G2SPL: Um Processo de Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software

Carla Silva
Centro de Ciências Aplicadas e Educação
Universidade Federal da Paraíba
Rio Tinto, Brasil
ctaciana@ccae.ufpb.br

Clarissa Borba, Jaelson Castro
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
ccb@cin.ufpe.br, jbc@cin.ufpe.br

***Abstract*— Abordagens orientadas a objetivos podem ser usadas como uma forma de descobrir requisitos variáveis e comuns de uma Linha de Produtos de Software (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 a *framework* i*. O i* fornece uma maneira de identificar e especificar tanto os objetivos dos *stakeholders* com relação ao sistema pretendido, como as características do próprio sistema. Este trabalho propõe uma extensão da linguagem de modelagem do i*, chamada** i*-c **(*i* with cardinality*), que permite a inserção de cardinalidade em alguns de seus elementos de modelagem. A abordagem** G2SPL **(*Goals to Software Product Line*) propõe um processo para identificar e modelar *features* comuns e variáveis de uma LPS em modelos i* com cardinalidade, como também para guiar a configuração de um produto específico na LPS.**

Keywords-Goal Oriented Requirements Engineering, Software Product Lines

I. INTRODUÇÃO

A Engenharia de Requisitos (ER) 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 [1]. A Engenharia de Requisitos Orientada a Objetivos (do inglês, *Goal-Oriented Requirements Enginering* ou GORE) tem crescido como uma forma promissora de descrever sistemas de software. Ela baseia-se nos objetivos dos *stakeholders*, com a finalidade de desenvolver o software que realmente satisfaça os desejos dos *stakeholders* [2]. Por objetivo se entende uma condição ou estado do mundo que um *stakeholder* gostaria de alcançar [3].

A ER tanto pode ser aplicada para desenvolver produtos de software únicos, como para desenvolver Linhas de Produtos de Software (LPS) [4]. Uma Linha de Produtos de Software, também chamada de família de produtos, 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 [5]. Uma *feature* pode ser vista como uma propriedade de sistema ou funcionalidade que é relevante para alguns *stakeholders* e usada para capturar *features* comuns e variáveis entre produtos de uma mesma família [6].

Na Engenharia de Requisitos para LPS, os modelos de *features* são utilizados para capturar similaridades e variabilidades de famílias de produtos [7]. Entretanto, é um grande desafio estabelecer um relacionamento entre as *features* de um produto de software e os objetivos dos *stakeholders*. Também não há uma forma sistemática para descobrir quais *features* farão parte da LPS e nem para determinar quais *features* serão opcionais, obrigatórias ou alternativas. Além disso, o modelo de *features* não permite justificar claramente porque um determinado conjunto de *features* foi selecionado para configurar um produto específico na LPS. Por outro lado, abordagens orientadas a objetivos podem ser usadas como uma forma de 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 têm sido propostas para modelar a variabilidade de requisitos, tais como o modelo de objetivos [8], o PL-AOVGraph [9] e o i* Aspectual [10]. Recentemente, uma comparação entre estas três abordagens apontou que elas têm expressividade limitada para representar variabilidade em LPSs [11]. Isto nos motivou a investigar o uso do *framework* i* como abordagem de Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software, permitindo estruturar os requisitos na fase inicial do desenvolvimento dos sistemas de acordo com as intenções dos *stakeholders* para uma família de produtos.

Este artigo está organizado da seguinte forma: a seção 2 apresenta uma visão geral do i* utilizando como exemplo a LPS Mobile Media, a seção 3 apresenta a linguagem **i*-c** e a seção 4, o processo definido pela abordagem **G2SPL**. A seção 5 apresenta os trabalhos relacionados e por fim, a seção 6 discute as contribuições do artigo e aponta para alguns trabalhos futuros.

II. VISÃO GERAL DO FRAMEWORK I*

As *features* 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 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 estes não capturam requisitos não-funcionais explicitamente e nem a influência positiva/negativa destes requisitos para alcançar configurações alternativas de um produto de software na LPS. Esta influência pode ajudar a guiar e justificar a escolha de uma configuração específica para um produto de software alcançar os objetivos da organização.

Abordagens orientadas a objetivos podem ser usadas para capturar *features* utilizando modelos mais significativos, que mantém o rastreamento entre as *features* da LPS e os objetivos dos stakeholders, bem como para justificar a configuração de um produto específico na LPS para a realização destes objetivos. O *framework* i* [3] 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 [12]. 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 [13] 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 [14].

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.

Para explicar os principais conceitos do i*, utilizaremos a LPS Mobile Media [15]. O principal objetivo do Mobile Media é manipular fotos, música e vídeo em dispositivos móveis como, por exemplo, telefones.

No modelo SR apresentado na Figura 1 temos dois atores (*Telefone Móvel* e *Usuário*) relacionados através de dependências externas. O ator *Usuário* tem como objetivo principal a *Manipulação de Mídia*. Uma forma de alcançar este objetivo é através da execução da tarefa *Adicionar Foto*. Portanto, esta tarefa está ligada ao objetivo através de uma ligação meio-fim (*means-end link*). A tarefa *Adicionar Foto* é dividida em quatro outras atividades através de uma ligação de decomposição de tarefa (*task-decomposition link*), significando que para que a tarefa raiz seja realizada, todas as sub-tarefas terão de ser obrigatoriamente realizadas. A sub-tarefa *Selecionar Álbum* será executada através do cumprimento das dependências externas *Integridade [Foto]* (dependência de *softgoal*) e *Album* (dependência de recurso).

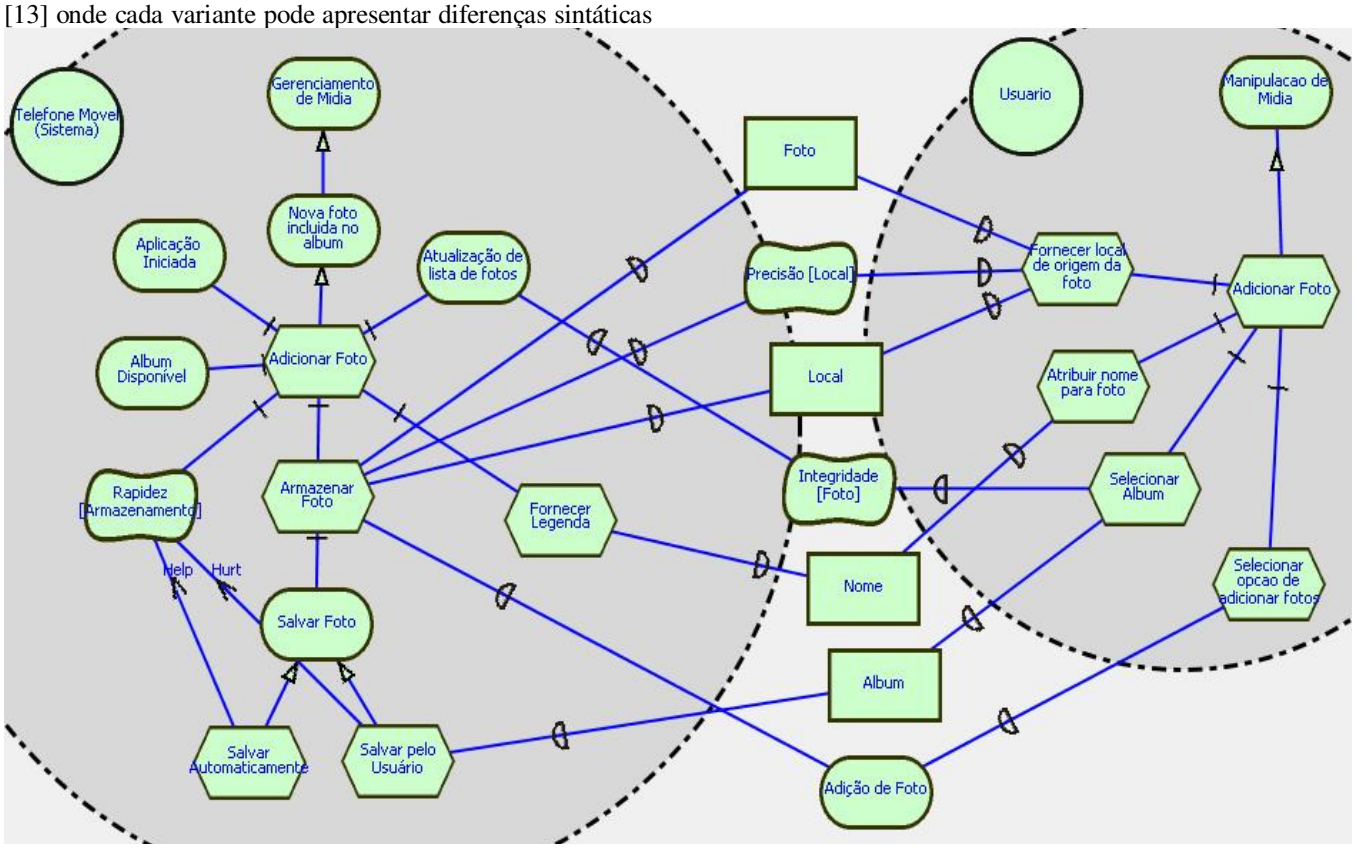


Figura 1. Modelo SR para o cenário Adicionar Foto.

Já a sub-tarefa *Selecionar opção de adicionar fotos* será executada através do cumprimento da dependência externa *Adição de Foto* (dependência de objetivo). Por outro lado, várias dependências externas dependem de elementos internos ao ator *Usuário* para serem alcançadas. É o caso das dependências de recurso *Foto*, de *softgoal Precisão [Local]* e de recurso *Local*, que dependem da tarefa *Fornecer local de origem da foto* para serem alcançadas. Além disso, a dependência de recurso *Nome* depende da tarefa *Atribuir nome para a foto* para ser alcançada. O ator *Telefone Móvel*, que representa a LPS pretendida, além dos elementos intencionais (e.g. tarefa, recurso, softgoal e objetivo), das ligações de refinamento e das ligações de dependência também presentes no ator *Usuário*, temos a ligação de contribuição. Esta ligação relaciona uma tarefa a um *softgoal*, significado que uma tarefa pode contribuir positivamente ou negativamente para a satisfação daquele *softgoal*. Por exemplo, a tarefa *Salvar Automaticamente* contribui positivamente (*Help*) para a satisfação do *softgoal Rapidez [Armazenamento]*, enquanto que a tarefa *Salvar pelo Usuário* contribui negativamente (*Hurt*) para a satisfação deste mesmo *softgoal*. Desta forma, destacamos duas formas distintas de alcançar o objetivo *Salvar Foto*.A ligação de meio-fim entre as tarefas *Salvar Automaticamente* e *Salvar pelo Usuário* com o objetivo *Salvar Foto* indicam um ponto de variação das *features* da LPS representada pelo ator *Telefone Móvel*. A capacidade de se ter formas alternativas de alcançar um objetivo, alinhado com a capacidade de se estabelecer relações de influência entre os requisitos funcionais (tarefas) e não-funcionais (*softgoals*) de um sistema, nos motivou a investigar o framework i* como técnica capaz de descobrir variabilidade e configurar (selecionar as *features* de) um ou mais produtos de software em uma LPS.

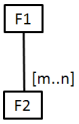
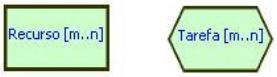
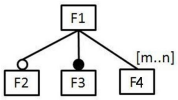
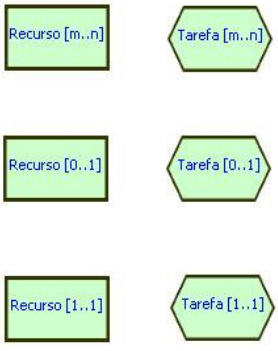
III. EXTENSÃO DO MODELO I* PARA INCLUSÃO DE CARDINALIDADE

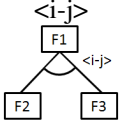
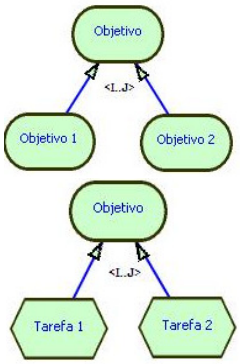
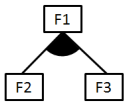
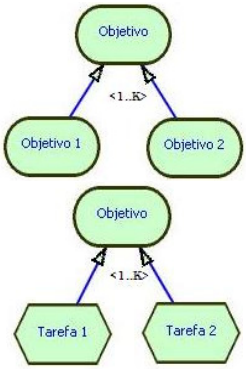
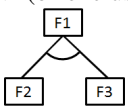
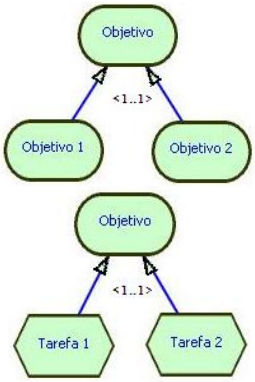
Modelos i* não foram desenvolvidos com a intenção de modelar LPS, portanto faz-se necessário ajustar este modelo para que se possa capturar informação relativa à presença opcional e obrigatória de algumas *features* em um produto da LPS. Dessa forma, propomos uma extensão da linguagem i*, chamada **i*-c** (*i* with cardinality*), onde foram adicionadas cardinalidade, permitindo capturar mais informação sobre a variabilidade de uma LPS com o mesmo poder de expressividade dos modelos de *features* de [16]. Os principais construtores de um modelo de *features* estão descritos na Tabela I.

Uma *feature* pode indicar qualquer característica ou funcionalidade do sistema e é usada para capturar similaridades ou variabilidades nos produtos de uma LPS [6]. Partindo desta definição, na linguagem **i*-c**, os elementos do tipo tarefa e recurso são os elementos intencionais do modelo i* com a possibilidade de terem cardinalidade, já que determinam funcionalidades e características de um sistema, respectivamente (vide conceitos **A**, **B**, **C** e **D** da Tabela I).

A ligação meio-fim (*MeansEndLink*) nos modelos i* significa que um objetivo (fim) pode ser alcançando através do sub-elemento de onde parte a ligação (ou seja, o meio), representando uma ligação *ou-inclusivo* (isto é, com cardinalidade <1-k>). Porém, para representarmos ligações do tipo *ou-exclusivo* (isto é, com cardinalidade <1-1>) e *ou com cardinalidade* (isto é, com cardinalidade <i-j>), adicionamos cardinalidade nas ligações meio-fim da linguagem **i*-c**. As *features* obtidas através dos sub-elementos (geralmente tarefas) dessa ligação estarão agrupadas no modelo de *features* e a cardinalidade pertencerá ao relacionamento (vide conceitos **E**, **F** e **G** da Tabela I). Porém, se a quantidade de sub-elementos dessa ligação for exatamente 1, então a cardinalidade pertencerá ao sub-elemento e não ao relacionamento.

TABELA I. RELAÇÃO ENTRE CONSTRUTORES DO MODELO DE *FEATURES* E DA LINGUAGEM I*-C

Modelo de *Features*		Linguagem i*-c
(A)	*Feature* com cardinalidade [m..n] 	Elementos do tipo recurso e tarefa com cardinalidade [m..n] 
(B)	*Feature* com cardinalidade [0..1] (opcional) 	Elementos do tipo recurso e tarefa com cardinalidade [0..1] 
(C)	*Feature* com cardinalidade [1..1] (obrigatória) 	Elementos do tipo recurso e tarefa com cardinalidade [1..1] 
(D)	Relações que incluem *features* opcionais, obrigatórias e com cardinalidade [17] 	Elementos do tipo tarefa ou recurso com cardinalidade [m..n], [0..1] e [1..1] 

(E)	*Features* agrupadas com cardinalidade <i-j> 	Ligação meio-fim com cardinalidade <i..j> 
(F)	*Features* agrupadas com cardinalidade <1-k>, onde k é tamanho do grupo (or-inclusive) 	Ligação meio-fim com cardinalidade <1..k> 
(G)	*Features* agrupadas com cardinalidade <1-1> (or-exclusive) 	Ligação meio-fim com cardinalidade <1..1> 

A ligação de decomposição de tarefas nos modelos i* denota obrigatoriedade, significando que seus sub-elementos (desde que sejam tarefas ou recursos) estarão presentes no produto de software. Assim, a cardinalidade pertencerá aos sub-elementos e será [1..1].

As ligações de contribuição servem para modelar a forma como os elementos podem contribuir para a satisfação ou cumprimento de um *softgoal*. Esses elementos (desde que sejam tarefas ou recursos) poderão ter cardinalidade, a depender da LPS sendo modelada. Dessa forma, a cardinalidade pertencerá aos elementos e não à ligação.

A Tabela I apresenta como os construtores da notação da linguagem **i*-c** representam os construtores do modelo de *features*.

A Figura 2 apresenta parte do metamodelo da linguagem **i*-c** que foi adaptado a partir de [13] para suportar cardinalidade. Os elementos *Resource*, *Task* e *MeansEndLink* (destacados no metamodelo) foram os adaptados para conter a informação de cardinalidade que ajudará a representar as *features* na abordagem **G2SPL**, apresentada em detalhes na próxima seção.

Existem algumas restrições quanto ao uso destes elementos na abordagem **G2SPL** e por isso foi necessária a construção de algumas regras escritas como expressões em OCL (*Object Constraint Language*) [18].

Essas restrições se referem quanto à utilização da cardinalidade 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 *MeansEndLink* (atributo *cardinality* não existe). O inverso também é verdade: quando a cardinalidade pertencer ao relacionamento *MeansEndLink*, então, os sub-elementos desta ligação estão agrupados (atributo *isGrouped* = true) e a cardinalidade não poderá pertencer a esses sub-elementos (atributo *cardinality* não existe). O atributo *isGrouped* determina se um elemento é agrupado ou não. Assim, apenas os elementos *Task* e *Resource* podem ser agrupados e o agrupamento só pode ser feito com o relacionamento *MeansEndLink*. A Tabela II apresenta as restrições como regras OCL para os elementos *Resource*, *Task* e *MeansEndLink* e sua respectiva representação em linguagem natural.

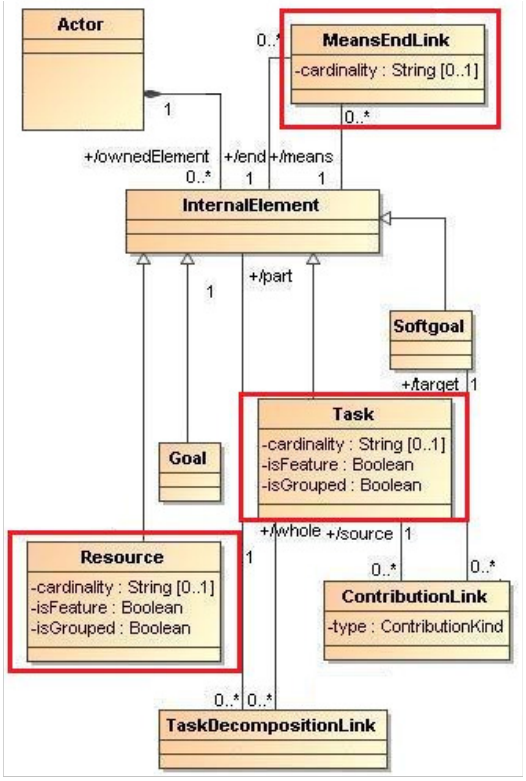


Figura 2. Metamodelo da linguagem i*-c (adaptado a partir de [13])

TABELA II. RESTRIÇÕES EM OCL DOS ELEMENTOS ADAPTADOS DO METAMODELO

Regra	Elemento no Metamodelo	Restrição em OCL	Restrição representada em Linguagem Natural
OCL1	*Resource*	*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*	Regra para um *Resource*: Se *Resource* é *feature* e é agrupado, então não existe cardinalidade no elemento e sim no seu relacionamento *MeansEndLink*. Se *Resource* é *feature* e não é agrupado, então existe cardinalidade no elemento e não no seu relacionamento *MeansEndLink*. Se *Resource* não é *feature*, então *isGrouped* é falso e não existe cardinalidade
OCL2	*Task*	*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*	Regra para uma *Task*: Se *Task* é *feature* e é agrupado, então não existe cardinalidade no elemento e sim no seu relacionamento *MeansEndLink*. Se *Task* é *feature* e não é agrupado, então existe cardinalidade no elemento e não no seu relacionamento *MeansEndLink*. Se Task não é *feature*, então *isGrouped* é falso e não existe cardinalidade
OCL3	*MeansEndLink*	*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 Task.cardinality[1] end if*	Regra para um *MeansEndLink*: Se existir cardinalidade no relacionamento *MeansEndLink* então, esse relacionamento possui mais de um sub-elemento e a cardinalidade não pertencerá aos seus sub-elementos. Se não existir cardinalidade no relacionamento *MeansEndLink* então, esse relacionamento possui apenas um sub-elemento e a cardinalidade 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 **G2SPL** a ser apresentada na próxima seção. Como o metamodelo da Figura 2 é bastante genérico, faz-se necessária a utilização de regras para que este metamodelo corresponda à versão do i* adotada nesse trabalho [14]. Dessa forma, duas regras foram adaptadas a partir de [13]. A Tabela III apresenta essas duas novas regras.

TABELA III. RESTRIÇÕES EM OCL PARA UTILIZAÇÃO DA VERSÃO I* DE [14]

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

IV. O PROCESSO G2SPL

O processo que descreve nossa abordagem, chamada de **G2SPL** (*Goal To Software Product Line*), tem como objetivo possibilitar a identificação de *features* nos modelos i* e a configuração de produtos específicos dentro da LPS. Para alcançar este propósito, definimos algumas atividades e heurísticas, formando um processo bem definido que será ilustrado através da sua aplicação a um caso de uso (Adicionar Foto) da LPS Mobile Media. A modelagem completa dessa LPS de acordo com a abordagem **G2SPL** pode ser encontrada em [19].

A Figura 3 apresenta uma visão geral do processo que descreve a abordagem **G2SPL**. Este processo foi modelado na notação BPMN (*Business Process Modeling Notation*) [20] com auxílio de uma ferramenta [21].

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 geralmente o papel de engenheiro de configuração.

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*) [22] que é um método aplicado na reengenharia de processos de negócio. Utilizamos apenas a fase 2 desse método, pois esta se refere à construção do modelo SR do i*.

O detalhamento da atividade de construção de modelos SR não é apresentado neste artigo por questões de espaço, mas pode ser encontrado em [19].

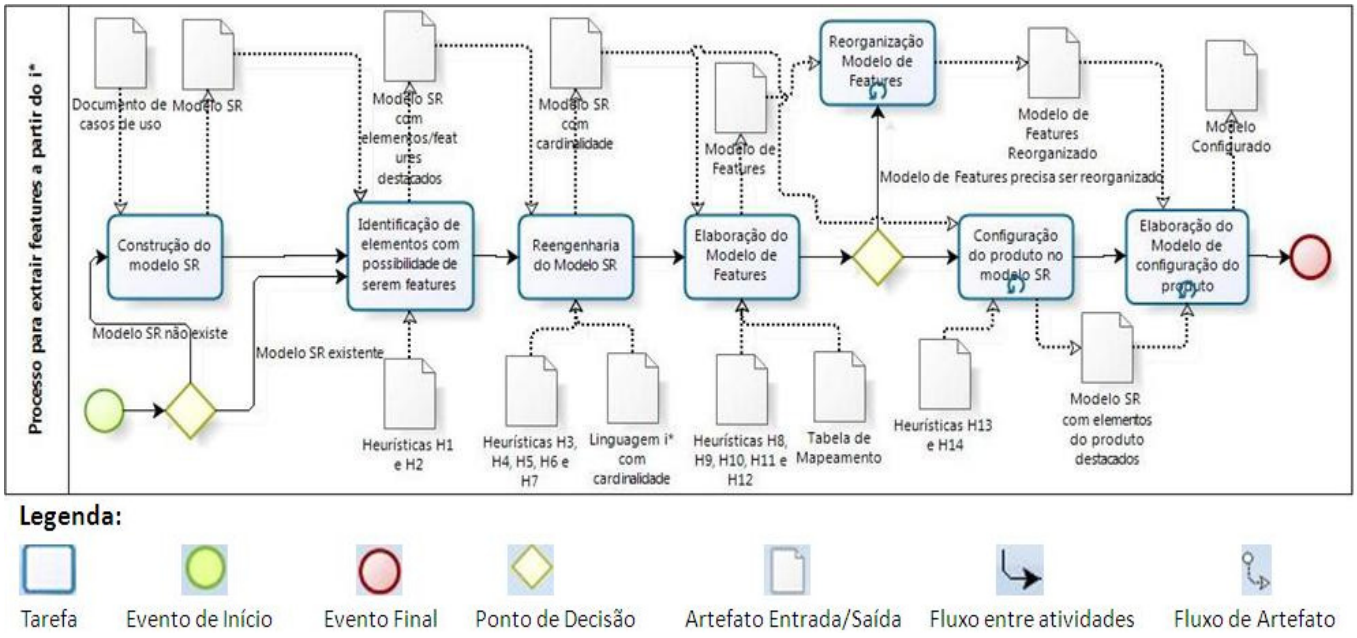


Figura 3. Processo para extrair features a partir do modelo SR do i* em BPMN

Uma vez que o modelo SR é obtido, o usuário precisa identificar quais elementos deste modelo estarão presentes no modelo de *features* (atividade *Identificação de elementos com possibilidade de serem features*). Como vimos anteriormente, uma *feature* pode indicar qualquer característica ou funcionalidade de um sistema e é usada para capturar similaridades ou variabilidades nos produtos de uma LPS [6]. 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.*

Como visto na seção II, 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 a LPS 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 como entrada o modelo SR da Figura 1, aplicamos as heurísticas H1 e H2, e obtemos o seguinte resultado: todos os elementos do tipo tarefa do ator que representa a LPS (*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 representa a saída desta atividade do processo, ou seja, um modelo SR com os elementos possíveis de serem *features* destacados.

Após as possíveis *features* terem sido identificadas, a atividade *Reengenharia do Modelo SR* preocupa-se em reestruturar o modelo SR gerado na atividade anterior, para que este modelo contenha cardinalidade. Esta reestruturação é feita através da aplicação de algumas heurísticas e da linguagem **i*-c**. A cardinalidade a ser usada nos modelos i* tem o mesmo significado que a dos modelos de *features* de [16] e é relembrada na Tabela IV.

Denominamos as heurísticas definidas nesta atividade (H3, H4, H5, H6 e H7) como heurísticas de domínio, pois são usadas para fazer a identificação e análise dos requisitos da LPS.

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 da LPS que está sendo modelada. Apenas indicamos que a cardinalidade a ser usada deve se basear na Tabela IV.

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).

TABELA IV. CARDINALIDADE PARA OS MODELOS I*

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

H4 *Se a quantidade de sub-elementos de uma ligação meio-fim que estão destacados como possíveis features, for maior do que 1* (e.g. ligação entre o objetivo *Salvar Foto* e as tarefas *Salvar Automaticamente* e *Salvar pelo Usuário* na Figura 1)*, então esses sub-elementos serão agrupados e a cardinalidade pertencerá ao relacionamento. Se a quantidade for exatamente 1* (e.g. ligação entre o objetivo *Nova foto incluída no álbum* e a tarefa *Inserir Foto* na Figura 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 (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, terão a cardinalidade [1..1], significando que a cardinalidade de obrigatoriedade pertencerá aos sub-elementos.*

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 esteja destacado como feature, terão a cardinalidade pertencente a eles próprios.*

Tendo como base o metamodelo da Figura 2 e a aplicação dessas heurísticas ao modelo SR gerado na atividade anterior, obtemos como saída desta atividade um modelo SR contendo cardinalidade (Figura 4).

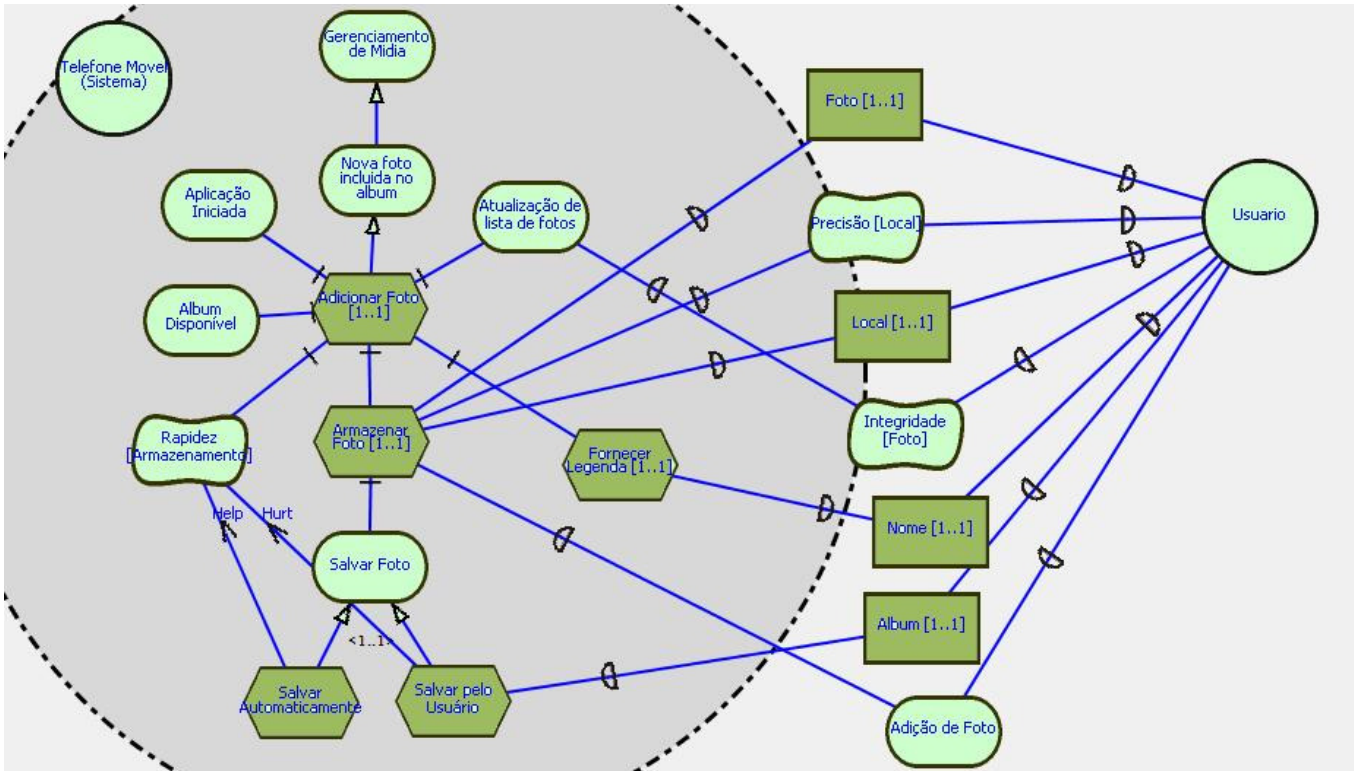


Figura 4. Modelo SR com cardinalidade do caso de uso Adicionar Foto

Aplicação no exemplo: Aplicando as heurísticas (H3, H4, H5, H6 e H7) e a linguagem **i*-c** ao modelo SR da Figura 1, obtemos o seguinte resultado:

H3 – *Os elementos destacados da Figura 4 (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> para o relacionamento, significando que os elementos são alternativas (relacionados através do ou-exclusivo).*

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 indicando que são obrigatórias.*

H7 – *A tarefa Adicionar Foto tem a cardinalidade [1..1] no próprio elemento, indicando que é obrigatória.*

Por questão de espaço, as atividades Elaboração do Modelo de *Features* e Reorganização do Modelo de *Features* não serão apresentadas neste artigo, porém a descrição detalhada de cada uma delas pode ser encontrada em [19]. Conseqüentemente, as heurísticas H8, H9, H10, H11 e-H12 foram omitidas.

A atividade de *Configuração do produto no modelo SR* tem como objetivo selecionar 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 de desenvolvimento do produto. Esta atividade se repetirá para cada produto a ser configurado na LPS.

Os modelos i* permitem expressar os objetivos dos *stakeholders* e as características do sistema pretendido usando elementos intencionais, que ilustram 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 um determinado produto de software. Por exemplo, de acordo com a Figura 4, se *Rapidez [Armazenamento]* é um *softgoal* do modelo SR que tem uma alta prioridade para o *stakeholder*, provavelmente o engenheiro de requisitos irá selecionar os elementos candidatos a *features* que melhor satisfazem este *softgoal* como, por exemplo, a tarefa *Salvar Automaticamente*.

Nesta fase, os *stakeholders* devem indicar quais *features* 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 o produto de software desenvolvido alcance os objetivos da organização. Algumas heurísticas foram definidas quanto à hierarquia dos elementos selecionados e a obrigatoriedade da presença deles no produto de software.

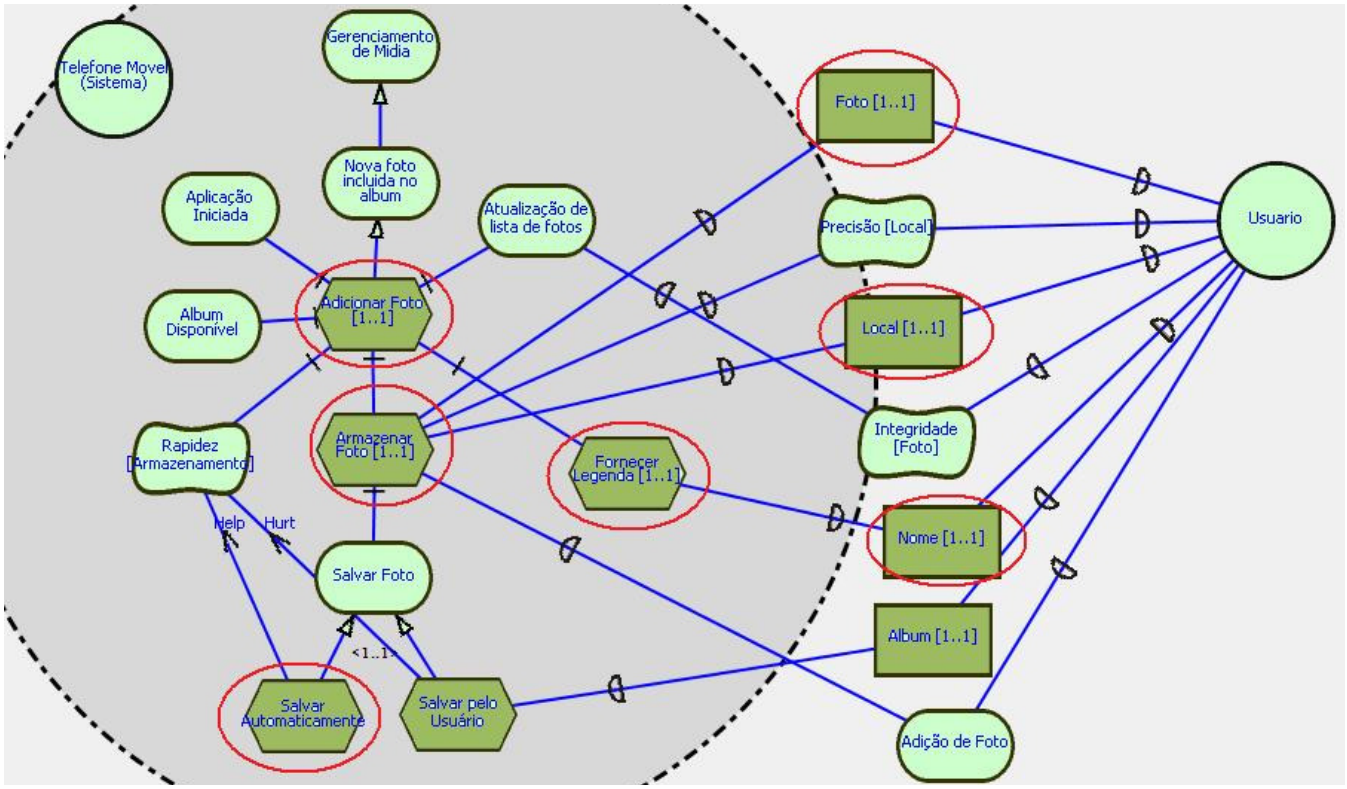


Figura 5. Modelo SR do caso de uso Adicionar Foto destacando os elementos que estarão presentes no modelo de configuração do produto

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 (ou elementos-raíz) também tenham sido selecionados. Os elementos com cardinalidade [0..1] poderão ou não ser selecionados, ficando à critério do stakeholder.*

H14 *Pelo menos um dos sub-elementos (meio) relacionados com as ligações meio-fim que possuem cardinalidade <1..1>, <1..k> ou <j..k>, desde que j seja maior ou igual a 1, deverá estar presente no modelo de configuração do produto.*

Aplicação no exemplo: Aplicando as heurísticas (H13 e H14) no modelo SR da Figura 4 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]. Observamos que o elemento Álbum, apesar de também ter a cardinalidade [1..1], não estará presente no modelo de configuração do produto, 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 (ou-exclusivo). 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].*

O objetivo da atividade *Elaboração do Modelo de configuração do produto* é construir o modelo de configuração de um produto específico (Figura 6), a partir do modelo SR que destaca (com elipses na Figura 5) os elementos que farão parte deste produto. Esta atividade se repetirá para cada produto a ser configurado na linha de produtos.

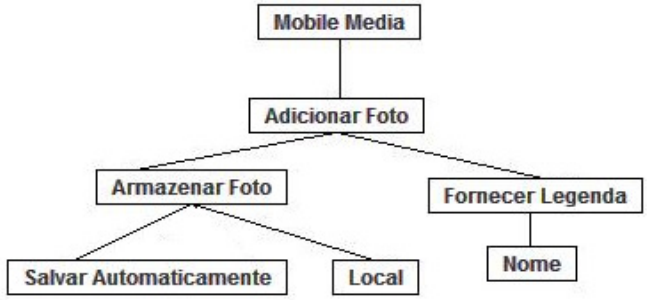


Figura 6. Modelo de configuração do produto

Aplicação no exemplo: observamos na Figura 5 que os elementos *Adicionar Foto, Armazenar Foto, Fornecer Legenda, Salvar Automaticamente, Foto, Local* e *Nome* foram destacados para fazerem parte do produto. Portanto, geramos o modelo de configuração válido para o caso de uso Adicionar Foto (Figura 6). Observe que os relacionamentos entre os elementos presentes no modelo de configuração são

obtidos diretamente dos relacionamentos presentes no modelo SR da Figura 5.

V. TRABALHOS RELACIONADOS

Algumas abordagens orientadas a objetivos para requisitos em LPS têm sido propostas para capturar as semânticas de modelos de *features*: modelos de objetivos [8], modelo PL-AOVGraph [9] e modelo i* aspectual [10]. Porém, uma comparação entre estas três abordagens realizada em [11] apontou que elas têm a expressividade limitada, pois não são capazes de representar a cardinalidade do modelo de *features*. Essa limitação nos levou a propor a linguagem **i*-c** (*i* with cardinality*) utilizada pela abordagem **G2SPL**, onde adicionamos cardinalidade, permitindo capturar mais informação sobre a variabilidade de uma LPS.

Outra abordagem baseada em modelos orientados a objetivos, mais especificamente em modelos i*, é IStarLPS [23]. Esta abordagem, similarmente à **G2SPL**, adapta a *framework* i* para Linhas de Produtos de Software, porém com significantes diferenças:

- A abordagem **G2SPL** utiliza as informações do modelo SR para guiar a configuração de um produto de software específico em uma LPS, além de utilizá-lo para identificar as *features* da LPS. A abordagem IStarLPS também se baseia no modelo SR para identificar as *features*, mas guia a seleção de *features* para configurar um produto específico;
- Na abordagem IStarLPS, todos os elementos intencionais dos modelos SD e SR são mapeados para *features*, enquanto que a abordagem **G2SPL** considera *features* apenas os elementos do tipo tarefa e recurso (vide heurística **H1**);
- Em IStarLPS, a cardinalidade é incluída em todos os elementos intencionais do i*, enquanto que em **G2SPL**, acrescentamos a cardinalidade apenas nos elementos que podem ser *features* (tarefas e recursos) e no relacionamento meio-fim (vide heurística **H4**). Em IStarLPS, quando o elemento do tipo objetivo tem a cardinalidade [0..1] significa a possibilidade deste elemento ser alcançado ou não num produto de software da LPS, porém não faz sentido a cardinalidade [m..n] no contexto de objetivos, já que este tipo de elemento é uma condição do mundo a ser alcançada (o que significa alcançar uma condição n vezes?). Devido a esta observação, optamos por considerar apenas tarefas e recursos como possíveis *features*;
- A abordagem **G2SPL** mantém o modelo i* com suas informações originais, ajudando na escolha e justificativa de uma configuração para um determinado produto de software, através dos relacionamentos de contribuição em relação à satisfação dos *softgoals*. Em IStarLPS, além dos elementos do tipo *softgoal* não terem sido considerados na abordagem, não há nenhum tipo de justificativa na escolha de uma determinada configuração para um produto de software específico. Porém, IStarLPS guia a identificação de *constraints* do tipo *requires* presentes nos modelos de *features* [7].

VI. CONCLUSÕES E TRABALHOS FUTUROS

Este artigo apresentou uma abordagem orientada a objetivos (**G2SPL** – *to Software Product Line*) que permite tanto a identificação de *features* comuns e variáveis em Linhas de Produtos de Software, como a configuração de um produto de software específico 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 LPS, como por exemplo: rastreamento entre as *features* do sistema e os objetivos dos *stakeholders*, bem como justificativa da configuração de um produto específico na LPS para satisfazer estes objetivos.

Neste trabalho, também foi apresentada 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 **G2SPL** 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 um determinado produto de software. De fato, os relacionamentos de contribuição dos requisitos funcionais com relação à satisfação de requisitos não-funcionais, auxiliam na atividade de configuração de um produto específico, reduzindo o tempo e os custos associados a esta atividade.

Como trabalhos futuros, nós planejamos: (i) realizar estudos de caso em diferentes domínios, de forma a avaliar com maior precisão as forças e fraquezas da abordagem **G2SPL**; (ii) desenvolver uma ferramenta que suporte a abordagem **G2SPL**; (iii) investigar como capturar *constraints* do modelo de *features* a partir dos modelos i* e (iv) avaliar o uso das fases iniciais da abordagem **G2SPL** como guia para a construção de modelos i*.

AGRADECIMENTOS

Este trabalho foi apoiado pelos órgãos brasileiros de fomento à pesquisa CNPq e CAPES.

REFERÊNCIAS

[1] P. Zave, "Classification of research efforts in requirements engineering", ACM Computing Surveys. New York, NY, USA: ACM Press, v. 29, n. 4, 1997, pp. 315-321.

[2] A. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, 2001.

[3] E. Yu, "Modelling Strategic Relationships for Business Process Reengineering", Ph.D Thesis, Dept. of Computer Science, University of Toronto, 1995.

[4] K. Pohl, G. Böckle, F. van der Linden, "Software Product Line Engineering: Foundations, Principles, and Techniques", Springer, 2005.

[5] P. Clements, L. Northrop, "Software Product Lines: Practices and Patterns". Addison-Wesley, 2002.

[6] K. Czarnecki, U. W. Eisenecker, “Generative Programming: Methods, Tools, and Applications”, ACM Press/Addison-Wesley Publishing Co, 2000.

[7] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study”. Software Engineering Institute, Technical report, CMU/SEI-90-TR-021, 1990.

[8] Y. Yu, J.C.S.P. Leite, A. Lapouchnian, J. Mylopoulos, “Configuring features with stakeholder goals”, Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), ACM, Fortaleza, Ceara, Brazil, pp. 645-649, 2008.

[9] T. Batista, M. Bastarrica, S. Soares, L. Fernandes, “A Marriage of MDD and Early Aspects in Software Product Line Development”, Early Aspects Workshop at 12th International Software Product Line Conference (LPSC’08), Limerick, Ireland, 2008, pp. 97-104.

[10] F. Alencar, J. Castro, A. Moreira, J. Araujo, C. Silva, R. Ramos, J. Mylopoulos, “Integration of Aspects with i* Models”, Agent-Oriented Information Systems IV, LNCS Vol. 4898, Springer-Verlag, 2008, pp. 183-201.

[11] C. Borba, C. Silva, “A comparison of goal-oriented approaches to model software product lines variability”. In: Ws on Requirements, Intentions and Goals in Conceptual Modeling in conjunction with the 29th International Conference on Conceptual modeling (ER 2009), Gramado. LNCS Vol. 5833, Springer-Verlag, 2009.

[12] E. Yu, “Towards modelling and reasoning support for early-phase requirements engineering”,: Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE 97). Annapolis, MD, USA: IEEE Computer Society, 1997. pp. 226-235.

[13] M. Lucena, E. Santos, C. Silva, F. Alencar, M.J. Silva, J. Castro, “Towards a Unified Metamodel for i*”, Second IEEE International Conference on Research Challenges in Information Science 2008, Marrakech, 2008.

[14] G. Grau, E. Yu, J. Horkoff, S. Abdulhadi, “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.

[15] E. Figueiredo, N. Cacho, C. Sant’Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F.C. Filho, F. Dantas, “Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability”, Proceedings of the 30th Intl. Conf. on Software Engineering (ICSE’08), Leipzig, Germany, 2008. pp 261-270.

[16] K. Czarnecki, S. Helsen, U. Eisenecker, “Formalizing cardinality-based feature models and their specialization”, Software Process Improvement and Practice, 10(1), 2005, pp.7-29.

[17] D. Benavides, S. Trujillo, P. Trinidad, “On the modularization of feature models”, 1st European Workshop on Model Transformation. Rennes, France, 2005.

[18] OMG, UML 2.0 OCL Specification, 2003. Disponível em: http://www.omg.org/docs/ptc/03-10-14.pdf.

[19] C. Borba, “Uma Abordagem Orientada a Objetivos para as Fases de Requisitos de Linhas de Produtos de Software”, Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, 2009.

[20] 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.

[21] BizAgi Process Modeler. Disponível em: <http://www.bizagi.com/esp/productos/ba-modeler/modeler.html>. Último acesso em: 06/2009.

[22] G. Grau, X. Franch, N. Maiden, “PRiM: An i*-based process reengineering method for information systems specification”, Information and Software Technology 50, 2008, pp. 76-100.

[23] S. António, J. Araújo, C. Silva, “Adapting the Framework i* for Software Product Lines”, Workshop on Requirements, Intentions and Goals in Conceptual Modeling in conjunction with the 29th International Conference on Conceptual modeling (ER 2009), 2009, Gramado. LNCS Vol. 5833, Springer-Verlag.