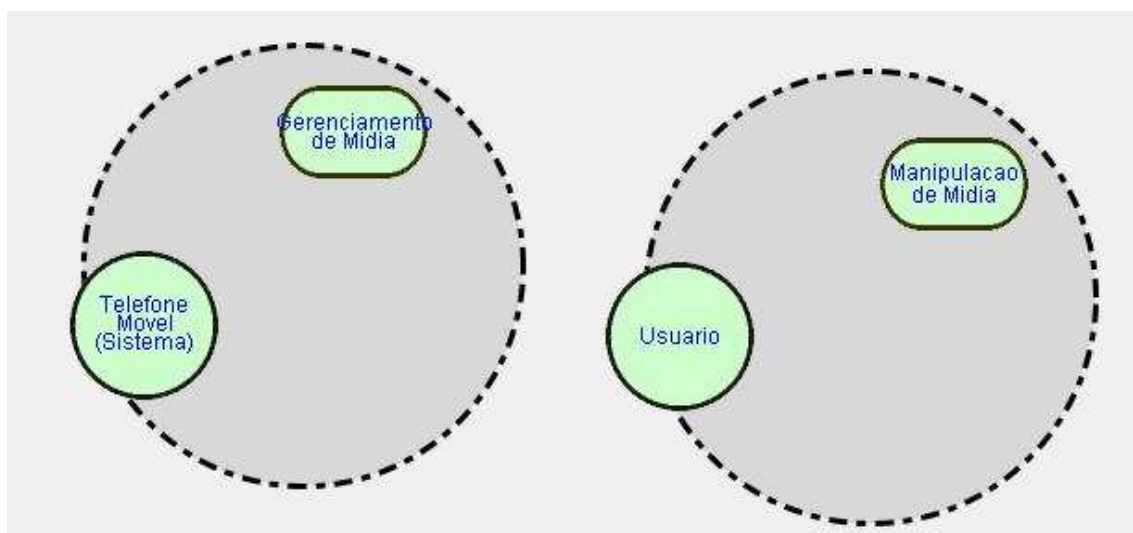


Figura 4.3 - Atores e seus principais objetivos

- **Passo 1.2 – Construção do modelo i* operacional**

- a. **Regra 1 – Modelagem de atividades:** cada atividade em que um ator está envolvido é modelada como uma tarefa no seu modelo SR. Por exemplo, para um documento de casos de uso, cada atividade em questão corresponde a um determinado caso de uso. Esta tarefa (derivada da atividade) é associada ao objetivo principal de cada ator identificado no passo anterior, utilizando o relacionamento meio-fim. Depois que todas as atividades são modeladas, nós obtemos o primeiro nível de decomposição no modelo SR de cada ator.

Aplicação no exemplo: De acordo com esta regra, o caso de uso **Adicionar Foto** é modelado como tarefa e associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim (vide Figura 4.4).

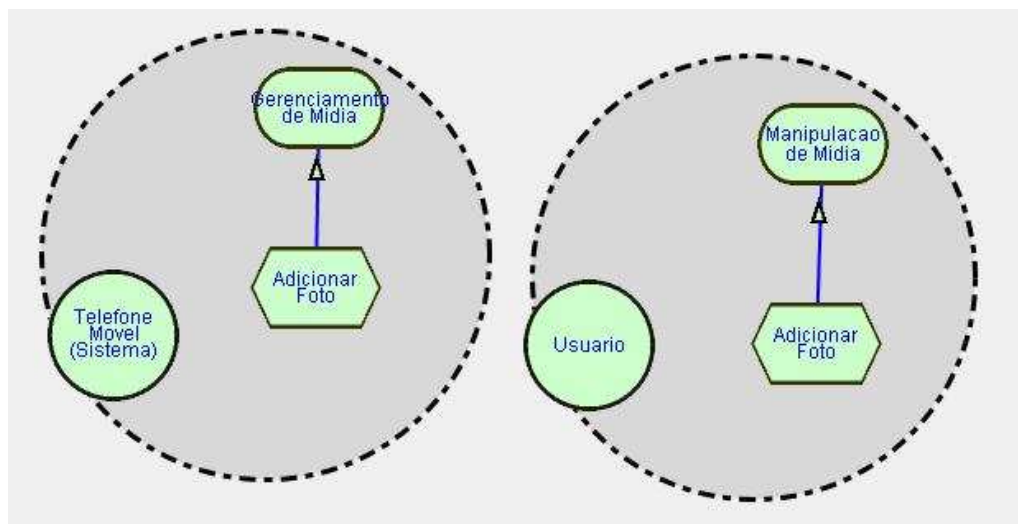


Figura 4.4 - Tarefa Adicionar Foto

- b. **Regra 2 – Modelagem de tarefas:** cada tarefa identificada na regra anterior é decomposta em sub-tarefas que correspondem aos passos do cenário principal dos casos de uso. Isto é, cada tarefa (derivada de cada caso de uso) é decomposta em sub-tarefas através do relacionamento de decomposição de tarefas. Estas tarefas correspondem aos passos do cenário principal do caso de uso em questão e são modeladas no ator que iniciou a tarefa.

Aplicação no exemplo: Os passos do cenário principal do caso de uso em questão são mapeados em tarefas e relacionados à tarefa **Adicionar Foto** através do relacionamento de decomposição de tarefas. As tarefas iniciada no ator **Telefone Móvel (sistema)** seriam: **Armazenar Foto** e **Atualizar lista de fotos**, porém, não vemos sentido em modelar **Atualizar lista de fotos** como tarefa e sim como objetivo, pois, entendemos como uma pós-condição do caso de uso (vide Regra 6). Assim, o ator **Telefone Móvel (sistema)** tem apenas a tarefa **Armazenar Foto**. As tarefas iniciadas no ator **Usuário** são: **Selecionar Álbum**, **Selecionar opção de adicionar fotos**, **Fornecer local da foto** e **Atribuir nome para foto**. A Figura 4.5 apresenta estas tarefas.

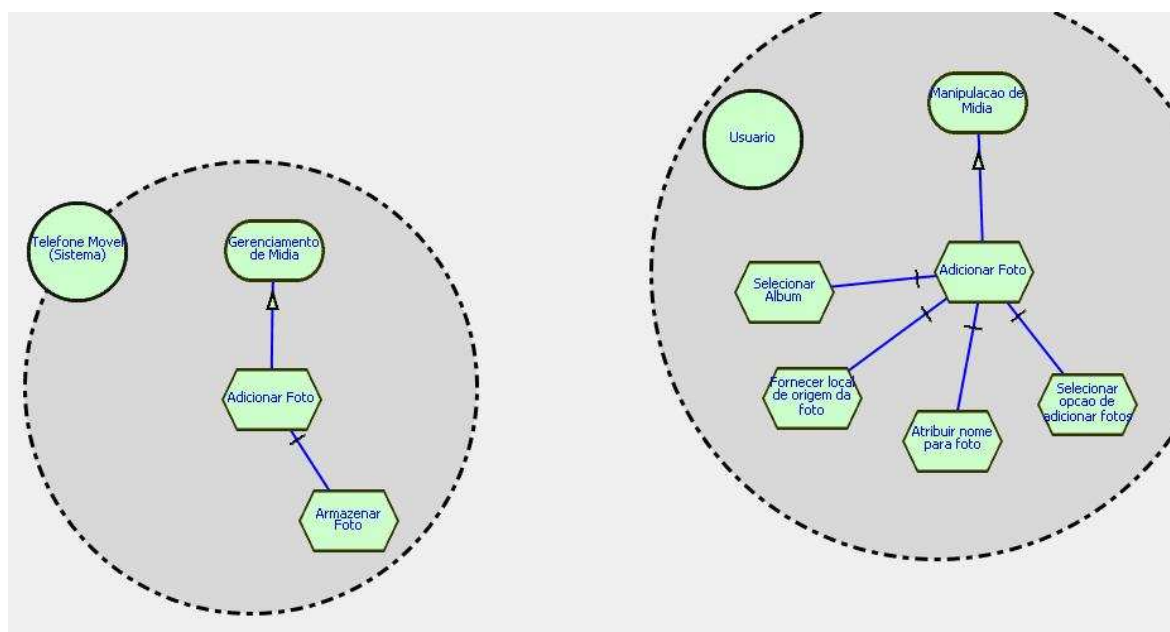


Figura 4.5 - Tarefas do caso de uso Adicionar Foto

- c. **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** se uma tarefa da regra anterior produz ou fornece recurso, este recurso se torna uma dependência (*dependum*) entre a tarefa que o produziu (no ator *dependee*) e, a tarefa que consumirá o recurso (no ator *dependor*).

Aplicação no exemplo: A tarefa **Fornecer local da foto** (*dependee*) produz os recursos **Foto** (*dependum*) e **Local** (*dependum*) que vão ser consumidos pela tarefa **Armazenar Foto** (*dependor*). A tarefa **Atribuir nome para foto** (*dependee*) fornece o recurso **Nome** (*dependum*) que vai ser consumido pela tarefa

Armazenar Foto (*dependor*). Essas dependências podem ser visualizadas na Figura 4.6.

- d. **Regra 4 – Modelagem de tarefas que consomem recursos:** uma tarefa pode consumir um determinado recurso. Se este recurso já existe no modelo (ou seja, já foi produzido por outra tarefa), então, é relacionado à tarefa que o produziu; caso contrário, é relacionado diretamente à tarefa do ator que contém a tarefa em questão.

Aplicação no exemplo: A tarefa **Armazenar Foto** consome recursos, mas já foi modelada pela regra anterior. A tarefa **Selecionar Álbum** (*dependor*) depende que o recurso **Álbum** (*dependum*) esteja disponível para que essa tarefa seja realizada. Como este recurso não foi produzido pela regra anterior, então, será produzido pela tarefa **Adicionar Foto** (*dependee*). Essa dependência também está modelada na Figura 4.6.

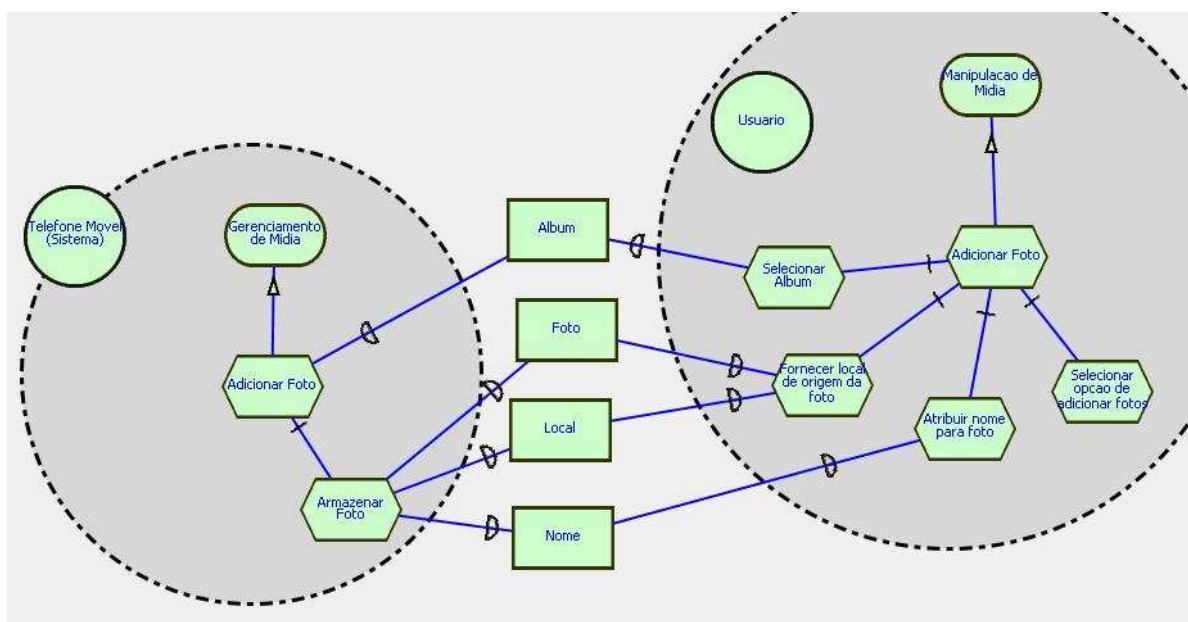


Figura 4.6 – Dependência de recurso entre as tarefas da tarefa Adicionar Foto

- e. **Regra 5 – Modelagem de tarefas alternativas:** no PRiM, esta regra diz que caso a documentação que está sendo analisada tenha cenários alternativos ou de extensão, então os passos destes cenários também serão modelados como tarefas. Porém, nesta dissertação, adaptamos essa regra para considerar apenas cenários alternativos e não os de exceção: caso a documen-

tação que está sendo analisada tenha algum passo alternativo no cenário principal, então esse passo dará origem a um objetivo genérico e tanto o passo no cenário principal como o passo alternativo serão modelados como tarefas alternativas para alcançar este objetivo. Porém, como nem sempre estas tarefas serão realizadas (já que são alternativas), o relacionamento entre estas tarefas e o objetivo genérico é a ligação meio-fim. Já o relacionamento entre esse objetivo e a tarefa oriunda do cenário principal é a ligação de decomposição de tarefas.

Aplicação no exemplo: Apesar da especificação original do caso de uso **Adicionar Foto** (vide Figura 4.2) não conter cenários alternativos, apenas de exceção, sentimos a necessidade de acrescentar informações que achamos relevantes enriquecer a especificação do caso de uso. Adicionamos cenário alternativo ao passo 5 do cenário principal: “5. (saved) The system saves automatically the new image in the default album”. A modelagem deste cenário alternativo fica da seguinte forma: a tarefa **Armazenar Foto** tem uma ligação de decomposição com o objetivo **Salvar Foto**, que por sua vez, tem as seguintes tarefas alternativas: **Salvar Automaticamente** e **Salvar pelo Usuário**. Nesse caso, verificamos que a dependência do recurso **Álbum** passou a ser com a tarefa alternativa **Salvar pelo Usuário**, ao invés de ser com a tarefa **Adicionar Foto**. A modelagem dessas tarefas alternativas pode ser vista na Figura 4.7.

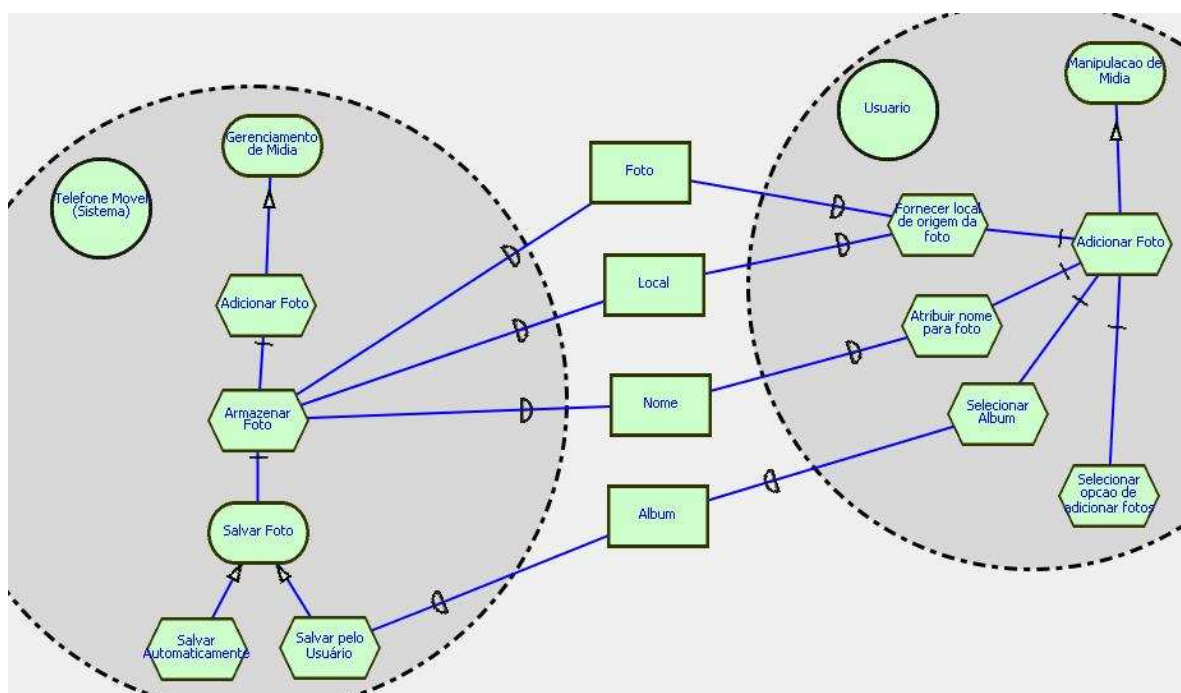


Figura 4.7 – Tarefas alternativas do caso de uso Adicionar Foto

- f. **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de *trigger*:** cada pré-condição, pós-condição e evento de *trigger* da atividade (ou caso de uso) têm que ser modelada explicitamente. Como eles representam a realização de um determinado estado, então, são modelados como objetivos no modelo SR. Pré-condições e pós-condições são adicionadas na tarefa utilizando o relacionamento de decomposição de tarefas. Já os eventos de *trigger* são modelados como dependências de objetivos entre uma tarefa do ator que dispara a *trigger* (*dependee*) e uma tarefa do ator que é atingido pela *trigger* (*dependor*).

Aplicação no exemplo: As pré-condições da tarefa **Adicionar Foto** são **Aplicação Iniciada** e **Álbum disponível**. De acordo com esta regra, essas pré-condições são modeladas, no ator que representa o sistema, como objetivos e relacionados através do relacionamento de decomposição de tarefas. A especificação original do caso de uso **Adicionar Foto** não contém pós-condições, porém, como vimos anteriormente na Regra 2, supomos e assim, modelamos **Atualização de lista de fotos** como uma pós-condição do caso de uso. Da mesma forma que as pré-condições, essa pós-condição é modelada, no ator que representa o sistema, como um objetivo relacionado através do relacionamento de decomposição de tarefas com a tarefa que representa o caso de uso. O evento de *trigger* da tarefa **Adicionar Foto** é **Adição de Foto**. Esta *trigger* é um *dependum* do tipo objetivo onde o *dependee* é a tarefa **Armazenar Foto** e o *dependor* é a tarefa **Selecionar opção de adicionar foto**. Essas modelagens podem ser vistas na Figura 4.8.

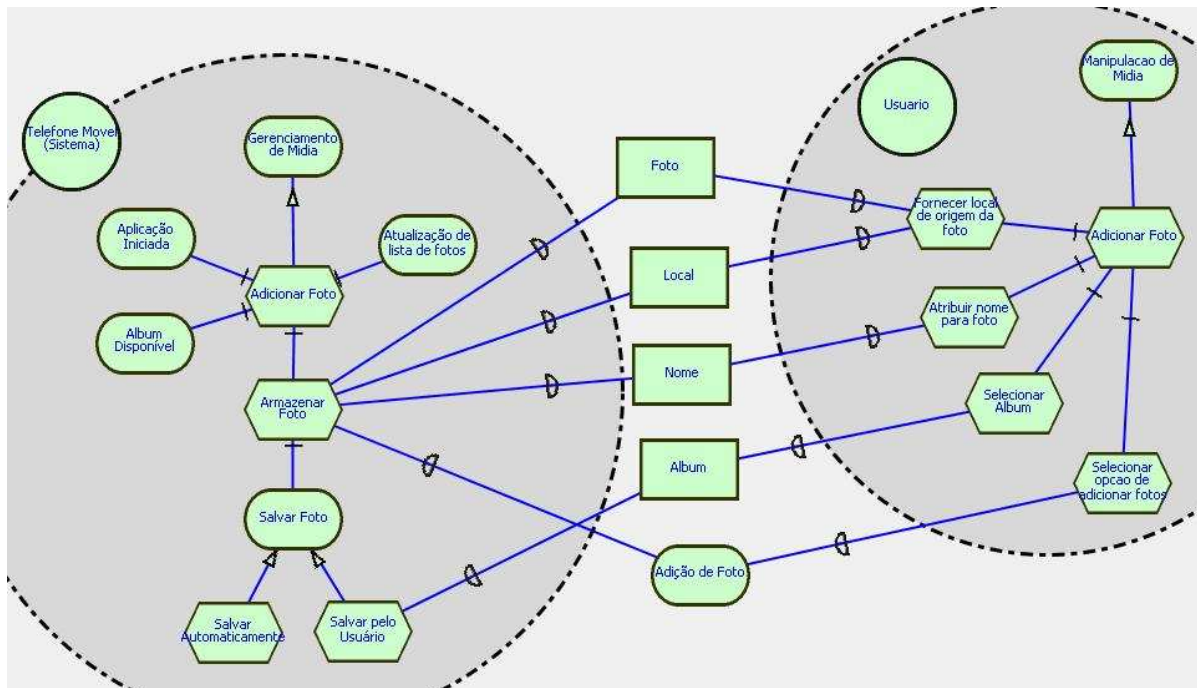


Figura 4.8 - Modelagem de pré-condições, eventos de *trigger* da tarefa Adicionar Foto

- g. **Regra 7 - Modelagem dos relacionamentos do tipo *includes* e *extends*:** o relacionamento do tipo *includes* significa que em algum ponto do caso de uso haverá uma chamada explícita à execução do caso de uso de inclusão. No caso de existir esse tipo de caso de uso, este é modelado como tarefa utilizando o relacionamento de decomposição de tarefas. O relacionamento do tipo *extends* modela uma variação no comportamento normal do caso de uso. Esse comportamento “extra” é separado num caso de uso denominado de extensão e é modelado como um objetivo. Como nem sempre os casos de uso de extensão serão realizados, o relacionamento é meio-fim com o objetivo principal do caso de uso que o contém.

Aplicação no exemplo: o caso de uso **Adicionar Foto** contém um relacionamento do tipo *includes* com o caso de uso **Fornecer Legenda**. De acordo com esta regra, **Fornecer Legenda** é modelada como sub-tarefa (através do relacionamento de decomposição de tarefas) da tarefa **Adicionar Foto**. Nesse caso, verificamos que a dependência do recurso **Nome** passou a ser com a tarefa **Fornecer Legenda**, ao invés de ser com a tarefa **Armazenar Foto**. A modelagem desse relacionamento *includes* pode ser vista na Figura 4.9.

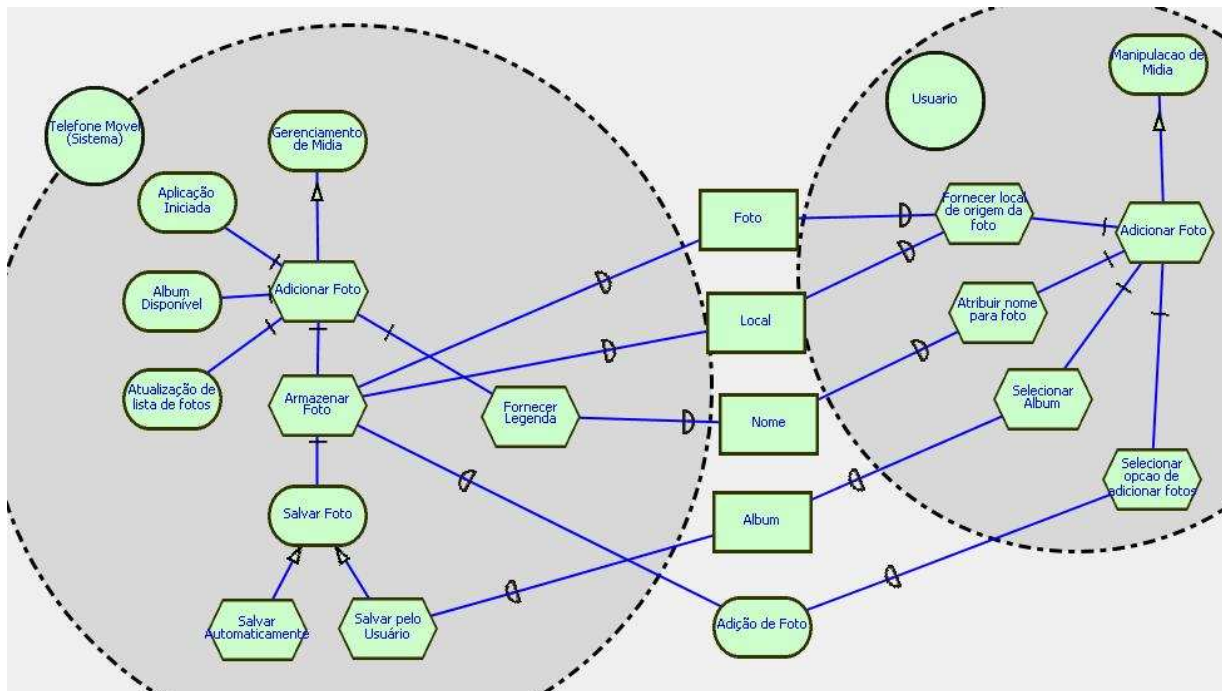


Figura 4.9 – Modelagem do relacionamento *includes* no caso de uso Adicionar Foto

- **Passo 1.3 – Construção do modelo i* intencional**

O PRiM considera dois tipos de modelos i*: operacional e intencional. No modelo i* operacional são consideradas tarefas e recursos, enquanto que no modelo i* intencional são considerados objetivos e *softgoals*. O modelo intencional complementa o operacional, porém é mais subjetivo, dependendo da estratégia da organização, do nível de detalhes da documentação do sistema a ser modelado, do grau de conhecimento do *stakeholder* que está realizando a modelagem, etc. Diante do exposto, algumas diretrizes são utilizadas como tentativa de tornar mais sistemática a descoberta dos elementos intencionais do modelo.

1. **Diretriz 1 – Análise dos elementos intencionais:** assumindo que cada tarefa modelada com a regra 1 do passo 1.2 representa uma realização de um objetivo, a resposta da seguinte pergunta se tornará um objetivo estratégico para cada atividade (ou caso de uso) do modelo:

- Qual o estado final alcançado através da execução da atividade em questão?

A identificação do *depender* e do *dependee* desse objetivo é obtida através das seguintes perguntas, respectivamente:

- Qual o ator precisa atingir este objetivo?
- De que ator depende para obter o objetivo?

Vale salientar que este objetivo não é a pós-condição da atividade (ou caso de uso). Para a modelagem deste objetivo, é necessária uma reorganização na hierarquia do modelo SR, pois este objetivo será inserido entre a tarefa e o objetivo principal do ator *depend*. Ou seja, a tarefa se tornará o meio para obter este objetivo, que por sua vez será o meio para alcançar o objetivo principal.

Aplicação no exemplo:

- Qual o estado final alcançado através da execução da tarefa “Adicionar Foto”?

A resposta pode ser: “Nova foto incluída no álbum selecionado”.

- Que ator precisa atingir este objetivo (*depend*)?

Resposta: o Telefone Móvel (sistema)

- De que ator depende a obtenção do objetivo (*dependee*)?

Resposta: depende do usuário

Este objetivo (**Nova foto incluída no álbum**) será inserido entre a tarefa e o objetivo principal do ator *depend*, ou seja, entre a tarefa **Adicionar Foto** e o objetivo **Gerenciamento de Mídia** do ator **Telefone Móvel (sistema)**. Ou seja, **Adicionar Foto** se tornará o meio para obter o objetivo **Nova foto incluída no álbum**, que por sua vez será o meio para alcançar o objetivo principal, **Gerenciamento de Mídia** (vide Figura 4.10).

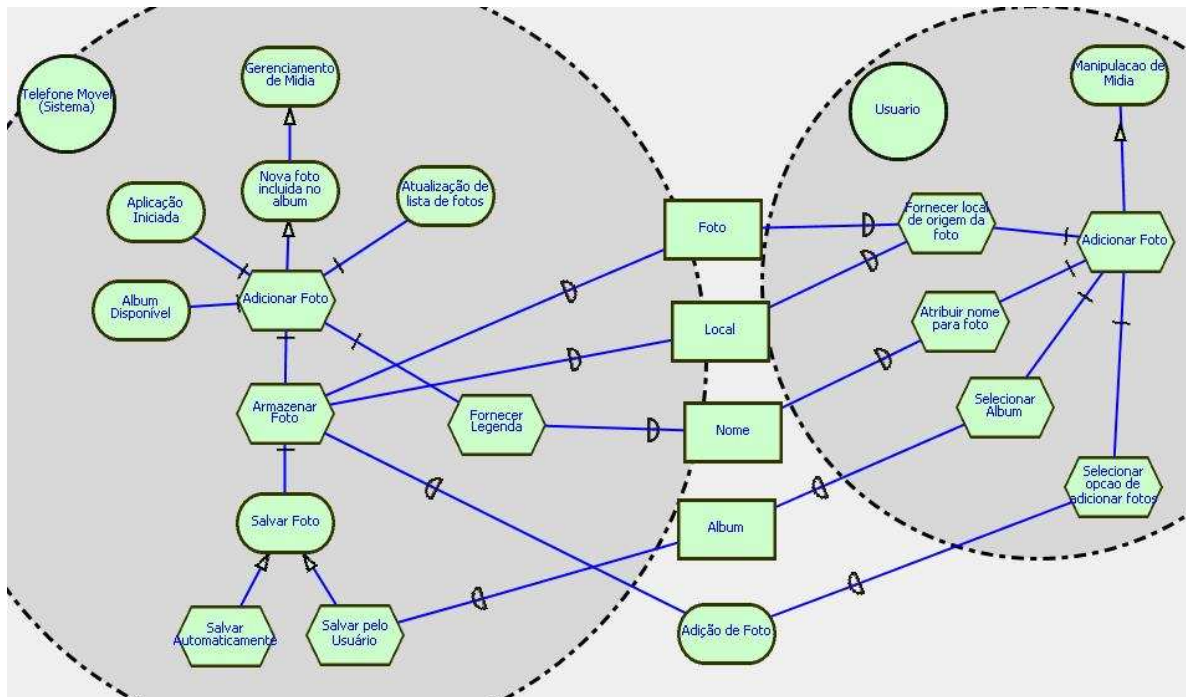


Figura 4.10 – Inclusão de objetivo entre a tarefa Adicionar Foto e o objetivo Gerenciamento de Mídia do ator sistema

2. **Diretriz 2 – Análise de atributos de qualidade:** atributos de qualidade podem ser relacionados com o tipo de elemento intencional: por exemplo, se o elemento analisado é um recurso, pode-se perguntar sobre precisão e segurança de dados; se o elemento é uma tarefa, pode-se perguntar sobre eficiência ou usabilidade. Por exemplo:

- Se um determinado recurso não é suficientemente correto, o processo falha? Que atores serão afetados?
- Se um determinado recurso tem a privacidade de seus dados violados, algum ator será prejudicado?
- Se uma determinada tarefa não é realizada eficientemente, o processo falha?

As respostas a essas perguntas auxiliam na obtenção e modelagem de atributos de qualidade, ou seja, *softgoals*, que serão úteis para a identificação de contribuições e conflitos entre os elementos do modelo.

Aplicação no exemplo: Neste ponto, obtemos e modelamos os requisitos não-funcionais do sistema. Observamos que para o sistema conseguir armazenar uma foto, o usuário depende que o celular (*dependee*) mantenha a

integridade da foto e o sistema depende que o usuário (*dependee*) tenha indicado o local com precisão. Todos estes desejos são representados pelos *softgoals* **Integridade [Foto]** e **Precisão [Local]**. Concluimos também que caso a foto seja salva pelo próprio sistema, ou seja, automaticamente, o armazenamento da mesma é mais rápido do que se for salva pelo usuário. Assim, a tarefa **Salvar Automaticamente** contribui positivamente para satisfazer o *softgoal* **Rapidez [Armazenamento]**. Já a tarefa **Salvar pelo Usuário** contribui negativamente para a satisfação do mesmo *softgoal*. O modelo SR final do caso de uso **Adicionar Foto** é apresentado na Figura 4.11.

Depois de aplicar todos os passos e regras descritos acima aos cenários de uso do sistema contidos em um documento de requisitos que está sendo analisado, obtêm-se um modelo SR que será utilizado como entrada para a atividade seguinte.

Vale salientar que o objetivo principal da atividade que foi apresentada é o de produzir um modelo SR, ficando a critério de cada *stakeholder* a forma de construção do modelo. Apenas sugerimos o método em que nos baseamos (PRiM) por ter uma atenção na construção de modelos *i** de maneira sistemática.

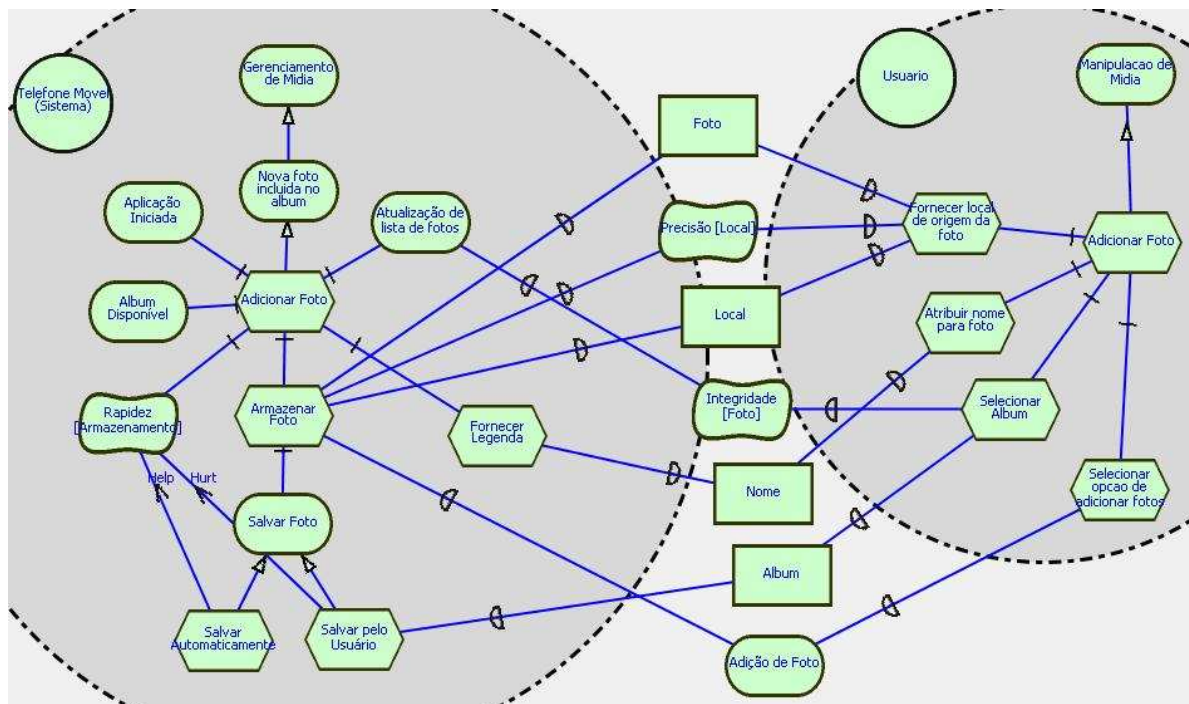


Figura 4.11 – Modelo SR final do caso de uso Adicionar Foto

4.3. Identificação de elementos com possibilidade de serem *features*

Uma vez que o modelo SR é obtido, o usuário precisa identificar quais elementos deste modelo estarão presentes no modelo de *features*. Como vimos anteriormente, uma *feature* pode indicar qualquer característica ou funcionalidade do sistema e é usada para capturar similaridades ou variabilidades nos produtos de uma LPS [Czarnecki, K. and Eisenecker, U., 2000]. Partindo desta definição, uma maneira direta para obter as possíveis *features* através dos elementos intencionais do modelo SR está descrita na heurística H1.

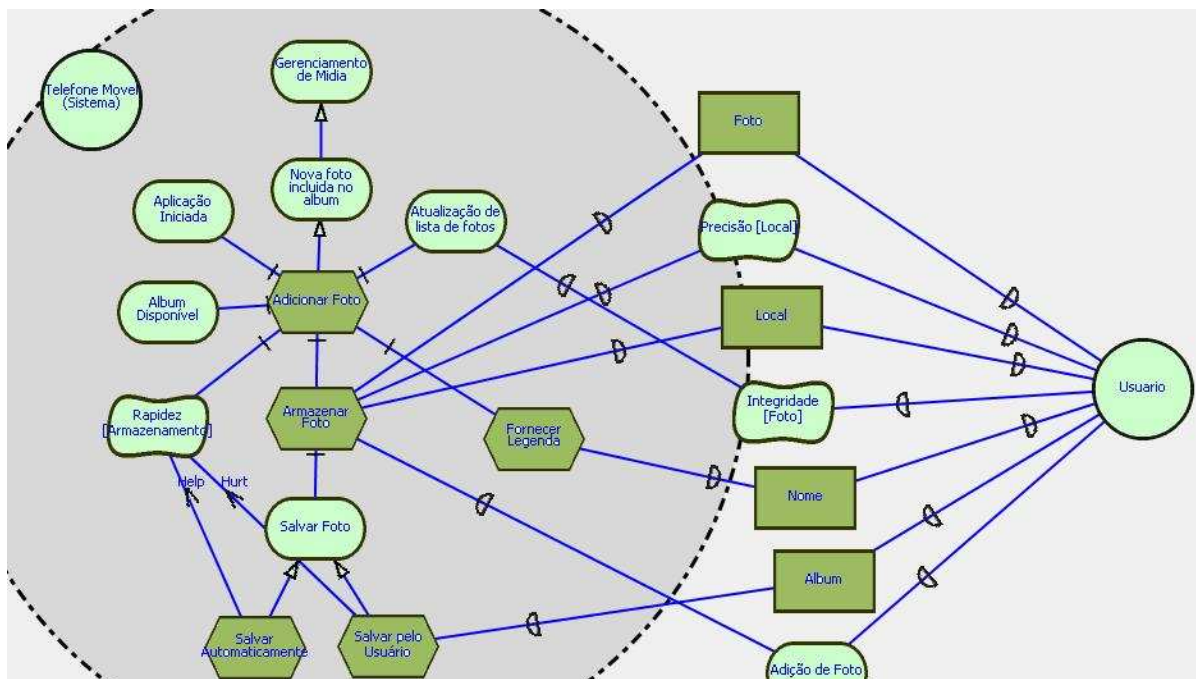
H1: As *features* podem ser extraídas a partir dos elementos do tipo tarefa e recurso, já que determinam funcionalidades e características do sistema, respectivamente.

Geralmente, os modelos SR têm um ator que representa o sistema. Em uma modelagem de LPS, este ator representará essa LPS. Assim, apenas os elementos internos e as dependências ligadas diretamente a esse ator serão considerados para a obtenção de *features*, descritas na heurística H2.

H2: As *features* podem ser extraídas a partir dos elementos internos do ator que representa o sistema e das dependências (particularmente do *dependum*) ligadas diretamente a esse ator. Os tipos dos elementos são os mesmos definidos na heurística H1, ou seja, tarefas e recursos.

Aplicação no exemplo: Utilizando com entrada o modelo SR da Figura 4.11, obtido na atividade anterior, aplicamos as heurísticas H1 e H2, e obtemos o seguinte resultado: todos os elementos do tipo tarefa do ator que representa o sistema (**Telefone Móvel**) são destacados como possíveis *features*. São elas: **Adicionar Foto, Armazenar Foto, Fornecer Legenda, Salvar Automaticamente e Salvar pelo Usuário**. As dependências do tipo recurso **Nome, Local, Foto e Álbum** ligados diretamente às tarefas do ator **Telefone Móvel** também são destacados como possíveis *features*.

A Figura 4.12 representa a saída desta atividade do processo, ou seja, um modelo SR com os elementos possíveis de serem *features* destacados (elementos com cor de fundo).



4.4. Reengenharia do Modelo SR

Após as possíveis *features* terem sido identificadas, nesta atividade deve-se reestruturar o modelo SR gerado na atividade anterior, de acordo com a aplicação de algumas heurísticas e da linguagem i* com cardinalidade (**i*-c** apresentada no capítulo anterior). A cardinalidade a ser usada nos modelos i* tem o mesmo significado que a dos modelos de *features* de [Czarnecki, K. et al., 2005a] e é relembrada na Tabela 4.1.

H3: Todos os elementos do tipo tarefa e recurso do modelo SR que foram destacados como possíveis *features* serão reestruturados para conter cardinalidade. Os demais elementos do modelo não sofrerão modificações.

Não definimos heurísticas para indicar que tipo de cardinalidade deverá ser utilizada, já que depende muito do sistema que está sendo modelado. Apenas indicamos que a cardinalidade a ser usada deve se basear na Tabela 4.1.

A ligação meio-fim nos modelos SR significa que um objetivo (fim) pode ser alcançado através do sub-elemento de onde parte a ligação (ou seja, o meio). Como desejamos modelar uma LPS, a cardinalidade associada a esse relacionamento indica que as *features* obtidas através dos sub-elementos dessa ligação estarão agrupadas no modelo de *features*,

desde que exista mais de um sub-elemento. Se apenas existir um sub-elemento “meio”, então a *feature* obtida através dele não será agrupada no modelo de *features*, significando que a cardinalidade pertencerá ao sub-elemento.

H4: Se a quantidade de sub-elementos de uma ligação meio-fim que estão destacados como possíveis *features*, for maior do que 1, então esses sub-elementos serão agrupados e a cardinalidade pertencerá ao relacionamento. Se a quantidade for exatamente 1, então a cardinalidade pertencerá ao sub-elemento e não ao relacionamento.

A ligação de decomposição de tarefas nos modelos SR denota obrigatoriedade, significando que a cardinalidade da(s) *feature(s)* obtida(s) através do(s) sub-elemento(s) dessa ligação será(ão) [1..1].

H5: Os sub-elementos de uma ligação de decomposição de tarefas que estão destacados como possíveis *features*, serão modelados como *features* obrigatórias, significando que a cardinalidade pertencerá aos sub-elementos e terá a cardinalidade [1..1].

Os elementos que não estão englobados pelas heurísticas H4 e H5 estão presentes nas definições das heurísticas H6 e H7.

H6: As dependências (*dependum*), que estão destacadas como possíveis *features*, terão a cardinalidade no próprio elemento.

H7: Os elementos destacados como possíveis *features*, que não sejam sub-elementos de nenhum outro elemento que também seja *feature*, terão a cardinalidade pertencente a eles próprios e as *features* obtidas através deles estarão ligadas diretamente a *feature* raiz no modelo de *features*.

As heurísticas definidas nesta atividade (H3, H4, H5, H6 e H7) denominamos como heurísticas de domínio. Tendo como base o metamodelo da Figura 3.7 e a aplicação das heurísticas de domínio ao modelo SR gerado na atividade anterior, obtemos como saída desta atividade um modelo SR contendo cardinalidade.

Tabela 4.1 - Cardinalidade para os modelos i*

Notação	Conceito nos modelos de <i>features</i>	Em um produto específico
[0..1]	<i>Feature</i> opcional	Pode estar presente ou não
[1..1]	<i>Feature</i> obrigatória	Deve estar presente exatamente uma vez
[n..m]	<i>Feature</i> com cardinalidade	No mínimo n e no máximo m cópias da <i>feature</i> devem estar presentes
<1..1>	<i>Feature</i> agrupada alternativa <i>xor</i> (ou-exclusivo)	Aparece exatamente 1 alternativa do agrupamento de <i>features</i>
<1..k>	<i>Feature</i> agrupada alternativa <i>or</i> (ou-inclusivo)	Aparece pelo menos 1 e no máximo k alternativas do agrupamento de <i>features</i>
<j..k>	<i>Feature</i> agrupada com cardinalidade	Pode aparecer entre j e k alternativas do agrupamento de <i>features</i>

Aplicação no exemplo: Aplicando as heurísticas (H3, H4, H5, H6 e H7) e a linguagem **i*-c** (i* com cardinalidade apresentada no capítulo anterior) no modelo SR da Figura 4.12, obtemos o seguinte resultado:

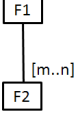
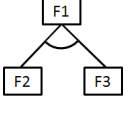
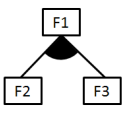
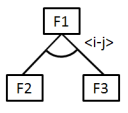
H3 – Os elementos destacados da Figura 4.12 (**Adicionar Foto**, **Armazenar Foto**, **Salvar Automaticamente**, **Salvar pelo Usuário**, **Fornecer Legenda**, **Nome**, **Local**, **Foto** e **Álbum**) serão reestruturados para conter cardinalidade.

H4 Os sub-elementos **Salvar Automaticamente** e **Salvar pelo Usuário** são agrupados e, portanto, a cardinalidade pertence ao relacionamento. Para este exemplo, definimos a cardinalidade <1..1>, indicando que as *features* daí obtidas pertencerão a uma alternativa *xor*.

H5 – Os elementos **Armazenar Foto** e **Fornecer Legenda** são sub-elementos de uma ligação de decomposição de tarefas, portanto são obrigatórios, indicando que a cardinalidade pertence aos próprios elementos e tem o valor [1..1].

H6 – As dependências (*dependum*) **Nome**, **Local**, **Foto** e **Álbum** têm a cardinalidade no próprio elemento. Definimos a cardinalidade de todos esses elementos como [1..1], indicando que são obrigatórias.

Tabela 4.2 – Mapeamento entre a cardinalidade nos modelos i* e nos modelos de *features*

Cardinalidade no modelo i*		Cardinalidade no modelo de <i>Features</i>	
Tipo	Valor	Notação	Explicação
Elemento	[0..1]		Uma <i>feature</i> opcional pode ou não estar presente em um produto concreto. A cardinalidade é [0..1].
Elemento	[1..1]		Uma <i>feature</i> obrigatória deve estar presente exatamente uma vez em um produto concreto. A cardinalidade é [1..1].
Elemento	[n..m]		Uma cardinalidade [n..m] indica que pelo menos n e no máximo m ocorrências de f2 devem estar presentes em um produto concreto.
Grupo	<1..1>		Uma cardinalidade de grupo <1..1> indica que exatamente uma <i>feature</i> pode ser selecionada do grupo.
Grupo	<1..k>		Uma cardinalidade de grupo <1..k> indica que pelo menos uma e no máximo k <i>features</i> podem ser selecionadas do grupo e que k é o número total de <i>features</i> no grupo.
Grupo	<j..k>		Uma cardinalidade de grupo é um intervalo inserido em um grupo de <i>features</i> que indica quantas <i>features</i> podem ser selecionadas do grupo em um produto concreto.

Como o nome já diz, modelos de *features* são compostos por *features* organizadas em um diagrama hierárquico como uma árvore, onde cada *feature* pode ser refinada em *sub-features*. Todo modelo de *features* tem uma raiz que geralmente representa um conceito (por exemplo, um sistema de software). Isso define a heurística H8.

H8: A *feature* raiz (em inglês, *root feature*) de um modelo de *features* será o sistema que está sendo modelado.

Para a elaboração do modelo de *features*, primeiramente precisamos nomear as *features* a partir do modelo SR. Na seção 2.1 definiu-se uma *feature* como sendo uma propriedade do sistema relevante para alguns *stakeholders* e usada para capturar características iguais e

diferentes entre produtos de uma mesma família. Pretende-se obter *features* que representem os conceitos mais simples, porém permitindo uma identificação o mais completa possível. Portanto, o modo de obtenção dos nomes das *features* vai ao encontro das propriedades que descrevem os elementos intencionais, em vez de se associar toda a descrição do elemento intencional com uma *feature* [António, S., 2009]. Uma maneira direta para obter o nome das *features* através dos elementos intencionais dos modelos i^* está descrita na heurística H9.

H9: Os nomes das *features* podem ser extraídas através das propriedades que descrevem os elementos intencionais. Se o elemento intencional for um recurso, o nome da *feature* será o mesmo nome do recurso. Se o elemento intencional for uma tarefa, o nome da *feature* poderá ser uma simplificação do nome da tarefa, caso seja necessário. Por exemplo, a tarefa “Realizar Pagamento com cartão de crédito” poderá ser mapeada para a *feature* “Realizar Pagamento”.

Depois de nomeada as *features*, criamos uma tabela para inserir algumas informações com o objetivo de facilitar a construção do modelo de *features*. A Tabela 4.3 é um exemplo da tabela a ser preenchida:

- i. A coluna “Elemento” deve ser preenchida com o nome do elemento no modelo i^* ;
- ii. A coluna “Cardinalidade” é subdividida em duas colunas: “Tipo” e “Valor”. O campo Tipo deve ser preenchido com os valores “elemento” ou “grupo” indicando se a cardinalidade é do elemento ou do relacionamento, respectivamente. O campo Valor deve ser preenchido com o valor da cardinalidade;
- iii. A coluna “Elemento Pai” deve ser preenchida com o nome do elemento que tem o elemento em questão como sub-elemento. Porém, (i) caso o “pai” do elemento em questão não esteja destacado no modelo SR como possível *feature* (por exemplo, se o “pai” for um objetivo), então se deve escolher o elemento seguinte na hierarquia como “elemento pai”; (ii) caso o elemento em questão não seja sub-elemento, ou seja, esteja no topo da árvore, então este campo será preenchido com o valor “-”. Essa informação será útil para a hierarquia do modelo de *features*;
- iv. A coluna “Feature” deve ser preenchida com o nome da *feature* definido na heurística H9.

Tabela 4.3 – Template de tabela a ser preenchida com informações para o modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		

Essa tabela deve ser preenchida com as informações de todos os elementos que foram destacados como possíveis *features*.

H10: Todas as linhas da tabela serão mapeadas para o modelo de *features* com o nome preenchido na coluna “Feature”. Caso duas ou mais *features* tenham o mesmo nome, as informações de todas elas serão unidas para se tornarem apenas uma *feature*.

Primeiramente, vão ser analisadas as linhas que não tenham a coluna “Elemento Pai” preenchida, pois, esse tipo de informação significa que o elemento da linha em questão não é sub-elemento.

H11: Todas as *features* da tabela serão relacionadas à *feature* raiz quando não tiverem o campo “Elemento Pai” preenchido.

Na sequência, devem ser avaliadas as linhas da tabela que tenham o campo “Elemento Pai” preenchido com o nome do elemento identificado na heurística anterior. Isso para facilitar a construção do modelo de *features*, respeitando a sua hierarquia. Repete-se este passo até esgotar a análise de todas as linhas da tabela.

H12: Todas as *features* da tabela serão *sub-features* quando tiverem a coluna “Elemento Pai” preenchida. Essas *sub-features* serão relacionadas à *feature* correspondente ao valor do campo “Elemento Pai”.

Os relacionamentos entre as *features* no modelo de *features* são obtidos a partir da coluna “Cardinalidade” da tabela avaliada. Essa coluna é subdividida em duas colunas: “Tipo” e “Valor”. Se o campo Tipo tiver preenchido com o valor “elemento”, significa que a cardinalidade é do elemento. Se o campo Tipo tiver preenchido com o valor “grupo”, significa que a cardinalidade é do relacionamento. O campo Valor contém o valor da cardinalidade.

Aplicação no exemplo: Utilizando os artefatos de entrada desta atividade, obtemos o seguinte resultado:

H8 A *feature* raiz (em inglês, *root feature*) de um modelo de *features* será o sistema que está sendo modelado, ou seja, **Mobile Media**.

H9 - Os nomes dos elementos do tipo recurso **Foto, Local, Nome e Album** serão os mesmos no modelo de *features*. Os nomes dos elementos do tipo tarefa **Adicionar Foto, Armazenar Foto, Fornecer Legenda, Salvar Pelo Usuário e Salvar Automaticamente** também terão os mesmos nomes no modelo de *features*.

Geramos a Tabela 4.4 de acordo com o template da Tabela 4.3 para facilitar a construção do modelo de *features*.

Tabela 4.4 – Informações dos elementos do caso de uso Adicionar Foto que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Adicionar Foto	Elemento	[1..1]	–	Adicionar Foto
Armazenar Foto	Elemento	[1..1]	Adicionar Foto	Armazenar Foto
Fornecer Legenda	Elemento	[1..1]	Adicionar Foto	Fornecer Legenda
Salvar Pelo Usuário	Grupo	<1..1>	Armazenar Foto	Salvar Pelo Usuário
Salvar Automaticamente	Grupo	<1..1>	Armazenar Foto	Salvar Automaticamente
Foto	Elemento	[1..1]	Armazenar Foto	Foto
Local	Elemento	[1..1]	Armazenar Foto	Local
Nome	Elemento	[1..1]	Fornecer Legenda	Nome
Album	Elemento	[1..1]	Salvar Pelo Usuário	Album

H10 As linhas da tabela 4.4 serão mapeadas para o modelo de *features* com o nome preenchido na coluna “*Feature*”, ou seja, **Adicionar Foto, Armazenar Foto, Fornecer Legenda, Salvar Pelo Usuário, Salvar Automaticamente, Foto, Nome, Local e Album** serão *features*.

H11 De acordo com a Tabela 4.4, a *feature* **Adicionar Foto** será relacionada à *feature* raiz (**Mobile Media**), pois, não tem o campo “Elemento Pai” preenchido. A Figura 4.14 mostra essa representação.



Figura 4.14 – Modelo de *Features* com as *features* **Mobile Media** e **Adicionar Foto**.

H12 - 1 Analisando as linhas da Tabela 4.4 que tenham o campo “Elemento Pai” preenchido com o nome do elemento identificado na heurística anterior (**Adicionar Foto**), encontramos as *features* **Armazenar Foto** e **Fornecer Legenda** que são mapeadas para o modelo de *features* de acordo com a Figura 4.15.

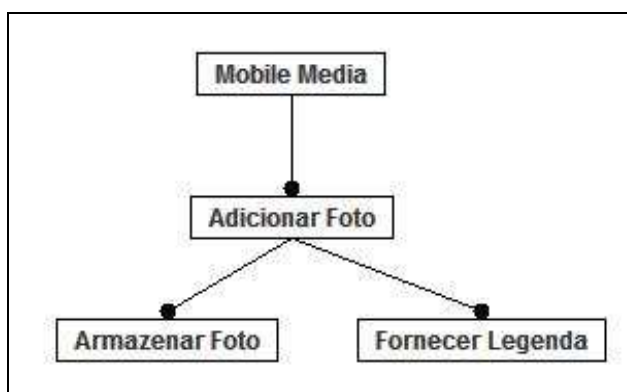


Figura 4.15 – Modelo de *Features* com as *sub-features* **Armazenar Foto** e **Fornecer Legenda**

H12 – 2 Continuamos a análise das linhas da Tabela 4.4 que tenham o campo “Elemento Pai” preenchidos com o nome **Armazenar Foto** e **Fornecer Legenda**. Para a *feature* **Armazenar Foto**, encontramos as *features* **Salvar Automaticamente**, **Salvar Pelo Usuário**, **Foto** e **Local**; para a *feature* **Fornecer Legenda**, encontramos a *feature* **Nome**. Todas essas *features* são mapeadas para o modelo de *features* de acordo com a Figura 4.16.

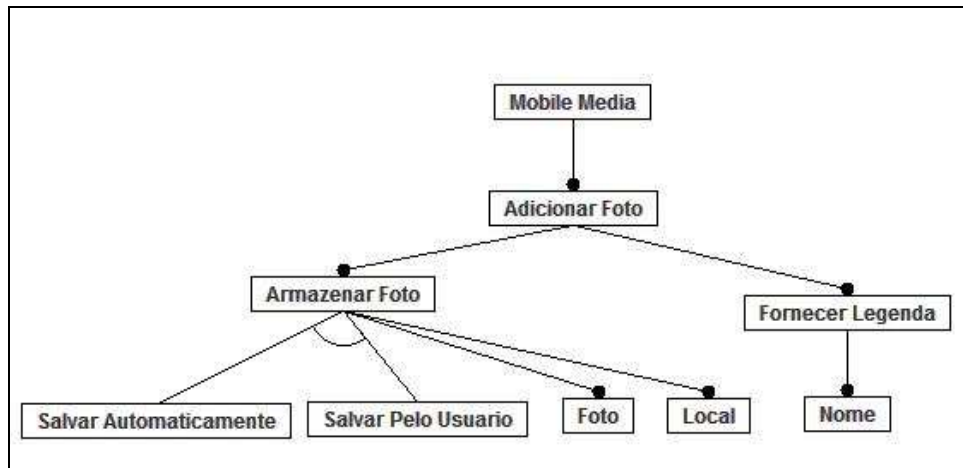


Figura 4.16 – Modelo de *Features* com as *sub-features* Salvar Automaticamente, Salvar Pelo Usuário, Foto, Local e Nome

H12 – 3 A última linha que resta ser analisada da Tabela 4.4 é a que tem o campo “Elemento Pai” preenchido com o nome **Salvar Pelo Usuário** e o campo “Feature” preenchido com o valor **Album**. Mapeamos esta última *feature* para o modelo de *features* e obtemos o artefato de saída desta atividade, representado na Figura 4.17.

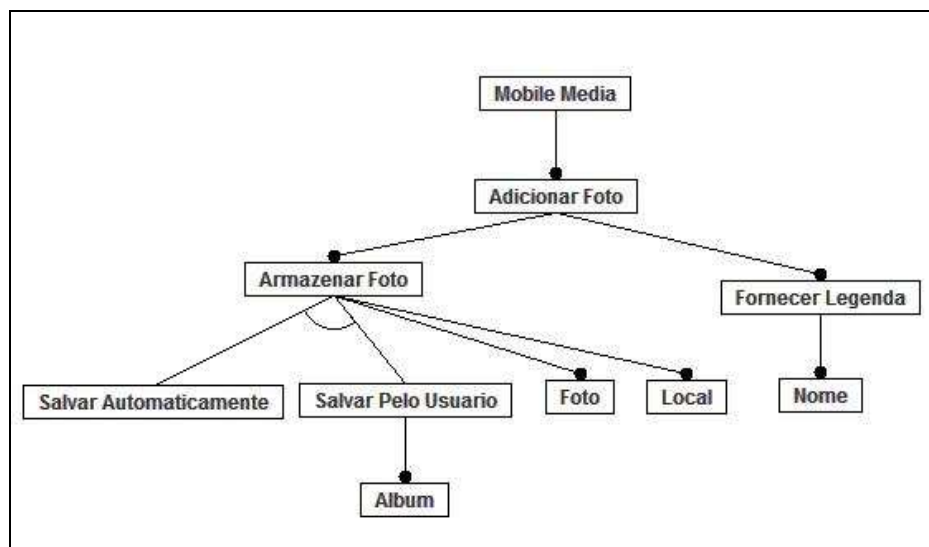


Figura 4.17 – Modelo de *features* para o caso de uso Adicionar Foto

4.6. Reorganização do Modelo de *Features* (opcional)

Esta atividade do processo é a reorganização do modelo de *features* obtido na atividade anterior. Esta atividade é considerada opcional, pois só será executada caso haja a necessidade de uma reorganização do modelo de *features*, porém, poderá ser executada tantas vezes enquanto o engenheiro do domínio achar necessário. O custo desta atividade vai depender de como o modelo SR foi construído.

Existem diversas situações onde uma reorganização faz-se necessária: *sub-feature* com mais de um pai, *features* repetidas, *features* colocadas em lugar inapropriado, *features* diferentes com mesma semântica, *features* que podem ser agrupadas por fazerem referência ao mesmo conceito, etc. Dependendo da situação, essas *features* podem ser removidas do modelo ou realocadas ou agrupadas. Por exemplo: (i) se uma *sub-feature* estiver relacionada a mais de uma *feature*, provavelmente esta *sub-feature* será realocada; (ii) se existirem *features* repetidas no modelo com a mesma semântica, as repetições serão eliminadas; (iii) se existirem diferentes *features* com mesma semântica, essas *features* serão agrupadas em uma só. Não definimos heurísticas para esta atividade, ficando a critério da experiência do engenheiro de domínio a identificação desses “*bad smells*”.

O artefato de saída desta atividade é um modelo de *features* reorganizado.

Aplicação no exemplo: Analisando o modelo de *features* da Figura 4.17, verificamos que todas as *features* do modelo se referem ao mesmo conceito, **Foto**, sendo redundante a presença desta feature. Dessa forma, removemos do modelo a feature **Foto**. A Figura 4.18 apresenta o artefato de saída desta atividade, um modelo de *features* reorganizado.

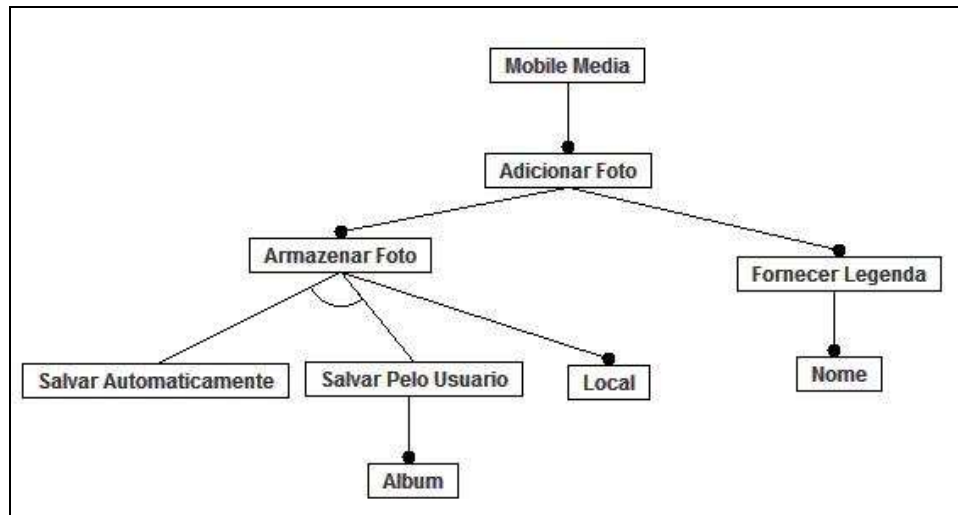


Figura 4.18 – Modelo de Features reorganizado do caso de uso Adicionar Foto

4.7. Configuração do produto no modelo SR

Tendo como entrada algumas heurísticas e um modelo SR com cardinalidade gerado na atividade de Reengenharia do Modelo SR, esta atividade tem como objetivo configurar os elementos candidatos a *features* que farão parte de um produto específico. É a partir desta atividade que nossa abordagem começa a dar suporte à fase da Engenharia de Aplicação, já que esta é a fase de desenvolvimento do produto. Esta atividade se repetirá para cada produto a ser configurado na linha de produtos.

Os modelos i* permitem expressar os interesses dos *stakeholders* e as características do sistema pretendido usando elementos intencionais, que ilustram as motivações envolvidas e as possíveis alternativas a serem consideradas em um processo de negócio. Essa informação adicional, encontrada nos modelos i*, complementa o modelo de *features*, ajudando na escolha e na justificativa de uma configuração para uma determinada aplicação. Por exemplo, se **Rapidez [Armazenamento]** é um *softgoal* do modelo SR que tem uma alta prioridade na aplicação, provavelmente, o *stakeholder* irá selecionar os elementos candidatos a *features* que melhor satisfazem este *softgoal*, como, por exemplo, a tarefa **Salvar Automaticamente**.

Nesta fase, o modelo de *features* já está modelado e organizado. Os *stakeholders* devem indicar quais desses elementos farão parte do produto. Apesar de esta indicação ser subjetiva, os *softgoals* dos modelos i* podem ajudar na escolha de uma configuração específica para que a aplicação desenvolvida alcance os objetivos da organização. Algumas heurísticas

foram definidas quanto à hierarquia dos elementos selecionados e a obrigatoriedade da presença deles na aplicação.

H13: Todos os elementos intencionais que possuírem cardinalidade [1..1] ou [n..m], onde n seja maior ou igual a 1, deverão estar presentes no modelo de configuração do produto, desde que seus elementos pais também tenham sido selecionados. Os elementos com cardinalidade [0..1] poderão ou não ser selecionados, ficando à critério do stakeholder.

H14: Os sub-elementos relacionados com as ligações meio-fim que possuem cardinalidade deverão estar presentes no modelo de *features* de acordo com a cardinalidade específica: <1..1>, <1..k> ou <j..k>, desde que j seja maior ou igual a 1.

Vale salientar que o modelo de *features* poderá ser usado para configurar vários produtos diferentes, dependendo apenas das preferências dos *stakeholders* com relação aos requisitos não-funcionais.

Aplicação no exemplo: Aplicando as heurísticas (H13 e H14) no modelo SR da Figura 4.13, produzido na atividade de Reengenharia do Modelo SR, obtemos o seguinte resultado:

H13 – Os elementos **Adicionar Foto**, **Armazenar Foto**, **Fornecer Legenda**, **Foto**, **Local** e **Nome** foram selecionados para estarem presentes no modelo de configuração do produto, já que possuem cardinalidade [1..1]. Resolvemos que o elemento **Album**, apesar de também ter a cardinalidade [1..1], não estará presente no modelo de *features*, pois seu elemento pai (a tarefa **Salvar pelo Usuário**) não foi selecionado.

H14 – Neste exemplo, definimos a cardinalidade do agrupamento **Salvar Automaticamente e Salvar pelo Usuário** como <1..1>, indicando a foto ou será salva pelo sistema ou pelo usuário (as *features* daí obtidas pertencerão a uma alternativa *xor*). Nesse caso, como apenas uma alternativa deve ser selecionada por grupo, escolhemos **Salvar Automaticamente**, devido a essa tarefa ter uma influência positiva em relação ao *softgoal* **Rapidez [Armazenamento]**.

A Figura 4.19 apresenta o artefato gerado nesta atividade, um modelo SR com os elementos que farão parte do produto destacados.

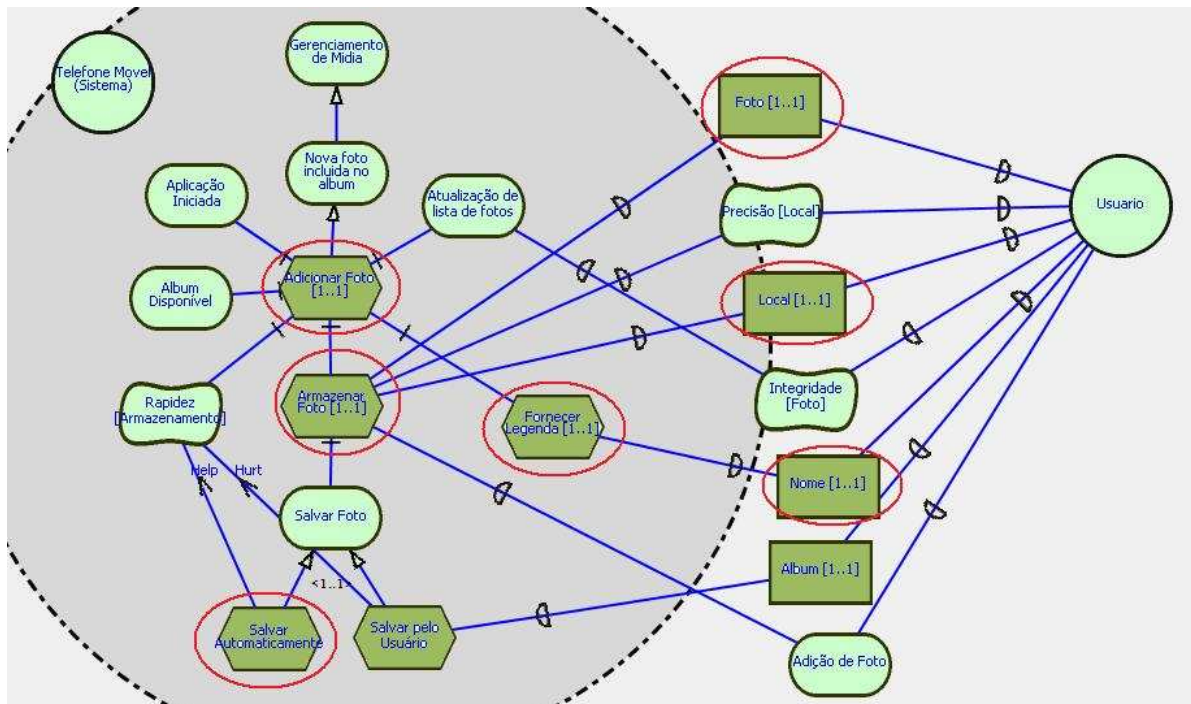


Figura 4.19 – Modelo SR do caso de uso Adicionar Foto com elementos que estarão presentes no modelo de configuração do produto

4.8. Elaboração do Modelo de Configuração do Produto

O objetivo desta atividade é construir o modelo de configuração de um produto específico, a partir do modelo SR que contém os elementos que farão parte do produto destacados (gerado na atividade anterior) e do modelo de *features* reorganizado (caso a atividade Reorganização do Modelo de *Features* tenha sido executada). Esta atividade se repetirá para cada produto a ser configurado na linha de produtos.

De acordo com [Gheyi, R. et al., 2008], uma configuração contém um conjunto de nomes de *features* (seleção de *features*), que é válida caso todas as relações e fórmulas do modelo sejam satisfeitas. Dessa forma, depois de definidas as *features* que estarão presentes no modelo de configuração do produto, comparamos essas *features* com o modelo de *features* gerado anteriormente, com o objetivo de validar se as *features* da configuração satisfazem as relações do modelo de *features*. Por fim, geramos um conjunto de nomes referentes às *features* selecionadas na atividade anterior.

Aplicação no exemplo: Analisando o modelo SR da Figura 4.19, observamos que os elementos **Adicionar Foto**, **Armazenar Foto**, **Fornecer Legenda**, **Salvar Automaticamente**

te, **Foto**, **Local** e **Nome** foram destacados para fazerem parte do produto. Porém, comparando esses elementos com o modelo de *features* da Figura 4.18, verificamos que a *feature* **Foto** não existe no modelo de *features* devido à sua reorganização feita anteriormente. Dessa forma, essa *feature* apesar de ter sido selecionada, não estará presente no modelo de configuração do produto. Geramos o seguinte modelo de configuração válido (conjunto de nomes) para o caso de uso **Adicionar Foto**:

Modelo Configuração: {Mobile Media, Adicionar Foto, Armazenar Foto, Fornecer Legenda, Salvar Automaticamente, Local e Nome}.

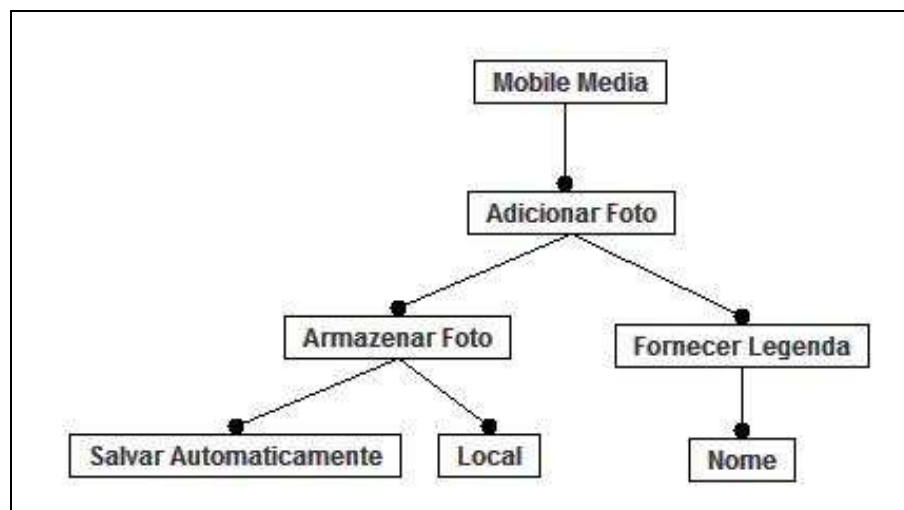


Figura 4.20 – Modelo de Configuração do caso de uso Adicionar Foto

4.9. Resumo das Heurísticas

Esta seção apresenta uma tabela (Tabela 4.5) com o resumo de todas as heurísticas apresentadas ao longo deste capítulo.

Tabela 4.5 – Resumo das heurísticas

Heurística	Definição
H1	As <i>features</i> podem ser extraídas a partir dos elementos do tipo tarefa e recurso.
H2	As <i>features</i> podem ser extraídas a partir dos elementos internos do ator que representa o sistema e das dependências (particularmente do <i>dependum</i>) ligadas diretamente a esse ator.
H3	Todos os elementos do modelo SR que foram destacados como possíveis <i>features</i> serão reestruturados para conter cardinalidade.
H4	Se a quantidade de sub-elementos de uma ligação meio-fim que estão destacados como possíveis <i>features</i> , for maior do que 1, então esses sub-elementos serão agrupados e a cardinalidade pertencerá ao relacionamento. Se a quantidade for exatamente 1, então a cardinalidade pertencerá ao sub-elemento e não ao relacionamento.
H5	Os sub-elementos de uma ligação de decomposição de tarefas que estão destacados como possíveis <i>features</i> , serão modelados como <i>features</i> obrigatórias, significando que a cardinalidade pertencerá aos sub-elementos e terá a cardinalidade [1..1].
H6	As dependências (<i>dependum</i>), que estão destacadas como possíveis <i>features</i> , terão a cardinalidade no próprio elemento.
H7	Os elementos destacados como possíveis <i>features</i> , que não sejam sub-elementos de nenhum outro elemento que também seja feature, terão a cardinalidade pertencente a eles próprios e as <i>features</i> obtidas através deles estarão ligadas diretamente a <i>feature</i> raiz no modelo de <i>features</i> .
H8	A <i>feature</i> raiz de um modelo de <i>features</i> será o sistema que está sendo modelado.
H9	Os nomes das <i>features</i> podem ser extraídas através das propriedades que descrevem os elementos intencionais. Se o elemento intencional for um recurso, o nome

	da <i>feature</i> será o mesmo nome do recurso. Se o elemento intencional for uma tarefa, o nome da <i>feature</i> poderá ser uma simplificação do nome da tarefa, caso seja necessário.
H10	Todas as linhas da tabela serão mapeadas para o modelo de <i>features</i> com o nome preenchido na coluna “ <i>Feature</i> ”. Caso duas ou mais <i>features</i> tenham o mesmo nome, as informações de todas elas serão unidas para se tornarem apenas uma <i>feature</i> .
H11	Todas as <i>features</i> da tabela serão relacionadas à <i>feature</i> raiz quando não tiverem o campo “Elemento Pai” preenchido.
H12	Todas as <i>features</i> da tabela serão <i>sub-features</i> quando tiverem a coluna “Elemento Pai” preenchida. Essas <i>sub-features</i> serão relacionadas à <i>feature</i> correspondente ao valor do campo “Elemento Pai”.
H13	Todos os elementos intencionais que possuem cardinalidade [1..1] ou [n..m], onde n seja maior ou igual a 1, deverão estar presentes no modelo de configuração do produto, desde que seus elementos pais também tenham sido selecionados. Os elementos com cardinalidade [0..1] poderão ou não ser selecionados, ficando à critério do <i>stakeholder</i> .
H14	Os sub-elementos relacionados com as ligações meio-fim que possuem cardinalidade deverão estar presentes no modelo de <i>features</i> de acordo com a cardinalidade específica.

4.10.Discussão

Como já vimos anteriormente, algumas abordagens orientadas a objetivos para requisitos em LPS têm sido propostas para capturar as semânticas de modelos de *features*. Na seção 3.1, já apresentamos algumas dessas abordagens: modelos de objetivos [Yu, Y. et al., 2008], modelo PL-AOVGraph [Batista, T. et al., 2008], modelo i* aspectual [Silva, C. et al., 2008] e modelo i* [Yu, E., 1995]. Na seção 3.2, apresentamos um modelo i* com cardinalidade cons-

truído a partir do modelo de *features* apenas para ilustrar a notação da linguagem **i*-c**, mas os modelos **i*** são úteis para descobrir *features* e não o contrário.

Outra abordagem baseada em modelos orientados a objetivos, mais especificamente em modelos **i***, é IStarLPS [Antônio, S., 2009]. Esta abordagem, similarmente à **G2FM**, adapta o *framework* **i*** para Linhas de Produtos de Software, porém com significantes diferenças:

- i. Tanto os modelos SD quanto os modelos SR são considerados na abordagem IStarLPS, enquanto que consideramos apenas o modelo SR na abordagem **G2FM**. Isso porque é o modelo SR que possui informações detalhadas sobre como o ator alcança as dependências estratégicas, o que inclui a modelagem destas dependências. Diferentemente da abordagem IStarLPS, a abordagem **G2FM** utiliza as informações do modelo SR para guiar a configuração de uma aplicação específica em uma LPS, além de utilizá-lo para identificar as *features* da LPS;
- ii. Todos os elementos intencionais dos modelos SD e SR da abordagem IStarLPS são mapeados para *features*, enquanto que a abordagem **G2FM** mapeia apenas para *features* os elementos do tipo tarefa e recurso (vide heurística **H1**);
- iii. Em IStarLPS, a cardinalidade é incluída em todos os elementos intencionais, enquanto que em **G2FM**, acrescentamos a cardinalidade apenas nos elementos que poderão ser *features* (tarefas e recursos) e nos relacionamentos meio-fim (vide heurística **H4**). Não faz sentido existir cardinalidade no contexto de elementos intencionais como objetivos já que este tipo de elemento é uma condição a ser alcançada (o que significa alcançar uma condição x vezes?);
- iv. A abordagem **G2FM** mantém o modelo **i*** com suas informações originais, ajudando na escolha e justificativa de uma configuração para uma determinada aplicação, através dos relacionamentos de contribuição em relação à satisfação dos *softgoals*. Em IStarLPS, além dos elementos do tipo *softgoal* não serem considerados na abordagem, todos os outros elementos intencionais do modelo **i*** são usados para gerar o modelo de *features* e não há nenhum tipo de justificativa na escolha de uma determinada configuração para uma aplicação específica.

4.11. Considerações Finais

Neste capítulo, apresentamos todas as fases do processo que descreve nossa abordagem, **G2FM**, mostrando detalhadamente as heurísticas definidas e os artefatos de saída. Utilizamos como exemplo o caso de uso **Adicionar Foto** do sistema **MobileMedia**, aplicado juntamente com a explicação de cada fase do processo.

Além da abordagem **G2FM** suportar cardinalidade do modelo de *features*, outra vantagem de nossa abordagem é que mantemos o modelo *i** com suas informações originais, ajudando na escolha e justificativa de uma configuração para uma determinada aplicação.

Apresentamos também uma comparação entre a abordagem **G2FM** e outra abordagem baseada em modelos *i** para Linhas de Produtos de Software (ISStarLPS), ressaltando as suas principais diferenças.

O capítulo seguinte exibirá um estudo de caso completo do sistema **MobileMedia**.

Capítulo

5. Estudo de Caso: Mobile Media

Este capítulo apresenta a utilização da abordagem **G2FM** (apresentada no capítulo anterior) com um sistema de Linha de Produtos de Software, o **Mobile Media**.

Neste capítulo apresentamos a utilização da abordagem **G2FM** (apresentada no Capítulo anterior) no contexto de uma linha de produtos de software denominado de **MobileMedia** [Figueiredo, E. et al., 2008]. O principal objetivo desse sistema é manipular fotos, música e vídeo em dispositivos móveis como, por exemplo, telefones celulares. A Figura 5.1 apresenta a tela da funcionalidade de adicionar foto. Para desenvolver o **MobileMedia**, os desenvolvedores estenderam a implementação base de uma LPS anterior chamada MobilePhoto [Young, T., 2005] incluindo novas *features* obrigatórias, opcionais e alternativas [Figueiredo, E. et al., 2008].



Figura 5.1 – Tela do sistema MobileMedia da funcionalidade de adicionar foto

Este estudo de caso tem o propósito de: (i) mostrar como a abordagem **G2FM** pode ser usada para extrair *features* de um modelo *i**; e (ii) avaliar se nossa proposta contribuirá para a escolha e justificativa da configuração de um produto específico.

Escolhemos esse sistema para realização do nosso estudo de caso por diversas razões: (i) é uma LPS que vem sendo utilizada em diversos estudos de caso [Procad, 2009]; (ii) é representativo para o domínio de dispositivos móveis, pois possui diversos pontos variáveis

relacionados a plataformas heterogêneas; e (iii) apresenta muitas *features* alternativas e opcionais [Figueiredo, E. et al., 2008].

5.1. Construção do modelo SR (opcional)

A primeira atividade do nosso processo é a construção do modelo SR do *framework* *i**, destacada na Figura 5.2. Vimos anteriormente que esta atividade é opcional, desde que já exista um modelo SR para o sistema a ser modelado. Como o sistema **MobileMedia** não possui este tipo de modelo, então faz-se necessária a execução desta atividade. O documento de casos de uso do sistema é o artefato consumido por esta atividade que, para nosso estudo de caso, é a *release 1* do documento de casos de uso do **MobileMedia**.

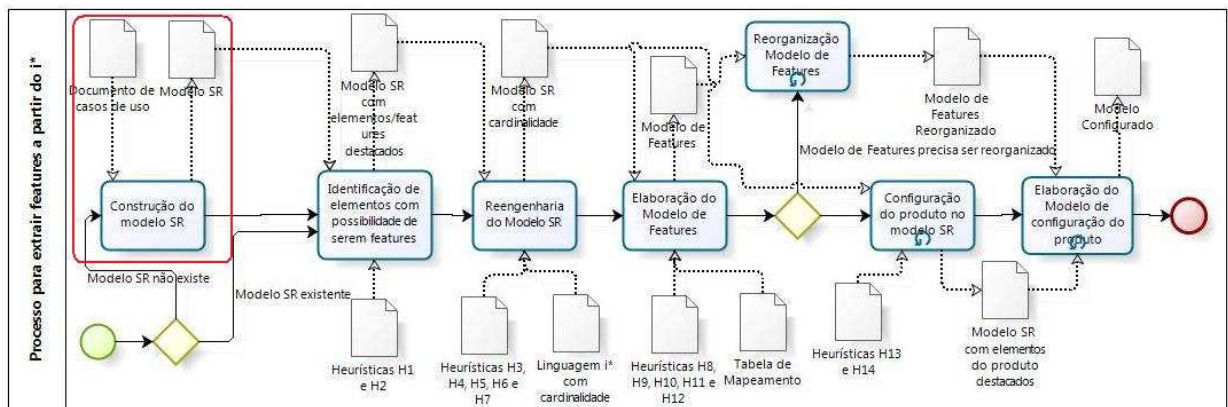


Figura 5.2 – Primeira atividade do processo G2FM: Construção do modelo SR

O documento de casos de uso do **MobileMedia** é composto por oito *releases*, disponibilizadas de forma incremental, ou seja, a cada nova *release* são adicionadas novas funcionalidades. Optamos por aplicar o nosso processo na *release 1*, que contem os seguintes casos de uso: **Adicionar Foto**, **Apagar Foto**, **Visualizar Conteúdo**, **Adicionar Álbum**, **Apagar Álbum** e **Fornecer Legenda**. Os passos definidos para esta atividade são aplicados a cada caso de uso.

5.1.1. Caso de Uso: Adicionar Foto

O modelo SR deste caso de uso já foi modelado no capítulo anterior, durante a apresentação da abordagem **G2FM**, visualizado na Figura 4.11.

5.1.2. Caso de Uso: Apagar Foto

A especificação original do caso de uso **Apagar Foto** encontra-se na Figura 5.3 abaixo.

Usecase: Delete Photo
Actor: Mobile Phone (system) and User
Description: The user can delete image from an album in the device
Precondition: Application must be launched, An image must exist in an album.
Trigger: User chooses to delete image
Scenario: 1. (select) The user selects an album from which images or audio file has to be removed 2. (listing) The device populates the list of available image in an album 3. (delete) The user selects the option to remove the image. The image is removed from the device memory 4. (update) The list is updated
Extensions: 1a. (null data) There are no images or audio files available on the device memory 3a. (delete) Problem occurs with deleting image from album and/or device memory 4a. (update) Problem with updating the modified information

Figura 5.3 – Especificação original do caso de uso Apagar Foto

A seguir, encontram-se os passos da primeira atividade do processo, aplicados ao caso de uso Apagar Foto. O modelo SR do caso de uso **Apagar Foto** é apresentado na Figura 5.4.

Passo 1.1 - Identificação e modelagem dos atores e objetivos principais: de acordo com a especificação do caso de uso **Apagar Foto**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente.

Passo 1.2 – Construção do modelo i* operacional:

- **Regra 1 – Modelagem de atividades:** de acordo com esta regra, o caso de uso (ou atividade) **Apagar Foto** é modelado como uma tarefa associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim.
- **Regra 2 – Modelagem de tarefas:** os passos do cenário principal do caso de uso em questão são mapeados em sub-tarefas e relacionados à tarefa **Apagar Foto** através do relacionamento de decomposição de tarefas. As tarefas iniciadas no ator **Te-**

Telefone Móvel (sistema) seriam: **Listar Fotos disponíveis**, **Remover Foto** e **Atualizar lista de fotos**, porém, da mesma forma que o caso de uso **Adicionar Foto**, não vemos sentido em modelar **Atualizar lista de fotos** como tarefa, pois, entendemos como uma pós-condição do caso de uso. Assim, o ator **Telefone Móvel (sistema)** tem as tarefas **Listar Fotos disponíveis** e **Remover Foto**. As tarefas iniciadas no ator **Usuário** são: **Selecionar Álbum**, **Selecionar opção de apagar fotos** e **Selecionar Foto a ser removida**. Esta última tarefa não aparece explicitamente nos passos do caso de uso, mas entendemos que ela é necessária para a realização do caso de uso, justificando a modelagem desta tarefa. Mais adiante, na Regra 5, com a modelagem das tarefas alternativas, a tarefa **Remover Foto** sofrerá modificação.

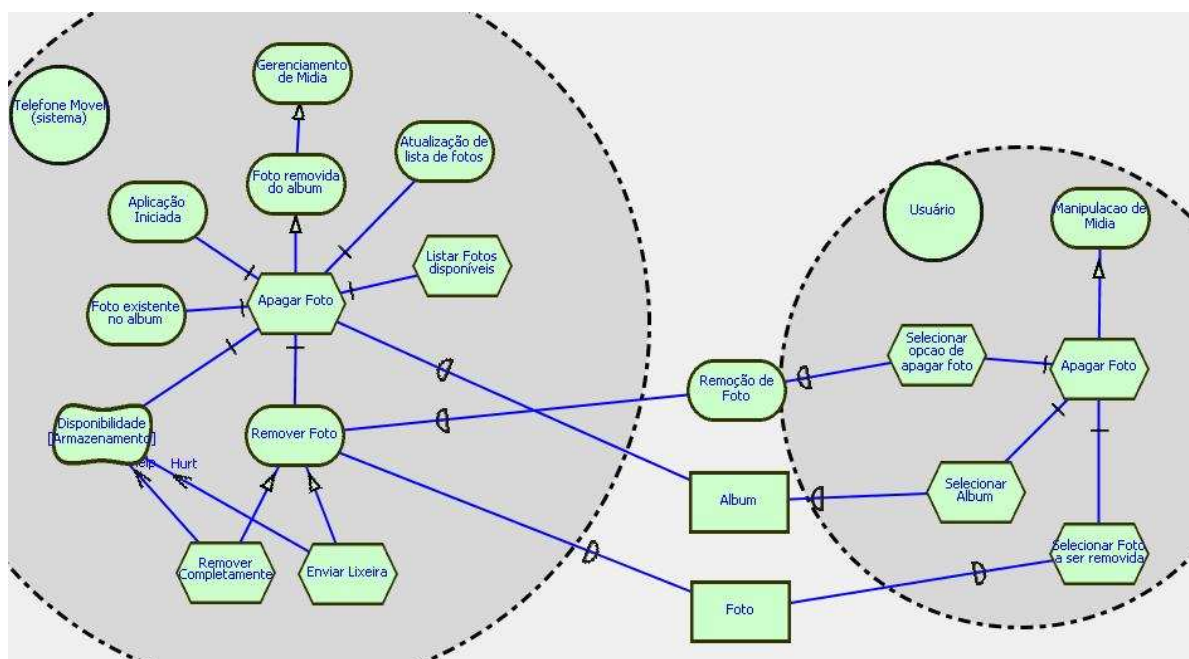


Figura 5.4 – Modelo SR do caso de uso Apagar Foto

- **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** a tarefa **Selecionar foto a ser removida** (*dependee*) fornece o recurso **Foto** (*dependum*) que vai ser consumido pela tarefa **Remover Foto** (*dependor*).
- **Regra 4 – Modelagem de tarefas que consomem recursos:** a tarefa **Selecionar Álbum** (*dependor*) depende que o recurso **Álbum** (*dependum*) esteja disponível para que essa tarefa seja realizada. Como este recurso não foi produzido pela regra anterior, então, será produzido pela tarefa **Apagar Foto** (*dependee*).

- **Regra 5 – Modelagem de tarefas alternativas:** apesar da especificação original do caso de uso **Apagar Foto** (vide Figura 5.3) conter apenas cenários de exceção, sentimos a necessidade de adicionar um cenário alternativo ao passo 3 do cenário principal: “3. *(delete)* The user chooses to send the image to the trash.”. A modelagem deste cenário alternativo fica da seguinte forma: a tarefa **Remover Foto** torna-se um objetivo que pode ser alcançado pelas seguintes tarefas alternativas: **Remover Completamente** e **Enviar Lixeira**.
- **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de trigger:** as pré-condições da tarefa **Apagar Foto** são **Aplicação Iniciada** e **Foto existente no álbum**. De acordo com esta regra, essas pré-condições são modeladas como objetivos e relacionados através do relacionamento de decomposição de tarefas, no ator que representa o sistema. A especificação original do caso de uso **Apagar Foto** não contém pós-condições, porém, como vimos anteriormente na Regra 2, supomos e assim, modelamos **Atualização de lista de fotos** como uma pós-condição do caso de uso. Da mesma forma que as pré-condições, essa pós-condição é modelada, no ator que representa o sistema, como um objetivo relacionado, através do relacionamento de decomposição de tarefas, com a tarefa que representa o caso de uso. O evento de *trigger* da tarefa **Apagar Foto** é **Remoção de Foto**. Esta *trigger* é um *dependum* do tipo objetivo, onde o *dependee* é a tarefa **Remover Foto** e o *dependor* é a tarefa **Selecionar opção de apagar foto**.
- **Regra 7 - Modelagem dos relacionamentos do tipo *includes* e *extends*:** esta regra não é aplicada neste caso de uso, pois, o mesmo não possui relacionamentos do tipo *includes* ou *extends*.

Passo 1.3 – Construção do modelo i* intencional:

- **Diretriz 1 – Análise dos elementos intencionais:**
 - Qual o estado final alcançado através da execução da tarefa “Apagar Foto”?
A resposta pode ser: “Foto removida do álbum selecionado”.
 - Que ator precisa atingir este objetivo (*dependor*)?
Resposta: o Telefone Móvel (sistema)
 - De que ator depende a obtenção do objetivo (*dependee*)?

Resposta: depende do usuário

Este objetivo (**Foto removida do álbum**) será inserido entre a tarefa e o objetivo principal do ator *depende*, ou seja, entre a tarefa **Apagar Foto** e o objetivo **Gerenciamento de Mídia** do ator **Telefone Móvel (sistema)**. Dessa forma, **Apagar Foto** se tornará o meio para obter o objetivo **Foto removida do álbum**, que por sua vez será um meio para alcançar o objetivo principal, **Gerenciamento de Mídia**.

- **Diretriz 2 – Análise de atributos de qualidade:** caso a foto seja removida completamente do celular, o sistema ganhará espaço de armazenamento. Assim, a tarefa **Remover Completamente** contribui positivamente para satisfazer o *softgoal* **Disponibilidade[Armazenamento]**, enquanto que a tarefa **Enviar Lixeira** contribui negativamente para a satisfação do mesmo *softgoal*.

5.1.3. Caso de Uso: Visualizar Conteúdo

A Figura 5.5 apresenta a especificação original do caso de uso **Visualizar Conteúdo**.

Usecase: View Content
Actor: Mobile Phone (system) and User
Description: The user can view a image or album content on the device memory
Precondition: Application must be launched A image must exist on an album
Trigger: User chooses to view an album or image in an album
Scenario: 1. (select)The user selects an album 2. (listing) The device populates the list of available images in an album 3. (view) User views the content of the album
Extensions: 1a. (null data) There are no images available on the device memory 3b. (view) View an image: 3a1. User selects an image from the list of images in an album. 3a2. Image is displayed on the screen 3c. (view) Problem occurs with retrieving 3c1. Problem in retrieving albums on the device 3c2. Problem in retrieving the images in an album

Figura 5.5 – Especificação original do caso de uso Visualizar Conteúdo

A seguir, encontram-se os passos da primeira atividade do processo, aplicados ao caso de uso Visualizar Conteúdo. A Figura 5.6 apresenta o modelo SR do caso de uso **Visualizar Conteúdo**.

Passo 1.1 - Identificação e modelagem dos atores e objetivos principais: de acordo com a especificação do caso de uso **Visualizar Conteúdo**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente.

Passo 1.2 – Construção do modelo i* operacional:

- **Regra 1 – Modelagem de atividades:** de acordo com esta regra, o caso de uso (ou atividade) **Visualizar Conteúdo** é modelado como tarefa e associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim.
- **Regra 2 – Modelagem de tarefas:** os passos do cenário principal do caso de uso em questão são mapeados em sub-tarefas e relacionados à tarefa **Visualizar Conteúdo** através do relacionamento de decomposição de tarefas. A tarefa iniciada no ator **Telefone Móvel (sistema)** é: **Listar Fotos disponíveis** e **Disponibilizar Foto**. Esta última tarefa não aparece explicitamente nos passos do caso de uso, mas foi modelada, pois achamos necessária para a realização do caso de uso. As tarefas iniciadas no ator **Usuário** são: **Selecionar Álbum** e **Visualizar Foto(s)**.
- **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** a tarefa **Disponibilizar Foto** (*dependee*) fornece o recurso **Foto** (*dependum*) que vai ser consumida pela tarefa **Visualizar Foto(s)** (*dependor*). Mais adiante, na Regra 5, com a modelagem das tarefas alternativas, essa dependência sofrerá modificação.
- **Regra 4 – Modelagem de tarefas que consomem recursos:** a tarefa **Selecionar Álbum** (*dependor*) depende que o recurso **Álbum** (*dependum*) esteja disponível para que essa tarefa seja realizada. Como este recurso não foi produzido pela regra anterior, então, será produzido pela tarefa **Visualizar Conteúdo** (*dependee*).
- **Regra 5 – Modelagem de tarefas alternativas:** a especificação original do caso de uso **Visualizar Conteúdo** (vide Figura 5.5) contém o seguinte cenário alternativo: “3a1. User selects an image from the list of images in an album” e “3a2. Image is displayed on the screen”. A modelagem destes cenários alternativos fica da seguinte

te forma: a tarefa **Visualizar Foto(s)** do ator **Usuário** tem uma ligação de decomposição com o objetivo **Selecionar Foto**. Já a tarefa **Disponibilizar Foto** do ator **Telefone Móvel** tem uma ligação de decomposição com o objetivo **Foto disponibilizada na tela**. O primeiro objetivo (**Selecionar Foto**) tem as seguintes tarefas alternativas: **Selecionar pelo nome** e **Selecionar sequencialmente**. Nesse caso, verificamos que a dependência do recurso **Foto** passou a ser com o objetivo **Selecionar Foto**, ao invés de ser com a tarefa **Visualizar Foto(s)**. O objetivo **Foto disponibilizada na tela**, tem as tarefas **Disponibilizar tamanho original** e **Disponibilizar tamanho reduzido** como tarefas alternativas.

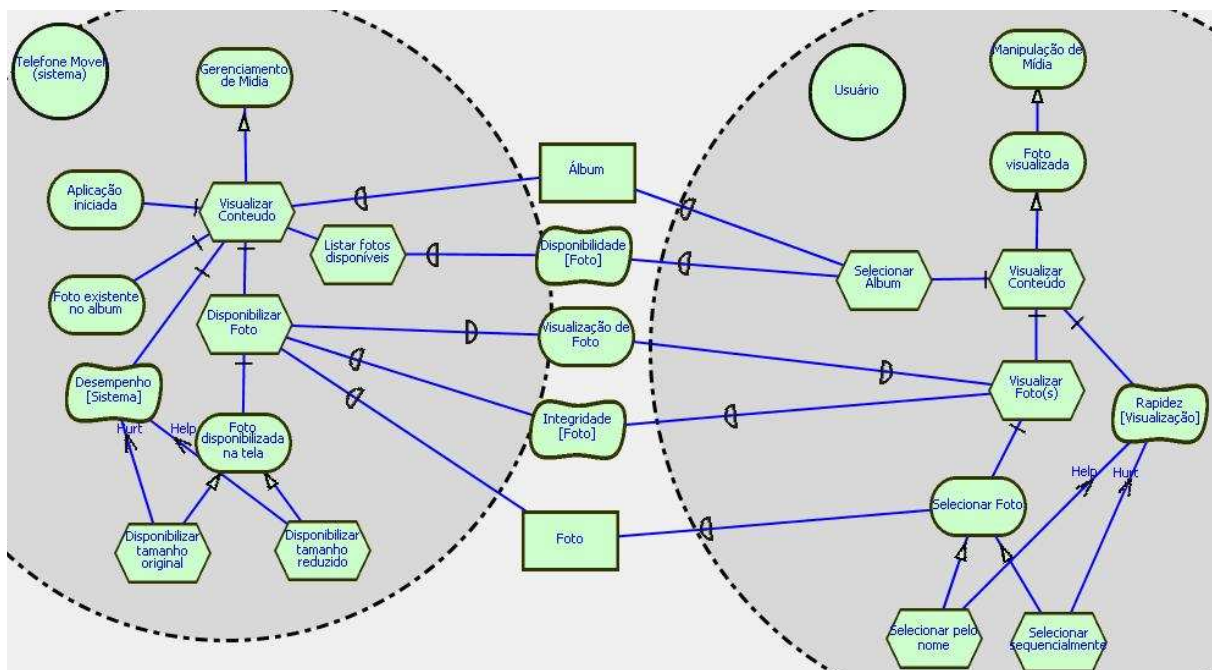


Figura 5.6 – Modelo SR do caso de uso Visualizar Conteúdo

- **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de trigger:** as pré-condições da tarefa **Visualizar Conteúdo** são **Aplicação Iniciada** e **Foto existente no álbum**. De acordo com esta regra, essas pré-condições são modeladas como objetivos e relacionados através do relacionamento de decomposição de tarefas, no ator que representa o sistema. Este caso de uso não contém pós-condições. O evento de *trigger* deste caso de uso é **Visualização de Foto**. Esta *trigger* é um *dependum* do tipo objetivo onde o *dependee* é a tarefa **Visualizar Foto(s)** e o *depend* é a tarefa **Disponibilizar Foto**.

- **Regra 7 - Modelagem dos relacionamentos do tipo *includes* e *extends*:** esta regra não é aplicada neste caso de uso, pois, o mesmo não possui relacionamentos do tipo *includes* ou *extends*.

Passo 1.3 – Construção do modelo i* intencional:

- **Diretriz 1 – Análise dos elementos intencionais:**
 - Qual o estado final alcançado através da execução da tarefa “Visualizar Conteúdo”?
A resposta pode ser: “Foto visualizada no álbum selecionado”.
 - Qual ator precisa atingir este objetivo (*dependor*)?
Resposta: o usuário
 - De que ator depende a obtenção do objetivo (*dependee*)?
Resposta: depende do Telefone Móvel (sistema)

Este objetivo (**Foto visualizada**) será inserido entre a tarefa e o objetivo principal do ator *dependor*, ou seja, entre a tarefa **Visualizar Conteúdo** e o objetivo **Manipulação de Mídia** do ator **Usuário**. Ou seja, **Visualizar Conteúdo** se tornará o meio para obter o objetivo **Foto visualizada**, que por sua vez será um meio para alcançar o objetivo principal, **Manipulação de Mídia**.

- **Diretriz 2 – Análise de atributos de qualidade:** para o sistema conseguir visualizar uma foto, o usuário depende que o Telefone Móvel (*dependee*) mantenha as fotos íntegras e disponíveis. Este desejo é representado pelos *softgoals* **Disponibilidade [Foto]** e **Integridade [Foto]**. Concluimos também que caso a foto seja visualizada sequencialmente, a visualização da mesma é mais lenta do que se for visualizada através do nome. Assim, a tarefa **Selecionar sequencialmente** contribui negativamente para satisfazer o *softgoal* **Rapidez [Visualização]**. Já a tarefa **Selecionar pelo nome** contribui positivamente para a satisfação do mesmo *softgoal*. O mesmo raciocínio é aplicado às tarefas alternativas **Disponibilizar tamanho original** e **Disponibilizar tamanho reduzido**. A primeira contribui negativamente para satisfazer o *softgoal* **Desempenho [Sistema]** e a segunda contribui positivamente para a satisfação do mesmo *softgoal*.

5.1.4. Caso de Uso: Adicionar Álbum

A especificação original do caso de uso **Adicionar Álbum** encontra-se na Figura 5.7. Comparando esta especificação com a do caso de uso **Adicionar Foto** (vide Figura 4.11), verificamos que são similares e assim, muito provavelmente, os modelos SR desses casos de uso também serão parecidos. O modelo SR do caso de uso **Adicionar Álbum** é apresentado na Figura 5.8

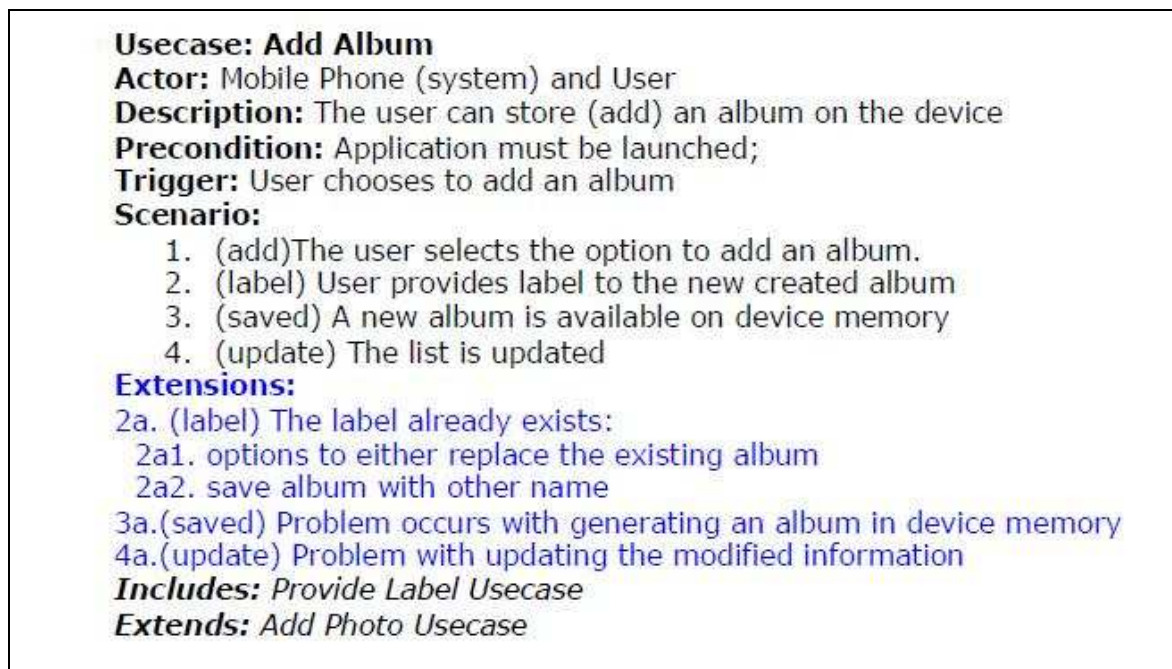


Figura 5.7 – Especificação original do caso de uso Adicionar Álbum

Passo 1.1 - Identificação e modelagem dos atores e objetivos principais: de acordo com a especificação do caso de uso **Adicionar Album**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente.

Passo 1.2 – Construção do modelo i* operacional:

- **Regra 1 – Modelagem de atividades:** de acordo com esta regra, o caso de uso **Adicionar Álbum** é modelado como tarefa e associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim.
- **Regra 2 – Modelagem de tarefas:** os passos do cenário principal do caso de uso em questão são mapeados em sub-tarefas e relacionados à tarefa **Adicionar Álbum**

através do relacionamento de decomposição de tarefas. As tarefas iniciada no ator **Telefone Móvel (sistema)** seriam: **Armazenar Álbum** e **Atualizar lista de álbuns**, porém, não vemos sentido em modelar **Atualizar lista de álbuns** como tarefa, pois, entendemos como uma pós-condição do caso de uso (vide Regra 6). Assim, o ator **Telefone Móvel (sistema)** tem apenas a tarefa **Armazenar Álbum**. As tarefas iniciadas no ator **Usuário** são: **Selecionar opção de adicionar álbum** e **Atribuir nome para álbum**.

- **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** a tarefa **Atribuir nome para álbum** (*dependee*) fornece o recurso **Nome** (*dependum*) que vai ser consumido pela tarefa **Armazenar Álbum** (*dependor*).
- **Regra 4 – Modelagem de tarefas que consomem recursos:** a tarefa **Armazenar Álbum** consome o recurso **Nome**, que já foi modelado pela regra anterior.
- **Regra 5 – Modelagem de tarefas alternativas:** apesar da especificação original do caso de uso **Adicionar Álbum** (vide Figura 5.7) não conter cenários alternativos, mas apenas de exceção, sentimos a necessidade de acrescentar informações que achamos relevantes para enriquecer a especificação do caso de uso. Adicionamos cenário alternativo ao passo 3 do cenário principal: “3. (*saved*) *The system saves automatically the new album on the device*”. A modelagem deste cenário alternativo fica da seguinte forma: a tarefa **Armazenar Álbum** tem uma ligação de decomposição com o objetivo **Salvar Álbum** que tem as seguintes tarefas alternativas: **Salvar Automaticamente** e **Salvar pelo Usuário**.
- **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de trigger:** a pré-condição da tarefa **Adicionar Álbum** é **Aplicação Iniciada**. De acordo com esta regra, essa pré-condição é modelada como objetivo e relacionado através do relacionamento de decomposição de tarefas, no ator que representa o sistema. A especificação original do caso de uso **Adicionar Álbum** não contém pós-condições, porém, como vimos anteriormente na Regra 2, supomos e assim, modelamos **Atualização de lista de álbuns** como uma pós-condição do caso de uso. Essa pós-condição é modelada da mesma forma que a pré-condição. O evento de *trigger* da tarefa **Adicionar Álbum** é **Adição de Álbum**. Esta *trigger* é um *dependum* do tipo objetivo onde o *dependee* é a tarefa **Armazenar Álbum** e o *dependor* é a tarefa **Selecionar opção de adicionar álbum**.

- **Regra 7 - Modelagem dos relacionamentos do tipo *includes* e *extends*:** o caso de uso **Adicionar Álbum** contém um relacionamento do tipo *includes* com o caso de uso **Fornecer Legenda** e um relacionamento do tipo *extends* com o caso de uso **Adicionar Foto**. De acordo com esta regra, **Fornecer Legenda** é modelada como sub-tarefa (através do relacionamento de decomposição de tarefas) da tarefa **Adicionar Álbum**. Nesse caso, verificamos que a dependência do recurso **Nome** passou a ser com a tarefa **Fornecer Legenda**, ao invés de ser com a tarefa **Armazenar Álbum**. **Adicionar Foto** é modelado como um objetivo relacionado ao objetivo **Gerenciamento de Mídia** através da ligação meio-fim. Mais adiante, na Diretriz 1, com a análise dos elementos intencionais, essa ligação sofrerá modificação.

Passo 1.3 – Construção do modelo i* intencional:

- **Diretriz 1 – Análise dos elementos intencionais:**
 - Qual o estado final alcançado através da execução da tarefa “Adicionar Álbum”?
A resposta pode ser: “Novo álbum incluído no celular”.
 - Qual ator precisa atingir este objetivo (*dependor*)?
Resposta: o Telefone Móvel (sistema)
 - De que ator depende a obtenção do objetivo (*dependee*)?
Resposta: depende do usuário

Este objetivo (**Novo álbum incluído**) será inserido entre a tarefa e o objetivo principal do ator *dependor*, ou seja, entre a tarefa **Adicionar Álbum** e o objetivo **Gerenciamento de Mídia** do ator **Telefone Móvel (sistema)**. Desse modo, **Adicionar Álbum** se tornará o meio para obter o objetivo **Novo álbum incluído**, que por sua vez será um meio para alcançar o objetivo principal, **Gerenciamento de Mídia**. Nesse caso, verificamos que o relacionamento do objetivo **Adicionar Foto** passou a ser com o objetivo **Novo álbum incluído**, ao invés de ser com o objetivo **Gerenciamento de Mídia**.

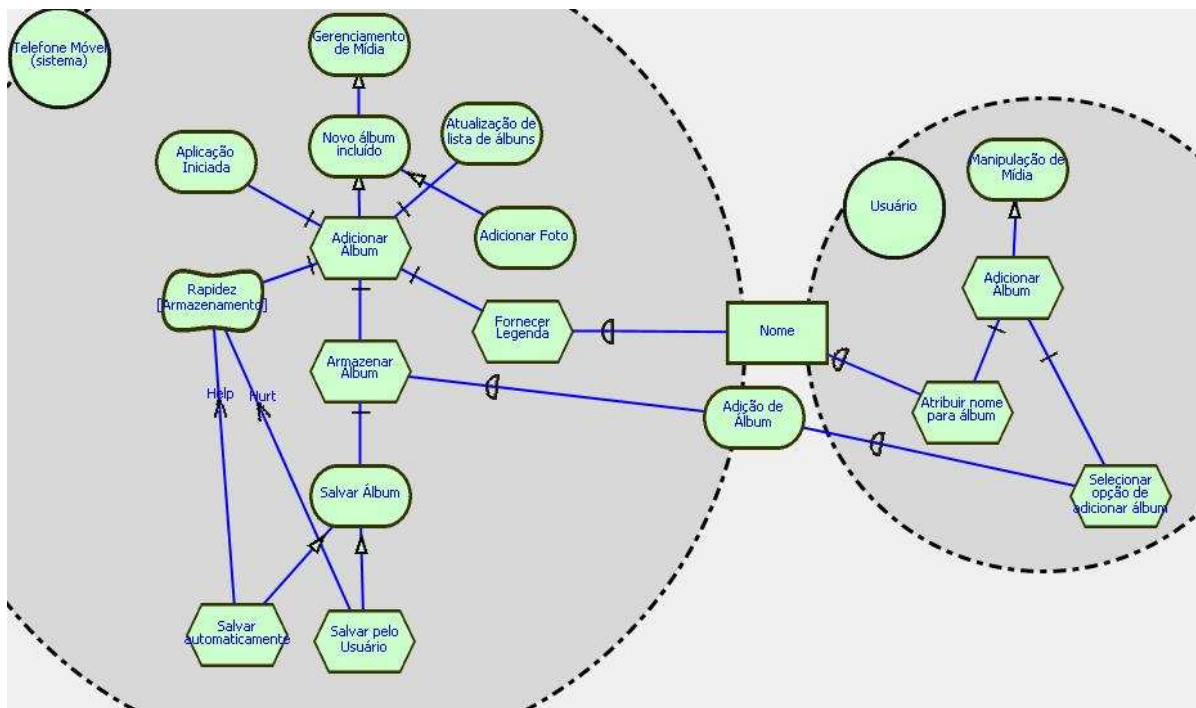


Figura 5.8 – Modelo SR do caso de uso Adicionar Álbum

- **Diretriz 2 – Análise de atributos de qualidade:** observamos que caso o álbum seja salvo pelo próprio sistema, ou seja, automaticamente, o armazenamento do mesmo é mais rápido do que se for salvo pelo usuário. Assim, a tarefa **Salvar Automaticamente** contribui positivamente para satisfazer o *softgoal* **Rapidez [Armazenamento]**. Já a tarefa **Salvar pelo Usuário** contribui negativamente para a satisfação do mesmo *softgoal*.

5.1.5. Caso de Uso: Apagar Álbum

Comparando a especificação original desse caso de uso (Figura 5.9) com a do caso de uso **Apagar Foto** (vide Figura 5.3), verificamos que são similares e assim, os modelos SR desses casos de uso, provavelmente, serão parecidos. A Figura 5.10 apresenta a configuração final do modelo SR do caso de uso **Apagar Álbum**.

Passo 1.1 - Identificação e modelagem dos atores e objetivos principais: de acordo com a especificação do caso de uso **Apagar Álbum**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente.

Passo 1.2 – Construção do modelo i* operacional:

- **Regra 1 – Modelagem de atividades:** de acordo com esta regra, o caso de uso **Apagar Álbum** é modelado como tarefa e associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim.

Usecase: Delete Album Actor: Mobile Phone (system) and User Description: The user can delete an album from the mobile phone Precondition: Application must be launched; The album must exist on the mobile phone, Trigger: User chooses to delete Album Scenario: 1. (listing) The albums stored in the mobile phone memory are listed down. 2. (delete) The user selects the option to delete the selected album 3. (delete) Album removed from the mobile phone 4. (update) The list is updated Extensions: 1a. (null data) There is no album available on the device 1b. (listing) Problem with listing down the albums 3a. (delete) Problem occurs while deleting the album 4a. (update) Problem with updating the modified information Extends: Delete Photo Usecase

Figura 5.9 – Especificação original do caso de uso Apagar Álbum

- **Regra 2 – Modelagem de tarefas:** os passos do cenário principal do caso de uso em questão são mapeados em sub-tarefas e relacionados à tarefa **Apagar Álbum** através do relacionamento de decomposição de tarefas. As tarefas iniciadas no ator **Telefone Móvel (sistema)** seriam: **Listar Álbuns existentes**, **Remover Álbum** e **Atualizar lista de álbuns**, porém, não vemos sentido em modelar **Atualizar lista de álbuns** como tarefa, pois, entendemos como uma pós-condição do caso de uso. Assim, o ator **Telefone Móvel (sistema)** tem as tarefas **Listar Álbuns existentes** e **Remover Álbum**. As tarefas iniciadas no ator **Usuário** são: **Selecionar opção de apagar álbum** e **Selecionar Álbum a ser removido**. Esta última tarefa não aparece explicitamente nos passos do caso de uso, mas foi modelada, pois achamos que ela é necessária para a realização do caso de uso. Mais adiante, na Regra 5, com a modelagem das tarefas alternativas, a tarefa **Remover Álbum** sofrerá modificação.

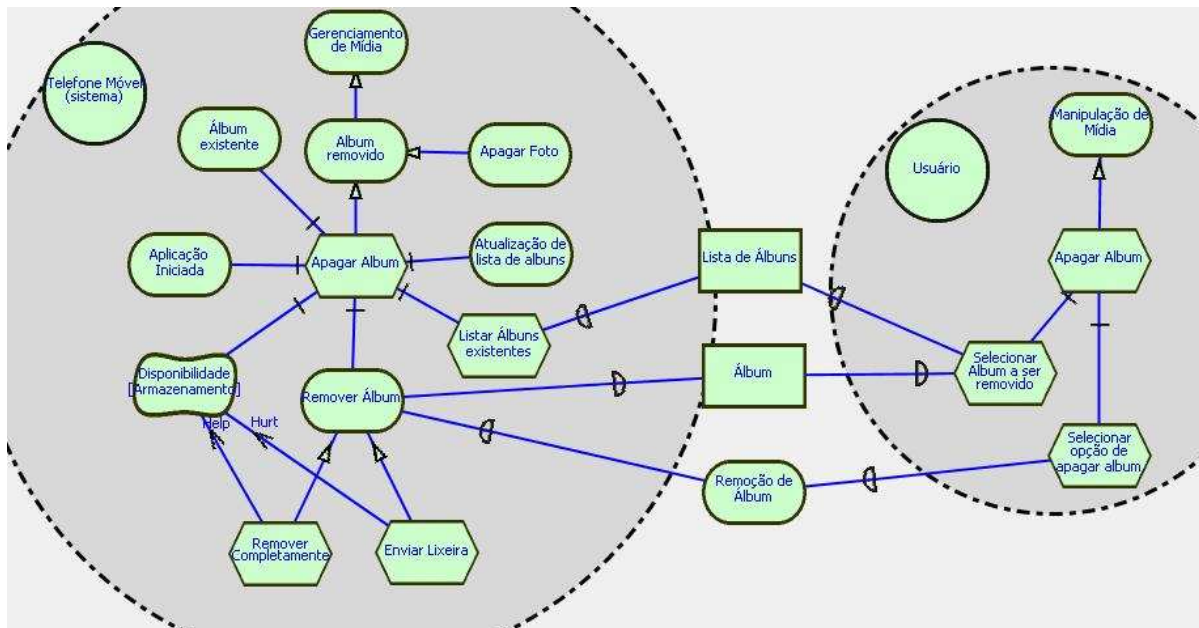


Figura 5.10 – Modelo SR do caso de uso Apagar Álbum

- **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** a tarefa **Listar Álbuns existentes** (*dependee*) fornece o recurso **Lista de Álbuns** (*dependum*), que vai ser consumida pela tarefa **Selecionar Álbum a ser removido** (*dependor*). Esta tarefa (**Selecionar Álbum a ser removido**) por sua vez, fornece o recurso **Album** (*dependum*), que vai ser consumido pela tarefa **Remover Álbum** (*dependor*).
- **Regra 4 – Modelagem de tarefas que consomem recursos:** os recursos consumidos neste caso de uso (**Lista de Álbuns** e **Album**) foram produzidos pela regra anterior.
- **Regra 5 – Modelagem de tarefas alternativas:** apesar da especificação original do caso de uso **Apagar Álbum** (vide Figura 5.9) conter apenas cenários de exceção, sentimos a necessidade de adicionar cenários alternativos ao passo 3 do cenário principal: “3. (*delete*) The user chooses to send the album to the trash.”. A modelagem deste cenário alternativo fica da seguinte forma: a tarefa **Remover Álbum** torna-se um objetivo que pode ser alcançado pelas seguintes tarefas alternativas: **Remover Completamente** e **Enviar Lixeira**.
- **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de trigger:** as pré-condições da tarefa **Apagar Álbum** são **Aplicação Iniciada** e **Álbum existente**. De acordo com esta regra, essas pré-condições são modeladas como objetivos e

relacionados através do relacionamento de decomposição de tarefas, no ator que representa o sistema. A especificação original do caso de uso **Apagar Álbum** não contém pós-condições, porém, como vimos anteriormente na Regra 2, supomos e assim, modelamos **Atualização de lista de álbuns** como uma pós-condição do caso de uso. Essa pós-condição é modelada da mesma forma que as pré-condições. O evento de *trigger* da tarefa **Apagar Álbum** é **Remoção de Álbum**. Esta *trigger* é um *dependum* do tipo objetivo, onde o *dependee* é a tarefa **Remover Álbum** e o *dependor* é a tarefa **Selecionar opção de apagar album**.

- **Regra 7 - Modelagem dos relacionamentos do tipo *includes* e *extends*:** o caso de uso **Apagar Álbum** contém um relacionamento do tipo *extends* com o caso de uso **Apagar Foto**. De acordo com esta regra, **Apagar Foto** é modelado como um objetivo relacionado ao objetivo **Gerenciamento de Mídia** através da ligação meio-fim. Mais adiante, na Diretriz 1, com a análise dos elementos intencionais, essa ligação sofrerá modificação.

Passo 1.3 – Construção do modelo i* intencional:

- **Diretriz 1 – Análise dos elementos intencionais:**
 - Qual o estado final alcançado através da execução da tarefa “Apagar Álbum”?
A resposta pode ser: “Álbum removido do celular”.
 - Qual ator precisa atingir este objetivo (*dependor*)?
Resposta: o Telefone Móvel (sistema)
 - De que ator depende a obtenção do objetivo (*dependee*)?
Resposta: depende do usuário

Este objetivo (**Álbum removido**) será inserido entre a tarefa e o objetivo principal do ator *dependor*, ou seja, entre a tarefa **Apagar Álbum** e o objetivo **Gerenciamento de Mídia** do ator **Telefone Móvel (sistema)**. Assim, **Apagar Álbum** se tornará o meio para obter o objetivo **Álbum removido**, que por sua vez será um meio para alcançar o objetivo principal, **Gerenciamento de Mídia**. Nesse caso, verificamos que o relacionamento do objetivo **Apagar Foto** passou a ser com o objetivo **Álbum removido**, ao invés de ser com o objetivo **Gerenciamento de Mídia**.

- **Diretriz 2 – Análise de atributos de qualidade:** caso o álbum seja removido completamente do celular, o sistema ganhará espaço de armazenamento. Assim, a tarefa **Remover Completamente** contribui positivamente para satisfazer o *softgoal* **Disponibilidade [Armazenamento]**, enquanto que a tarefa **Enviar Lixeira** contribui negativamente para a satisfação do mesmo *softgoal*.

5.1.6. Caso de Uso: Fornecer Legenda

A especificação original do caso de uso Fornecer Legenda encontra-se na Figura 5.11 e a Figura 5.12 apresenta o modelo SR deste caso de uso.

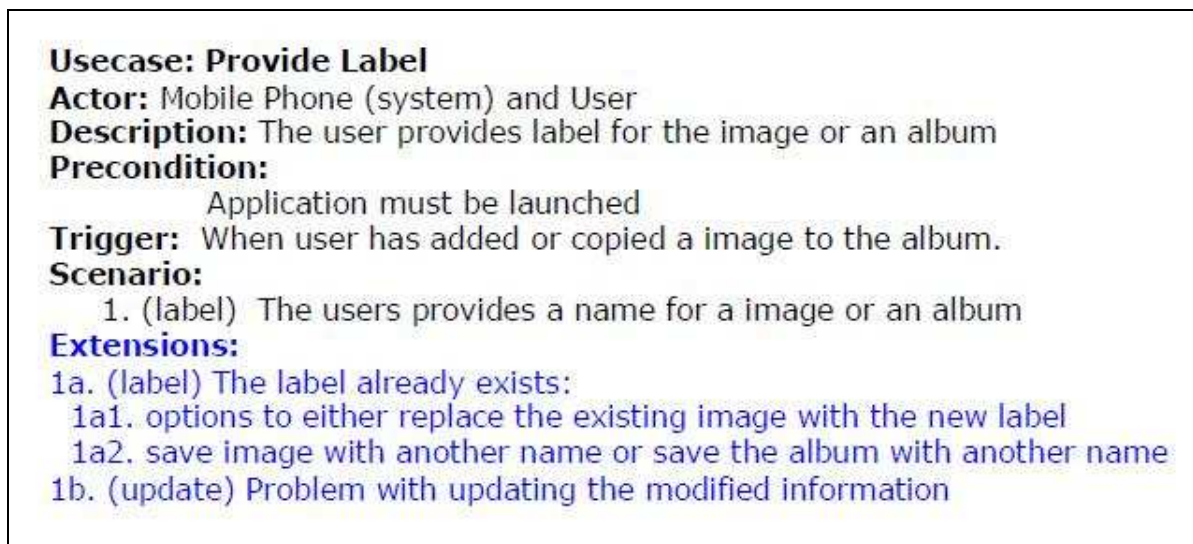


Figura 5.11 – Especificação original do caso de uso Fornecer Legenda

Passo 1.1 - Identificação e modelagem dos atores e objetivos principais: de acordo com a especificação do caso de uso **Fornecer Legenda**, os atores identificados são **Telefone Móvel (sistema)** e **Usuário**, e seus principais objetivos são **Gerenciamento de Mídia** e **Manipulação de Mídia**, respectivamente.

Passo 1.2 – Construção do modelo i* operacional:

- **Regra 1 – Modelagem de atividades:** de acordo com esta regra, o caso de uso **Fornecer Legenda** é modelado como tarefa e associada ao objetivo principal dos atores **Telefone Móvel (sistema)** e **Usuário**, através do relacionamento meio-fim.
- **Regra 2 – Modelagem de tarefas:** só existe um passo no cenário principal desse caso de uso que é mapeada na sub-tarefa **Atribuir nome**, no ator **Usuário**.

- **Regra 3 – Modelagem de tarefas que produzem/fornecem recursos:** a tarefa **Atribuir nome** (*dependee*) fornece o recurso **Nome** (*dependum*) que vai ser consumido pela tarefa **Fornecer Legenda** (*depende*). Mais adiante, na Regra 5, com a modelagem das tarefas alternativas, essa dependência sofrerá modificação.
- **Regra 4 – Modelagem de tarefas que consomem recursos:** o recurso consumido neste caso de uso (**Nome**) já foi modelado na regra anterior.
- **Regra 5 – Modelagem de tarefas alternativas:** apesar da especificação original do caso de uso **Fornecer Legenda** (vide Figura 5.11) não conter cenários alternativos, mas apenas de exceção, sentimos a necessidade de acrescentar informações que achamos relevantes para enriquecer a especificação do caso de uso. Assim, adicionamos um cenário alternativo ao passo 1 do cenário principal: “1. (*label*) The system assigns a label automatically to the photo or album”. A modelagem deste cenário alternativo fica da seguinte forma: a tarefa **Fornecer Legenda** tem uma ligação de decomposição com o objetivo **Nomear Álbum ou Foto** que por sua vez, tem as seguintes tarefas alternativas: **Nomear Automaticamente** e **Nomear pelo Usuário**. Nesse caso, verificamos que a dependência do recurso **Nome** passou a ser com a tarefa alternativa **Nomear pelo Usuário**, ao invés de ser com a tarefa **Fornecer Legenda**.
- **Regra 6 – Modelagem de pré-condições, pós-condições e eventos de trigger:** a pré-condição da tarefa **Fornecer Legenda** é **Aplicação Iniciada**. De acordo com esta regra, essa pré-condição é modelada, no ator que representa o sistema, como objetivo e relacionado através do relacionamento de decomposição de tarefas. O evento de *trigger* da tarefa **Fornecer Legenda** é **Adição de Foto/Adição de Álbum**. Esta *trigger* é um *dependum* do tipo objetivo onde o *dependee* é a tarefa **Fornecer Legenda** e o *depende* é a tarefa **Atribuir Nome**.
- **Regra 7 - Modelagem dos relacionamentos do tipo includes e extends:** esta regra não é aplicada neste caso de uso, pois, o mesmo não possui relacionamentos do tipo *includes* ou *extends*.

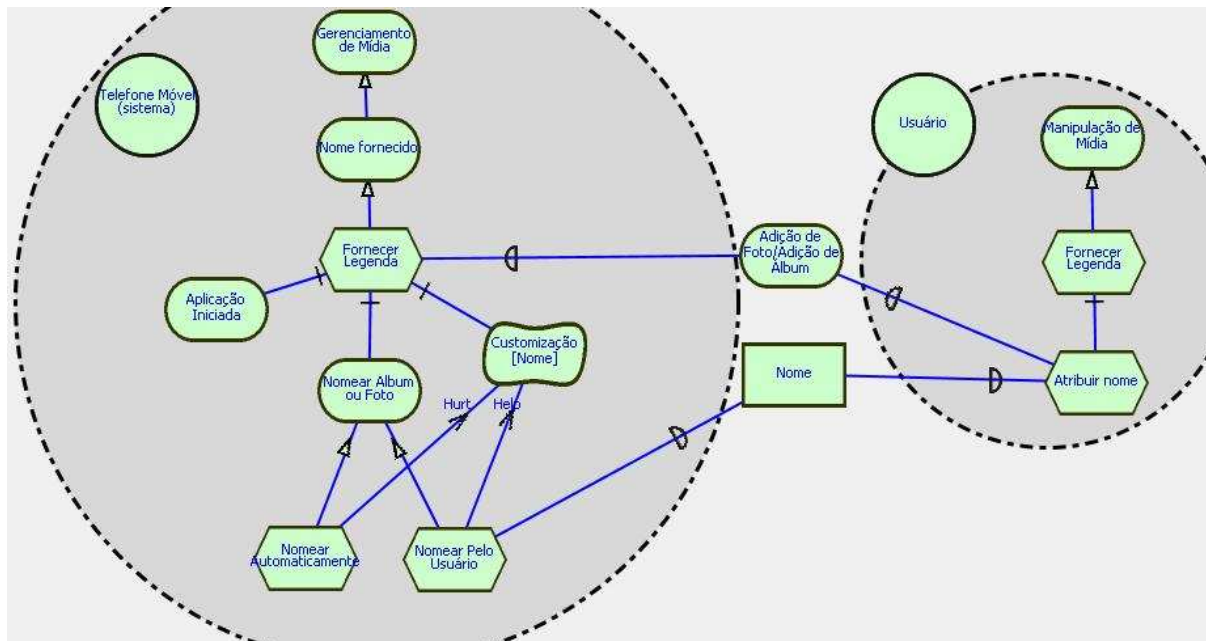


Figura 5.12 – Modelo SR do caso de uso Fornecer Legenda

Passo 1.3 – Construção do modelo i* intencional:

- **Diretriz 1 – Análise dos elementos intencionais:**

- Qual o estado final alcançado através da execução da tarefa “Fornecer Legenda”?

A resposta pode ser: “Nome fornecido a foto ou álbum”.

- Qual ator precisa atingir este objetivo (*dependor*)?

Resposta: o Telefone Móvel (sistema)

- De que ator depende a obtenção do objetivo (*dependee*)?

Resposta: depende do usuário

Este objetivo (**Nome fornecido**) será inserido entre a tarefa e o objetivo principal do ator *dependor*, ou seja, entre a tarefa **Fornecer Legenda** e o objetivo **Gerenciamento de Mídia** do ator **Telefone Móvel (sistema)**. Assim, **Fornecer Legenda** se tornará o meio para obter o objetivo **Nome fornecido**, que por sua vez será um meio para alcançar o objetivo principal, **Gerenciamento de Mídia**.

- **Diretriz 2 – Análise de atributos de qualidade:** observamos que caso a foto ou o álbum seja nomeado pelo usuário, o nome será customizado. Assim, a tarefa **Nomear Automaticamente** contribui negativamente para satisfazer o *softgoal* **Customi-**

zação [Nome] e a tarefa **Nomear pelo Usuário** contribui positivamente para a satisfação do mesmo *softgoal*.

5.2. Identificação de elementos com possibilidade de serem *features*

A segunda atividade do processo **G2FM** é a identificação dos possíveis elementos que poderão ser *features*. De acordo com a Figura 5.13, observamos que os artefatos consumidos nesta atividade são o modelo SR e a aplicação de algumas heurísticas.

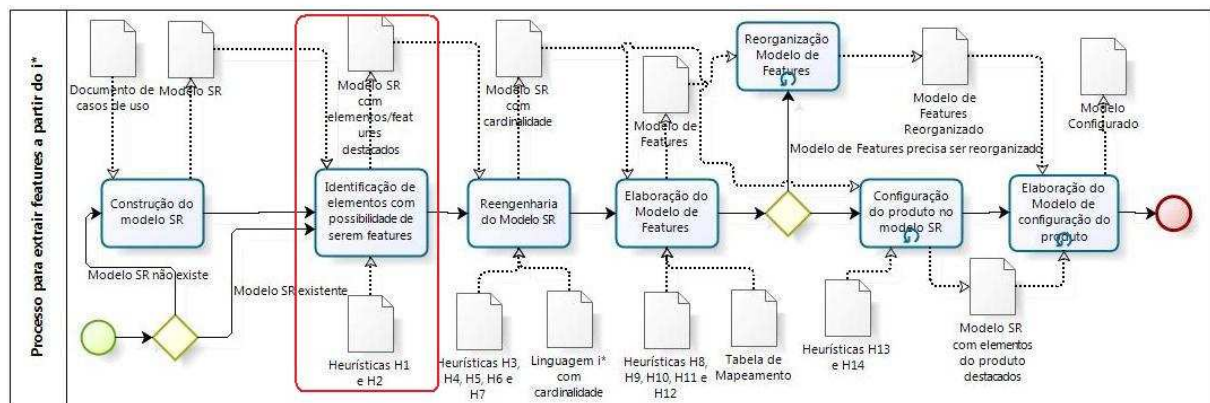


Figura 5.13 – Segunda atividade do processo G2FM: Identificação de elementos com possibilidade de serem *features*

Por questões de legibilidade, aplicaremos as heurísticas aos modelos SR dos casos de uso separadamente, ao invés de unificar os modelos em um só. O resumo das heurísticas utilizadas nesta atividade encontra-se na Tabela 5.1.

Tabela 5.1 – Resumo das heurísticas H1 e H2

Heurística	Definição
H1	As <i>features</i> podem ser extraídas a partir dos elementos do tipo tarefa e recurso.
H2	As <i>features</i> podem ser extraídas a partir dos elementos internos do ator que representa o sistema e das dependências (particularmente do <i>dependum</i>) ligadas diretamente a esse ator.

Aplicando as heurísticas H1 e H2 aos modelos SR dos casos de uso **Apagar Foto** (Figura 5.4), **Visualizar Conteúdo** (Figura 5.6), **Adicionar Álbum** (Figura 5.8), **Apagar Álbum** (Figura 5.10) e **Fornecer Legenda** (Figura 5.12), obtemos as figuras 5.14, 5.15, 5.16, 5.17 e 5.18, respectivamente. Essas figuras obtidas representam a saída desta atividade do processo, ou seja, modelos SR com destaque nos elementos possíveis de serem *features*.

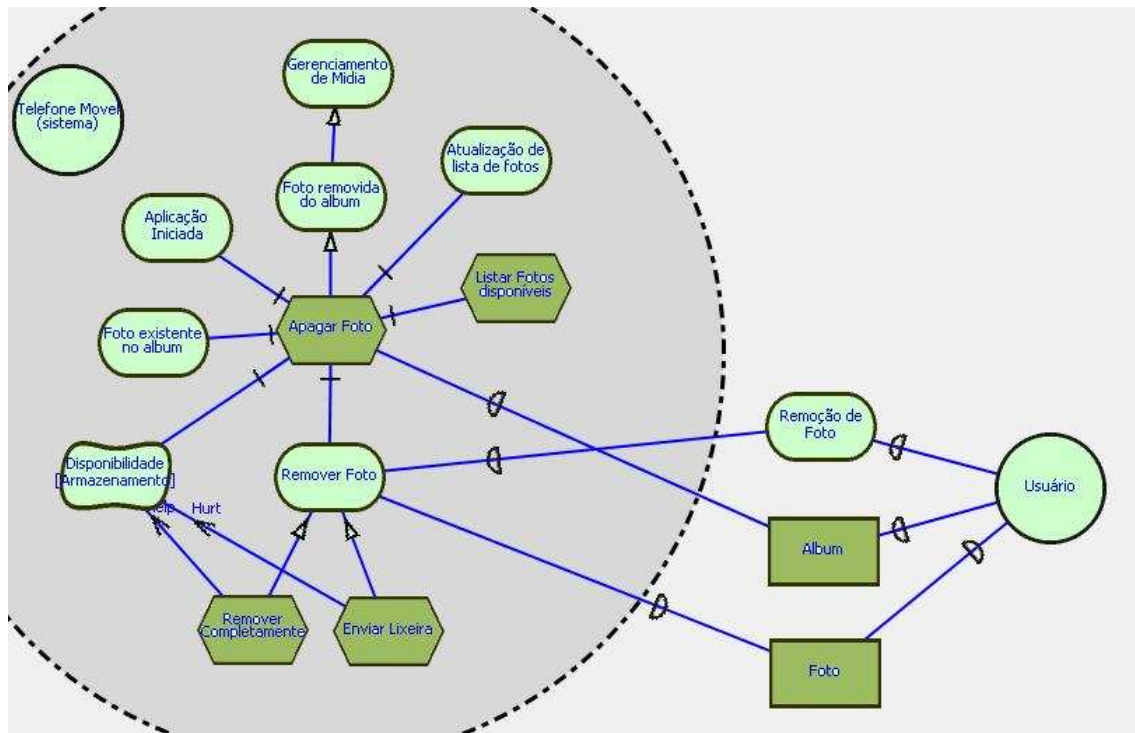


Figura 5.14 – Modelo SR do caso de uso Apagar Foto com possíveis elementos de serem *features*

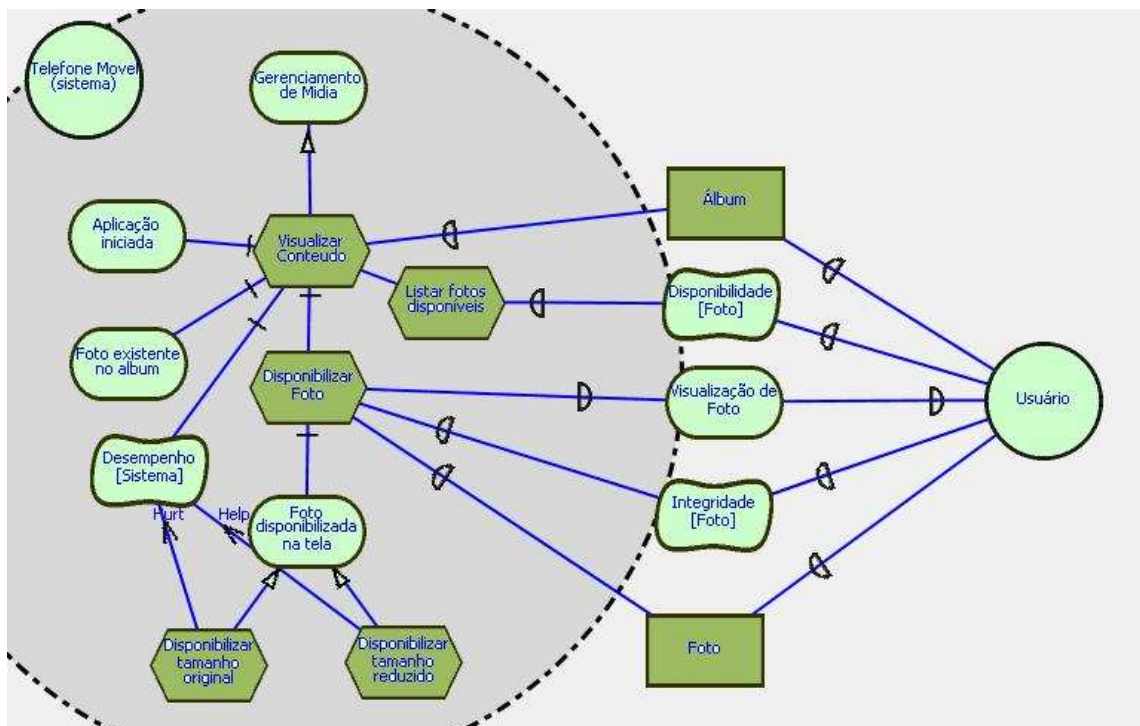


Figura 5.15 – Modelo SR do caso de uso Visualizar Conteúdo com possíveis elementos de serem *features*

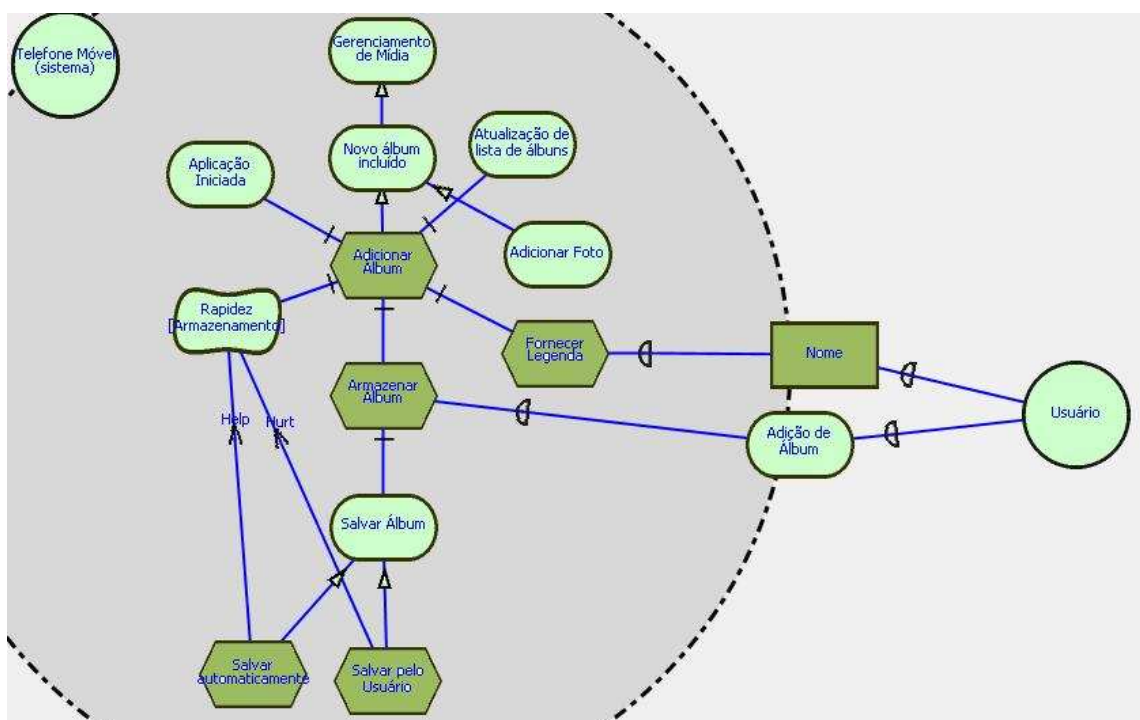


Figura 5.16 – Modelo SR do caso de uso Adicionar Álbum com possíveis elementos de serem *features*

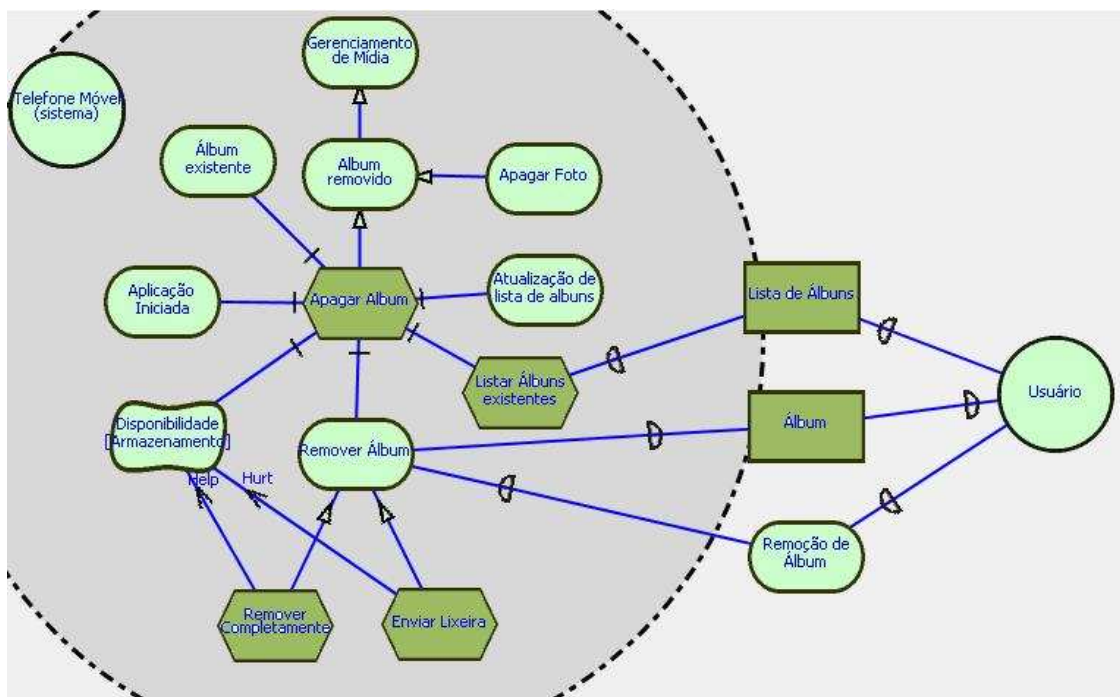


Figura 5.17 – Modelo SR do caso de uso Apagar Album com possíveis elementos de serem *features*

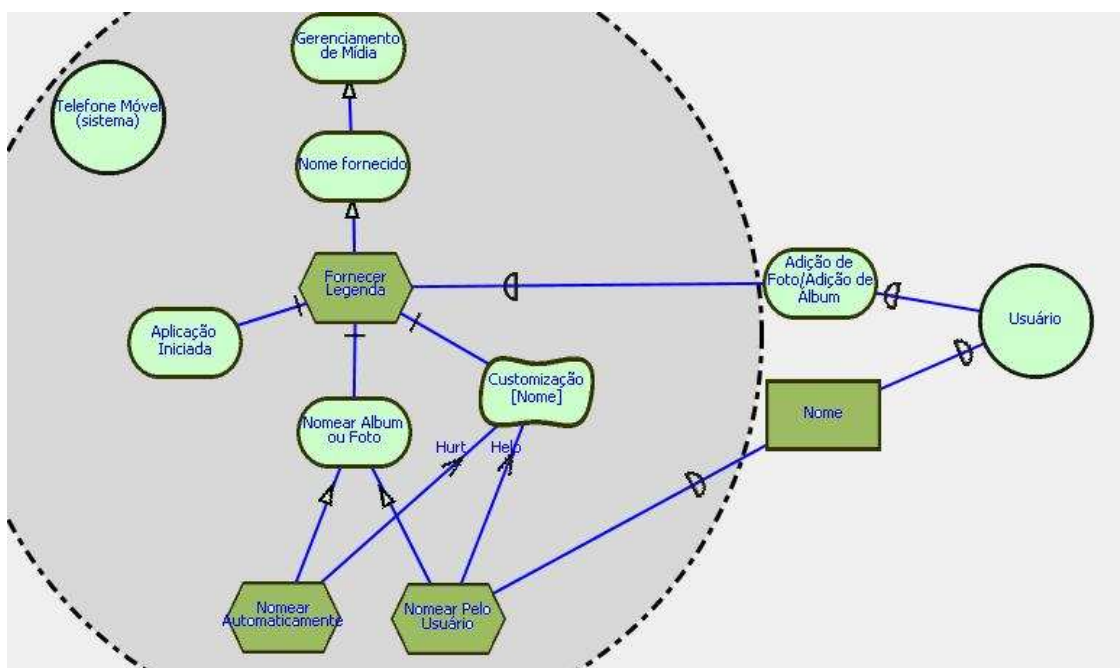


Figura 5.18 – Modelo SR do caso de uso Fornecer Legenda com possíveis elementos de serem *features*

5.3. Reengenharia do Modelo SR

Esta atividade tem como principal objetivo a adição de novas informações aos modelos SR gerados na atividade anterior, de acordo com a aplicação de algumas heurísticas e da linguagem **i*-c** (linguagem **i*** com cardinalidade). O artefato produzido nesta atividade é o modelo SR com cardinalidade (vide Figura 5.19).

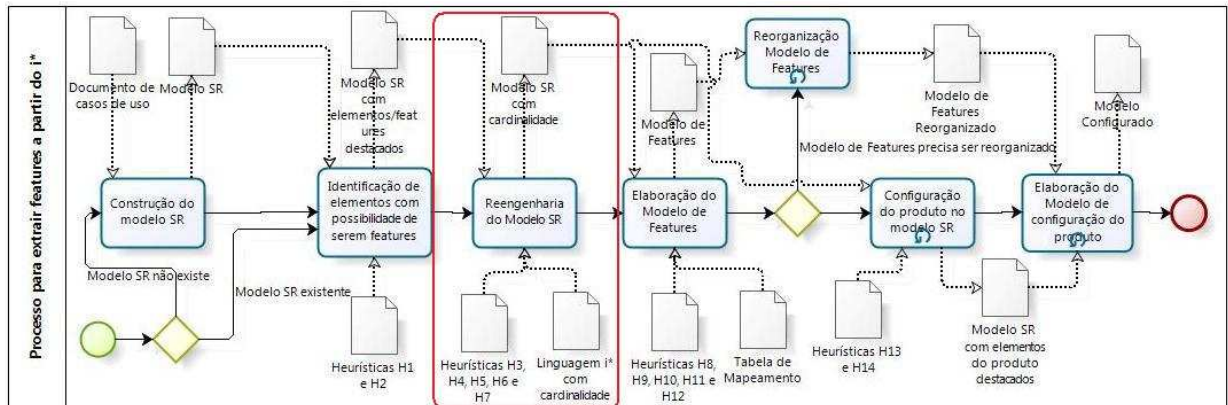


Figura 5.19 – Terceira atividade do processo G2FM: Reengenharia do Modelo SR

A Tabela 5.2 contém o resumo das heurísticas utilizadas nesta atividade.

Aplicando as heurísticas H3, H4, H5, H6 e H7 no modelo SR do caso de uso **Apagar Foto** (Figura 5.14), obtemos o modelo SR com cardinalidade da Figura 5.20.

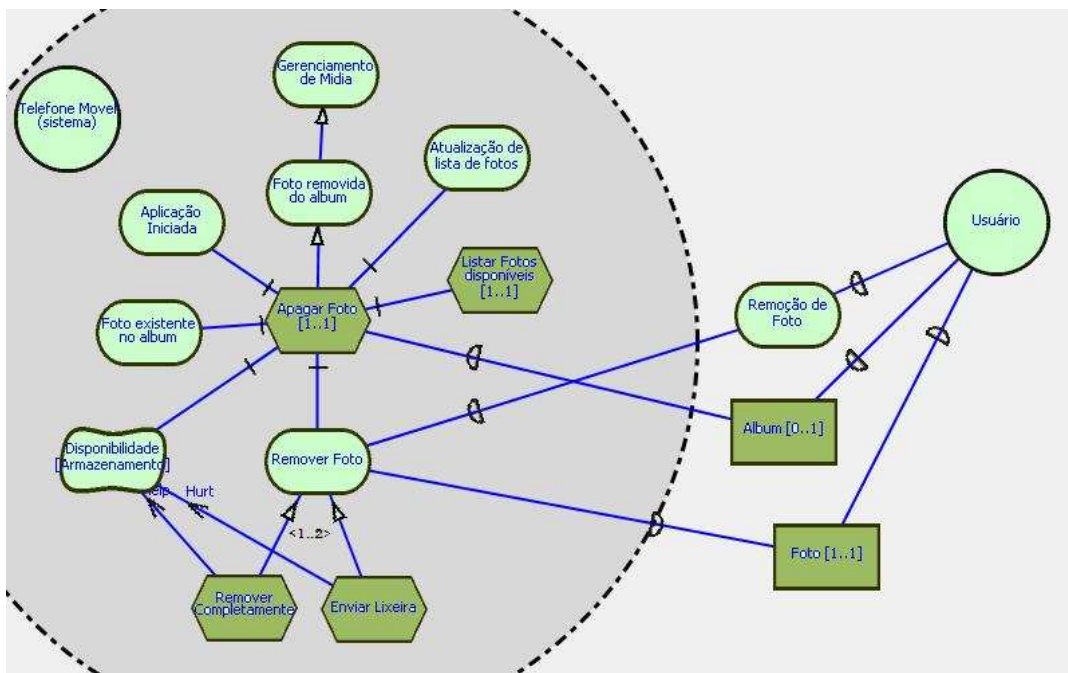


Figura 5.20 – Modelo SR com cardinalidade do caso de uso Apagar Foto

Tabela 5.2 – Resumo das heurísticas H3, H4, H5, H6 e H7

Heurística	Definição
H3	Todos os elementos do modelo SR que foram destacados como possíveis <i>features</i> serão reestruturados para conter cardinalidade.
H4	Se a quantidade de sub-elementos de uma ligação meio-fim que estão destacados como possíveis <i>features</i> , for maior do que 1, então esses sub-elementos serão agrupados e a cardinalidade pertencerá ao relacionamento. Se a quantidade for exatamente 1, então a cardinalidade pertencerá ao sub-elemento e não ao relacionamento.
H5	Os sub-elementos de uma ligação de decomposição de tarefas que estão destacados como possíveis <i>features</i> , serão modelados como <i>features</i> obrigatórias, significando que a cardinalidade pertencerá aos sub-elementos e terá a cardinalidade [1..1].
H6	As dependências (<i>dependum</i>), que estão destacadas como possíveis <i>features</i> , terão a cardinalidade no próprio elemento.
H7	Os elementos destacados como possíveis <i>features</i> , que não sejam sub-elementos de nenhum outro elemento que também seja feature, terão a cardinalidade pertencente a eles próprios e as <i>features</i> obtidas através deles estarão ligadas diretamente a <i>feature</i> raiz no modelo de <i>features</i> .

Aplicando as mesmas heurísticas no modelo SR do caso de uso **Visualizar Conteúdo** (Figura 5.15), obtemos o modelo SR com cardinalidade da Figura 5.21.

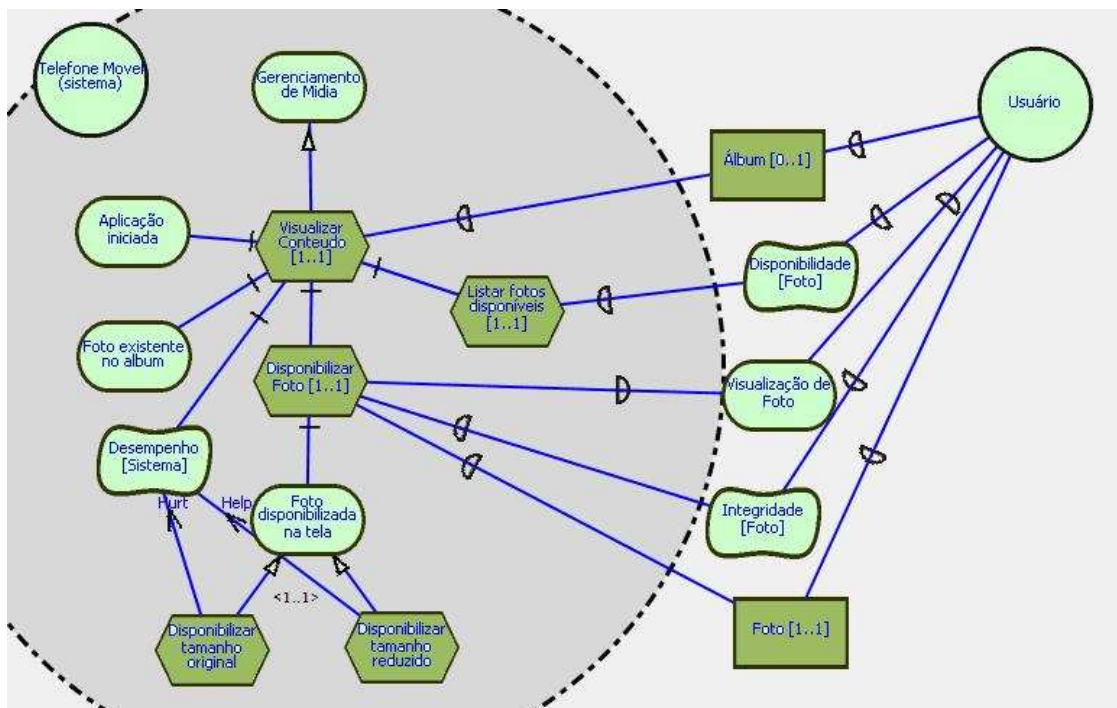


Figura 5.21 – Modelo SR com cardinalidade do caso de uso Visualizar Conteúdo

Repete-se a aplicação dessas heurísticas aos modelos SR dos casos de uso **Adicionar Álbum** (Figura 5.16), **Apagar Álbum** (Figura 5.17) e **Fornecer Legenda** (Figura 5.18), e obtém-se as figuras 5.22, 5.23 e 5.24, respectivamente.

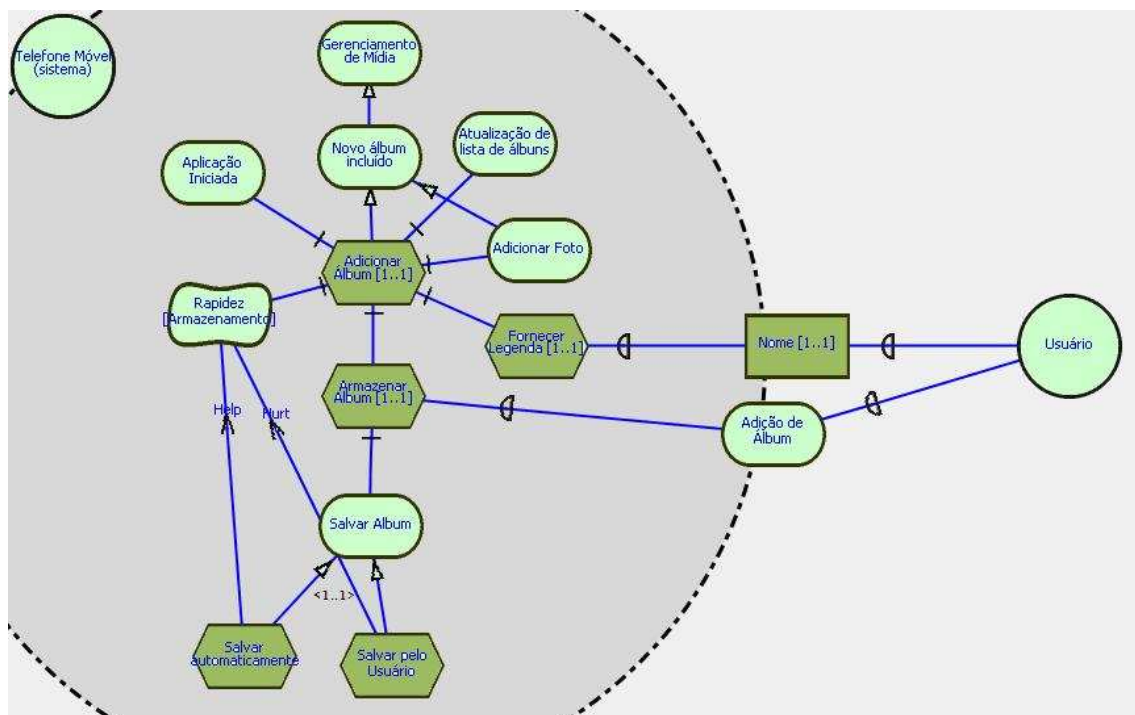


Figura 5.22 – Modelo SR com cardinalidade do caso de uso Adicionar Álbum

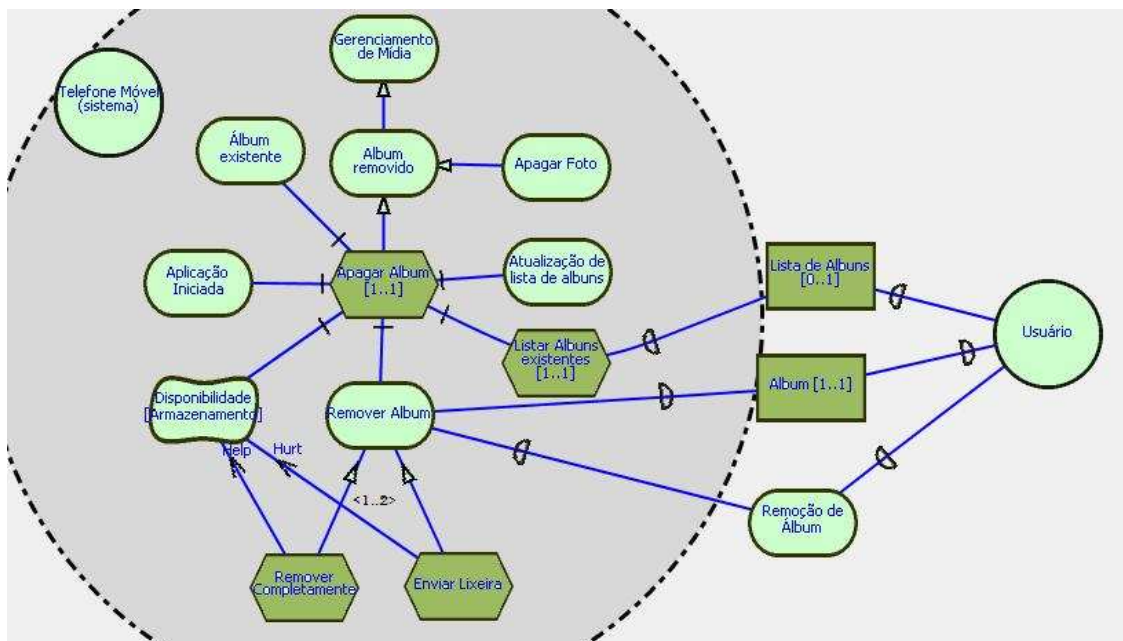


Figura 5.23 – Modelo SR com cardinalidade do caso de uso Apagar Álbum

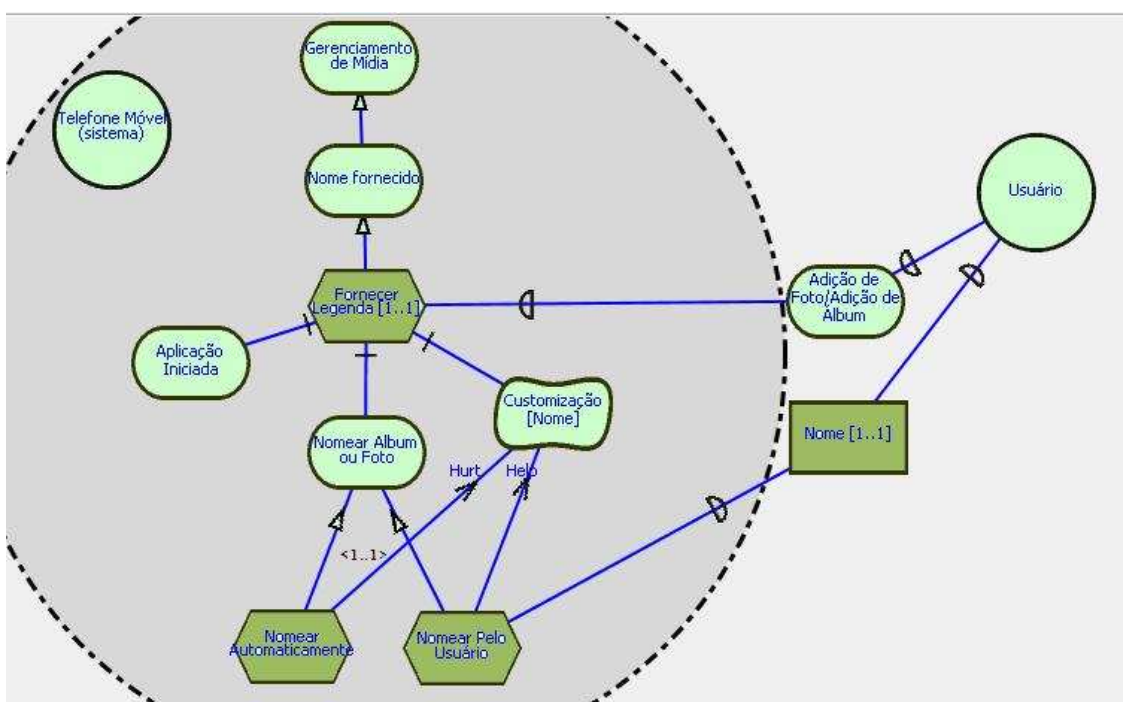


Figura 5.24 – Modelo SR com cardinalidade do caso de uso Fornecer Legenda

O modelo SR com cardinalidade do caso de uso **Adicionar Foto**, foi modelado no capítulo anterior e apresentado na Figura 4.13.

5.4. Elaboração do Modelo de *Features*

De acordo com a Figura 5.25, o objetivo desta atividade é construir o modelo de *features* de uma linha de produtos, a partir da aplicação de algumas heurísticas, uma tabela de mapeamento entre a cardinalidade dos modelos *i** e dos modelos de *features* (Tabela 5.4) e o artefato gerado na atividade anterior (um modelo SR com cardinalidade).

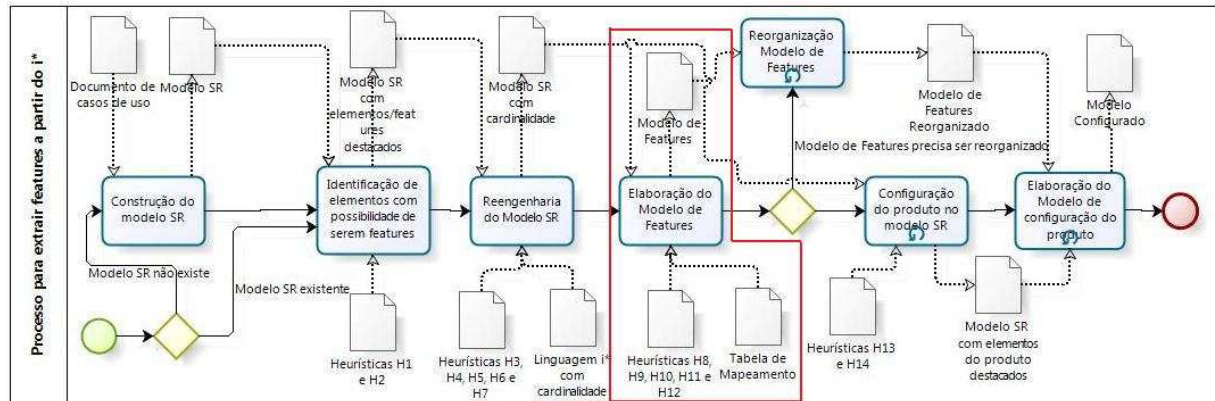


Figura 5.25 – Quarta atividade do processo G2FM: Elaboração do Modelo de *Features*

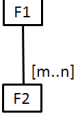
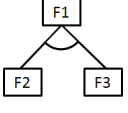
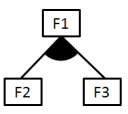
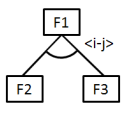
O resumo das heurísticas utilizadas nesta atividade encontra-se na Tabela 5.3.

Tabela 5.3 – Resumo das heurísticas H8, H9, H10, H11 e H12

Heurística	Definição
H8	A <i>feature</i> raiz de um modelo de <i>features</i> será o sistema que está sendo modelado.
H9	Os nomes das <i>features</i> podem ser extraídas através das propriedades que descrevem os elementos intencionais. Se o elemento intencional for um recurso, o nome da <i>feature</i> será o mesmo nome do recurso. Se o elemento intencional for uma tarefa, o nome da <i>feature</i> poderá ser uma simplificação do nome da tarefa, caso seja necessário.
H10	Todas as linhas da tabela serão mapeadas para o modelo de <i>features</i> com o nome preenchido na coluna “ <i>Feature</i> ”. Caso duas ou mais <i>features</i> tenham o mesmo nome, as informações de todas elas serão unidas para se tornarem apenas uma <i>feature</i> .
H11	Todas as <i>features</i> da tabela serão relacionadas à <i>featu-</i>

	re raiz quando não tiverem o campo “Elemento Pai” preenchido.
H12	Todas as <i>features</i> da tabela serão <i>sub-features</i> quando tiverem a coluna “Elemento Pai” preenchida. Essas <i>sub-features</i> serão relacionadas à <i>feature</i> correspondente ao valor do campo “Elemento Pai”.

Tabela 5.4 – Mapeamento entre a cardinalidade nos modelos i* e nos modelos de *features*

Cardinalidade no modelo i*		Cardinalidade no modelo de <i>Features</i>	
Tipo	Valor	Notação	Explicação
Elemento	[0..1]		Uma <i>feature</i> opcional pode ou não estar presente em um produto concreto. A cardinalidade é [0..1].
Elemento	[1..1]		Uma <i>feature</i> obrigatória deve estar presente exatamente uma vez em um produto concreto. A cardinalidade é [1..1].
Elemento	[n..m]		Uma cardinalidade [n..m] indica que pelo menos n e no máximo m ocorrências de f2 devem estar presentes em um produto concreto.
Grupo	<1..1>		Uma cardinalidade de grupo <1..1> indica que exatamente uma <i>feature</i> pode ser selecionada do grupo.
Grupo	<1..k>		Uma cardinalidade de grupo <1..k> indica que pelo menos uma e no máximo k <i>features</i> podem ser selecionadas do grupo e que k é o número total de <i>features</i> no grupo.
Grupo	<j..k>		Uma cardinalidade de grupo é um intervalo inserido em um grupo de <i>features</i> que indica quantas <i>features</i> podem ser selecionadas do grupo em um produto concreto.

A partir de cada modelo obtido na atividade anterior, preenchemos uma tabela de acordo com o template da Tabela 4.3 definida no capítulo anterior. Os resultados encontram-se nas subseções seguintes.

5.4.1. Caso de Uso: Adicionar Foto

O modelo de *features* deste caso de uso foi modelado no capítulo anterior e visualizado na Figura 4.17.

5.4.2. Caso de Uso: Apagar Foto

A Tabela 5.5 é preenchida com informações para facilitar a construção do modelo de *features* do caso de uso **Apagar Foto**.

Tabela 5.5 – Informações dos elementos do caso de uso Apagar Foto que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Apagar Foto	Elemento	[1..1]	–	Apagar Foto
Listar Fotos Disponíveis	Elemento	[1..1]	Apagar Foto	Listar Fotos
Enviar Lixeira	Grupo	<1..2>	Apagar Foto	Lixeira
Remover Completamente	Grupo	<1..2>	Apagar Foto	Remover Completamente
Foto	Elemento	[1..1]	Apagar Foto	Foto
Album	Elemento	[0..1]	Apagar Foto	Album

A Figura 5.26 apresenta o modelo de *features* do caso de uso **Apagar Foto**, obtido a partir da Tabela 5.5.

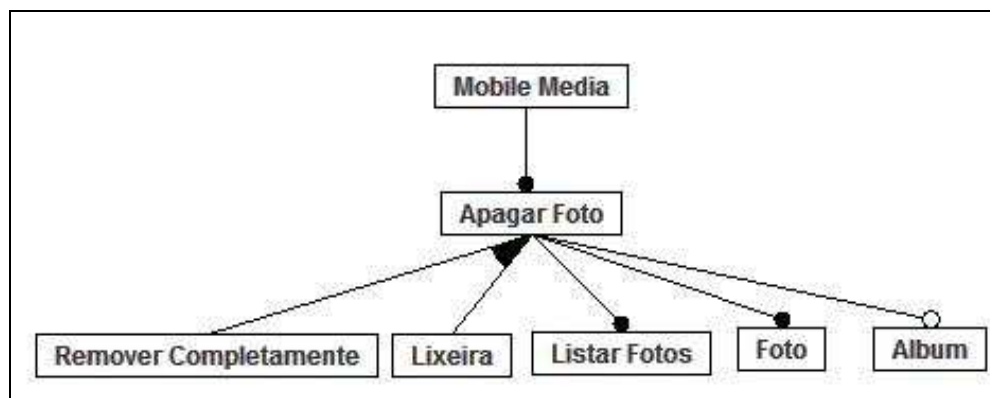


Figura 5.26 - Modelo de *Features* do caso de uso Apagar Foto

5.4.3. Caso de Uso: Visualizar Conteúdo

A Tabela 5.6 é preenchida com informações para facilitar a construção do modelo de *features* do caso de uso **Visualizar Conteúdo**.

Note que o elemento **Visualizar Conteúdo** será representado no modelo de *features* com o nome de **Visualizar Foto**, já que a palavra conteúdo se refere à foto durante a descrição do caso de uso.

Tabela 5.6 – Informações dos elementos do caso de uso Visualizar Conteúdo que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Visualizar Conteúdo	Elemento	[1..1]	–	Visualizar Foto
Disponibilizar Foto	Elemento	[1..1]	Visualizar Conteúdo	Disponibilizar Foto
Listar Fotos Disponíveis	Elemento	[1..1]	Visualizar Conteúdo	Listar Fotos
Album	Elemento	[0..1]	Visualizar Conteúdo	Album
Foto	Elemento	[1..1]	Disponibilizar Foto	Foto
Disponibilizar tamanho reduzido	Grupo	<1..1>	Disponibilizar Foto	Tamanho Reduzido
Disponibilizar Tamanho Original	Grupo	<1..1>	Disponibilizar Foto	Tamanho Original

A Figura 5.27 apresenta o modelo de *features* do caso de uso **Visualizar Conteúdo**, obtido a partir da Tabela 5.6.

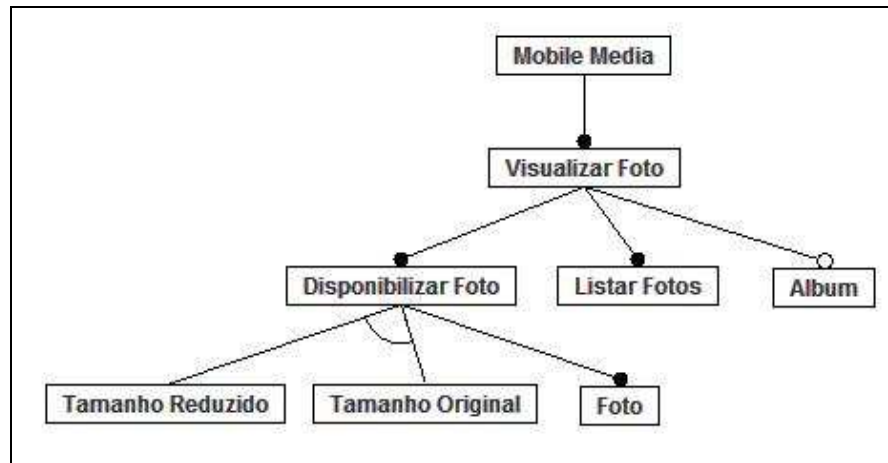


Figura 5.27 – Modelo de Features do caso de uso Visualizar Conteúdo

5.4.4. Caso de Uso: Adicionar Álbum

A Tabela 5.7 é preenchida com informações para facilitar a construção do modelo de *features* do caso de uso **Adicionar Álbum**.

Tabela 5.7 – Informações dos elementos do caso de uso Adicionar Álbum que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Adicionar Álbum	Elemento	[1..1]	–	Adicionar Álbum
Armazenar Álbum	Elemento	[1..1]	Adicionar Álbum	Armazenar Álbum
Fornecer Legenda	Elemento	[1..1]	Adicionar Álbum	Fornecer Legenda
Nome	Elemento	[1..1]	Fornecer Legenda	Nome
Salvar Automaticamente	Grupo	<1..1>	Armazenar Álbum	Salvar Automaticamente
Salvar pelo Usuário	Grupo	<1..1>	Armazenar Álbum	Salvar Pelo Usuário

A Figura 5.28 apresenta o modelo de *features* do caso de uso **Adicionar Álbum**, obtido a partir da Tabela 5.7.

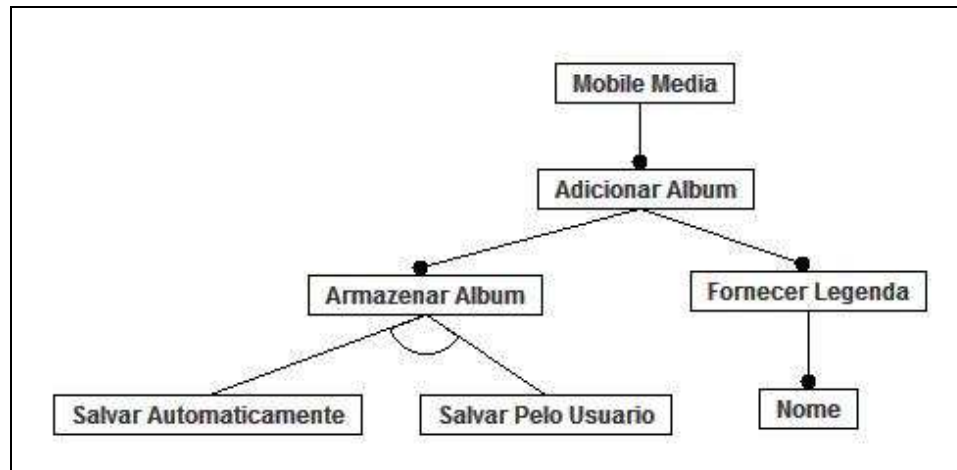


Figura 5.28 – Modelo de *Features* do caso de uso Adicionar Álbum

5.4.5. Caso de Uso: Apagar Álbum

A Tabela 5.8 contém informações para facilitar a construção do modelo de *features* do caso de uso **Apagar Álbum**.

Tabela 5.8 – Informações dos elementos do caso de uso Apagar Álbum que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Apagar Álbum	Elemento	[1..1]	–	Apagar Álbum
Listar Álbuns Existentes	Elemento	[1..1]	Apagar Álbum	Listar Álbuns
Álbum	Elemento	[1..1]	Apagar Álbum	Álbum
Remover Completamente	Grupo	<1..2>	Apagar Álbum	Remover Completamente
Enviar Lixeira	Grupo	<1..2>	Apagar Álbum	Lixeira
Lista de Álbuns	Elemento	[0..1]	Listar Álbuns Existentes	Lista de Álbuns

A Figura 5.29 apresenta o modelo de *features* do caso de uso **Apagar Álbum**, obtido a partir da Tabela 5.8.

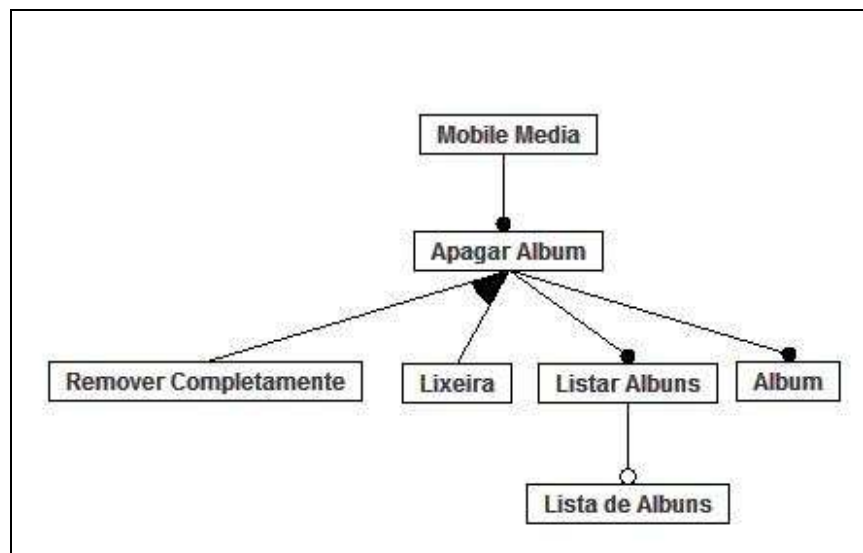


Figura 5.29 – Modelo de *Features* do caso de uso Apagar Álbum

5.4.6. Caso de Uso: Fornecer Legenda

Na Tabela 5.9 encontramos informações para facilitar a construção do modelo de *features* do caso de uso **Fornecer Legenda**.

Tabela 5.9 – Informações dos elementos do caso de uso Fornecer Legenda que estarão presentes no modelo de *features*

Elemento	Cardinalidade		Elemento Pai	Feature
	Tipo	Valor		
Fornecer Legenda	Elemento	[1..1]	–	Fornecer Legenda
Nomear Automaticamente	Grupo	<1..1>	Fornecer Legenda	Nomear Automaticamente
Nomear pelo Usuário	Grupo	<1..1>	Fornecer Legenda	Nomear Pelo Usuário
Nome	Elemento	[1..1]	Nomear pelo Usuário	Nome

A Figura 5.30 apresenta o modelo de *features* do caso de uso **Fornecer Legenda**, obtido a partir da Tabela 5.9.

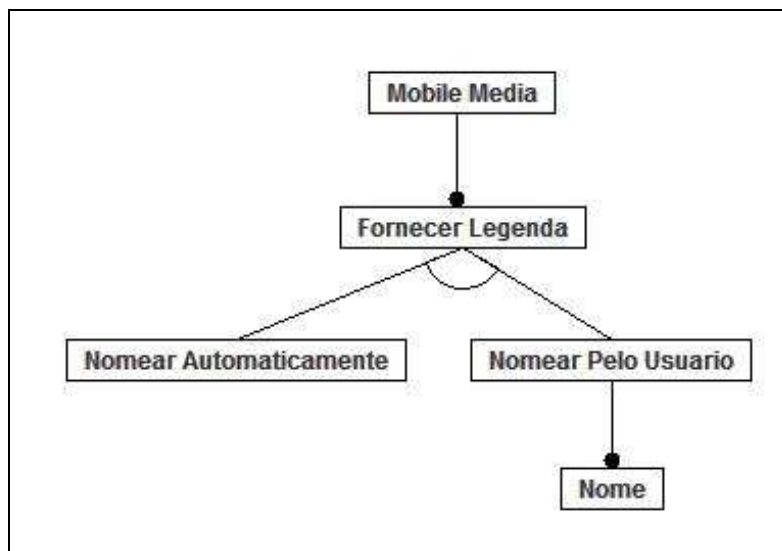


Figura 5.30 – Modelo de *Features* do caso de uso Fornecer Legenda

A Figura 5.31 apresenta o artefato de saída desta atividade: a unificação de todos os modelos de *features* deste capítulo.

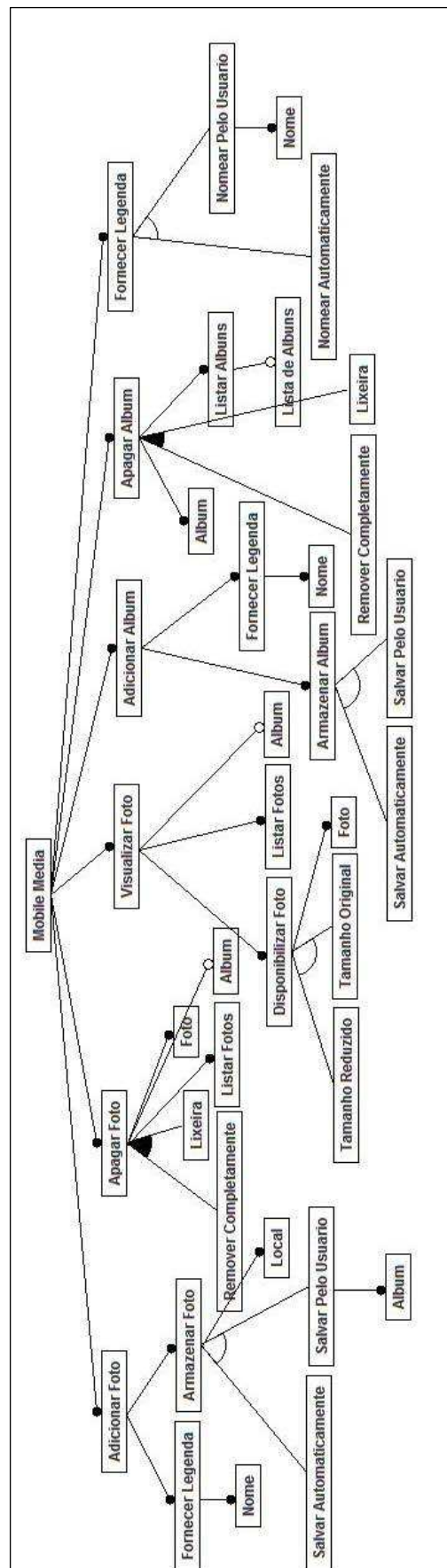


Figura 5.31 – Modelo de *Features* do sistema Mobile Media

5.5. Reorganização do Modelo de *Features* (opcional)

A quinta atividade do nosso processo é a reorganização do modelo de *features*, destacada na Figura 5.32. Vimos anteriormente que esta atividade é opcional, desde que o modelo de *features* obtido na atividade anterior não precise ser reorganizado.

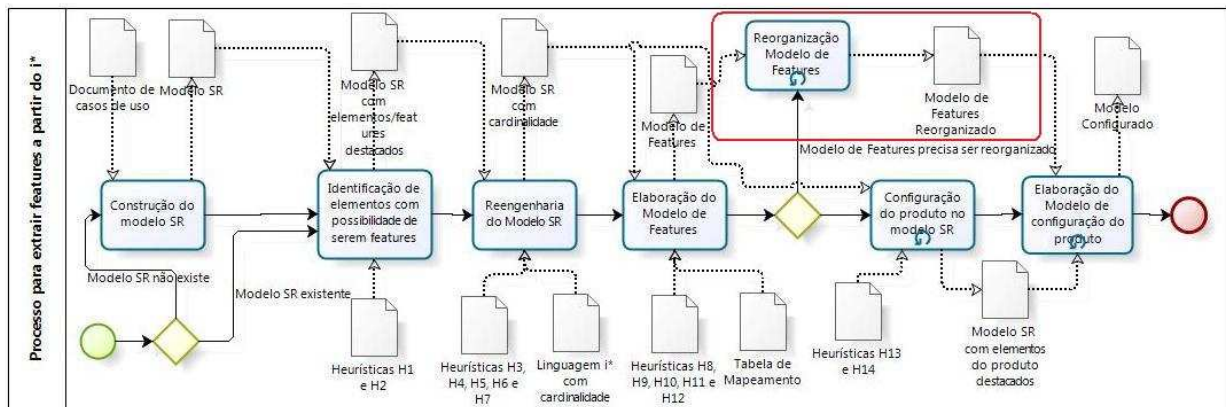


Figura 5.32 – Quinta atividade do processo G2FM: Reorganização do Modelo de *Features*

De acordo com o modelo de *features* obtido na atividade anterior (Figura 5.31), observamos que existem *features* repetidas, *features* diferentes, mas com a mesma semântica e *features* que podem ser agrupadas por fazerem referência ao mesmo conceito, tornando-se necessária a execução desta atividade.

Inicialmente, verificamos que há dois grupos de *features* nesse modelo: um com operações relacionadas à foto e outro com operações relacionadas a álbum (vide Figura 5.33).

Reorganizamos esse modelo para contemplar esses dois grupos de operações: criamos uma *feature* **Gerenciamento de Foto** e outra **Gerenciamento de Álbum**, e colocamos como *sub-features* as *features* relacionadas a foto e álbum, respectivamente. Com isso, removemos as *features* **Foto** que aparecem relacionadas às operações de foto e removemos as *features* **Álbum** que aparecem relacionadas às operações de álbum. (vide Figura 5.34).

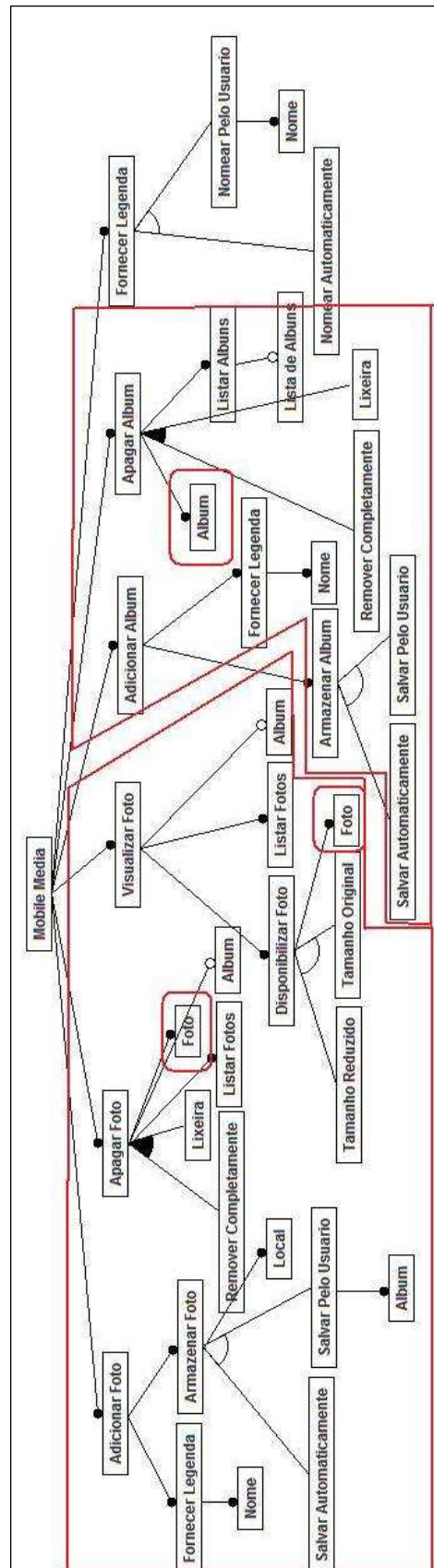


Figura 5.33 – Operações com Foto e Álbum destacadas

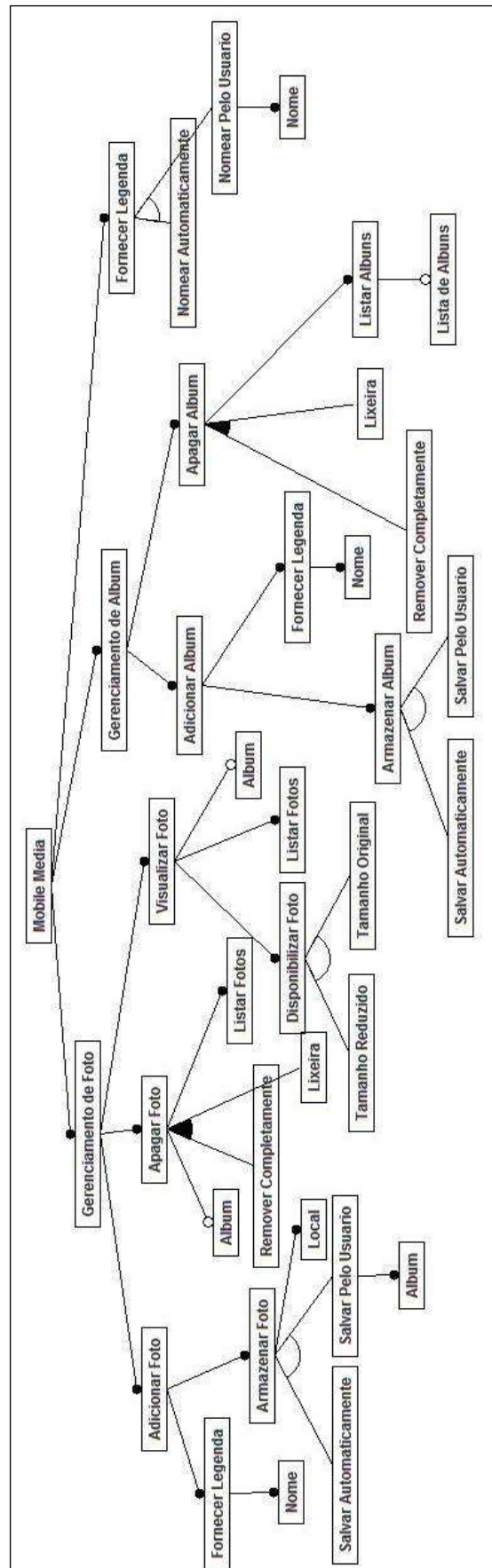


Figura 5.34 – Primeira reorganização do Modelo de *Features* do sistema Mobile Media

Continuando a análise do modelo de *features*, verificamos que a *feature* **Fornecer Legenda** aparece repetidas vezes: como *sub-feature* do grupo de operações **Gerenciamento de Foto**, como *sub-feature* do grupo de operações **Gerenciamento de Album** e como *sub-feature* da *feature* raiz, **Mobile Media**. Essa repetição é desnecessária já que em todas as ocorrências, ela tem o mesmo propósito e a mesma cardinalidade (vide Figura 5.35). Dessa forma, removemos essa *sub-feature* de **Adicionar Foto** e **Adicionar Álbum** (vide Figura 5.36).

Observando a Figura 5.36, verificamos que ainda existem *features* repetidas (**Salvar Automaticamente**, **Salvar Pelo Usuário**, **Album**, **Remover Completamente**, **Lixeira**, e **Listar Fotos**), porém não devem ser removidas, pois, fazem parte de uma determinada configuração do produto, podendo ser substituída por outras *features*, dependendo de outra configuração. Entretanto, existem *features* com nomes diferentes, mas com mesmo significado (vide Figura 5.37).

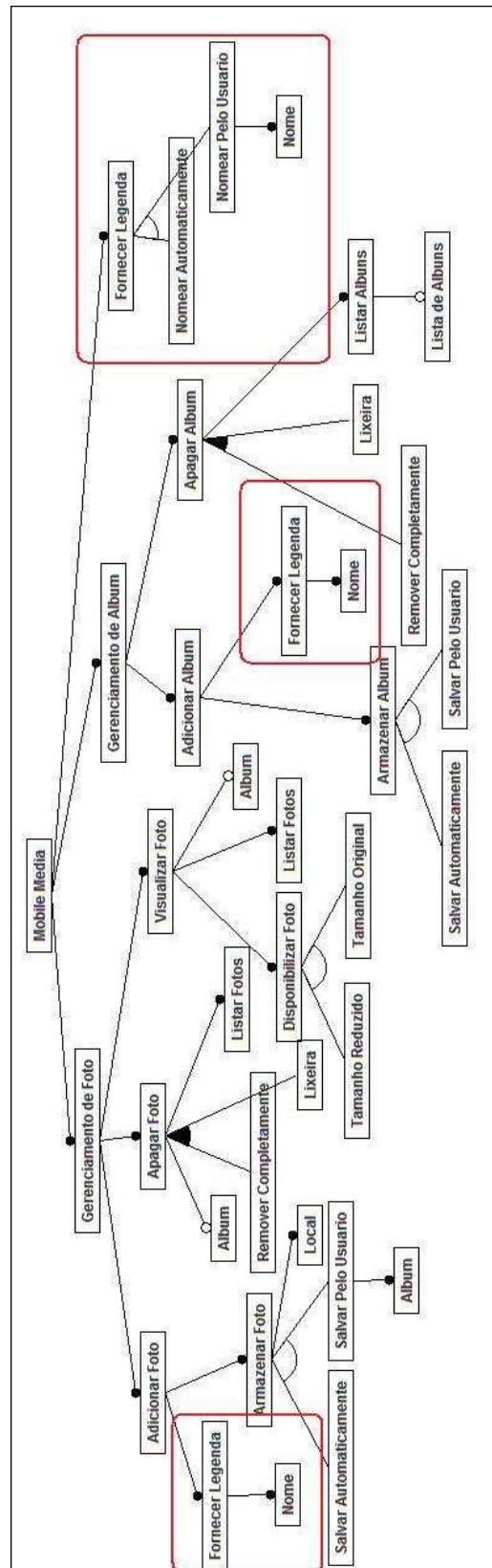


Figura 5.35 – Modelo de *Features* com a *feature* Fornecer Legenda destacada várias vezes

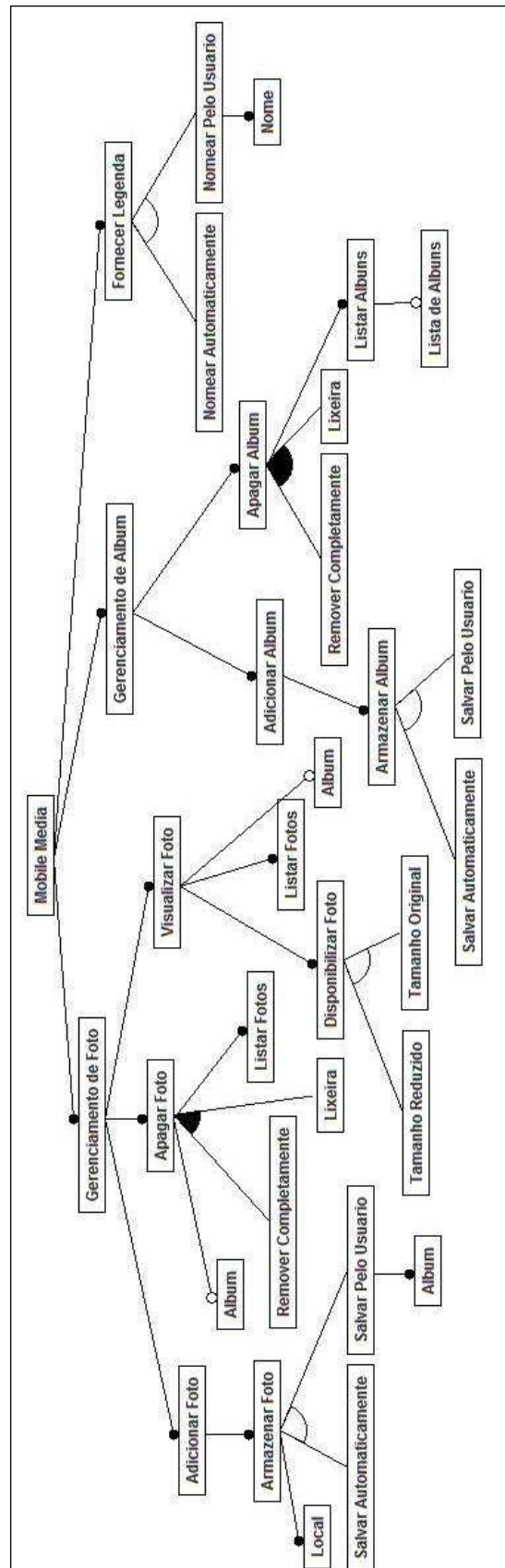


Figura 5.36 – Segunda reorganização do modelo de *features* do sistema Mobile Media

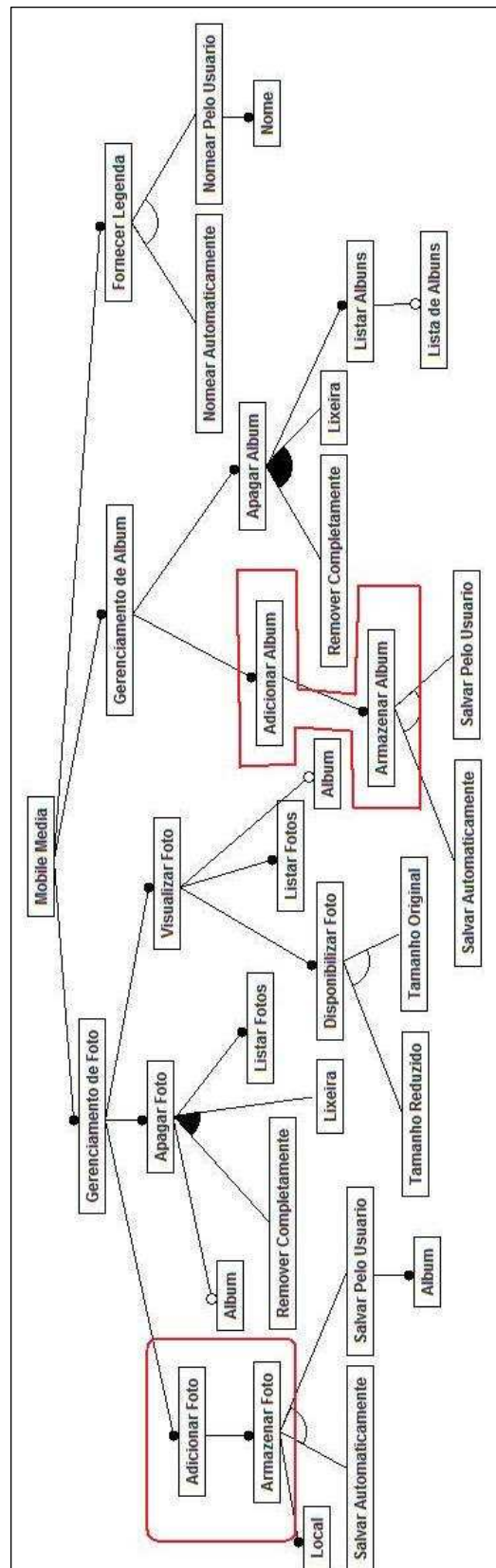


Figura 5.37 – Modelo de *Features* com *features* com mesmo significado destacadas

Como as *features* destacadas têm o mesmo significado, mantemos uma e removemos a outra, deixando o modelo de *features* mais enxuto, porém com a mesma semântica. A Figura 5.38 apresenta esta última reorganização e consequentemente, o modelo de *features* final reorganizado do **Mobile Media**.

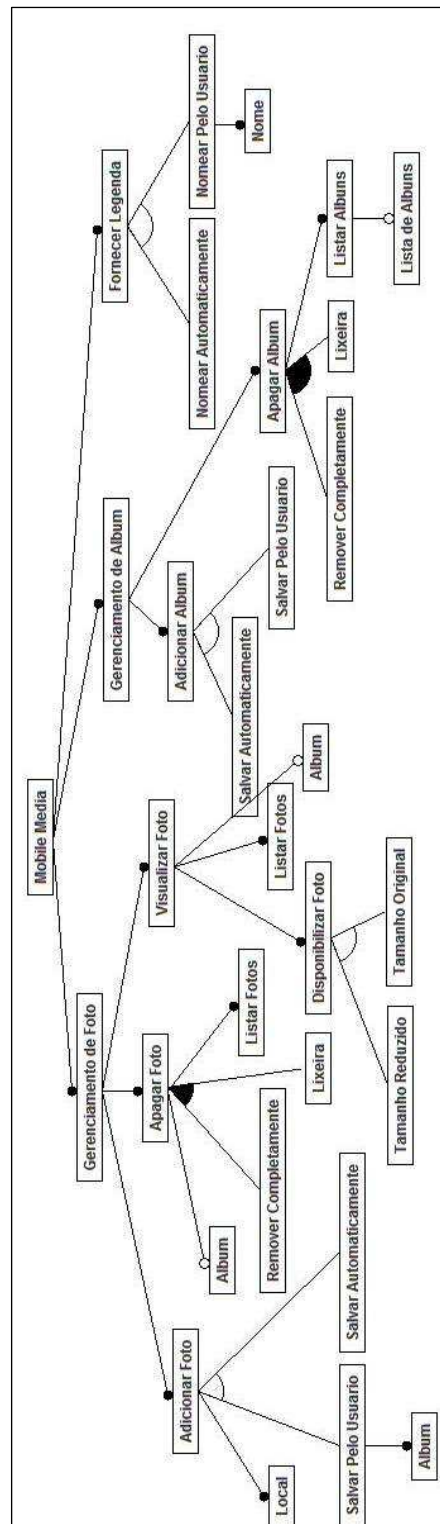


Figura 5.38 – Modelo de *Features* reorganizado do sistema Mobile Media

5.6. Configuração do produto no modelo SR

De acordo com a Figura 5.39, esta atividade tem como objetivo selecionar os elementos candidatos a *features* que farão parte de um produto específico, a partir de algumas heurísticas e do modelo SR com cardinalidade gerado na atividade de Reengenharia do Modelo SR.

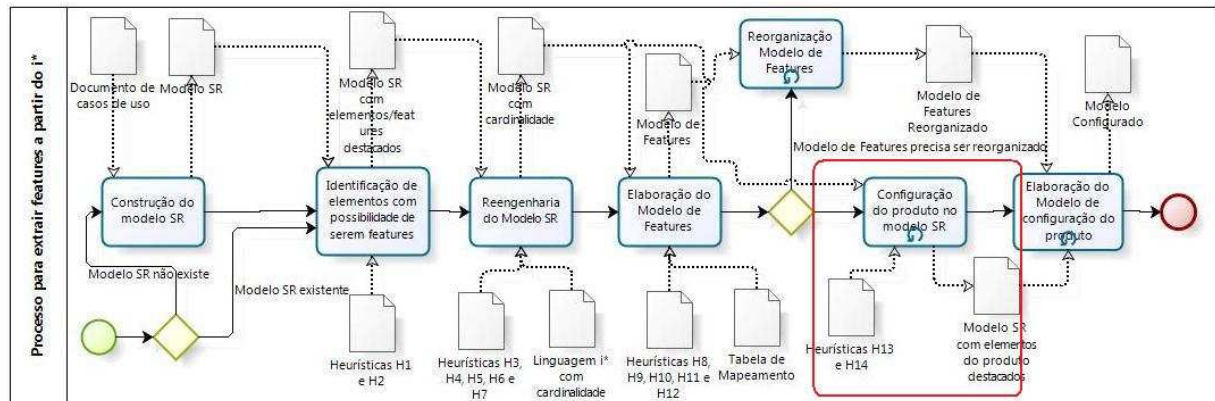


Figura 5.39 – Sexta atividade do processo G2FM: Configuração do produto no modelo SR

O resumo das heurísticas utilizadas nesta atividade encontra-se na Tabela 5.10.

Tabela 5.10 – Resumo das heurísticas H13 e H14

Heurística	Definição
H13	Todos os elementos intencionais que possuírem cardinalidade [1..1] ou [n..m], onde n seja maior ou igual a 1, deverão estar presentes no modelo de configuração do produto, desde que seus elementos pais também tenham sido selecionados. Os elementos com cardinalidade [0..1] poderão ou não ser selecionados, ficando à critério do <i>stakeholder</i> .
H14	Os sub-elementos relacionados com as ligações meio-fim que possuem cardinalidade deverão estar presentes no modelo de <i>features</i> de acordo com a cardinalidade específica.

Aplicamos as heurísticas (H13 e H14) nos modelos SR com cardinalidade produzidos na atividade de Reengenharia do Modelo SR para obter novos modelos SR com os elementos

que farão parte da configuração do produto. Os resultados encontram-se nas subseções seguintes.

5.6.1. Caso de Uso: Adicionar Foto

O modelo SR deste caso de uso com os elementos que farão parte da configuração do produto foi modelado no capítulo anterior e visualizado na Figura 4.19.

5.6.2. Caso de Uso: Apagar Foto

A Figura 5.40 apresenta o modelo SR do caso de uso **Apagar Foto** com os elementos, que farão parte da configuração do produto, destacados. Selecionamos a alternativa **Remover Completamente**, pois essa tarefa contribui positivamente em relação ao *softgoal* **Disponibilidade [Armazenamento]**.

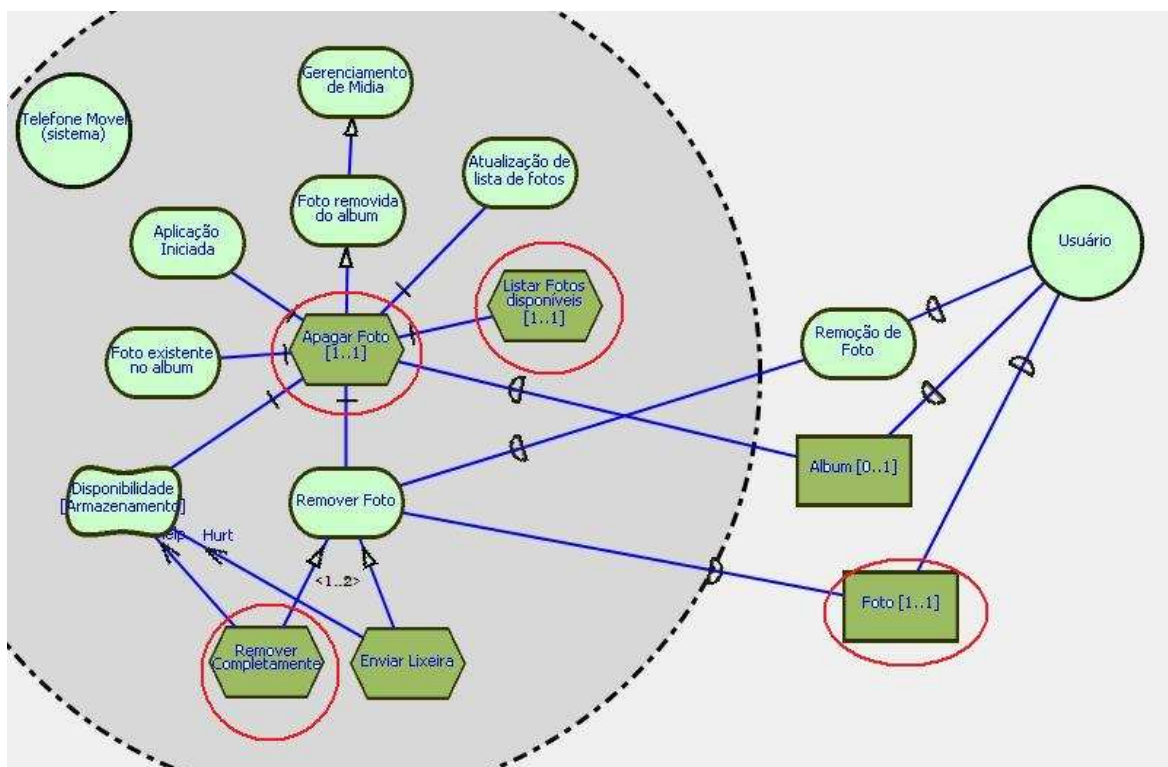


Figura 5.40 - Modelo SR do caso de uso Apagar Foto com elementos que estarão presentes na configuração do produto

5.6.3. Caso de Uso: Visualizar Conteúdo

A Figura 5.41 apresenta o modelo SR do caso de uso **Visualizar Conteúdo** com os elementos, que farão parte da configuração do produto, destacados. Optamos por selecionar a alternativa **Disponibilizar tamanho reduzido**, devido a essa tarefa ter uma influência positiva em relação ao *softgoal* **Desempenho [Sistema]**.

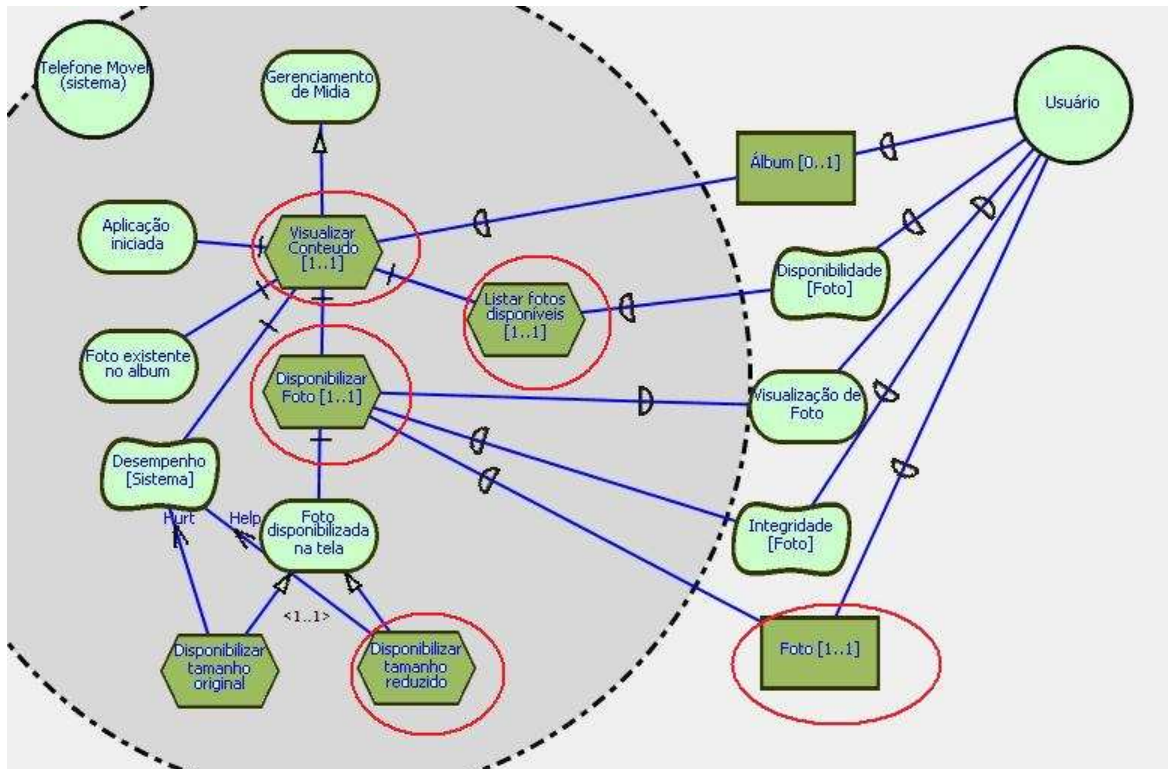


Figura 5.41 - Modelo SR do caso de uso Visualizar Conteúdo com elementos que estarão presentes na configuração do produto

5.6.4. Caso de Uso: Adicionar Álbum

A Figura 5.42 apresenta o modelo SR do caso de uso **Adicionar Álbum** com os elementos que farão parte do produto destacados. Selecionamos a alternativa **Salvar Automaticamente**, por ter uma influência positiva em relação ao *softgoal* **Rapidez [Armazenamento]**.

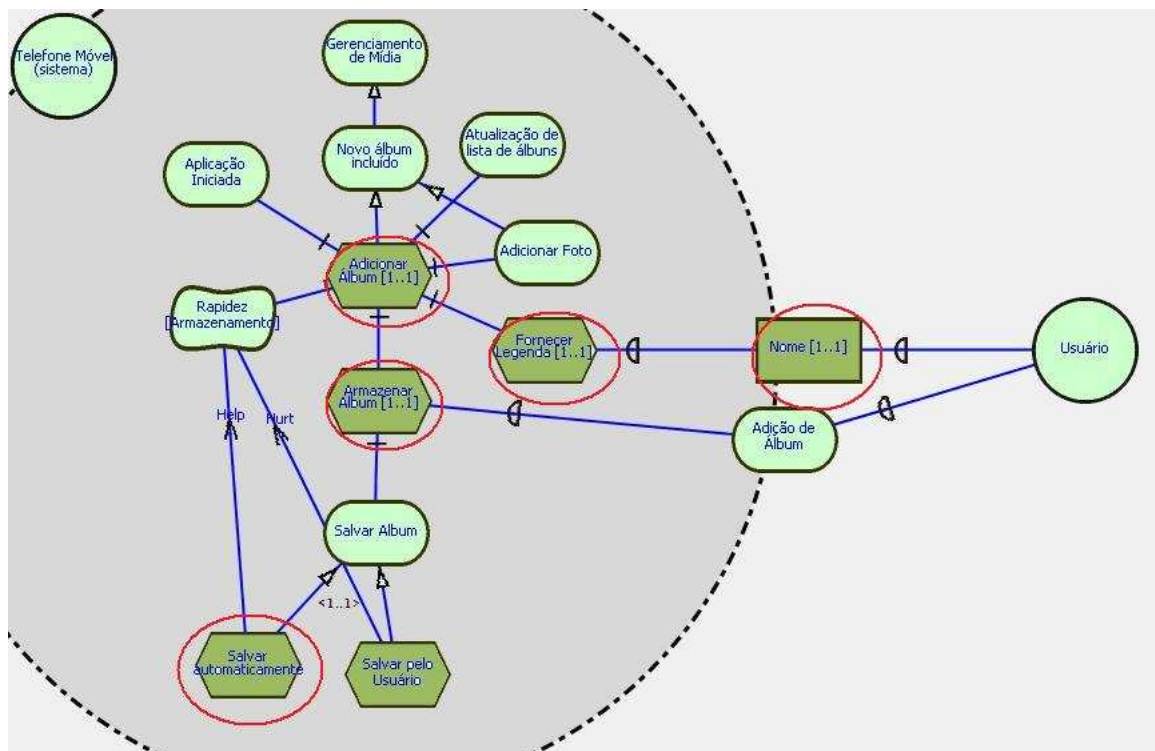


Figura 5.42 - Modelo SR do caso de uso Adicionar Álbum com elementos que estarão presentes na configuração do produto

5.6.5. Caso de Uso: Apagar Álbum

A Figura 5.43 apresenta o modelo SR do caso de uso **Apagar Álbum** com os elementos que farão parte da configuração do produto destacados. Selecionamos a alternativa **Remover Completamente**, por essa tarefa ter uma influência positiva em relação ao *softgoal* **Disponibilidade [Armazenamento]**.

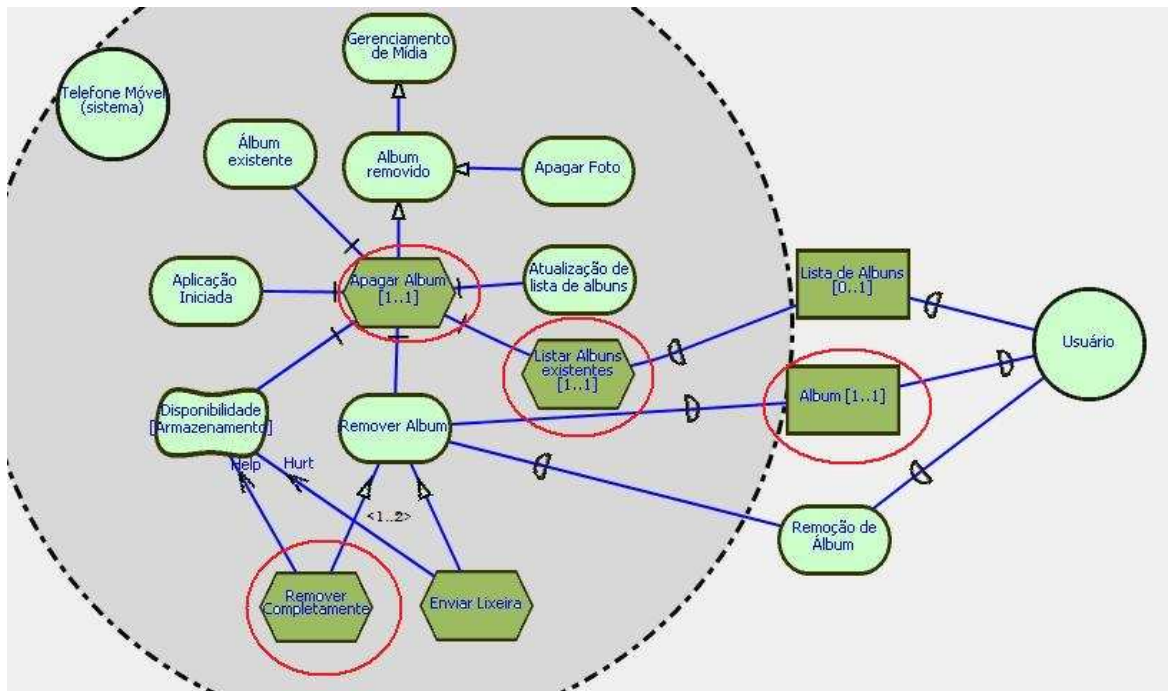


Figura 5.43 - Modelo SR do caso de uso Apagar Álbum com elementos que estarão presentes na configuração do produto

5.6.6. Caso de Uso: Fornecer Legenda

A Figura 5.44 contém o modelo SR do caso de uso **Fornecer Legenda** com os elementos que farão parte da configuração do produto destacados. Selecionamos a alternativa **Nomear pelo Usuário**, devido a essa tarefa influenciar positivamente o *softgoal* **Customização [Nome]**.

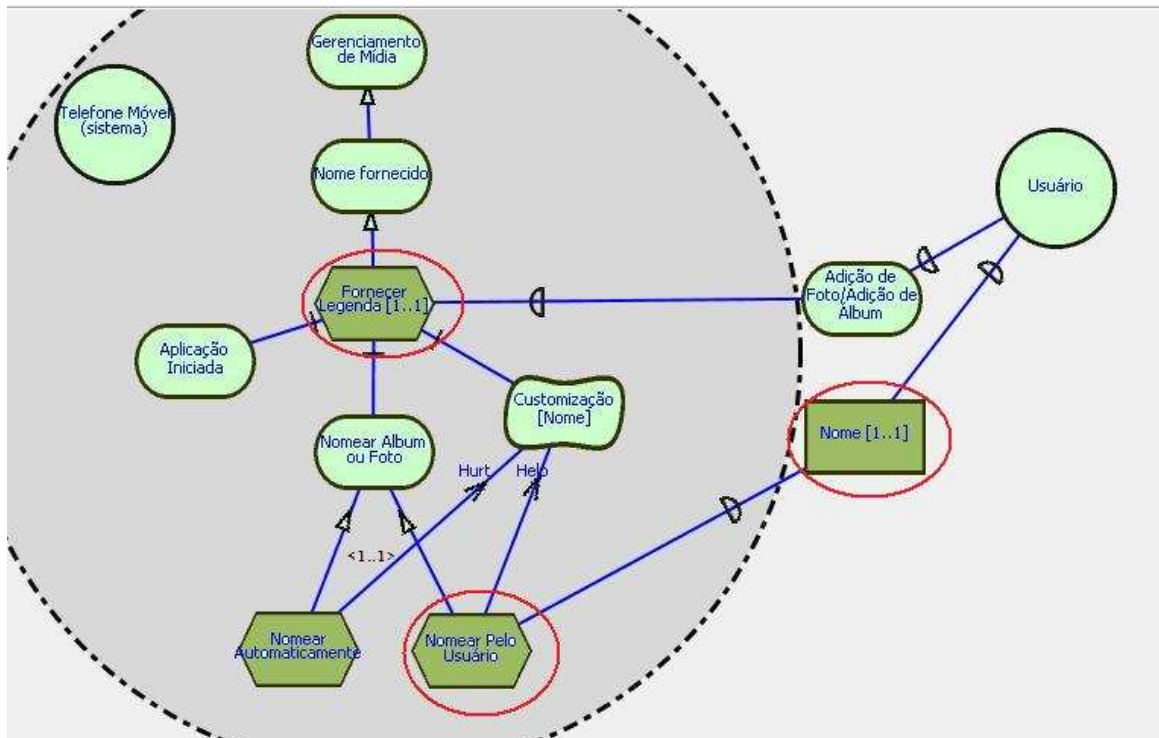


Figura 5.44 - Modelo SR do caso de uso Fornecer Legenda com elementos que estarão presentes na configuração do produto

5.7. Elaboração do Modelo de Configuração do Produto

A última atividade do nosso processo (vide Figura 5.45) tem como objetivo a construção do modelo de configuração de um produto específico, a partir do modelo SR que contém os elementos que farão parte do produto destacados (gerado na atividade anterior) e do modelo de *features* reorganizado (caso a atividade Reorganização do Modelo de *Features* tenha sido executada).

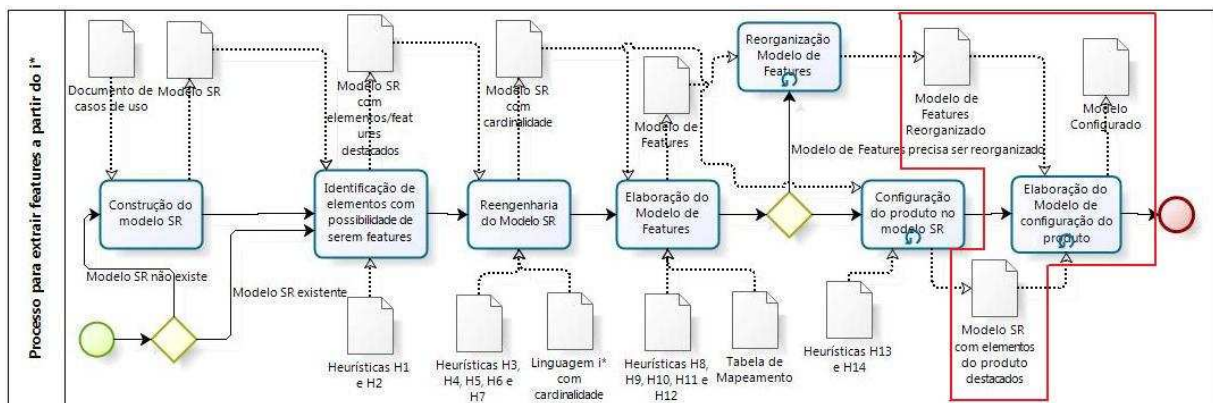


Figura 5.45 – Última atividade do processo G2FM: Elaboração do Modelo de Configuração do produto

Analisando os modelos SR gerados na atividade anterior (Figura 4.19 do caso de uso **Adicionar Foto**, Figura 5.40 do caso de uso **Apagar Foto**, Figura 5.41 do caso de uso **Visualizar Conteúdo**, Figura 5.42 do caso de uso **Adicionar Álbum**, Figura 5.43 do caso de uso **Apagar Álbum** e Figura 5.44 do caso de uso **Fornecer Legenda**) e comparando os elementos destacados de todos esses modelos SR com o modelo de *features* da Figura 5.38, geramos o seguinte modelo de configuração válido (conjunto de nomes) para a *release* 1 do sistema **Mobile Media**:

Modelo Configuração: {Mobile Media, Gerenciamento de Foto, Adicionar Foto, Salvar Automaticamente, Local, Apagar Foto, Remover Completamente, Listar Fotos, Visualizar Foto, Disponibilizar Foto, Tamanho Reduzido, Gerenciamento de Álbum, Adicionar Álbum, Apagar Álbum, Listar Álbuns, Fornecer Legenda, Nomear pelo Usuário, Nome}.

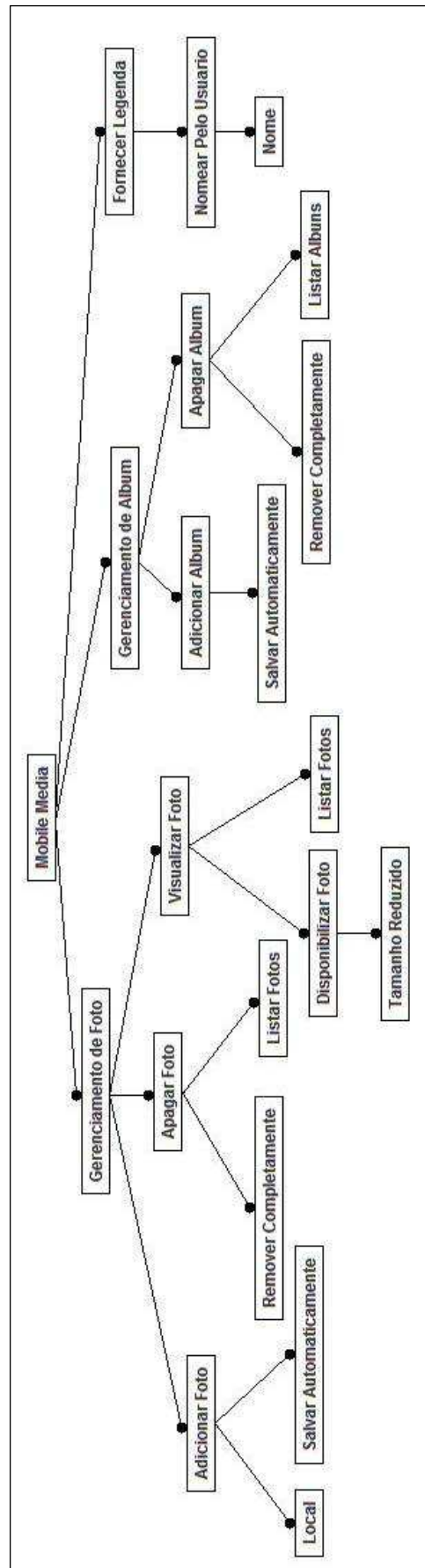


Figura 5.46 – Modelo de Configuração de um produto do MobileMedia

Vimos anteriormente que o modelo de *features* pode ser usado para configurar vários produtos diferentes, caso na atividade de **Configuração do produto no modelo SR**, outros elementos sejam selecionados como preferências dos *stakeholders* com relação aos requisitos não-funcionais. Por exemplo: (i) caso o elemento **Salvar Pelo Usuário** das Figuras 4.19 e 5.42 fosse selecionado no lugar do elemento **Salvar Automaticamente**, (ii) caso o elemento **Enviar Lixeira** das Figuras 5.40 e 5.43 fosse selecionado no lugar do elemento **Remover Completamente**, (iii) caso o elemento **Disponibilizar Tamanho Original** da Figura 5.41 fosse selecionado no lugar do elemento **Disponibilizar Tamanho Reduzido** e (iv) caso o elemento **Nomear Automaticamente** da Figura 5.44 fosse selecionado no lugar do elemento **Nomear pelo Usuário** teríamos um outro modelo de configuração válido para a *release* 1 do sistema **Mobile Media**:

Modelo Configuração 2: {Mobile Media, Gerenciamento de Foto, Adicionar Foto, Salvar pelo Usuário, Album, Local, Apagar Foto, Lixeira, Listar Fotos, Visualizar Foto, Disponibilizar Foto, Tamanho Original, Gerenciamento de Álbum, Adicionar Álbum, Apagar Álbum, Listar Álbuns, Fornecer Legenda, Nomear Automaticamente}.

5.8. Discussão

Comparando o modelo de *features* final obtido durante o estudo de caso deste capítulo (Figura 5.38) e o modelo de *features* da *release* 1 do sistema **MobileMedia** original (Figura 5.47 abaixo), concluímos que a Figura 5.38 contém uma quantidade bem maior de *features*. Isso significa que a utilização da abordagem **G2FM** para a identificação e modelagem de *features* permite a representação de modelos mais significativos e detalhados.

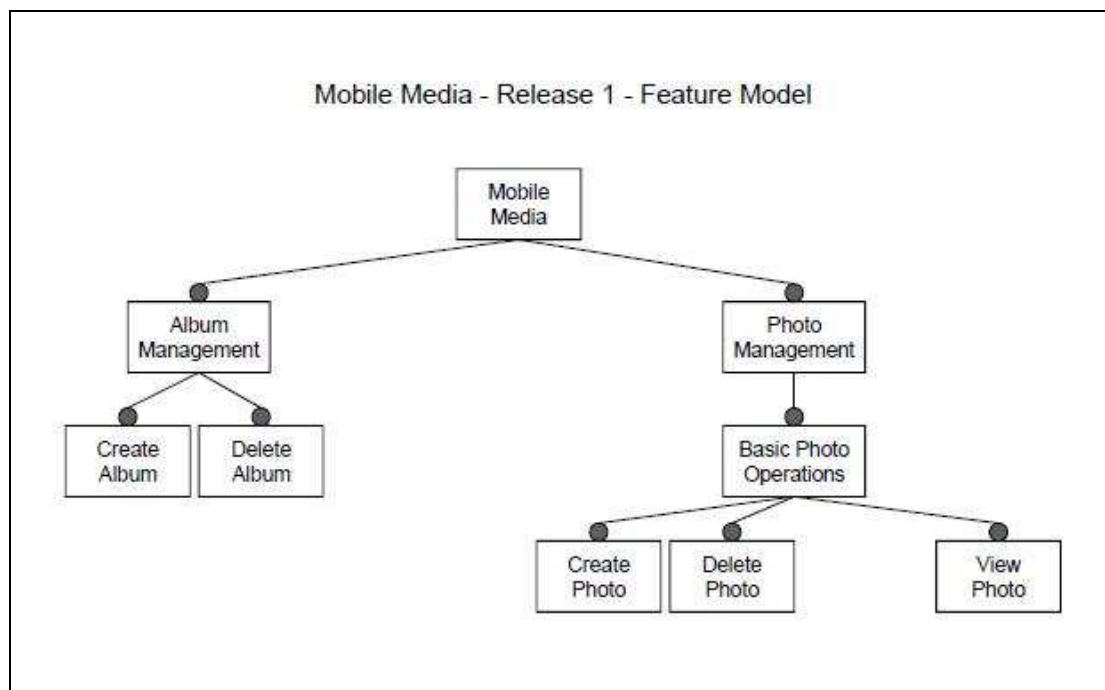


Figura 5.47 – Modelo de *Features* da *release 1* do sistema **MobileMedia** original

5.9. Considerações Finais

Neste capítulo, apresentamos um estudo de caso completo do sistema **MobileMedia**, o que nos possibilitou a utilização do processo **G2FM**, mostrando detalhadamente as heurísticas definidas e os artefatos de saída. Utilizamos como exemplo os casos de uso: **Adicionar Foto**, **Apagar Foto**, **Visualizar Conteúdo**, **Adicionar Álbum**, **Apagar Álbum** e **Fornecer Legenda** da *release 1* do sistema **MobileMedia**. Posteriormente, mostramos uma comparação entre o modelo de *features* obtido como resultado deste estudo de caso e o modelo de *features* do sistema **MobileMedia** original.

O capítulo seguinte apresentará as conclusões e trabalhos futuros desta dissertação de mestrado.

Capítulo

6. Conclusão

Este capítulo apresenta um resumo de tudo que foi realizado ao longo deste trabalho, salientando as contribuições da abordagem **G2FM** e sugerindo alguns tópicos para trabalhos futuros.

Nesta dissertação, primeiramente discutimos o estado da arte das duas principais subáreas envolvidas neste trabalho: a Engenharia de Linhas de Produto de Software (ELPS) e a Engenharia de Requisitos Orientada a Objetivos (ou *Goal-Oriented Requirements Engineering* – GORE), onde destacamos o modelo de *features* e o *framework* *i**, respectivamente, por terem sido utilizados durante a realização de todo este trabalho. Em seguida, apresentamos uma comparação da expressividade entre abordagens orientadas a objetivos (modelos de objetivos, PL-AOVGraph, *i** Aspectual e *i**) e um modelo de *features*. Com o resultado desta comparação, concluiu-se que essas três abordagens têm a expressividade limitada, pois não são capazes de modelar a representação da cardinalidade do modelo de *features*. Essa limitação nos motivou a estender o *i** para suportar cardinalidade, definindo a linguagem ***i*-c*** (*i* with cardinality*).

Na sequência, apresentamos em detalhes um processo que descreve a abordagem **G2FM** (*Goal To Feature Model*). Esta abordagem possibilitou a identificação de *features* nos modelos SR do *i** para a construção do modelo de *features* da LPS, bem como a configuração de aplicações específicas a partir de modelos *i**. Por fim, realizamos um estudo de caso completo do sistema **MobileMedia**, no qual utilizou-se o processo **G2FM**, mostrando detalhadamente as heurísticas definidas e os artefatos de saída.

6.1. Contribuições

Esta dissertação apresentou uma abordagem orientada a objetivos (**G2FM – Goal To Feature Model**) que permite tanto a identificação e modelagem de *features* comuns e variáveis em Linhas de Produtos de Software, como a configuração de uma aplicação específica de uma família de software. Portanto, o uso de uma abordagem de Engenharia de Requisitos Orientada a Objetivos tal como o *framework* *i** trouxe vários benefícios para a Engenharia de Linhas de Produto de Software, como por exemplo: rastreamento entre as *features* do sistema e suas motivações, bem como justificativa da configuração de alternativas do sistema para realização dos objetivos dos *stakeholders*.

Nesta dissertação, também foi proposta a linguagem ***i*-c*** (*i* with cardinality*), uma extensão do *framework* *i** com cardinalidade. A inserção de cardinalidade nos modelos *i** permite representar a variabilidade de uma linha de produtos com boa expressividade,

tornando possível a identificação de *features* comuns e variáveis e o relacionamento entre elas, a partir de modelos orientados a objetivos.

Além da abordagem **G2FM** suportar cardinalidade do modelo de *features*, outra contribuição desta abordagem é manter o modelo *i** com suas informações originais, ajudando na escolha e justificativa de uma configuração para uma determinada aplicação.

A utilização dos requisitos não-funcionais com relação aos relacionamentos de contribuição, auxiliam na atividade de configuração de um produto específico, reduzindo o tempo e os custos associados a esta atividade.

Outra contribuição da abordagem **G2FM** é a adaptação do método PriM (*Process Reengineering i* Method*) [Grau, G. et al., 2008], que pode ser usado caso não exista um modelo SR do *i** pronto e torna a construção destes modelos mais simples, sistemáticos e flexíveis, além de mais focados em aplicações para LPS.

6.2. Trabalhos Futuros

Apesar de termos conseguido realizar o objetivo inicialmente proposto de estabelecer uma abordagem que permitisse a identificação e modelagem de *features* comuns e variáveis em LPS, bem como a configuração de aplicações específicas a partir de modelos *i**, ainda existe muito trabalho a ser explorado nessa área de Engenharia de Requisitos para LPS. Como trabalhos futuros, podemos listar:

- Aplicar a abordagem **G2FM** nas outras *releases* do documento de casos de uso do **MobileMedia**;
- Investigar como o modelo de configuração de um produto do **MobileMedia** obtido a partir da abordagem **G2FM**, pode ser utilizado com o sistema **MobileMedia** já implementado;
- Realizar outros estudos de caso com a abordagem **G2FM**;
- Validar o modelo de *features* obtido com a abordagem **G2FM** com o obtido por outra técnica sem a abordagem GORE;
- Desenvolver uma ferramenta que suporte a abordagem **G2FM**;

- Gerar arquivos de *builds* de produtos a partir de modelos de configuração, obtidos com a utilização da abordagem **G2FM**. Para a geração desses arquivos de *builds*, podem-se utilizar ferramentas já existentes, como por exemplo, FLiP [Borba, P. et al., 2008];
- Definir e aplicar regras de transformação nos modelos de *features* para melhorar a organização dos mesmos;
- Investigar como capturar *constraints* do modelo de *features* a partir dos modelos *i**;
- Estudar como as redundâncias dos modelos de *features* possam ser resolvidas ainda nos modelos **i*-c** da abordagem **G2FM**;
- Verificar a possibilidade do uso das fases iniciais da abordagem **G2FM** como guia para a construção de modelos *i**.

Referências

[Alencar, F. et al., 2008]	Alencar, F. et al., 2008, F., Castro, J., Moreira, A., Araujo, J., Silva, C., Ramos, R., Mylopoulos, J.: Integration of Aspects with i* Models, Agent-Oriented Information Systems IV, LNCS, Vol. 4898, Springer-Verlag, 2008, pp. 183-201.
[António, S., 2009]	António, S. I. A. Adaptação da Framework i* para Linhas de Produtos. Dissertação (Mestrado) – Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2009.
[Batista, T. et al., 2008]	Batista, T., Bastarrica, M., Soares, S., Fernandes, L.: A Marriage of MDD and Early Aspects in Software Product Line Development. In: Early Aspects Workshop at 12th International Software Product Line Conference (LPSC'08), pp. 97-104, Limerick, Ireland, 2008.
[Bell, T. E. and Thayer, T. A., 1976]	Bell, T. E., Thayer, T. A. Software Requirements: Are They Really a Problem?. Proceedings on 2nd International Conference on Software Engineering, San Francisco, 1976.
[Benavides, D. et al., 2005]	Benavides, D., Trujillo, S., Trinidad, P.: On the modularization of feature models. In: 1st European Workshop on Model Transformation. Rennes, France, 2005.
[Bezivin, J., 2005]	Bezivin, J. On the unification power of models. Software and System Modeling, v. 4, n. 2, p. 171-188, 2005.
[BPMN, 2008]	Business Process Modeling Notation, V1.1. OMG Available Specification, 2008. Disponível em: < http://www.omg.org/spec/BPMN/1.1/PDF/ >. Último acesso: Julho de 2009.
[BizAgi, 2009]	BizAgi Process Modeler. Disponível em:

	< http://www.bizagi.com/esp/productos/ba-modeler/modeler.html >. Último acesso em: Junho de 2009.
[Boehm, B., 1981]	Boehm, B. Software Engineering Economics. Prentice Hall, 1981.
[Borba, C. and Silva, C., 2009]	Borba, C., Silva, C. A comparison of goal-oriented approaches to model software product lines variability. A ser publicado no Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGIM) In conjunction with the 28th International Conference on Conceptual Modeling (ER 2009), Gramado, Brazil, November 2009.
[Borba, P. et al., 2008]	Borba, P., Calheiros, F., Nepomuceno, V., Menezes, A., Soares, S., Alves, V. Flip - product line derivation tool. In Forum Demonstration at Seventh International Conference on Aspect-Oriented Software Development, AOSD.08, Brussels, Belgium, April 2008. ACM.
[Chung, L. K. et al., 2000]	Chung, L. K.; Nixon, B. A.; Yu, E.; Mylopoulos, J. Non-Functional Requirements in Software Engineering. [S.l.]: Kluwer Publishing, 2000. (International Series in Software Engineering, v. 5).
[Clements, P. and Northrop, L., 2002]	Clements, P., Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.
[Czarnecki, K. and Eisenecker, U., 2000]	Czarnecki, K., Eisenecker, U. W. Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co. 2000.
[Czarnecki, K. et al., 2005a]	Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process Improvement and Practice, 10(1):7-29, 2005.
[Czarnecki, K. et al., 2005b]	Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice,

	10(2): 143-169, 2005.
[Davis, A. M., 1993]	Davis, A. M. Software Requirements: Objects, Functions and States. New Jersey,USA: Prentice Hall, 1993.
[ESI, 1996]	European Software Institute, European User Survey Analysis. Report USV_EUR 2.1 ESPITI Project, 1996.
[Estrada, H., 2008]	Estrada, H.: A service-oriented architecture for the i* Framework. Ph.D. Thesis. DSIC. Universidad Politècnica de Valencia, 2008.
[Figueiredo, E. et al., 2008]	Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F. C., Dantas, F. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In Proceedings of the 30th international conference on Software engineering (ICSE 2008), Leipzig, Germany, 2008. pp 261-270.
[Gheyi, R. et al., 2008]	Gheyi, R., Massoni, T., Borba, P. Algebraic laws for feature models. Journal of Universal Computer Science (JUCS), 14:3573-3591, 2008.
[Grau, G. et al., 2008]	Grau, G., Franch, X., Maiden, N. PRiM: An i*-based process reengineering method for information systems specification. Information and Software Technology 50 (2008) 76-100.
[Grau, G. et al., 2009]	Grau, G., Yu, E., Horkoff, J., Abdulhadi, S.: i* Guide. Disponível em: http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67 , 2009. Último acesso em: 9 de Julho de 2009.
[Kang, K. et al., 1990]	Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Technical report, CMU/SEI-90-TR-021, 1990.
[Kang, K. et al., 1998]	Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M. FORM: A feature-oriented reuse method with domain-specific refer-

	ence architectures. In J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, pp. 143-168, 1998.
[Kotonya, G. and Sommerville, I., 1998]	Kotonya, G., Sommerville, I. Requirements Engineering: Processes and Techniques. John Wiley & Sons, Inc. New York, NY, USA, 1998. ISBN 0471972088
[Lamsweerde, A., 2000]	Lamsweerde, A. Requirements Engineering in the Year 00: A Research Perspective. 22nd Proceedings of International Conference on Software Engineering, Limerick, Ireland, 2000.
[Lamsweerde, A., 2001]	Lamsweerde, A. Goal-Oriented Requirements Engineering: A Guided Tour. RE'01, 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, 2001.
[Lamsweerde, A., 2003]	Lamsweerde, A. KAOS Tutorial. Cediti, 2003.
[Lapouchnian, A., 2005]	Lapouchnian, A. Goal-Oriented Requirements Engineering: An Overview of the Current Research. University of Toronto, Canada, 2005.
[Lucena, M. et al., 2008]	Lucena, M.; Santos, E., Silva, C.; Alencar, F. et al., 2008, F.; Silva, M. J.; Castro, J. Towards a unified metamodel for i*. In: Proceeding of 2nd IEEE Research Challenges in Information Science - RCIS 2008.
[OMG, 2003]	OMG, UML 2.0 OCL Specification, 2003. Disponível em: http://www.omg.org/docs/ptc/03-10-14.pdf .
[Pohl, K. et al., 2005]	Pohl, K., Böckle, G., van der Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.
[Procad, 2009]	Projeto Procad (Programa Nacional de Cooperação Acadêmica), Desenvolvimento de Linhas de Produtos de Software usando Técnicas Orientadas a Aspectos. Disponível em: < http://twiki.cin.ufpe.br/twiki/bin/view/ProjetoProcad/WebHome >. Último acesso: Junho de 2009.

[Rashid, A. et al., 2003]	Rashid, A., Moreira, A., Araújo, J.: Modularisation and Composition of Aspectual Requirements. In: 2nd Intl. Conf. on Aspect-Oriented Soft. Develop., USA, pp. 11-20, 2003.
[Santander, V. F. A., 2002]	Santander, V. F. A. Integrando Modelagem Organizacional com Modelagem Funcional. Tese (Doutorado), Centro de Informática, Universidade Federal de Pernambuco, 2002.
[Silva, L. and Leite, J., 2005]	Silva, L. F., Leite, J. C. S. P.: Integração de Características Transversais Durante a Modelagem de Requisitos, Simpósio Brasileiro de Engenharia de Software (SBES-2005) Uberlândia-MG, 2005.
[Silva, C. et al., 2008]	Silva, C., Alencar, F. et al., 2008, F., Araújo, J., Moreira, A., Castro, J.: Tailoring an Aspectual Goal-oriented Approach to Model Features. In: 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08), pp. 472-477, Knowledge Systems Institute Graduate School, San Francisco, CA, USA, 2008.
[STA, 1995]	The Standish Group, Software Chaos. http://www.standishgroup.com/chaos.html , 1995.
[Travassos, G. et al., 2002]	Travassos, G., Gurov, D., Amaral, E.: Introdução à Engenharia de Software Experimental. Relatório Técnico, RTES-590/02, Rio de Janeiro, 2002.
[Wohlin, C. et al., 2000]	Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, USA, 2000.
[Young, T., 2005]	Young, T. Using AspectJ to Build a Software Product Line for Mobile Devices. Dissertação Mestrado. University of British Columbia, 2005.
[Yu, E., 1995]	Yu, E. Modelling Strategic Relationships for Business Process Reengineering. Tese (Doutorado), Dept. of Computer Science, University of Toronto, 1995.

[Yu, E., 1997]	Yu, E. S. K. Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE 97). Annapolis, MD, USA: IEEE Computer Society, 1997. p. 226-235.
[Yu, Y. et al., 2004]	Yu, Y.; Leite, J.; Mylopoulos, J. From goals to aspects: discovering aspects from requirements goal models. In: Proceedings of 12th IEEE International Requirements Engineering Conference, 2004. Kyoto, Japan: IEEE Computer Society, 2004. p. 38-47.
[Yu, Y. et al., 2008]	Yu, Y., Leite, J.C.S.P., Lapouchnian, A., Mylopoulos, J. Configuring features with stakeholder goals. Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), ACM, Fortaleza, Ceara, Brazil, pp. 645-649, 2008
[Zave, P., 1997]	Zave, P. Classification of research efforts in requirements engineering. In: ACM Computing Surveys. New York, NY, USA: ACM Press, 1997. v. 29, n. 4, p. 315-321.