Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado em Engenharia Informática

1º Semestre, 2008/2009

Adaptação da Framework i* para Linhas de Produtos

Nº 26294 Sandra Isabel Alves António

Orientador

Prof. Doutor João Baptista da Silva Araújo Júnior

Co-Orientadora

Prof. Doutora Carla Silva

20 de Fevereiro de 2009

Nº do aluno: 26294

Nome: Sandra Isabel Alves António

Título da dissertação:

Adaptação da Framework i* para Linhas de Produtos

(Adapting the i* Framework for Product Lines)

Palavras-Chave:

Framework i*
Linhas de Produto de *Software*
Engenharia de Requisitos
Modelos de *Feature*
Abordagens Orientadas a objectivos

Keywords:

i* Framework
Software Product Lines
Requirements Engineering
Feature Models
Goal-orientation Approach

Agradecimentos

Agradeço a Deus por me acompanhar e ajudar ao longo da minha vida. Aos meus pais Leonídio e Judite, à minha irmã Andrea e ao meu namorado Rui pelo amor, carinho, compreensão e apoio desde sempre e em particular durante este ano.

A todos os meus amigos por continuarem a ser meus amigos mesmo sem eu dar sinal de vida durante um bom tempo. Ao Dagmar, meu companheiro de trabalho que me apoiou durante esta dissertação. Ao Professor João Araújo e à Professora Carla Silva por me orientarem e ajudarem a decidir o caminho a seguir.

Muito obrigado a todos.

Resumo

A investigação de requisitos em linhas de produtos de *software* tem explorado as maneiras pelas quais se podem definir, de forma apropriada, uma plataforma capaz de servir de base para a derivação rentável dos produtos para os utilizadores individuais. Os modelos de *features* são uma técnica importante para capturar os pontos comuns e diferentes nas linhas de produtos. Uma *feature* pode representar uma característica funcional ou não-funcional de requisitos, arquitectura, ou de qualquer outro nível de abstracção do *software*. Contudo, os modelos de *features* mostram uma perspectiva muito específica das linhas de produtos, por isso é necessário ter uma abordagem que mostre outras perspectivas a nível de requisitos e dar significado a essas *features* de maneira a ser mais compreensível o que se pretende representar.

Os paradigmas orientados a objectivos ou agentes têm sido usados para o desenvolvimento de sistemas complexos e, algumas abordagens como a *framework* i* têm sido desenvolvidas para serem utilizadas na engenharia de requisitos. A *framework* de modelação organizacional i* oferece conceitos sociais e intencionais, e os desejos dos *stakeholders* são considerados realmente importantes para o desenvolvimento de sistemas que satisfaçam melhor nas suas necessidades. Os modelos de objectivos fornecem um modo natural de identificar a variabilidade na fase inicial de requisitos, permitindo a captura de formas alternativas para os *stakeholders* alcançarem os seus objectivos. Este modo natural é muito importante para se conseguir utilizar modelos orientados a objectivos no desenvolvimento de linhas de produtos. Todavia, o uso da *framework* i* para linhas de produtos não foi ainda suficientemente explorado. Portanto, o objectivo desta dissertação é adaptar a *framework* i* para o desenvolvimento de linhas de produtos de *software* e fornecer uma abordagem mais expressiva para a engenharia de requisitos destes produtos.

Abstract

Research in requirements for software product lines has been exploring ways by which one can define a platform capable of serving as the basis for cost-effective derivation of products for individual users. Feature modeling is an important technique for capturing commonalities and variabilities in product lines. A feature may denote any functional or non-functional characteristic at the requirements, architecture, or any other abstraction level of software. However, feature models show a very specific perspective of a product line, so it is necessary to have an approach that showed other perspectives at requirements level, and give them semantic to make it understandable.

The goal and agent -oriented paradigms have been used to develop complex systems and some approaches such as i* framework, have been developed to be used in requirements engineering. i* organizational modeling framework offer social and intentional concepts, and stakeholder's desires are considered really important to development systems that best meet their needs. Goal models provide a natural way to identify variability at the early requirements phase, by allowing the capture of alternative ways by which stakeholders achieve their goals. This natural way is very important to use in product lines with goal-oriented model. However, the use of i* framework to describe product lines has not been explored sufficiently yet. Therefore, the purpose of this dissertation is to adapt i* framework to develop software product lines in order to provide a more expressive approach for requirement engineering their product.

Acrónimos

Lista de acrónimos utilizados ao longo desta dissertação.

Acrónimo	**Significado**
ELPS	Engenharia de Linhas de Produto de Software
EROO	Engenharia de Requisitos Orientada a Objectivos
ER	Engenharia de Requisitos
GRL	Goal-oriented Requirements Language
KAOS	Keep All Objectives Satisfied
LPS	Linha de Produtos de Software
MATA	Modeling Aspects using a Transformation Approach
NFR	Non-Functional Requirement
NFRF	Non-Functional Requirement Framework
SD	Strategic Dependency
SR	Strategic Rational
UML	Unified Modeling Language
UMLT	Unified Modeling Language Transformation

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1 Engenharia de Requisitos e Linhas de Produto de Software

Um dos objectivos da Engenharia de *Software* é definir os requisitos de um sistema de *software*. Um requisito é um atributo necessário num sistema, uma declaração que identifica uma capacidade, característica ou factor de qualidade no sistema para que o mesmo tenha valor e utilidade para um cliente ou utilizador [1]. O processo de determinação desses requisitos é levado a cabo pela Engenharia de Requisitos (ER) [2].

A ER é útil para descobrir os objectivos, as restrições e as funcionalidades de sistemas complexos, e produzir documentação sobre os requisitos para posterior desenvolvimento e manutenção. Quem fornece essa informação são os *stakeholders*, ou seja, as pessoas que irão intervir com o sistema, quer usando-o, patrocinando-o ou construindo-o [3]. A informação obtida neste processo será tida como a base de um sistema, por isso esta fase tem grande importância. O processo de ER inclui várias actividades, sem uma sequência obrigatória. As actividades, segundo Lamsweerde [4] estão descritas na Tabela 1.1.

A aplicação da ER é complexa devida à quantidade de factores envolvidos. Por exemplo, o factor humano, os vários conflitos de interesse e a quantidade de actividades interligadas a ER, dificultam a sua aplicação, sendo muito útil haver suporte e técnicas rigorosas que ajudem a ultrapassar as dificuldades.

A ER tanto pode ser aplicada para desenvolver produtos de *software* únicos, como para desenvolver Linhas de Produtos de *Software* (LPS) [5, 6]. A Engenharia de Linhas de Produto de *Software* (ELPS) tem sido explorada actualmente para ajudar a desenvolver vários produtos de forma mais eficiente e barata. A ELPS possibilita uma rentabilidade maior com uma família de produto do que com a produção de produtos semelhantes separadamente, ao captar as partes comuns e variáveis desses produtos. No entanto, isto também implica que haja menor diversificação de produtos [6]. Assim, a ELPS permite a produção em larga escala de produtos semelhantes que satisfazem as necessidades individuais dos clientes.

Tabela 1.1 Descrição das actividades de engenharia de requisitos

Fases	Actividades	Descrição
Desenvolvimento dos Requisitos	**Análise de domínio**	Permite fornecer o contexto do sistema existente, os seus *stakeholders*, os problemas e opções a serem analisados, tal como os principais objectivos;
	Identificação de requisitos	São determinados os requisitos, e várias modelações;
	Especificação de requisitos	Os requisitos são estruturados de um modo preciso;
	Análise da especificação de requisitos	As hipóteses são avaliadas, para se encontrar deficiências e a viabilidade dessas especificações, tendo em conta os riscos das alternativas;
	Negociação de requisitos	São tomadas decisões e são negociados, com os *stakeholders,* as características mais importantes;
	Documentação de requisitos	Informação necessária para se perceber posteriormente as escolhas efectuadas;
Manutenção dos Requisitos	**Evolução de requisitos**	Gestão dos requisitos em que se efectuam alterações aos requisitos ou quando aparecem novos requisitos.

Com a utilização de LPS, beneficia-se de uma maior variedade de produtos semelhantes, em menos tempo. As LPSs podem ser especificadas em modelos de *features*, que permitem representar a variabilidade de uma família de produtos. Porém, a representação dos pontos iguais e diferentes de uma LPS na forma taxonómica do modelo de *features* apenas apresentará símbolos. Portanto, se utilizar outros modelos para complementar o modelo de *features*, será possível dar semântica a esses símbolos e obter mais informação desses modelos [7]. De qualquer modo, a Engenharia de Requisitos em ELPS é um pouco diferente da aplicada a uma família de sistemas desenvolvidos separadamente.

Esta dissertação concentra-se na Engenharia de Requisitos das Linhas e Produtos de *Software*, permitindo estruturar os requisitos, na fase inicial dos sistemas de acordo com as intenções dos *stakeholders*, para uma família de produtos. Na próxima secção é apresentada a motivação deste trabalho.

1.2 Motivação

Actualmente muitos dos sistemas que são desenvolvidos acabam por ser descartados devido às más práticas de engenharia de requisitos adoptadas, levando a requisitos especificados incorrectamente e, consequentemente, ao desperdício de recursos. Para evitar esse

desperdício, existem várias metodologias de identificação e análise de requisitos, podendo estas ser baseadas em objectos [3], aspectos [8], objectivos [1], *viewpoints* [9], entre outras. No caso da Engenharia de Requisitos Orientados a Objectivos (EROO) a principal preocupação é conhecer as propriedades do sistema que os *stakeholders* desejam, de modo a obter resultados mais adequados às suas necessidades. Tem-se especial interesse em perceber o "Porquê" da existência das tarefas, que é uma informação mais estável, em vez de se perceber o modo como uma tarefa pode ser executada (o "Como" se pode executar uma tarefa). Para guiar a especificação de sistemas baseando-se em EROO podem-se referenciar, por exemplo, as abordagens KAOS [10], i* [11], GRL [12, 13], NFRF [14] e V-graph [15]. Tendo em conta que os modelos de objectivos 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.

A variabilidade entre os produtos de uma mesma família motivou a ELPS. Esses produtos têm características comuns e diferentes, sendo desenvolvidos a partir da mesma plataforma (características iguais), mas possuem variações (características diferentes) que ajudam a satisfazer as necessidades de diferentes clientes.

Existem vários motivos que tornam a engenharia de requisitos orientada a objectivos útil. Por exemplo, a maior estabilidade dos objectivos em vez dos requisitos. Pois, os objectivos são mais estáveis que os requisitos ou operações, porque como os requisitos são obtidos devido à existência de objectivos, se houver uma mudança num objectivo os requisitos serão alterados, mas se mudar um requisito não haverá a implicação de alterar o objectivo que lhe deu origem. A EROO também tem a potencialidade de poder especificar a variabilidade do domínio de problemas com as várias alternativas que são apresentadas pelos *stakeholders* para atingir um objectivo. Mas é necessário proceder a algumas adaptações para utilizar esta abordagem na engenharia de requisitos de uma LPS. De facto, as abordagens EROO existentes não têm sido ainda exploradas suficientemente para identificar e analisar requisitos para linhas de produto de *software*.

1.3 Objectivos e Contribuições Esperadas

Com este trabalho pretende-se adaptar a *framework* i*, uma abordagem de Engenharia de Requisitos Orientada a Objectivos, às Linhas de Produtos de *Software*. Como o modelo orientado a objectivos i* fornece um modo de identificar a variabilidade na fase inicial de requisitos, será possível contribuir com maior expressividade no desenvolvimento de LPSs. O

uso da *framework* i* para a Engenharia de Linhas de Produtos de *Software* não foi ainda suficientemente explorado, mas existe uma grande potencialidade na adaptação das abordagens EROO, pois os modelos orientados a objectivos identificam naturalmente a variabilidade das LPS nas fases iniciais dos requisitos.

Este trabalho é importante para a área de engenharia de *software*, na medida em que a *framework* i*, sendo uma metodologia simples e popular na análise e identificação de requisitos facilita a identificação da variabilidade, suportando o desenvolvimento de LPSs, que são cada vez mais utilizadas.

1.4 Organização do Documento

Este documento está organizado do seguinte modo: o capítulo 2 apresenta a Engenharia de Requisitos Orientada a Objectivos, e algumas metodologias orientadas a objectivos, dando maior ênfase à *framework* i*. O capítulo 3 aborda o tema de Linhas de Produto de *Software* e de modelo de *features*. No capítulo 4 são apresentados alguns trabalhos relacionados. O capítulo 5 descreve o processo de utilização da abordagem IStarLPS. O capítulo 6 ilustra a aplicação da abordagem IStarLPS com um caso de estudo real. E por fim, o capítulo 7 apresenta as conclusões desta dissertação.

2. Engenharia de Requisitos Orientada a Objectivos

A Engenharia de Requisitos Orientada a Objectivos (EROO) pretende preencher as lacunas existentes na criação de *software*, baseando-se nos objectivos dos *stakeholders*, de modo que o *software* a ser desenvolvido corresponda ao que realmente os *stakeholders* desejam [16]. Segundo Lamsweerde [4], um objectivo é aquilo que o sistema deverá atingir com a cooperação dos vários agentes, quando o projecto estiver concluído e a ser utilizado no seu ambiente.

As técnicas de EROO destacam-se de outros tipos de técnicas devido ao facto de cobrirem tanto os Requisitos Funcionais como os Requisitos Não-Funcionais (do inglês, *Non-Functional Requirements* – NFRs) [14]. Requisitos Funcionais descrevem as funcionalidades do sistema (por exemplo, fazer a pesquisa numa base de dados através de palavras-chave). Os NFRs são requisitos qualitativos (como segurança, fiabilidade, disponibilidade, tempo de resposta), ou seja, pretende-se que o sistema os possua, mas não é especificado o modo como esse requisito será satisfeito. Por vezes geram-se conflitos com a escolha de dois ou mais requisitos não-funcionais incompatíveis na sua satisfação simultânea. Por exemplo, se for tão importante que o sistema seja seguro como tenha um tempo de resposta reduzido, isso implica a existência de um conflito, que terá de ser resolvido preferencialmente com a ajuda dos *stakeholders* de modo a se perceber o que é mais importante e com isso tomar uma decisão. Como este exemplo, existem outros casos em que há conflitos, mas estes podem ser detectados com algumas técnicas EROO (e. g. NFRF, KAOS).

A abordagem EROO na aquisição de requisitos contrasta com outras técnicas (ex. orientada a objectos) porque elas não capturam o "Como", mas sim o "Porquê" da existência das relações em termos de objectivos. Pode-se clarificar requisitos, na sua identificação e especificação observando os objectivos dos actores tendo em conta o futuro sistema [17]. Esta abordagem conduz a vários benefícios para ER, e estão aqui enumerados, de acordo com Lapouchnian *et al.*[16]:

(1) Os objectivos conduzem à identificação dos requisitos a eles associados. Pois os requisitos são o modo de operacionalização dos objectivos. Por sua vez, os objectivos ajudam na execução de um processo de elaboração de requisitos sistemático.

(2) Os objectivos fornecem um critério preciso para uma especificação de requisitos suficientemente completa (isto verifica-se, tendo em conta um conjunto de objectivos e caso se possa provar que todos esses objectivos são alcançados a partir das especificações e das propriedades conhecidas sobre o domínio);

(3) Os objectivos fornecem um critério preciso para a pertinência dos requisitos, evitando assim os irrelevantes. Mesmo sem um método de análise formal, é possível ver num modelo de objectivos, se um objectivo em particular realmente contribui para um objectivo descrito pelo *stakeholder*;

(4) Os objectivos podem ser usados como base para detectar e gerir conflitos entre requisitos;

(5) Um requisito aparece devido à existência de um objectivo que necessita dele. Numa árvore de refinamento de objectivos encontram-se ligações de rastreamento de objectivos de alto nível para requisitos técnicos de baixo-nível;

(6) A modelação por objectivos aumenta a facilidade de leitura (*readability*) porque fornece um mecanismo natural de estruturar requisitos complexos;

(7) Um único modelo de objectivos captura a variabilidade no domínio do problema através do refinamento de objectivos alternativos e atribuições de responsabilidades alternativas;

(8) Esta modelação fornece o nível de abstracção certo para comunicar com os clientes sobre os requisitos e validar escolhas através de várias alternativas;

(9) A separação entre a informação volátil e estável é importante, e existem pesquisas que dizem que os objectivos são mais estáveis que os requisitos ou operações.

Por tudo isto, conclui-se que as abordagens EROO fornecem um tipo de modelação com muita potencialidade para a criação de sistemas mais fiéis às necessidades dos *stakeholders*.

Existem várias abordagens EROO, mas este trabalho dará ênfase ao i* que fará parte da abordagem a ser proposta. Na secção 2.1 apresenta-se a *framework* i*, os seus conceitos básicos e os seus tipos de modelos ilustrados num exemplo.

2.1 Framework i*

A *framework* i* (leia-se *i-estrela*) apareceu com o intuito de preencher lacunas que existem em várias organizações de modo a adaptar sistemas já concebidos, ou prever possíveis alterações de um novo sistema. Se houver um conhecimento profundo sobre a organização é possível responder ao "Porquê" fazer algo e assim permitir modelar um sistema mais flexível [11].

O i* é uma abordagem centrada nos *stakeholders* do sistema e nas dependências entre eles. Os *stakeholders* são representados como actores que dependem uns dos outros para alcançarem objectivos, realizarem tarefas e fornecerem recursos [4, 18]. Para ilustrar o tipo de modelos que se pode produzir com o i*, utilizou-se o caso de estudo da *Loja de Multimédia Online*, apresentada por Castro *et al.* [19], e produzida com a ferramenta *OME 3.1* [20].

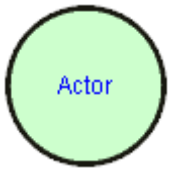
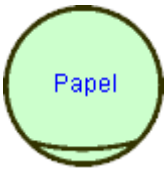
2.1.1 Conceitos Básicos

O i* está dividido em dois tipos de modelos, o modelo de Dependência Estratégica (*Strategic Dependency* – SD) e o de Razão Estratégica (*Strategic Rational* – SR). Tanto o modelo SD como o modelo SR terão as suas características descritas nesta secção. De modo a oferecer uma visão mais clara sobre a organização do i*, os conceitos básicos, de acordo com Yu [11], encontram-se em tabelas descritas a seguir.

Como aparece na Tabela 2.1 um actor é uma entidade capaz de realizar acções para atingir objectivos. O actor pode ser especificado num papel, posição ou agente.

Os actores dependem uns dos outros para conseguirem atingir os seus objectivos. Conforme Yu [11], existe o actor dependente (*depender*), o actor de quem se depende (*dependee*) e o objecto central dessa dependência (*dependum*), de modo a que as dependências ficam com o formato *depender* → *dependum* → *dependee* (Figura 2.1).

Tabela 2.1 Tipos de actores

Entidade			Descrição
Actor 	**Papel**		Usa-se o *papel* quando se deseja referir a função que um actor desempenha;
	Posição		Usa-se a *posição* quando se pretende que o actor exerça um determinado conjunto de funções;
	Agente		O a*gente* representa actores concretos, ou seja, podem ser uma pessoa ou um agente artificial.

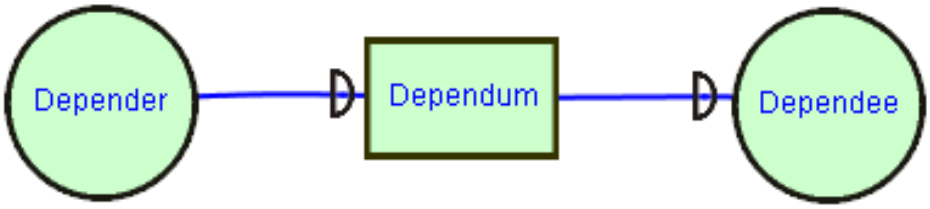


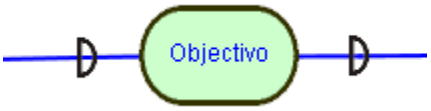
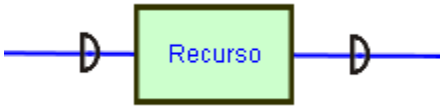
Figura 2.1 Formato da dependência

As dependências podem ser de quatro diferentes tipos. Existem dependências do tipo objectivo, tarefa, recurso ou *softgoal*, conforme descritos na Tabela 2.2.

O significado do *softgoal* encontra-se especificado em termos dos métodos que são escolhidos para realizar o objectivo. Numa dependência de objectivo, um *depender* tem a vantagem de ter a condição de realização definida, mas torna-se vulnerável no caso de o *dependee* não conseguir satisfazer essa dependência. A diferença reside nas condições para se alcançar o *softgoal*, pois estas são elaboradas à medida que as tarefas vão sendo desempenhadas [11].

Assim, no caso de ter um *softgoal* "*Pacotes de Qualidade*", que representa o objectivo de obter pacotes com qualidade, o significado de qualidade não é esclarecido. Assim sendo, considera-se esta como uma dependência de *softgoal* (exemplo representado no modelo da Figura 2.2).

Tabela 2.2 Tipos de dependências entre actores

Dependência	Representação	Descrição
Objectivo		É aquela que existe por um actor depender de outro para que uma condição no mundo seja verdadeira.
Tarefa		Existe quando um actor necessita de outro para que uma actividade seja executada.
Recurso		É usada se um actor depender de outro para ter disponível uma entidade (física ou de informação).
Softgoal		É uma variante da dependência de objectivo, ou seja, só difere deste por não ter *a priori* um critério para se atingir esse objectivo.

A cada dependência está associada um tipo de força. É possível distinguir três tipos de força: aberta, comprometida ou crítica. Na Tabela 2.3 encontram-se descritos estes tipos de força.

Tabela 2.3 Força das dependências entre actores

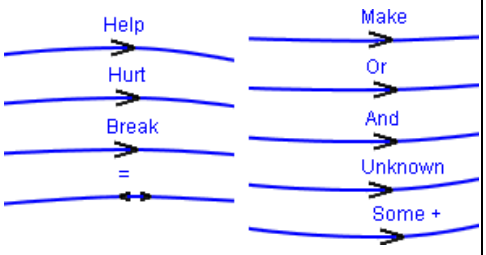
Força da dependência	Descrição
Aberta	O actor dependente gostaria de ter a dependência satisfeita, mas caso isso não se verifique, os seus objectivos não serão muito afectados.
Comprometida	O actor dependente tem objectivos que serão gravemente afectados caso a dependência não seja satisfeita. Essa dependência é motivo de preocupação para o actor dependente; enquanto o actor de que se depende tentará satisfazer esse objectivo através da satisfação das suas próprias dependências.
Crítica	O actor dependente terá objectivos que serão seriamente afectados se a dependência não for satisfeita, visto que todos os caminhos de acção conhecidos falharão.

Estas características verificam-se tanto para os modelos SD como para os SR. Mas para além destas, no caso do modelo SR ainda é possível encontrar ligações de Decomposição de Tarefas, de Meio-Fim e de Contribuição. Essas ligações são descritas na Tabela 2.4.

Na ligação de decomposição de tarefa existem quatro tipos de sub-elementos: objectivos, tarefas, recursos e *softgoals* [11].

Relativamente às ligações de contribuição, originalmente segundo Yu [11] apenas existiam as que contribuíam positiva (*Help*) ou negativamente (*Hurt*) para a satisfação de uma dependência. Mas actualmente existem mais, sendo alguns exemplos, as ligações de contribuição que realizam a dependência (*Make*), ou as que não se conhece qual será a sua influência na dependência (*Unknown*), ou ainda as que não permitem que a dependência seja satisfeita (*Break*). Este tipo de informação adicional às ligações de contribuição é útil para se ter uma melhor percepção da informação do modelo.

Tabela 2.4 Tipos de ligações

Tipos de Ligações	Representação	Descrição
Ligação de Meio-Fim		Representa um tipo de ligação que expressa um meio para atingir um fim.
Decomposição de Tarefa		Com esta ligação é possível perceber o que vai ser necessário para realizar uma tarefa.
Ligação de Contribuição	 Exemplos de contribuições	Esta ligação permite clarificar que tipo de contribuição existe para determinada situação se verificar.

Estes são os conceitos básicos necessários para desenvolver modelos i*. De seguida serão apresentados os modelos de Dependência Estratégica (SD) e de Razão Estratégica (SR), para

que se possam perceber melhor as suas composições. Para ilustrar este tipo de modelação, utilizou-se o caso de estudo de uma *Loja de Multimédia Online* descrita mais detalhadamente por Castro *et al.* [19], e desenvolvido com a ferramenta do *OME 3.1* [20].

2.1.2 Modelo de Dependência Estratégica

O modelo SD apresenta as dependências existentes entre os vários actores. São representados os actores como nós e as dependências como ligações entre esses nós.

As funcionalidades do modelo SD são apresentadas com o caso de estudo da *Loja de Multimédia Online.* Esta pretende desenvolver uma loja *online* designada por Multimédi@ para poder vender os seus produtos, como livros, jornais, CD's, vídeos, entre outros do mesmo tipo, pela internet. O sistema apresentado na Figura 2.2 representa o processo de negócio de compras antes de se desenvolver o sistema Multimédi@ para se perceber como é o funcionamento da loja. A filosofia da *Loja de Multimédia Online* baseia-se na consulta, por parte dos clientes, do catálogo fornecido pela loja e para que estes possam comprar os artigos que lhes interessam. Os artigos vendidos na Loja de Multimédia são fornecidos pela Fornecedor e este último adquire os artigos do Produtor. Portanto, com a Figura 2.2 é possível encontrar quatro actores diferentes, o Cliente, a Loja Multimédia, o Fornecedor e o Produtor enquanto entre eles existem várias dependências.

Para se construir este tipo de modelo é importante tomar algumas decisões e escolher uma maneira ou outra de as representar. Portanto, por exemplo, a compra de artigos por parte dos clientes é representada como uma dependência de objectivo da Loja de Multimédia porque existe mais do que um modo para se fazer esta compra, pois se só houvesse uma maneira de fazer compras, como ir directamente à loja, poder-se-ia usar uma dependência de tarefa em vez de objectivo. Por outro lado a consulta do catálogo é representada como uma tarefa porque está definida a maneira como se procede.

A dependência de recurso "Artigos Multimédia", que se verifica do Fornecedor para a Loja de Multimédia, existe porque o que será transaccionado é o recurso Artigo Multimédia mas também poderia ser colocada uma dependência de tarefa com o nome de "Entregar Artigos Multimédia". Esta seria previamente definida e estaria subjacente a existência do recurso "Artigos Multimédia".

No caso de se colocar a dependência de recurso e a dependência de tarefa, torna o modelo mais complexo, além de em princípio não oferecer mais informação. Consideram-se como

softgoals o "Aumentar Cota de Mercado", "Clientes Felizes", "Negócio Contínuo" e "Pacotes de Qualidade" porque deseja-se ter esses objectivos satisfeitos, porém a sua satisfação pode ser atingida em vários níveis (como por exemplo, parcialmente satisfeita, ao contrário do objectivo, que só possui dois níveis de satisfação: satisfeito e não satisfeito).

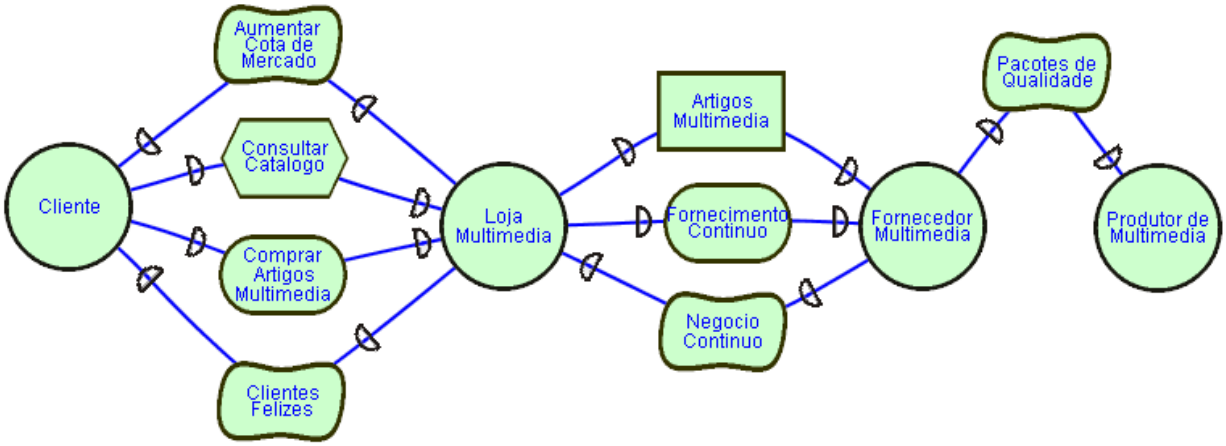


Figura 2.2 Modelo SD da Loja de Multimédia

Ilustrou-se um exemplo do modelo SD para a *Loja de Multimédia Online* dando a conhecer o processo de negócio de compra de uma forma geral, tal como cada um dos conceitos deste tipo de modelo.

2.1.3 Modelo de Razão Estratégica

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

Na Figura 2.3 modelou-se a maneira como o actor Loja Multimédia consegue satisfazer as várias dependências que estão ligadas a ele. Por exemplo, para a Loja Multimédia alcançar o objectivo "Processar Pedidos dos Clientes" pode receber as encomendas dos clientes através do telefone, por internet ou directamente na loja. Portanto, como cada uma destas três maneiras permite alcançar o objectivo utilizou-se a ligação meio-fim que representa o modo necessário (meio) para alcançar um objectivo (fim). No caso da tarefa "Pedidos pela Internet" a Loja Multimédia depende do actor Multimédi@ para lhe fornecer o que necessita, através da satisfação do objectivo "Pedidos da Internet Processados".

Para satisfazer a tarefa "Pedir Pessoalmente", que está decomposta através da ligação de decomposição em três sub-tarefas ("Seleccionar Artigos", "Determinar Quantia" e "Consultar

Catálogo"), é necessário executar cada uma das suas sub-tarefas. O *softgoal* "Serviço Melhorado" tem a contribuição da tarefa "Treinar Pessoal" e dos *softgoals* "Satisfazer Desejos dos Clientes" e "Ser Simpático". Ou seja, se os funcionários da loja tiverem formação em vendas, forem amigáveis e satisfizerem os desejos dos clientes isso contribuirá positivamente para que o *softgoal* "Serviço Melhorado" seja satisfeito. Estas ligações ajudam a perceber melhor como é possível satisfazer as várias dependências.

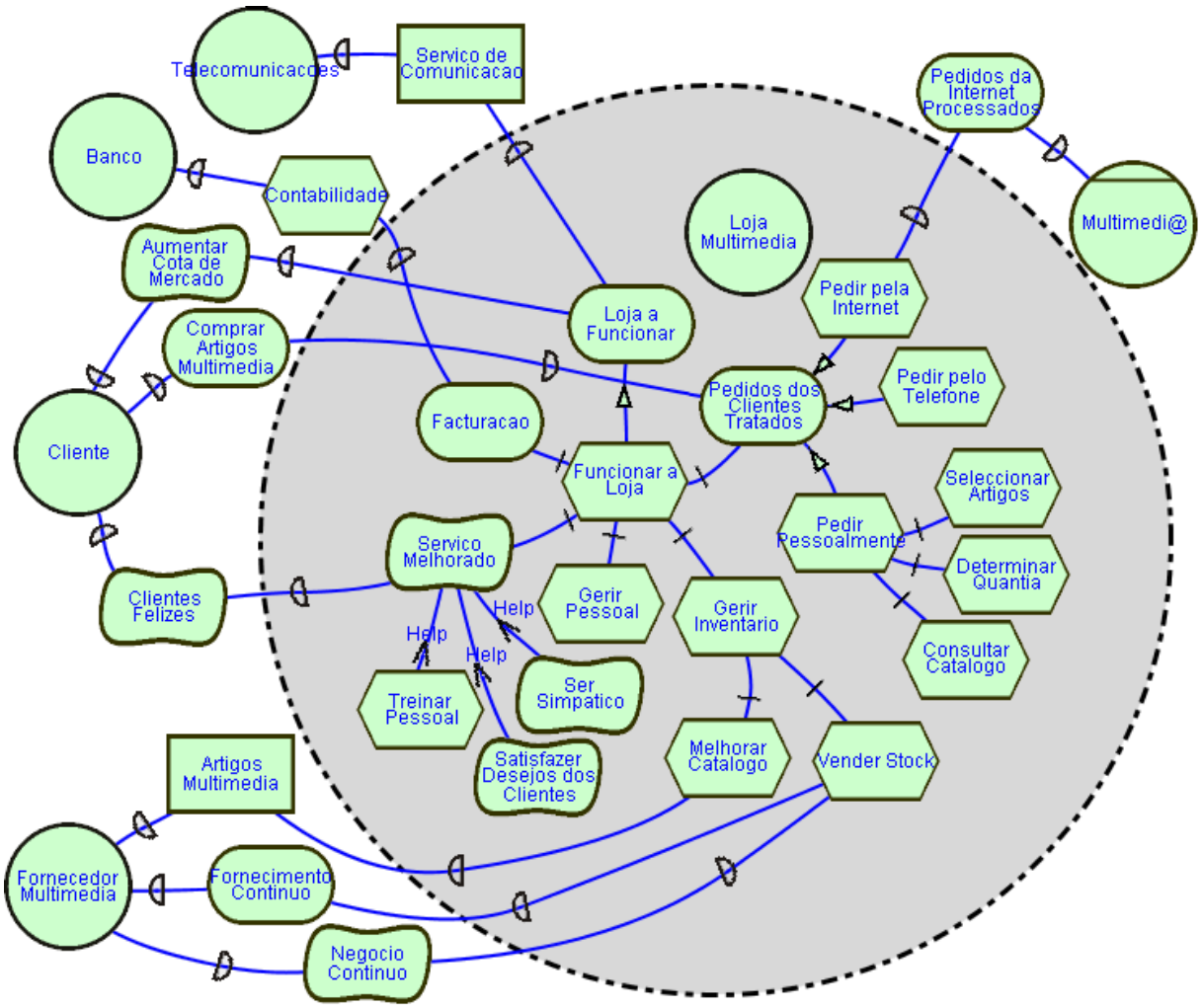


Figura 2.3 Modelo SR para a Loja de Multimédia

Os dois modelos representam o funcionamento da *Loja de Multimédia Online*, permitindo ter uma visão da organização da loja. A *framework* i* permite fazer facilmente a modelação de sistemas, mas no caso de estes serem mais complexos a compreensão dos modelos diminuirá. De qualquer modo, esta modelação é simples e ajuda a representar sistemas de acordo com os objectivos dos *stakeholders*.

Com as descrições e exemplos, pretendeu-se dar noções sobre a *framework* i*. Na secção 2.2, serão apresentadas outras abordagens baseadas em objectivos e assim proporcionar um conhecimento mais abrangente sobre EROO.

2.2 Outras Abordagens Orientadas a Objectivos

2.2.1 Framework NFR

A *framework* NFR (*Non-Functional Requirements Framework* - NFRF) [14] pretende representar e analisar requisitos não-funcionais, que são representados como *softgoals*. Por conseguinte, o *softgoal* é o conceito central desta *framework*, e representa um objectivo que não possui uma definição clara nem precisa de como é satisfeito. No NFRF os *softgoals* são classificados como satisfeitos ou rejeitados, e essa decisão é avaliada de acordo com a satisfação e contribuição dos seus descendentes. Os *softgoals* são interdependentes uns dos outros e as relações de dependência são úteis para perceber como um *softgoal* pode ser satisfeito, através da classificação dos *softgoals* que este depende.

Os principais componentes do NFRF são os *softgoals*, as interdependências, os procedimentos de avaliação, os métodos e as correlações. Os *softgoals*, que são identificados, necessitam de ser catalogados e arrumados de acordo com os seus tipos (exemplo: segurança, disponibilidade), e em hierarquias de relação *IsA* (tradução em Português "é uma") que refinam um *softgoal* inicial. Os catálogos de tipos e os métodos de decomposição são usados para decompor um *softgoal* específico. Os catálogos são úteis para futuras reutilizações em que se pretenda saber quais os requisitos que se omitiram. O NFRF não mostra como são identificados exactamente 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 NFRF está direccionado para clarificar o significado dos requisitos não-funcionais e fornecer alternativas para os satisfazer ao mais alto nível.

2.2.2 KAOS

Outra abordagem orientada a objectivos é o KAOS (*Keep All Objectives Satisfied*) [10, 16, 21]. O objectivo é o conceito mais importante do modelo KAOS, e pode ser formulado em diferentes níveis de abstracção. Este pode ser refinado através de ligações de decomposição *and* e *or*. Estas ligações relacionam um objectivo com um ou mais sub-objectivos, designados por requisitos ou expectativas [10]. Pode-se distinguir um objectivo em requisitos funcionais (*goal*) e não-funcionais (*softgoal*).

O KAOS é composto por quatro tipos de modelos: modelo de objectivos, modelo de responsabilidades, modelo de objectos e modelo de operações. Para cada um desses modelos segue-se uma breve descrição:

O modelo de objectivos modela as relações entre os objectivos.

O modelo de responsabilidades descreve para cada agente, as suas responsabilidades para com os vários requisitos e expectativas. Esses agentes encontram-se ligados a um requisito ou expectativa através de ligações de refinamento denominadas por atribuição ou responsabilidade [16].

O modelo de objectos tem as mesmas características de um diagrama de classes do modelo UML (*Unified Modeling Language*) [22], e pode ser derivado a partir da especificação formal de objectivos, desde que se refiram a objectos ou às suas propriedades.

O modelo de operações define os comportamentos necessários para que os vários agentes cumpram os seus requisitos, através de operações.

Estes são os principais conceitos da abordagem KAOS.

2.2.3 GRL

A linguagem de requisitos orientada a objectivos (do inglês, *Goal-oriented Requirement Language* – GRL) [12] faz parte da Notação de Requisitos de Utilizador (do inglês, *User Requirement Notion*), em que complementa a GRL com Mapas de Casos de Uso (do inglês, *Use Cases Maps*). O GRL é uma notação de modelação visual que combina o NFRF [13, 14] com a *framework* i*, já apresentados neste documento.

A sintaxe do GRL é baseada na sintaxe do i*, enquanto o conceito de um mecanismo de avaliação que suporta um raciocínio sobre o grafo de objectivos é baseado no NFRF. A modelação GRL ajuda no raciocínio sobre os requisitos tendo em destaque os requisitos não-funcionais.

Existem três categorias de conceitos: elementos intencionais, ligações e actores. Os elementos intencionais são os objectivos, tarefas, recursos e *softgoals* [23]. O grafo de objectivos pode documentar ainda as opiniões (*beliefs*) dos *stakeholders.*

Podem ser utilizados quatro tipos de ligações: dependência, decomposição, correlação ou contribuição. As ligações de contribuição podem ainda ter associado um identificador a referir

qual o tipo de contribuição que fornece. Também é possível ter níveis de satisfação relacionados com os elementos intencionais [13]. O GRL apoia o raciocínio sobre cenários, através da correspondência entre elementos intencionais GRL e elementos não intencionais no modelo de cenário da Notação de Requisitos de Utilizador. Com esta modelação é possível identificar novos objectivos e cenários importantes para os *stakeholders*, permitindo encontrar mais exaustivamente e com maior exactidão os vários requisitos [12].

2.2.4 V-graph

O V-graph tem esta denominação devido à forma global dos modelos gerados serem semelhantes à letra V. Nos extremos superiores do grafo encontram-se representados os requisitos funcionais e não-funcionais, respectivamente na forma de objectivos e *softgoals*. O vértice inferior representa tarefas que irão satisfazer tanto os requisitos funcionais como não-funcionais. De acordo com o que Yu *et al.* [15] referem, os requisitos não-funcionais são representados como *softgoals* porque não estão definidos especificamente como podem ser atingidos.

Baseando-se no NFRF, Yu *et al.* [15] declararam que, cada objectivo e tarefa seriam descritos com um tipo e um tópico. Assim, um tópico captura a informação contextual para o objectivo, tarefa ou *softgoal*. Para um objectivo ou tarefa, um tipo descreve uma função genérica; enquanto para um *softgoal*, um tipo descreve um atributo de qualidade (por exemplo, disponibilidade, tempo de resposta).

O V-graph pode ter correlação de objectivos para *softgoals* e contribuições de tarefas para objectivos ou *softgoals*, de objectivos para objectivos e de *softgoals* para *softgoals*. Na Tabela 2.5 [24] pode-se perceber quais os tipos de ligações que podem se verificar no V-graph e a semântica associada a cada um.

Tabela 2.5 Possíveis ligações de correlação e contribuição no V-graph

Ligação	***and***	***or***	***xor***	***make***	***help***	***unknow***	***hurt***	***break***
Contribuição	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Correlação	Não	Não	Não	Sim	Sim	Sim	Sim	Sim

2.3 Resumo

Neste capítulo ofereceu-se uma visão geral sobre alguns tipos de modelação que podem ser usados quando se pretende utilizar uma abordagem EROO. Ilustrou-se com o exemplo da *Loja de Multimédia Online* os vários conceitos dos modelos SD e SR da *framework* i*. Foram

brevemente descritas as abordagens NFRF, KAOS, GRL e por último V-graph que representam outros tipos de modelação orientadas a objectivos, pois as abordagens EROO estão a ser actualmente mais exploradas e utilizadas.

A *framework* i* permite uma boa representação dos objectivos, mas a modelação de sistemas mais complexos dificulta a leitura dos modelos. De qualquer modo, o i* permite a modelação de várias alternativas para satisfazer um objectivo nos modelos SR, de acordo com as intenções dos *stakeholders*, e isso permite que se possa tirar vantagem para utilizar na engenharia de requisitos de Linhas de Produto de *Software*. No capítulo 3 será abordado o tema de Linhas de Produto de *Software*, onde serão apresentados os conceitos básicos, as actividades envolvidas na engenharia de Linhas de Produto de *Software* e o modelo de *features*.

3. Linha de Produto de *Software*

A utilização de produtos de *software* em geral tem aumentado, o que influência as empresas a lançarem novos produtos com maior frequência. A produção separada de vários produtos semelhantes de *software* faz com que estes demorem mais tempo para serem lançados, do que se pertencessem a uma linha de produtos. Consequentemente, várias empresas adoptaram a abordagem de Linhas de Produto de *Software* (LPS), que se baseia na reutilização de *software*, construindo uma família de aplicações em vez de desenvolver produtos separadamente. Uma LPS é um conjunto de produtos de *software* que têm em comum a maioria das suas características, mas existem diferenças específicas entre elas (a esse conjunto denomina-se por família de produtos) [5].

A Engenharia de Linha de Produtos de *Software* (ELPS) é direccionada para o desenvolvimento de aplicações de *software* utilizando plataformas e produção em larga escala de bens adequados às necessidades individuais dos clientes [6]. Denomina-se por plataforma (do inglês, *platform* ou também designado de *core assets*), um conjunto de subsistemas e interfaces que formam uma estrutura comum para que cada conjunto de produtos derivados possa ser efectivamente desenvolvido e produzido [6].

Para ELPS são utilizados dois processos, o de Engenharia de Domínio e o de Engenharia de Aplicação. A Engenharia de Domínio modela todas as características possíveis das funcionalidades da linha de produtos; enquanto a Engenharia de Aplicação configura e produz vários produtos, combinando para cada produto, várias funcionalidades modeladas no processo de Engenharia de Domínio [6].

Com o uso de LPSs, conseguir-se-á utilizar mais eficientemente os recursos humanos, reduzindo o tempo de colocação de produtos no mercado, melhorando a qualidade do *software* e aumentando a produtividade em grande escala, o que implica uma maior satisfação do cliente. Estes são alguns dos benefícios da utilização de linhas de produto de *software* em vez de produzir vários sistemas únicos semelhantes.

Na Figura 3.1 encontra-se representada a *framework* de ELPS segundo Pohl [6]. Os subprocessos de engenharia de domínio e de aplicação, bem como, as suas actividades

encontram-se representados num ciclo, mas estes não são obrigatoriamente realizados sequencialmente.

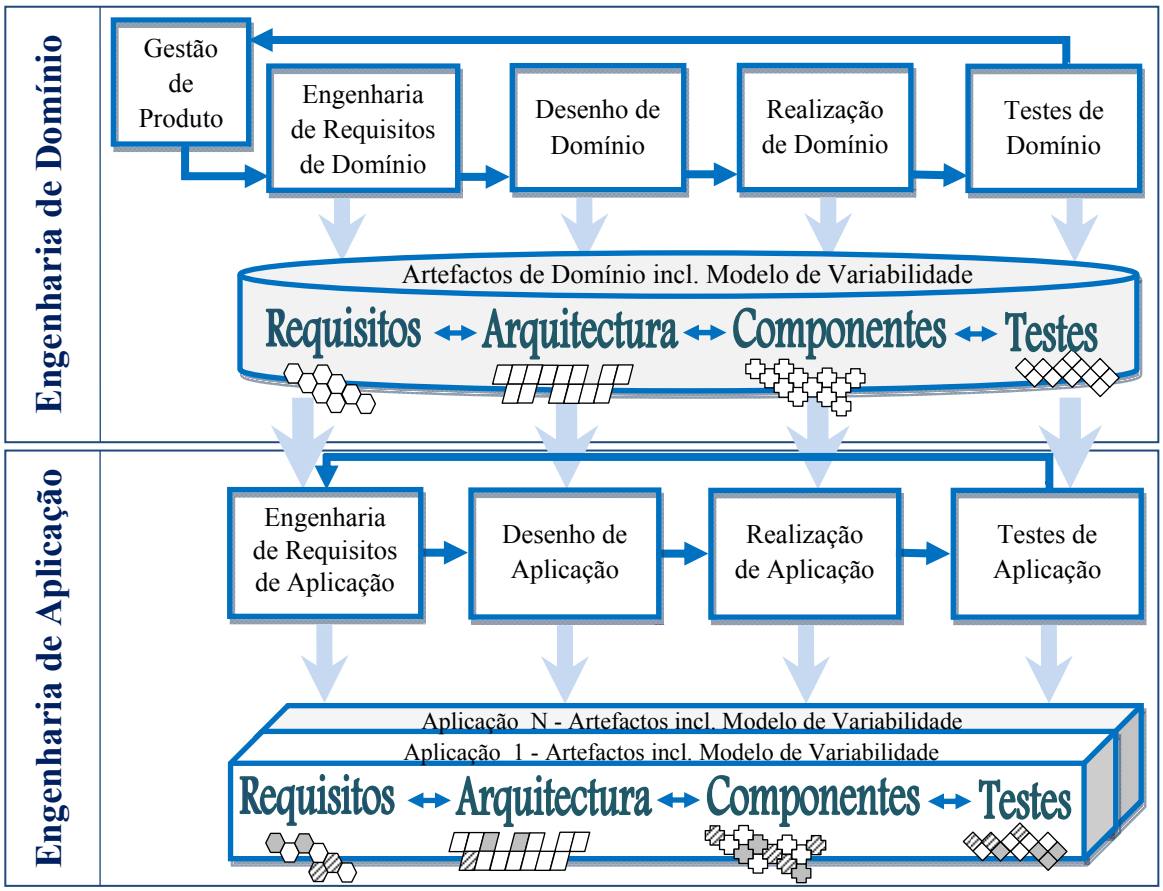


Figura 3.1 ***Framework*** **de engenharia de linha de produtos de** ***software***

Segundo Clements e Northrop [5], o processo de Engenharia de Domínio também pode ser designado por Desenvolvimento de Artefactos Base (*Core Assets Development*), e o processo de Engenharia de Aplicação por Desenvolvimento de Produto (*Product Development*). Tanto o desenvolvimento de artefactos base como o desenvolvimento de produto são duas actividades de uma LPS, e são supervisionadas pela actividade de gestão técnica e organizacional (*Management*). Estas são as três actividades principais de uma LPS.

Nas secções 3.1 e 3.2 são apresentados os dois processos de ELPS tal como cada um dos seus subprocessos.

3.1 Engenharia de Domínio

Segundo Pohl [6], a Engenharia de Domínio é o processo de ELPS em que os pontos comuns e variáveis da linha de produto são definidos e perceptíveis. Por isso, essas semelhanças e diferenças dos produtos de uma LPS são importantes para se analisar o domínio. Na análise

de domínio serão encontradas essas características do conjunto de produtos, e aos produtos derivados dessa linha dá-se o nome de família de produtos [25].

São descritos sumariamente os vários subprocessos de engenharia de domínio, de acordo com Pohl [6]:

- O objectivo da Gestão de Produto é integrar o desenvolvimento, a produção e o *marketing* de produtos para maximizar o investimento e ir ao encontro das necessidades dos clientes. A estratégia de *marketing* utilizada tem em conta os aspectos económicos da LPS. A gestão de produto utiliza técnicas objectivas para definir o que está dentro do escopo de uma linha de produto e o que está fora;

- Na Engenharia de Requisitos de Domínio pretende-se identificar os requisitos. Portanto é feita a análise de pontos comuns e variáveis entre produtos da LPS. Apresenta ainda a documentação de todos os requisitos (iguais e diferentes) da linha de produtos;

- Define-se no Desenho de Domínio a arquitectura 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 arquitectura de referência fornece uma estrutura de alto nível, comum a todas as aplicações da linha de produtos;

- No subprocesso de Realização de Domínio, são tratados os detalhes do desenho e da implementação dos componentes de *software* reutilizáveis e das suas interfaces;

- A responsabilidade dos Testes de Domínio consiste em validar e verificar os componentes reutilizáveis, para evitar que os erros sejam propagados para o processo de testes de aplicação. Pode-se desenvolver uma ou mais aplicações de exemplo, para se proceder aos testes como se fossem sistemas únicos. Ou então, produzir testes para as partes comuns da engenharia de domínio e, desenvolver testes variáveis para aplicar nas componentes específicas das aplicações durante os testes de aplicação.

A Engenharia de Domínio é constituída por estes cinco subprocessos, sendo os resultados destes importantes para se poder realizar a Engenharia de Aplicação eficientemente. Na secção 3.2 será apresentado o segundo processo da *framework* de ELPS, a Engenharia de Aplicação.

3.2 Engenharia de Aplicação

Segundo Pohl [6], a Engenharia de Aplicação é o processo de ELPS no qual as aplicações são construídas através da reutilização dos artefactos de domínio e pela exploração da variabilidade da linha de produtos. Esses artefactos de domínio são os resultados dos vários subprocessos da engenharia de domínio.

A engenharia de aplicação é constituída pela engenharia de requisitos de aplicação, o desenho de aplicação, a realização de aplicação e os testes de aplicação. Serão brevemente descritos esses quatro subprocessos, conforme Pohl [6]:

- O objectivo da Engenharia de Requisitos de Aplicação é identificar os requisitos dos *stakeholders* para a aplicação. É necessário fazer um mapeamento entre os requisitos da aplicação e os requisitos do processo de engenharia de domínio. Pretende-se reutilizar o máximo possível dos requisitos de domínio, mas isso depende bastante dos requisitos da aplicação. Assim, a principal preocupação deste subprocesso é detectar diferenças entre requisitos de aplicação e os disponíveis na plataforma de LPS (produzidos na engenharia de requisitos de domínio). É feita a documentação do modelo de variabilidade da aplicação, que contém o modelo de variabilidade de domínio com os dados correspondentes aos pontos que a aplicação reutiliza;

- O Desenho de Aplicação contém as actividades de produção da arquitectura da aplicação, que é semelhante ao desenho de *software* de um sistema único. A arquitectura da aplicação determina toda a estrutura de uma aplicação específica e tem de ser capaz de satisfazer os requisitos dessa aplicação. Utiliza-se a arquitectura de referência para iniciar a arquitectura da aplicação, pois ela contém o desenho de muitos dos requisitos da aplicação. Por isso, seleccionam-se e configuram-se as partes requeridas da arquitectura de referência e incorporam-se adaptações específicas da aplicação;

- No subprocesso de Realização de Aplicação são tratados todos os detalhes de desenho, de implementação e configuração de componentes. É produzida a aplicação requerida, tendo em conta, a selecção 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;

- Os Testes de Aplicação consistem em actividades necessárias para validar e verificar uma aplicação de acordo com a sua especificação. Para isso, é necessário definir um conjunto de testes que se apliquem adequadamente à aplicação a ser testada. É preciso também testar o código reutilizado, para garantir que não surgiram erros ao juntar com o código específico da aplicação. É útil ter uma documentação dos testes, para que se possa repetir o processo de teste da aplicação.

Estes são os processos e subprocessos que constituem a *framework* de ELPS. Para produzir uma linha de produtos é importante analisar a sua variabilidade, ou seja, o que permite produzir produtos com as mesmas características base, mas com diferenças que ajudam a dar maior resposta às necessidades individuais dos clientes. A variabilidade pode ser modelada com modelos de *features*. Esses modelos serão apresentados mais detalhadamente na secção 3.3.

3.3 Modelo de *Features*

Os modelos de *features* são usados para facilitar a documentação da informação da variabilidade [26]. Eles são desenvolvidos no subprocesso de Engenharia de Requisitos de Domínio, quando se faz a análise e identificação dos requisitos da LPS. Também são utilizados no subprocesso de Engenharia de Requisitos de Aplicação, quando se faz a selecção das *features* para uma aplicação específica.

Os modelos de *features* são importantes para a representação de *features* de uma LPS, pois eles descrevem as configurações válidas possíveis [27] e são de fácil compreensão. Originalmente o modelo de *features* foi proposto como parte do método de Análise de Domínio Orientada a *Features* (do inglês, *Feature-Oriented Domain Analysis*) [28]. Entretanto, acabou por ser aplicado noutros negócios e domínios técnicos [29], pois este tipo de modelo é útil para representar as capacidades essenciais que satisfazem as necessidades dos clientes, ou seja, ele foca as funcionalidades na perspectiva do utilizador (os "serviços" fornecidos pela aplicação) [28].

O modelo de *features* será apresentado de acordo com Czarnecki *et al*. [29], ou seja, suportando cardinalidade. Por isso, ele é designado por modelo de *features* baseado em cardinalidade (do inglês, *cardinality-based feature modeling*). Czarnecki *et al*. [29] afirmaram que os modelos de *features* baseados em cardinalidade são desejados principalmente a nível de requisitos, pois eles mostram directamente o impacto da selecção de uma dada *feature* no modelo final.

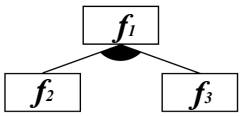
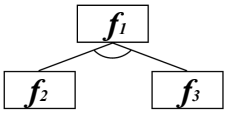
3.3.1 Conceitos do Modelo de *Features*

Cada *feature,* de um modelo de *features,* é uma propriedade do sistema relevante para alguns *stakeholders* e usada para capturar características iguais e diferentes entre produtos de uma mesma família [29]. As *features* estão organizadas num diagrama, sendo este uma árvore com a raiz a representar um conceito (e.g. um sistema de *software*). Um modelo de *features* é constituído por um ou mais diagramas de *features*, que as organiza hierarquicamente e tem informação adicional para obter melhor documentação sobre as *features* (tem por exemplo, a descrição da *feature*, as prioridades e os *stakeholders*) [29, 30]. Cada *feature* pode ser refinada em *sub-features*.

Uma *feature* pode ser do tipo obrigatória, opcional ou alternativa. No caso de ser alternativa, pode-se ainda fazer a separação entre as alternativas *or* (ou) ou *xor* (*ou-exclusivo*).

A Tabela 3.1 apresenta a descrição e a representação para cada um dos tipos de *features*.

Tabela 3.1 Descrição dos tipos de *feature*

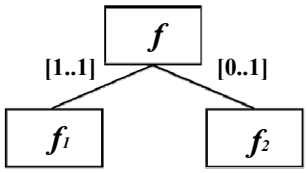
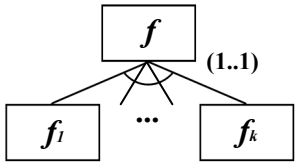
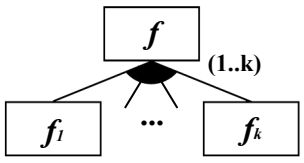
Tipo de *feature*		**Representação**	**Descrição**
Obrigatória			A *feature f* está presente em todas as aplicações possíveis.
Opcional			A *feature f* poderá ou não estar presente nas aplicações.
Alternativa	OR		Pelo menos uma das *sub-features* f_2, f_3 terá de estar presente na aplicação que tem a *feature* f_1 seleccionada.
	XOR		Apenas uma das *sub-features* f_2, f_3 poderá estar presente na aplicação que tem a *feature* f_1 seleccionada.

Czarnecki *et al.* [29] propuseram colocar cardinalidade nas *features* para poder remover as ambiguidades e representar a informação mais facilmente.

A cardinalidade é um intervalo que denota a quantidade de vezes que uma *feature*, com as suas *sub-features*, pode ser clonada como descendente, quando é especificada uma aplicação.

É possível criar grupos de *sub-features* para ajudar a identificar conjuntos em que é obrigatório ter pelo menos uma *sub-feature* escolhida, ou em que não se pode escolher *sub-features* simultaneamente. As possíveis cardinalidades encontram-se descritas na Tabela 3.2.

Tabela 3.2 Cardinalidade de uma *feature*

Multiplicidade	**Descrição**
	A cardinalidade da *sub-feature* f_1 implica a sua obrigatoriedade numa aplicação. No caso da *sub-feature* f_2 a cardinalidade indica que esta *feature* é opcional.
	Tem de se escolher uma única *sub-feature* no grupo de f_1 a f_k (*xor*).
	Tem de se escolher pelo menos uma *feature* de um grupo de *sub-features* (*or*).

A Figura 3.2 representa uma parte do modelo da *Loja de Multimédia Online* num modelo de *features* baseado em cardinalidade, conforme Czarnecki *et al.* [27], produzido com a ferramenta *CapitainFeature 1.0* [31].

Neste modelo são *features* obrigatórias, por exemplo, o "*Payment*" e o "*Shipping*". O "*PaymentTypes*" tem uma alternativa *or* com as *features "PurshaseOrder", "DebitCard*" e "*CreditCard*", em que é necessário escolher pelo menos uma das opções. A *feature* "*FraudDetection*" é opcional. A *feature* "*CustomMethods*" tem uma *sub-feature* com uma cardinalidade que indica que terá de aparecer pelo menos uma vez numa qualquer aplicação.

No caso da *feature* "*ShippingGateways*", de acordo com a notação da ferramenta, esta aparece com um quadrado no canto superior direito para identificar que tem *sub-features*, mas estas encontram-se noutro diagrama e não são relevantes para a ilustração do modelo. A *feature* "*Expiration*" contém um grupo de *sub-features* numa alternativa *xor*, ou seja, só uma delas poderá aparecer numa aplicação. No caso da *feature* "*Chars*", esta também tem um grupo de *sub-features* numa alternativa *xor* mas com uma cardinalidade que indica que entre duas a quatro *sub-features* terão de ser usadas numa aplicação. Utilizou-se uma seta a tracejado até à "*EMedia*" desde as três *sub-features* "*Payment*", "*PasswordPolicy*" e "*Shipping*", porque a "*EMedia*" requer essas *features*.

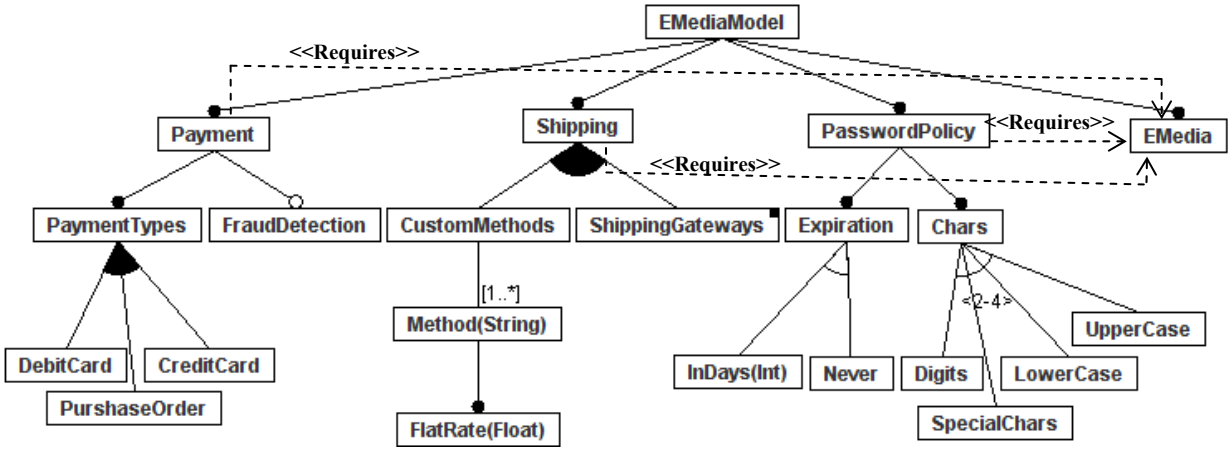


Figura 3.2 Parte do modelo de *features* sobre a *Loja de Multimédia Online,* retirada de [27]

Esta é uma das possíveis representações para um modelo de *features* (as *features* poderiam ter uma representação do tipo de directórios), e para se ilustrar esse modelo utilizou-se uma parte do exemplo da *Loja de Multimédia Online*.

3.4 Resumo

Nesta secção apresentou-se a ELPS que permite desenvolver famílias de produtos. São apresentados os processos de Engenharia de Domínio e de Aplicação, tal como os seus subprocessos, que descrevem os vários passos do desenvolvimento de uma linha de produtos. Foi explorado o modelo de *features* que representa a variabilidade de uma LPS. Foram explicadas noções deste modelo e ilustrou-se um modelo de *features* com o exemplo da *Loja de Multimédia Online* para descrever os vários conceitos de *features* e cardinalidades.

A ELPS está a ser cada vez mais utilizada para produzir produtos de uma forma mais económica e em maior escala. De qualquer modo poderão ser melhor aproveitadas se na engenharia de requisitos houver uma identificação e análise de requisitos que permita

encontrar mais eficientemente os requisitos do sistema. Portanto se for usada uma abordagem EROO nos subprocessos de Engenharia de Requisitos de Domínio e Aplicação poder-se-á fazer uma análise mais completa ao encontrar a variabilidade dos produtos numa fase inicial de desenvolvimento do *software*, pois os objectivos são definidos numa fase inicial do *software*.

No capítulo 4 serão apresentados alguns trabalhos relacionados com Linhas de Produtos de *Software*.

4. Trabalhos Relacionados

Actualmente as LPSs estão a ser mais utilizadas o que implica que apareçam abordagens que as relacionem com outros tipos de modelos (por exemplo, orientados a objectos ou objectivos). Seguidamente serão brevemente apresentadas algumas dessas abordagens.

4.1 Casos de Uso e LPS

Em [32] Gomaa apresenta uma abordagem orientada a casos de uso que pretende estender casos de uso para modelarem LPSs. Para que os casos de uso suportem as linhas de produtos, criaram-se três novos tipos de requisitos funcionais: os centrais (*kernel*), os opcionais e os alternativos. Foram também introduzidos novos conceitos de pontos de variação para modelar a variabilidade dentro dos casos de uso. Com essas novas alterações foi necessário adaptar também os conceitos de *extends* e *includes* para suportar a variabilidade em LPS. Para utilizar os casos de uso com linhas de produto Gomaa utilizou o modelo de *features*. Considerou que cada *feature* pode corresponder a: um único caso de uso, um grupo de casos de uso ou um ponto de variação dentro de um caso de uso e, assim os casos de uso e as *features* complementam-se.

De acordo com Gomaa, também pode haver o mapeamento dos casos de uso para *features* considerando as propriedades de reutilização de ambos. Para modelar os casos de uso para *features* Gomaa considerou: a modelação de um grupo de casos de uso reutilizáveis conjuntamente será então mapeado para uma *feature* e representados através de um pacote de casos de uso; um elemento *feature* poderá corresponder a uma metaclasse; ou então, poderá descrever o relacionamento de um caso de uso com uma *feature* em forma tabular, de um modo conciso. Deste modo, esta abordagem pretende permitir tirar vantagens da larga utilização do UML conjugando-o com LPS. No entanto, não são tratados os requisitos não-funcionais de uma LPS, e apenas os requisitos funcionais são modelados. Além disso, os casos de uso são orientados a objectos implicando a necessidade de desenvolver o modelo de *features* numa fase posterior à que os modelos orientados a objectivos permitem.

4.2 i* Aspectual e Modelo de *Features*

A abordagem descrita por Silva *et al.* em [33] pretende mostrar que o i* aspectual consegue suportar a variabilidade necessária para linhas de produto de *software*. São apresentadas heurísticas para permitir fazer o mapeamento para o modelo i* aspectual através do modelo de *features.* As heurísticas são: separação de *features* em modelos i*; tipos de *features* e relações entre *features* em modelos i*; verificar a corretude dos relacionamentos e; mapear o nome das *features* para os nomes dos elementos do modelo do i* aspectual. Com estas heurísticas procedesse ao mapeamento manual das *features* para o i* aspectual, através da extracção da informação comportamental dos *stakeholders.*

A variabilidade no modelo i* aspectual é apresentada tanto para a engenharia de domínio como para a engenharia de aplicação da LPS. Logo, segundo Silva *et al.*, esta abordagem representa a variabilidade existente nos objectivos dos *stakeholders* e responde ao porquê de uma determinada configuração ser necessária, dando significado às várias *features*. Contudo, esta abordagem está baseada na premissa que considera que cada *feature* opcional ou alternativa irá ser mapeada num aspecto, e isso nem sempre se verifica. Sabendo que um aspecto é um requisito que é transversal a vários conceitos do sistema (do inglês, *crosscuting concern*), pode haver uma *feature* opcional ou alternativa que não é um aspecto. Por exemplo, a *Loja de Multimédia Online* pode ter uma *feature* opcional "Catálogo" que não é um aspecto, mas segundo a heurística proposta nesta abordagem esta *feature* seria representada como tal. E ainda, esta abordagem necessita de ser melhorada para reduzir o aumento exponencial dos modelos, pois devido à adição do elemento que representa o conceito aspectual, o modelo fica muito denso dificultando a sua compreensão e legibilidade.

4.3 Abordagem EROO e Modelo de *Features*

Yu *et al.* apresentam em [34] uma abordagem dirigida por modelos para passar de um modelo de objectivos descritos como a abordagem *framework* i*, que forneça as várias possibilidades dos requisitos iniciais, para um modelo de *features,* que estrutura a variabilidade do sistema. A intenção deles é usar a satisfação de *softgoals* como critério para configurar as *features* a partir das preferências dos *stakeholders*.

Esta abordagem é apoiada por uma ferramenta chamada OpenOME que permite gerar um modelo de *features* inicial através dos objectivos dos *stakeholders* descritos em modelos i*. Yu *et al.* afirmam que a ferramenta que desenvolveram suporta a manutenção da rastreabilidade entre os objectivos e as *features*, e permite a configuração das *features* através

do uso de algoritmos de raciocínio sobre objectivos. No entanto, esta ferramenta apenas recorre aos objectivos e *softgoals* para desenvolver o modelo de *features* inicial, descartando com isto os outros conceitos fornecidos pelo modelo de objectivos que permitem especificar mais a variabilidade da LPS. Assim, seria útil tirar maior partido da ferramenta estendendo-a para ser utilizada num desenvolvimento mais detalhado do modelo de *features* e não só para obter uma primeira versão deste.

4.4 LPS e MATA

Em [35] é proposta por Jayaraman *et al.* uma abordagem para manter a separação das *features* durante a modelação do ciclo de vida e, uma abordagem para detectar interacções estruturais indesejadas entre os diferentes modelos de *features*. As *features* base são expressas em modelos UML (diagramas de classe, sequência e estados). As *features* que variam são especificadas em UMLT, ou seja, numa representação UML de transformação de grafos que indicam como é feita a modificação dos modelos base.

Assim, com a ajuda de uma ferramenta de suporte a esta abordagem, obtêm a composição de *features* automaticamente usando um mecanismo de reescrita de grafos e, utilizam uma dupla análise crítica para detectar interacções de *features* estruturais. Essas interacções servem para mostrar as inconsistências entre o diagrama de dependência de *features* com os modelos UML das *features*. Este é um exemplo em que se usou MATA (*Modeling Aspects using a Transformation Approach*) [36] mas para linhas de produtos, pois podem-se adicionar novas *features* ou combinar *features* facilmente. Entretanto, como é uma modelação orientada a objectos é necessário ter a informação mais detalhada, para se poder aplicar esta abordagem, do que quando se trata de uma abordagem orientada a objectivos, como no caso da *framework* i*.

4.5 Resumo

Estes trabalhos mostram algumas abordagens na área de Linhas de Produtos de *Software.* Também são apresentadas propostas de utilização de modelos orientados a objectivos, que oferecem benefícios na fase inicial do desenvolvimento de sistemas, com a identificação e análise de requisitos.

No próximo capítulo será apresentada a abordagem IStarLPS que permite adaptar a *framework* i* para linhas de produtos.

5. Abordagem IStarLPS

Neste capítulo será apresentado o processo que descreve uma abordagem designada por IStarLPS que adapta a *framework* i* para linhas de produto de *software*. Este processo mostra como desenvolver um modelo de *features* recorrendo à informação dos modelos i*. Também estão incluídos neste capítulo, um exemplo que ilustra o processo e os metamodelos relativos às abordagens da *framework* i*, do modelo de *features* e da IStarLPS.

Na secção 5.1 será apresentado o processo detalhado de utilização da abordagem IStarLPS, que se encontra em conformidade com o metamodelo da secção 5.3.3. Este processo será ilustrado com um exemplo desenvolvido na secção 5.2. No fim deste capítulo serão apresentadas algumas considerações finais sobre a abordagem proposta.

5.1 Processo de Utilização da Abordagem IStarLPS

O processo da abordagem IStarLPS é composto por duas fases, tal como na engenharia de linhas de produtos apresentada no Capítulo 3. Inicialmente obtém-se uma modelação a nível de engenharia de domínio para a linha de produtos de *software*, ilustrada na Figura 5.1 e posteriormente configura-se um produto a nível de engenharia de aplicação, como na Figura 5.2.

Segundo a Figura 5.2 configura-se primeiro o modelo de *features* (pois aqui são seleccionadas mais facilmente as *features* de um determinado produto), e depois os modelos i*. Assim é possível ficar com a documentação completa sobre estes modelos para uma aplicação específica.

O processo apresentado nas Figura 5.1 e Figura 5.2 encontra-se dividido por engenharia de domínio e engenharia de aplicação, como acontece na engenharia de LPS. A Figura 5.1 indica que se produz primeiro o modelo SD e depois o modelo SR, como na *framework* i*. Assim, o processo que se apresenta nesta secção tira partido dos conhecimentos previamente adquiridos da utilização das abordagens de engenharia de LPS e da *framework* i*, em separado.

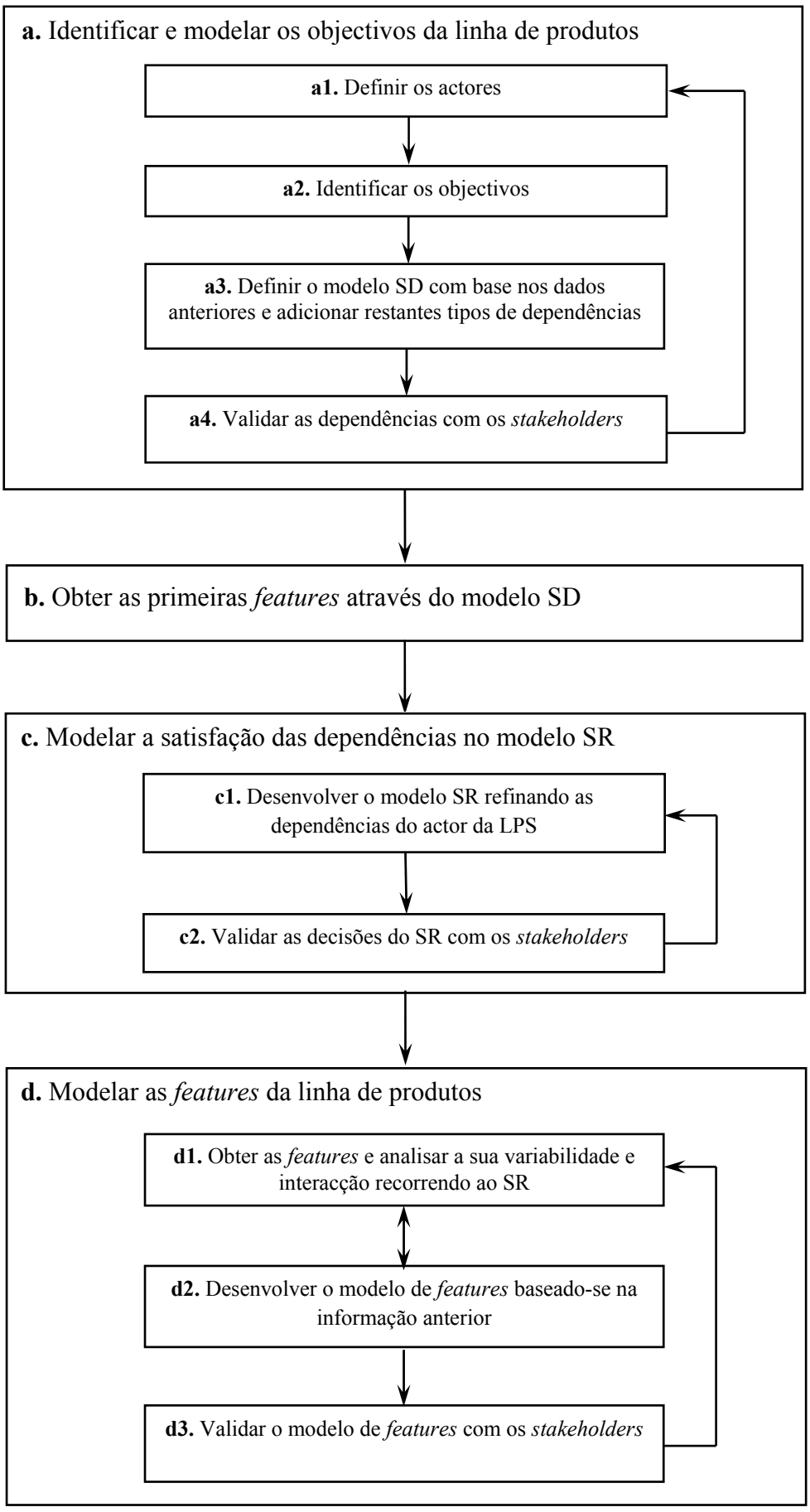


Figura 5.1 Processo a nível de engenharia de domínio

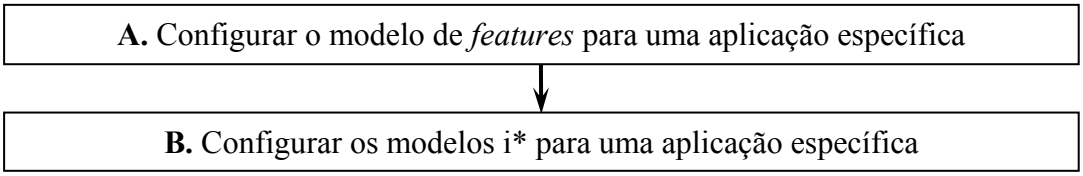


Figura 5.2 Processo a nível de engenharia de aplicação

A *framework* i* permite modelar os objectivos dos *stakeholders* para a LPS, estruturando as várias opções para a satisfação das dependências entre os actores. Após validar essas opções, pode-se desenvolver o modelo de *features* baseando-se em informação estruturada. O processo apresentado para a engenharia de domínio, na Figura 5.1, revela como se pode obter o modelo de *features* através da *framework* i*. Enquanto, o processo de engenharia de aplicação apresentado na Figura 5.2 mostra como se procederá a configuração dos modelos.

Na secção 5.2 será ilustrado cada passo do processo com o exemplo da *Loja de Multimédia Online* [19], apresentado no Capítulo 2 para ajudar na compreensão da abordagem.

5.2 Aplicação da Abordagem IStarLPS

Como visto na Secção 5.1, este processo está dividido em engenharia de domínio e engenharia de aplicação. Assim pode ser definido o domínio da linha de produtos com as várias opções e, posteriormente pode-se configurar os modelos para uma aplicação concreta.

5.2.1 Processo a Nível de Engenharia de Domínio

Utiliza-se o exemplo do sistema da *Loja de Multimédia Online*, para ilustrar os vários passos do processo da abordagem IStarLPS.

a. Identificar e modelar os objectivos da linha de produtos.

Inicialmente será produzido o modelo SD da *framework* i*. Para isso, definem-se os principais objectivos e os *stakeholders*, tal como no desenvolvimento dos modelos i* para um único sistema. O modelo SD será construído através da definição de actores, identificação de objectivos e desenvolvimento do modelo, sendo validado com a ajuda dos *stakeholders*. Caso exista algo a ser alterado, volta-se a proceder a cada uma das actividades, prosseguindo com os restantes passos após a validação positiva por parte dos *stakeholders*. Os dados sobre a LPS são adquiridos do mesmo modo como para um sistema único, ou seja, através dos *stakeholders* e descrição do sistema.

Definir actores. Os actores identificados para a *Loja de Multimédia Online* são:

- Banco;
- Cliente;
- Loja Multimédia;
- Fornecedor;
- Telecomunicações;
- Multimédi@.

Identificar os objectivos. Os objectivos identificados são:

- Artigos Multimédia Comprados;
- Transacção de Dinheiro *Online* Processada;
- Encontrados Novos Artigos Necessários aos Clientes;
- Pagamentos Processados;
- Estatísticas Fornecidas;
- Fornecimento Contínuo.

Definir o modelo SD. Nesta fase estabelecem-se as ligações de dependência entre os vários actores. Também são identificados os restantes tipos de dependências. As várias dependências vão sendo acrescentadas gradualmente, tal como no modelo SD para um único sistema (i.e., que não seja uma LPS). Pois, usualmente os modelos SD têm um actor a representar o sistema único, mas neste caso, como se está a modelar uma LPS, o actor do sistema que se está a desenvolver representará essa LPS, ou seja, o actor Multimédi@.

Na Figura 5.3 estão representados alguns dos actores definidos anteriormente. Entre os actores encontram-se as dependências de objectivos, também definidos anteriormente mas agora com a indicação do actor *dependee* e do actor *depender*. Neste modelo o Cliente depende da Loja Multimédia para comprar os artigos multimédia. Para a Loja Multimédia satisfazer o pedido do Cliente, necessita de ter um fornecimento contínuo do Fornecedor. O Fornecedor poderá saber quais os novos artigos que devem disponibilizar através da Multimédi@. A Multimédi@ necessita do Banco para saber se uma transacção de dinheiro *online* foi processada. Por fim, a Loja Multimédia necessita da Multimédi@ para obter estatísticas e para saber quais foram as encomendas processadas *online*. As Telecomunicações não dependem nem têm dependências de objectivos e por isso não aparecem nesta versão.

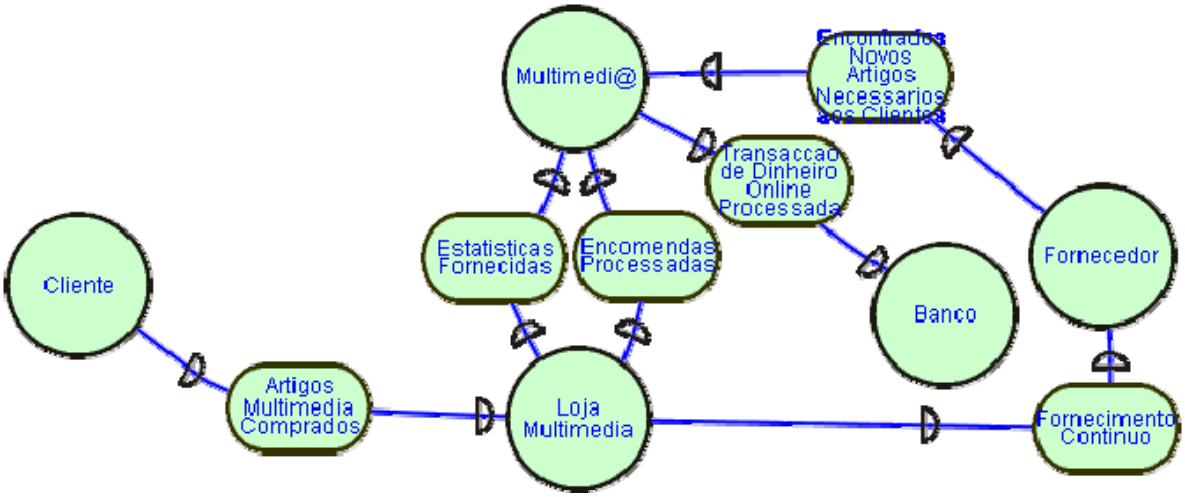


Figura 5.3 Primeira versão do modelo SD para a *Loja de Multimédia Online*

Seguidamente, procede-se à introdução das restantes dependências que se obtêm devido à informação dos *stakeholders* ou, caso sejam dependências vitais para o funcionamento das aplicações da LPS, como ilustrado na Figura 5.4. As restantes dependências não têm uma ordem específica para serem referidas, mas para facilitar a descrição, apresentam-se primeiro as dependências de tarefa e depois de recurso.

É importante relembrar que a *framework* i* foi desenvolvida para modelar sistemas únicos, por isso não foi criada tendo em conta informação que é indispensável a uma linha de produtos. Para ajustar os modelos i* foi necessário adicionar-lhes informação. Essa informação é a cardinalidade que aparecerá directamente na descrição dos elementos intencionais nos modelos i*.

Por exemplo, se numa linha de produtos a pesquisa utilizando uma palavra-chave é opcional, é útil ter essa informação nos modelos i*. A opcionalidade pode ser indicada nos modelos através da cardinalidade [0..1], directamente na descrição do elemento intencional que representa essa situação. Portanto, neste exemplo poderá aparecer essa informação num elemento intencional designado por "Pesquisar [0..1] Palavra-Chave" ou só "[0..1] Palavra-Chave" e, facilita a compreensão caso essa cardinalidade anteceda a expressão donde se obterá a *feature*. A cardinalidade nos elementos intencionais do modelo SD reflecte-se no refinamento do modelo SR, visto o SR apresentar opções para satisfazer as dependências do modelo SD. Esta cardinalidade vai ajudar no desenvolvimento do modelo de *feature* que será obtido através da informação dos modelos i*, como será ilustrado no ponto **d.**. É importante salientar que a cardinalidade pode ser ajustada durante o desenvolvimento dos modelos i*.

Relativamente ao modelo SD da Figura 5.3 não houve necessidade de inserir cardinalidade em nenhum elemento, como se poderá perceber após explicar as várias situações em que é útil adicionar essa cardinalidade, no ponto **c.**. Por sua vez, o modelo SD da Figura 5.4 apresenta o recurso "[0..1] Palavra-Chave" onde se indica através da cardinalidade [0..1] que a pesquisa por palavra-chave é opcional. Desta maneira, adianta-se que a *feature* obtida a partir do recurso "Palavra-Chave" será opcional no modelo de *feature*, de qualquer modo serão apresentados outros tipos de cardinalidades no modelo SR.

Na Figura 5.4 o Cliente depende da Multimédi@ para realizar as tarefas "Mostrar Catálogo" e "Fazer Encomenda", por sua vez a Multimédi@ necessita do Cliente para obter o recurso "Palavra-Chave" e, todas estas dependências existem porque são importantes para os *stakeholders.* Os recursos "Serviço de Internet" e "Serviço de Comunicação" existem porque são vitais para o funcionamento dos sistemas, pois sem os quais não poderá haver a Multimédi@.

A Loja Multimédia depende do Fornecedor por causa do recurso "Artigos Multimédia" que são essenciais para a Loja Multimédia poder vender os seus artigos. Além disso, a Loja Multimédia também depende do Banco para obter a tarefa "Contabilidade" sobre os artigos pagos *online*.

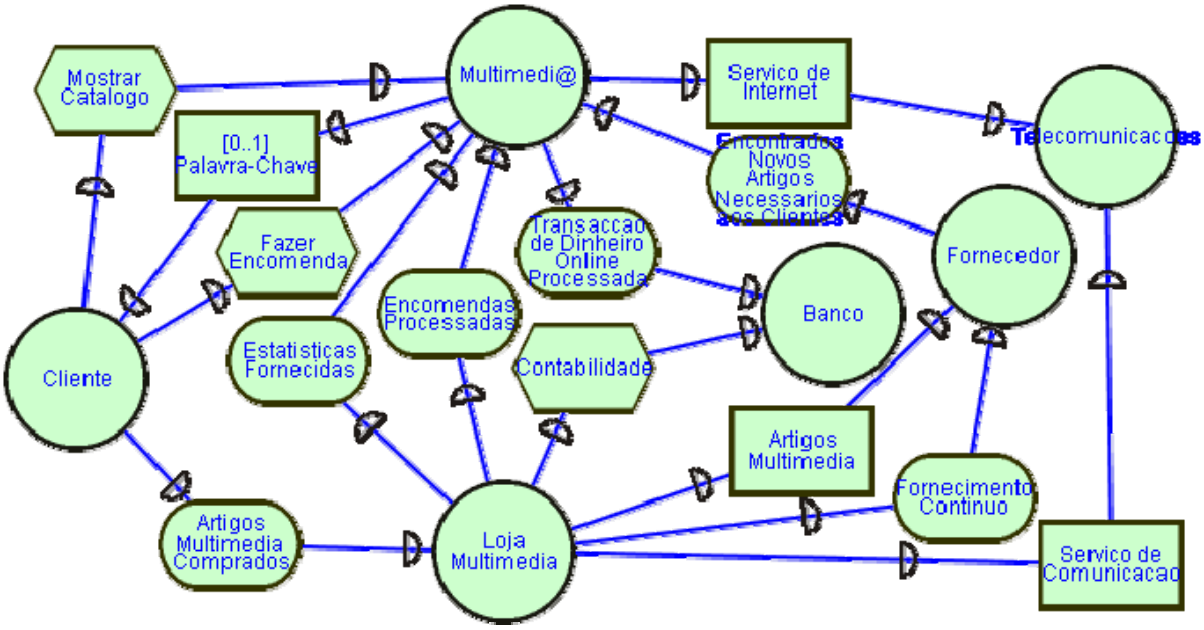


Figura 5.4 Segunda versão do modelo SD para a *Loja de Multimédia Online*

É preciso referir que não há nenhuma ordem específica para colocar qualquer um dos tipos de dependências no modelo. De qualquer modo, optou-se por explicar o modelo SD seguindo a

ordem das dependências de objectivo, tarefas e recursos, apenas para se tentar minimizar a complexidade da explicação.

Como a obtenção de *features* através de *softgoals* não está a ser tratada nesta dissertação, não serão apresentadas as dependências por *softgoals* nos modelos.

Validar objectivos e dependências. A validação das dependências entre os vários actores é feita ao longo, ou após, o desenvolvimento do SD, através dos *stakeholders* que irão confirmar ou rectificar as várias decisões tomadas para a construção do modelo SD.

b. Obter as *features* através do modelo SD. Após desenvolver o modelo SD é possível obter algumas possíveis *features* através das dependências, porque neste momento ainda se desconhece se serão ou não aceites para o modelo de *features*.

No Capítulo 3 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. Uma maneira directa para obter as possíveis *features* através dos elementos intencionais dos modelos i* está descrita na heurística **H.1**.

H.1 As *features* podem ser extraídas através das propriedades que descrevem os elementos intencionais. Por exemplo, na dependência por objectivo "Artigos Multimédia Comprados" a propriedade é "Artigos Multimédia" e por isso obtém-se "Artigos Multimédia", que posteriormente pode ser simplificada para "Artigo", visto só haver um tipo de artigo.

Pretende-se obter *features* que representem os conceitos mais simples, mas permitindo uma identificação o mais completa possível. Portanto, o modo de obtenção 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*, o que poderia dificultar a obtenção das *features* e, acrescentar complexidade ao processo.

Na Tabela 5.1 relacionam-se os conceitos do modelo SD e as *features*. Nesta primeira obtenção, encontram-se algumas possíveis *features*, pois ainda não foram validadas. O recurso "Palavra-Chave" tem uma cardinalidade [0..1] e por isso a *feature* "Palavra-Chave" será opcional. As restantes *features* serão consideradas por omissão como obrigatórias, mas neste momento ainda não será usada essa informação. Mais detalhes serão obtidos e usados no passo **d.**, onde é feita a modelação das *features*.

Apenas as dependências directamente ligadas ao actor Multimédi@ serão consideradas para se obter *features*, porque são esses conceitos que serão refinados posteriormente no modelo SR.

Tabela 5.1 Primeira obtenção de possíveis *features*

Elemento Intencional		***Features***	
Tipo	**Nome**		
Objectivo	Estatísticas Fornecidas	Estatística	
Objectivo	Encomendas Processadas	Encomenda	
Objectivo	Transacção de Dinheiro *Online* Processada	-	
Objectivo	Encontrados Novos Artigos Necessários aos Clientes	Artigo	
Recurso	Serviço de Internet	Serviço	Internet
Recurso	Palavra-Chave	Palavra-Chave	
Tarefa	Mostrar Catálogo	Catálogo	
Tarefa	Fazer Encomenda	Encomenda	

Como se pode verificar através da Tabela 5.1 não é obrigatório obter *features* a partir de todos os elementos intencionais, mas outros exemplos serão apresentados ao longo desta secção. A justificação para não se obter uma *feature* a partir do objectivo "Transacção de Dinheiro *Online* Processada" será apresentada na secção d..

Nesta fase não será apresentada uma versão do modelo de *features* porque a informação é insuficiente. É necessário saber, por exemplo, a relação hierárquica entre as *features* para se poder começar a desenvolver o modelo de *features*, mas só se obtém essa informação ao longo do desenvolvimento do modelo SR.

c. Modelar a satisfação das dependências no modelo SR. Seguidamente, é desenvolvido o modelo SR da *framework* i* para se perceber como são satisfeitas as dependências do actor Multimédi@.

Desenvolver modelo SR. O modelo SR apenas expande o actor Multimédi@, mostrando como satisfaz as dependências a ele associado. Isto acontece porque este é o único actor que representa a linha de produtos de *software*, que está a ser modelada. A Figura 5.9 representa o modelo SR com as ligações às dependências associadas ao actor Multimédi@.

Como já foi referido nesta secção, a *framework* i* não foi desenvolvida a pensar em modelar LPS, portanto ajustou-se os modelos SD e SR para que possam ter informação adicional e assim, ajudar no desenvolvimento de uma linha de produtos. Toda a informação que é

modelada em i* para um sistema único será implementada, mas numa LPS nem todas as dependências e opções para as satisfazer aparecerão obrigatoriamente nos produtos, pois isso dependerá da especificação de cada aplicação.

Para lidar com essas situações adicionou-se cardinalidade nos próprios modelos i*, permitindo transmitir mais informação sobre a variabilidade de uma LPS, semelhante ao que acontece nos modelos de *features* que estão de acordo com Czarnecki *et al*. [29]. A cardinalidade a ser usada tem o mesmo significado que a dos modelos de *features* e, é relembrada na Tabela 5.2.

Tabela 5.2 Cardinalidade para os modelos i*

Notação	**Cardinalidade**	**Numa aplicação específica**
[0..1]	Opcional	Pode aparecer ou não
[1..1]	Obrigatório	Aparece sempre
(1..1)	Alternativa *xor*	Aparece apenas 1 alternativa
(j..k)	Alternativa *or*, com $1 \leq j < k \leq$ nº máx. diferentes *features*	Pode aparecer entre j e k alternativas

A cardinalidade obrigatória [1..1] não aparecerá explicitamente nos elementos intencionais dos modelos i* sendo considerada por omissão. Na Figura 5.5 mostra-se um objectivo, uma tarefa e um recurso sem cardinalidade explícita, logo as *features* daí obtidas serão, em princípio, obrigatórias.

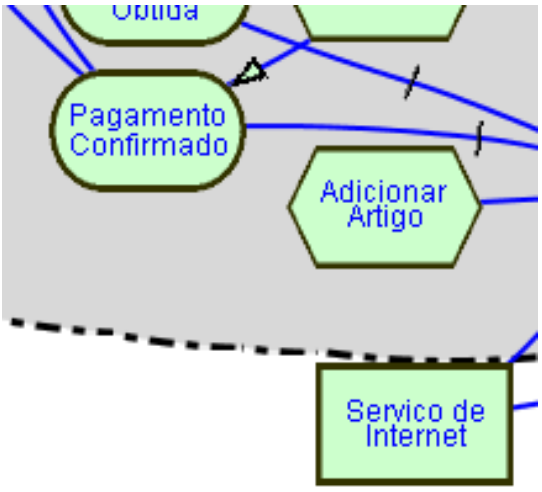


Figura 5.5 Obtenção de *features* obrigatórias a partir de elementos intencionais

Um elemento opcional não precisa ter [0..1] explícita em todos os elementos intencionais, bastando ser identificado uma vez como opcional. Por exemplo, quando um mesmo elemento se encontra num caso com cardinalidade [0..1] e depois sem cardinalidade, a cardinalidade opcional prevalecerá sobre a obrigatória, que se poderia assumir por omissão. Na Figura 5.6 tem-se o exemplo da Palavra-Chave, em que o recurso "[0..1] Palavra-Chave" indica que a *feature* que se obtém "Palavra-Chave" será opcional, mas na tarefa "Pesquisar Palavra-Chave" já não consta a cardinalidade [0..1] podendo se pensar que esta seria obrigatória, mas visto já ter sido referida uma vez como opcional, esta opcionalidade prevalecerá sobre a obrigatoriedade.

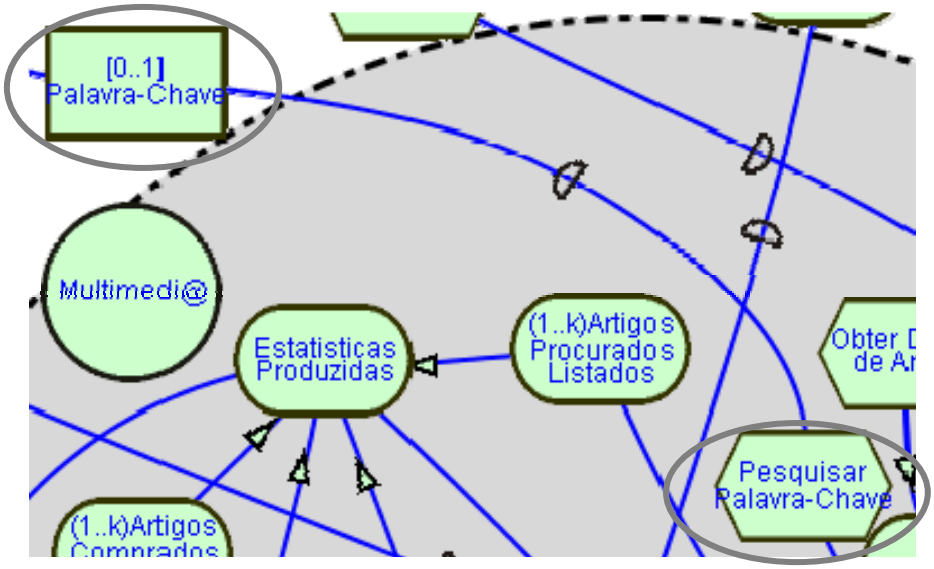


Figura 5.6 Elemento intencional com cardinalidade opcional num modelo i*

Assim, não se sobrecarrega a descrição dos elementos intencionais bastando indicar apenas uma vez a cardinalidade opcional. A cardinalidade, quando é colocada explicitamente no modelo i*, deve anteceder a palavra donde se irá obter a *feature*, ajudando assim a sua leitura para se perceber a qual *feature* pertence a cardinalidade, pois há casos em que se obtêm mais do que uma *feature* a partir de um mesmo elemento intencional, como será apresentado com mais detalhe no passo **d.**. Caso não se saiba onde colocar a cardinalidade, esta pode ser inserida no início da descrição do elemento intencional.

Por sua vez, a cardinalidade que representa a alternativa depende do contexto em que está inserida, e é usada em sub-elementos com ligações meio-fim para um objectivo.

A ligação meio-fim nos modelos SR significa que um objectivo pode ser alcançando através do sub-elemento de onde parte a ligação (ou seja, o meio) e, quando se trata de um único sistema todas as opções têm de estar implementadas. Como neste caso está a ser modelada

uma LPS, a cardinalidade associada aos sub-elementos que representam o meio, pode indicar que as *features* obtidas através deles estarão agrupadas como alternativas no modelo de *features*. Assim, a cardinalidade alternativa só pode ser apresentada nos sub-elementos de uma ligação meio-fim como (1..1) ou (j..k), sendo definidos como $1 \leq j < k \leq$ número máximo de diferentes *features*, e número de diferentes *features* ≥ 2.

A cardinalidade (1..1) representa que a *feature* obtida pelo sub-elemento pertence a um grupo *xor*, ou seja, apenas se pode escolher uma única opção desse grupo numa aplicação específica. A Figura 5.7 exemplifica a alternativa *xor*, com o registo do método de pagamento a ser satisfeito através do registo com cartão de crédito ou transferência bancária. Portanto, com a cardinalidade (1..1) uma aplicação só terá disponível o pagamento por cartão de crédito ou por transferência bancária, e nunca ambas.

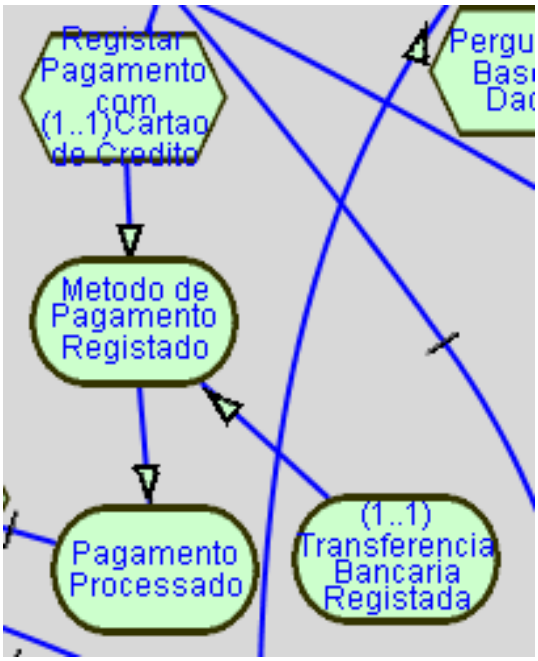


Figura 5.7 Cardinalidade de alternativa *xor* num modelo SR

No caso da alternativa (j..k), com $1 \leq j < k \leq$ número máximo de diferentes *features*, e número de diferentes *features* ≥ 2, representa que a *feature* obtida pelo sub-elemento pertence a um grupo *or*. O exemplo da Figura 5.8 mostra quatro ligações meio-fim para satisfazer o objectivo "Estatísticas Produzidas". Em cada um desses elementos tem (1..k), representando j=1, ou seja, tem de se escolher pelo menos uma alternativa e, k=k indicando que k será igual ao máximo número de *features* diferentes, portanto neste caso k=3, pois obter-se-á 3 *features* diferentes como se verá no ponto **d.**.

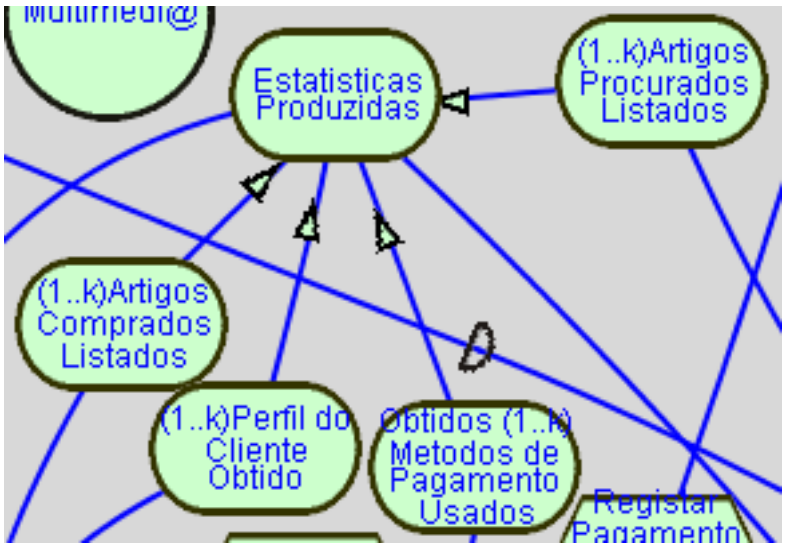


Figura 5.8 Cardinalidade de alternativa *or* num modelo SR

Quando aparece cardinalidade de um grupo de alternativas, é obrigatória a existência de pelo menos dois sub-elementos com ligações meio-fim para um mesmo elemento intencional e devem-se obter *features* diferentes (de outro modo, deixa-se de ter uma alternativa passando a ter uma relação com sub-*features*).

A Figura 5.9 ilustra uma primeira versão do modelo SR da *Loja de Multimédia Online*. A tarefa principal "Gerir Loja *Online*" vai permitir à Loja Multimédia satisfazer o objectivo "Encomendas Processadas", mas para isso necessita do recurso "Serviço de Internet" fornecido pelas Telecomunicações. Na gestão da loja *online* também precisa de ter satisfeito o objectivo "Estatísticas Produzidas" e realizar a tarefa "Carrinho de Compras".

O objectivo "Estatísticas Produzidas" vai satisfazer o objectivo "Estatísticas Fornecidas" da Loja Multimédia. No caso do "Carrinho de Compras" este vai precisar de adicionar artigos à compra, ter a informação do cliente e ter o pagamento confirmado, para que possa satisfazer o objectivo de ter os artigos transaccionados.

Um artigo transaccionado pode ser encontrado pelos Clientes através da consulta do catálogo, permitindo seleccionar os artigos que lhe interessam e, ao perguntar à base de dados estes podem estar disponíveis ou indisponíveis. Caso estejam indisponíveis o Cliente pode encomendá-los e, isso vai permitir ao Fornecedor conhecer os novos artigos necessários aos clientes. A Multimédi@ tem disponível a opção do Cliente fazer a pesquisa por palavras-chave, fazendo perguntas à base de dados sobre os artigos.

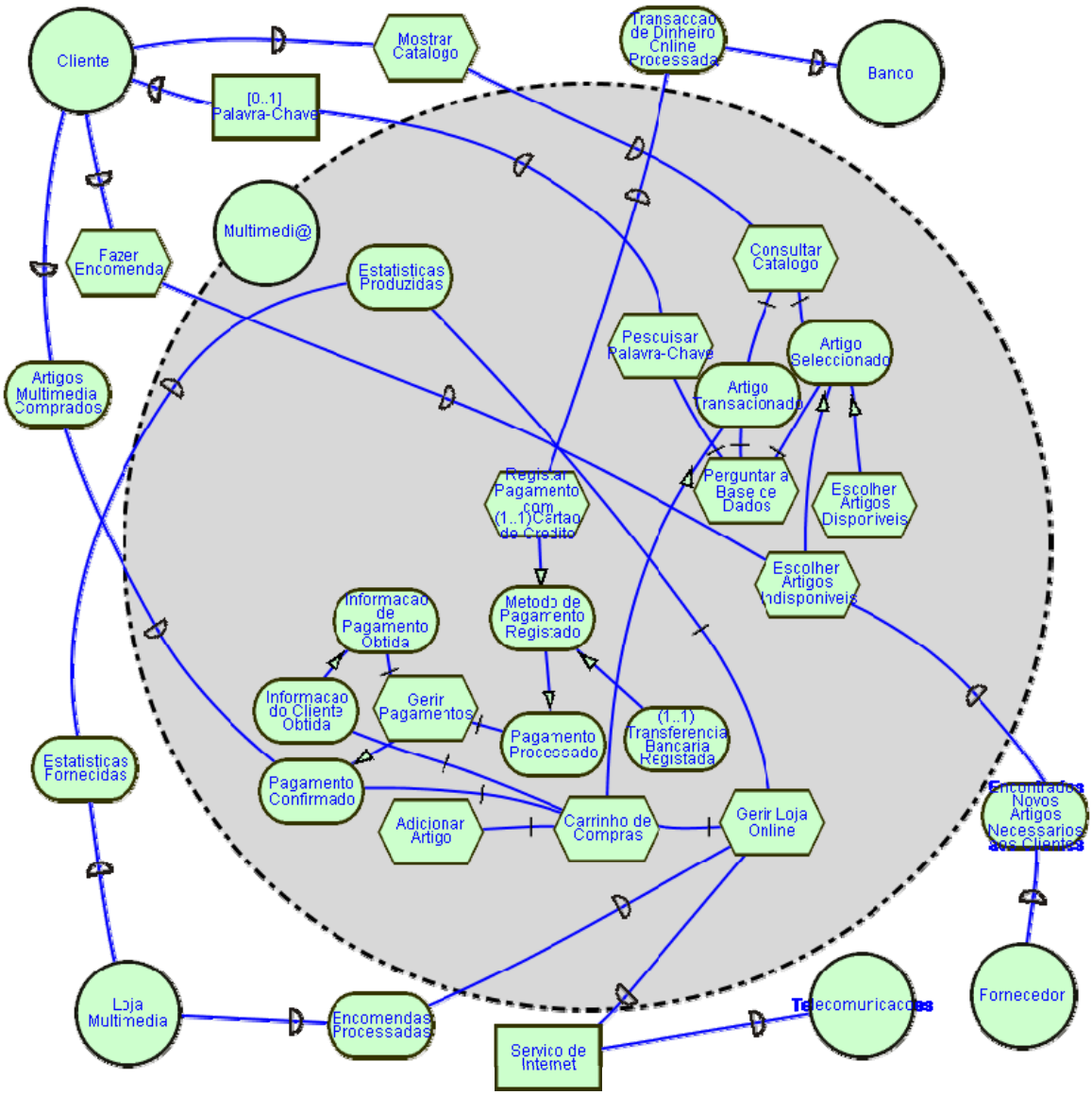


Figura 5.9 Primeira versão do modelo SR do actor Multimédi@

Pode-se obter informação sobre um pagamento através da informação do cliente que fez esse pagamento. Para gerir os pagamentos tem de se ter a informação dos pagamentos e saber quais os pagamentos que foram processados.

É possível realizar a tarefa de registar o pagamento por cartão de crédito após obter a informação do Banco que informa sobre a transacção de dinheiro *online*. A tarefa "Registar Cartão de Crédito" e o objectivo "Transferência Bancária Registada" satisfazem o objectivo "Método de Pagamento Registado". Com o método de pagamento registado pode-se satisfazer o objectivo "Pagamento Processado". Como o registo do cartão de crédito e o registo da transferência bancária têm ligações meio-fim com cardinalidade (1..1) conclui-se que uma aplicação apenas pode ter uma das opções, ou seja, as *features* que se obtêm a partir desses sub-elementos serão agrupadas por uma alternativa *xor* no modelo de *features*.

Na Figura 5.10 encontra-se a segunda versão do modelo SR para o actor Multimédi@. Nesta versão é possível encontrar mais pormenores sobre o modo como este actor satisfaz as dependências. Os elementos intencionais na área sombreada já foram apresentados na Figura 5.9.

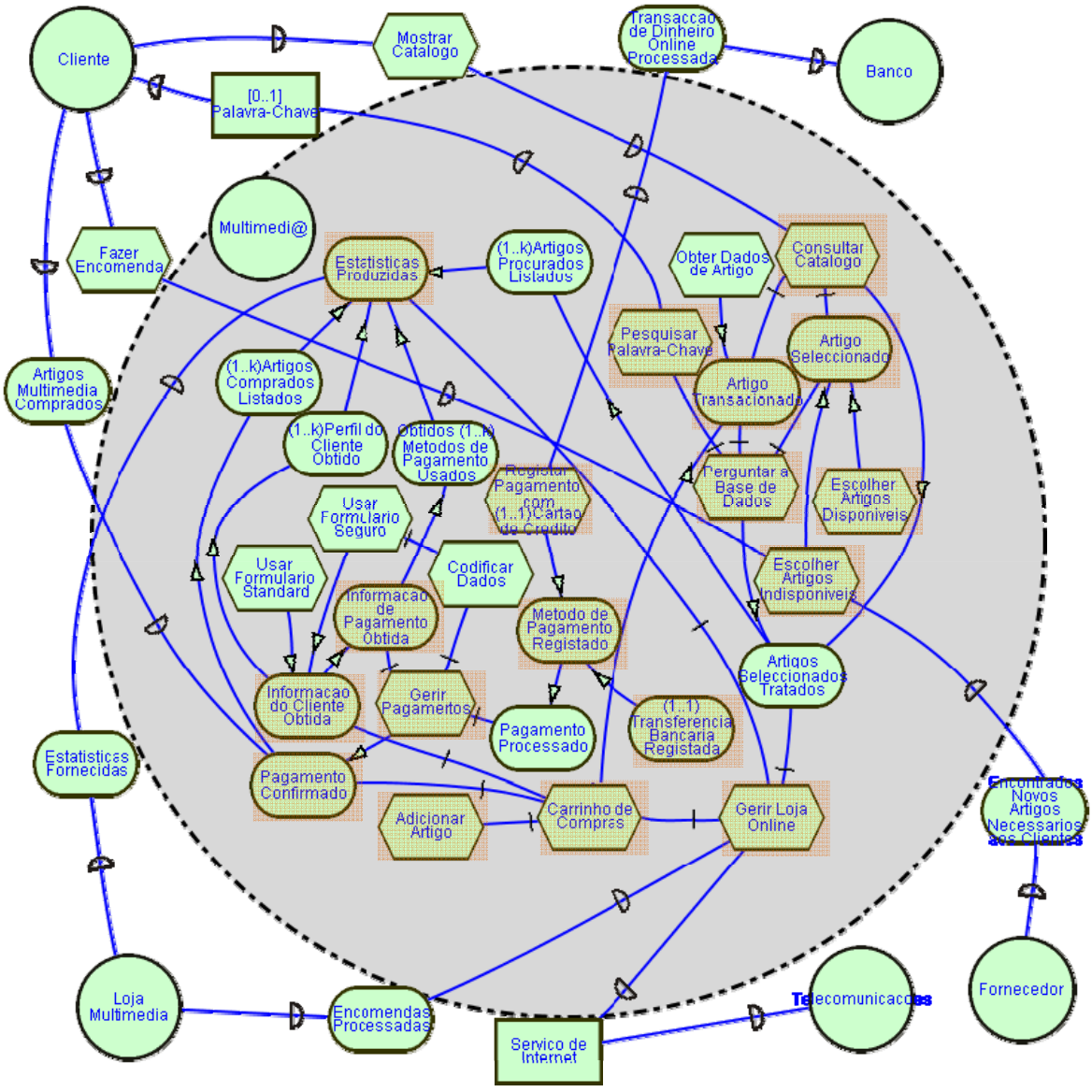


Figura 5.10 Segunda versão do modelo SR do actor Multimédi@

Nesta segunda versão pode-se analisar que o objectivo "Artigo Transaccionado" necessita de obter dados sobre esse artigo. A tarefa "Codificar Dados" é necessária para a gestão dos pagamentos e para "Usar Formulário Seguro". A utilização de um formulário seguro e de um formulário *standard* permitem obter informação sobre o cliente. A informação do cliente permite definir um perfil. As estatísticas produzidas podem ser baseadas no perfil dos clientes, nos artigos comprados, nos métodos de pagamento usados ou nos artigos procurados e, como

esses 4 objectivos têm cardinalidade (1..k) significa que as *features* obtidas através desses sub-elementos intencionais pertencerão a um grupo *or* no modelo de *features*.Alcança-se o objectivo "Artigos Comprados Listados" através dos pagamentos confirmados. O objectivo "Obtidos Métodos de Pagamento Usados" é alcançado através da informação de métodos de pagamento. Alcança-se o objectivo "Artigos Procurados Listados" através do objectivo "Artigos Seleccionados Tratados", fazendo este último parte da gestão da loja e alcançado através das tarefas "Perguntar à Base de Dados" e "Consultar Catálogo".

Os modelos i* apresentados são apenas para exemplificar a cardinalidade e várias opções para se obter *features*, por isso todas as opções da *Loja de Multimédia Online* não se encontram aqui apresentadas.

Validar as decisões do SR. A validação é feita através dos *stakeholders* que irão confirmar as várias decisões para a construção do modelo SR.

d. Modelar as *features* restantes. Desenvolve-se o modelo de *features* com a informação que se obtém através dos modelos anteriores.

Obter e analisar as *features*. Para se produzir o modelo de *features* tem de se obter as restantes *features* através do modelo SR, do modo como será seguidamente apresentado.

A obtenção de *features* a partir dos elementos intencionais dos modelos i* é dependente do modo como se descreve esse elemento intencional, pois quanto mais completo for, mais fácil se torna a obtenção das *features* e a compreensão sobre a hierarquia entre elas.

A escolha das palavras na descrição dos elementos intencionais também pode facilitar ou dificultar a obtenção de *features*. Na obtenção de *features* também é preciso ter em atenção até onde se pretende refinar o modelo.

A Tabela 5.3 apresenta a junção entre a informação da Tabela 5.1 e as restantes *features* obtidas a partir do modelo SR da Figura 5.10, recorrendo à heurística **H.1**, descrita acima.

Entretanto, pode-se ir construindo o modelo de *features* à medida que são encontradas as *features*. Mas neste caso, segmentou-se o desenvolvimento do modelo de *features* de modo a apresentar a progressão do modelo, à medida que se vão justificando as várias decisões tomadas.

Na Tabela 5.3 os campos que não têm *features* apresentam-se com um travessão (-).

Tabela 5.3 Obtenção das restantes *features*

Elemento Intencional		*Feature*	
Tipo	**Nome**		
Tarefa	Gerir Loja *Online*	Loja *Online*	
Recurso	Serviço de Internet	Serviço	Internet
Objectivo	Encomendas Processadas	Encomenda	
Tarefa	Carrinho de Compras	Carrinho	
Tarefa	Adicionar Artigo	Artigo	
Objectivo	Pagamento Confirmado	Pagamento	
Objectivo	Artigos Multimédia Comprados	Artigo	
Tarefa	Gerir Pagamentos	Pagamento	
Objectivo	Informação de Pagamento Obtida	Informação	Pagamento
Tarefa	Codificar Dados	-	
Tarefa	Usar Formulário Seguro	Formulário	
Objectivo	Informação do Cliente Obtida	Informação	Cliente
Tarefa	Usar Formulário *Standard*	Formulário	
Objectivo	Pagamento Processado	Pagamento	
Objectivo	Método de Pagamento Registado	Método	Pagamento
Objectivo	Transferência Bancária Registada	Transferência Bancária	
Tarefa	Registar Pagamento com Cartão de Crédito	Cartão Crédito	Pagamento
Objectivo	Transacção de Dinheiro *Online* Processada	-	
Objectivo	Obtidos os Métodos de Pagamento Usados	Método	Pagamento
Objectivo	Estatísticas Produzidas	Estatística	
Objectivo	Estatísticas Fornecidas	Estatística	
Objectivo	Artigos Comprados Listados	Artigo	
Objectivo	Perfil do Cliente Obtido	Perfil	Cliente
Objectivo	Artigos Procurados Listados	Artigo	
Objectivo	Artigos Seleccionados Tratados	Artigo	
Tarefa	Perguntar à Base de Dados	-	
Tarefa	Pesquisar Palavra-Chave	Palavra-Chave	
Recurso	Palavra-Chave	Palavra-Chave	
Objectivo	Artigo Transaccionado	Artigo	
Tarefa	Obter Dados de Artigo	Dados	Artigo
Objectivo	Artigo Seleccionado	Artigo	
Tarefa	Escolher Artigos Disponíveis	Artigo	
Tarefa	Escolher Artigos Indisponíveis	Artigo	
Tarefa	Fazer Encomenda	Encomenda	
Objectivo	Encontrados Novos Artigos Necessários aos Clientes	Artigo	
Tarefa	Consultar Catálogo	Catálogo	
Tarefa	Mostrar Catálogo	Catálogo	

Os casos em que não se obtêm *features* a partir de um elemento intencional existem devido a um dos factores que se seguem:

- A *feature* que se obteria do elemento intencional corresponderia a informação de baixo nível, podendo não ser desejável que seja representada no modelo de *features*. Por exemplo "Base de Dados" poderia ser obtida através da tarefa "Perguntar à Base de Dados".

- A descrição do elemento intencional é vaga. Como acontece no caso da tarefa "Codificar Dados", onde os dados a codificar são relativos à informação dos clientes e dos pagamentos, segundo o modelo SR da Figura 5.10.

- A descrição do elemento intencional é omissa e as *features* que se obteriam não fornecem mais informação do que a já existente. Caso se considere que a informação omitida é relevante para o desenvolvimento do modelo de *features*, então renomeia-se o elemento intencional adicionando a informação em falta, ou pode-se proceder de acordo com a heurística **H.7** descrita à frente.

No caso do objectivo "Transacção de Dinheiro *Online* Processada" poder-se-ia obter a *feature* "Dinheiro" mas pode-se considerar esta *feature* como sendo de baixo nível, pois esse dinheiro será transaccionado através de uma transferência bancária ou de um pagamento por cartão de crédito, não sendo por isso relevante a extracção desta *feature*.

Seguem-se algumas heurísticas para obter *features* e informação sobre essas *features*.

H.2 Possivelmente, uma *feature* X terá pelo menos 1 sub-*feature* Y, quando X se relaciona com uma acção que necessita de um objecto directo ou indirecto. Por exemplo, a "Encomenda" relaciona-se com a acção encomendar, portanto a "Encomenda" poderá ser detalhada com sub-*features*, onde neste caso, é necessário saber o que se vai encomendar e, após a análise do modelo SR verifica-se que será "Artigo".

H.3 Através da descrição de um elemento intencional pode-se obter mais do que uma *feature.* No caso genérico "X de Y" onde X é uma propriedade e Y se relaciona com X, podem-se obter as *features* X e Y. Por exemplo, do objectivo "Informação do Pagamento Obtida" podem-se obter as *features* "Informação" que representa o X e "Pagamento" que representa o Y. Mas, dependendo do nível de detalhe que se pretenda, pode-se deixar uma única *feature* como acontece no recurso "Serviço de Internet" em que se poderia obter a *feature* "Serviço" e "Internet" que seriam o X e Y respectivamente, mas neste caso não é necessária essa divisão, obtendo-se então apenas uma *feature* "Serviço de Internet".

H.4 Pode-se obter informação sobre a hierarquia das *features*, quando através dos elementos intencionais em que aparece "X de Y" se obtêm as *features* X e Y, então possivelmente X será sub-*feature* de Y, ou vice-versa. Como no exemplo, de "Informação do Cliente Obtida" obtêm-se as *features* "Informação" e "Cliente", que estão relacionadas hierarquicamente,

onde "Informação" é relativa ao Cliente e por isso será sub-*feature* de "Cliente". Esta hierarquia depende do nível de detalhe que se pretenda que o modelo de *features* tenha.

H.5 Como se verificou que a *framework* i* descrita por Yu [11] não tem conceitos que indiquem a possibilidade de se obter daí *features* opcionais, visto este tipo de modelação não ter sido desenvolvida inicialmente para representar LPS, adicionou-se a cardinalidade nos modelos SD e SR. Então, se a partir de um elemento intencional se puder obter uma *feature* que é indicada com a cardinalidade [0..1], essa *feature* será assinalada como opcional no modelo de *feature*. Por exemplo, o recurso "[0..1] Palavra-Chave" é opcional, por isso tem a cardinalidade [0..1], logo a *feature* daí obtida também é opcional.

Ao longo da construção do modelo de *features* vão-se identificando as várias *features* e desde o primeiro modelo pode-se usar a cardinalidade. No caso de não se saber ainda qual a cardinalidade adequada a uma determinada *feature* indica-se esta como obrigatória, por omissão, até se encontrar informação que indique outra cardinalidade e, quando isso acontecer, reajusta-se a cardinalidade.

Um primeiro modelo de *features* é representado na Figura 5.11 em que as *features* foram obtidas a partir do modelo SD da Figura 5.4 e do modelo SR da Figura 5.9.

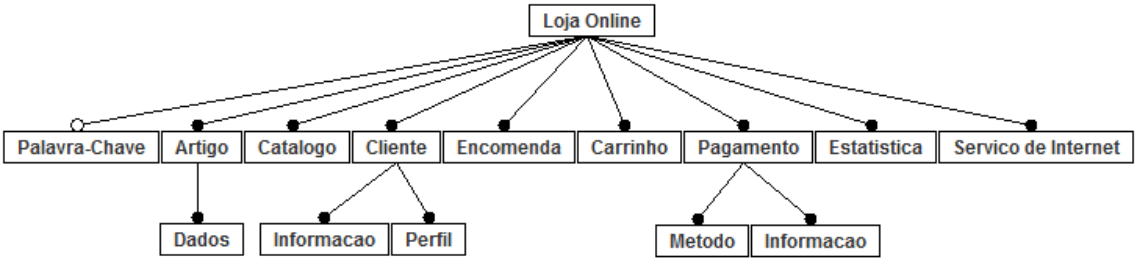


Figura 5.11 Primeira versão do modelo de *features*

Pretende-se que este primeiro modelo seja mais simples para que a sua complexidade possa ser gradualmente aumentada à medida que se vai obtendo mais informação sobre as *features*, a sua hierarquia e as suas interacções.

H.6 A análise dos modelos i* com cardinalidade ajudam na identificação das *features* alternativas *or,* ou seja, alternativas que poderão ser todas ou algumas escolhidas para uma aplicação. As *features* obtidas a partir dos elementos intencionais com cardinalidade (j..k), que têm ligações de meio-fim para um objectivo, podem ser agrupadas como alternativas *or*. Por exemplo, o objectivo "Estatísticas Produzidas" pode ser alcançado através da satisfação dos objectivos "(1..k) Artigos Comprados Listados", "(1..k) Perfil do Cliente Obtido",

"Obtidos os (1..k) Métodos de Pagamento Usados" ou "(1..k) Artigos Procurados Listados", e estes 4 objectivos têm uma cardinalidade (1..k). Portanto, as *features* que se obtêm vão pertencer a um grupo *or* no modelo de *features*. Do objectivo "Estatísticas Produzidas" obtém-se a *feature* "Estatísticas" e esta terá um grupo de alternativas *or* composto pelas *features* "Artigo", "Perfil" do Cliente e "Método" de Pagamento. Como elas representam uma manipulação de dados sobre artigos, perfis e métodos serão designadas por "EArtigo", "EPerfil", "EMétodo" respectivamente, pois são obtidas através dos 4 objectivos i* donde se obtêm 3 diferentes *features*, logo k=3.

H.7 Nos modelos i* podem-se identificar *features* alternativas *xor,* ou seja, tem de se escolher uma única opção dentro de um grupo. As alternativas encontram-se nos elementos intencionais com cardinalidade (1..1) e têm ligações de meio-fim para um objectivo. Por exemplo, o Pagamento pode ser feito pelos métodos de transferência bancária ou cartão de crédito, segundo o modelo SR da Figura 5.10. Então, o "Método" do Pagamento terá as sub-*features* "Transferência Bancária" e "Cartão de Crédito", mas numa aplicação só uma delas será permitida porque os objectivos "Transferência (1..1) Bancária Registada" e "(1..1) Cartão de Crédito Registado" têm cardinalidade (1..1) logo as *features* pertencem a uma alternativa *xor*.

A Figura 5.12 representa uma segunda versão do modelo de *features*. Neste modelo ilustram-se mais algumas variabilidades. Pode-se adicionar a informação sobre variabilidade e interacção à medida que esta vai sendo encontrada, mas neste exemplo está a ser apresentada gradualmente para facilitar a explicação.

É útil analisar os nomes das *features* e as ligações entre elas de modo a clarificar os conceitos, especificar as *features* que podem ser mais pormenorizadas e reajustar as suas ligações. Algumas *features* podem ser renomeadas, ou o seu nome pode ser ajustado, para que na construção do modelo de *features* seja claro o que se está a referir. Por exemplo, as *features* "Informação" do Pagamento e "Informação" do Cliente serão alteradas para "InformaçãoP" e "InformaçãoC" respectivamente, de modo a distingui-las, pois a informação relativa ao pagamento é diferente da informação do cliente.

Para se simplificar os modelos i* pode-se agrupar um conjunto de características identificando-o através de uma palavra nas descrições dos elementos intencionais. Por exemplo, o elemento intencional "Informação do Pagamento Obtida" pode ser usado em vez de se colocar vários elementos intencionais como "Método de Pagamento Obtido", "Preço

Total do Pagamento Obtido", "Data do Pagamento Obtida" e "Número do Pagamento Obtido". Ao ter um único elemento intencional que representa um conjunto de conceitos facilita a compreensão do modelo i* e ajuda a agrupar conceitos que não são relevantes ter em separado para se satisfazer os objectivos do sistema. De qualquer modo, caso essa informação estivesse explicitamente no modelo i* as *features* seriam extraídas directamente, e poder-se-iam obter logo as *features* "Método", "Preço Total", "Data" e "Número". Entretanto, caso a informação esteja omissa pode-se recorrer aos *stakeholders* ou à descrição da LPS para se alcançar esse nível de detalhe.

H.8 As *features* que se obtêm por elementos intencionais que omitem informação podem ter sub-*features* com os detalhes. Por exemplo, a *feature* "InformaçãoP" do Pagamento, referida acima, pode ser detalhada com as sub-*features* "Método", "Preço Total", "Data" e "Número" de Pagamento.

A *feature* "Encomenda" contém os artigos que não estão disponíveis em *stock*. A Encomenda refere-se aos artigos que precisam de ser encomendados ao Fornecedor. A *feature* "Carrinho" contém os artigos que estão disponíveis em *stock* e podem ser comprados imediatamente.

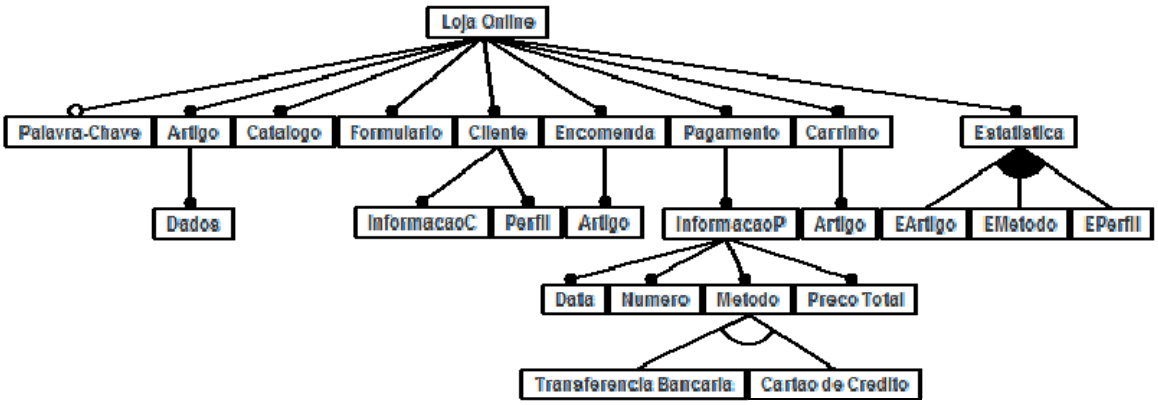


Figura 5.12 Segunda versão do modelo de *features*

Uma terceira versão do modelo de *features* encontra-se ilustrada na Figura 5.13. Nesta versão aparecem referências a *features*, ou seja, uma representação onde se indica que uma *feature* requer outra.

Pode-se descobrir a hierarquia de *features* recorrendo ao modelo SR e analisando como se relacionam os elementos intencionais de onde se extraem *features*.

H.9 Duas diferentes *features*, obtidas através de elementos intencionais que se relacionam através de uma ligação meio-fim ou de uma decomposição de tarefa, estarão relacionadas hierarquicamente no modelo de *features*. Por exemplo, existe uma ligação meio-fim da tarefa

"Consultar Catálogo" para o objectivo "Artigos Seleccionados Tratados" donde se obtêm as *features* "Catálogo" e "Artigo" respectivamente, e implica que "Catálogo" tenha a sub-*feature* "Artigo". No caso da ligação de decomposição de tarefas, têm-se por exemplo a tarefa "Carrinho de Compras" que permite obter a *feature* "Carrinho" e é decomposta em "Adicionar Artigo", "Pagamento Confirmado" e "Informação do Cliente Obtida" donde se obtém as *features* "Artigo", "Pagamento", "InformaçãoC" e "Cliente", logo no modelo de *features* "Carrinho" terá sub-*features* "Artigo", "Pagamento" e "InformaçãoC". A *feature* "Cliente" não será sub-*feature* porque "Carrinho" apenas necessita da informação do cliente, em vez de toda a descrição do cliente.

No caso da tarefa "Usar Formulário Seguro", de onde se obter a *feature* "Formulário", esta tarefa permite satisfazer o objectivo "Informação do Cliente Obtida", donde se extrai duas *features* "InformaçãoC" e "Cliente", segundo a heurística **H.3**. Como se verificou que o formulário continha os dados pessoais dos clientes em vez de se manter uma *feature* "Formulário" como sub-*feature* de "Loja *Online*" (conforme na Figura 5.12) passa a ser uma sub-*feature* de "Cliente" com o nome "Dados Pessoais", porque que consta no formulário são os dados pessoais do cliente.

H.10 Duas diferentes *features*, obtidas através de elementos intencionais que se relacionam através de uma ligação de dependência, estarão relacionadas hierarquicamente no modelo de *features*. Por exemplo, a *feature* "Encomenda" necessita de "Artigo" porque no modelo SR da Figura 5.10 a tarefa "Fazer Encomenda" depende da tarefa "Escolher Artigos Indisponíveis" donde se extrai "Artigo".

H.11 Uma *feature* X requer outra Y, quando se verifica, utilizando as heurísticas anteriores, que X necessita de Y como sua sub-*feature*, mas Y é independente de X, assim Y encontra-se no modelo de *features* num ramo diferente do de X. Por exemplo, a *feature* "Catálogo" requer a *feature* "Artigo" pois este é indispensável para si, mas "Artigo" é independente de "Catálogo" e por isso mantém-se como sub-*feature* de "Loja *Online*". Conclui-se portanto que se trata de uma sub-*feature* quando esta necessita da super-*feature* e, utiliza-se a ligação *require* quando a *feature* não necessita da super-*feature*, mas a super-*feature* necessita dela.

As sub-*features* de "Estatística", ou seja EArtigo, EPerfil e EMétodo, requerem as 3 *features* "Artigo", "Perfil" e "Método" respectivamente para poderem manipular os seus dados, pois as sub-*features* de "Estatística" necessitam dessas *features*, mas estas últimas não necessitam

dessas manipulações. Portanto, cada uma dessas *features* será requerida pelas sub-*features* de "Estatística", segundo a heurística **H.11**.

A *feature* "Carrinho" requer as *features* "Pagamento", "Cliente" e "Artigo", segundo a heurística **H.11**. Isto acontece porque no modelo SR a tarefa "Carrinho de Compras", que se relaciona com a *feature* "Carrinho", decompõe-se nos objectivos "Pagamento Confirmado" e "Informação do Cliente Obtida", donde se obtêm as *features* "Pagamento", "InformaçãoC" e "Cliente" e, na tarefa "Adicionar Artigo" de onde se obtém a *feature* "Artigo". Daí a *feature* "Carrinho" requer as *features* "InformaçãoC", "Pagamento" e "Artigo". A *feature* "Cliente" não é requerida porque o importante é a sua sub-*feature* "InformaçãoC" que contém os dados que a *feature* "Carrinho" necessita. As *features* requeridas encontram-se representadas com uma seta a tracejado com uma etiqueta ***<<Requires>>***, visto ter sido esta a notação apresentada no capítulo de LPS.

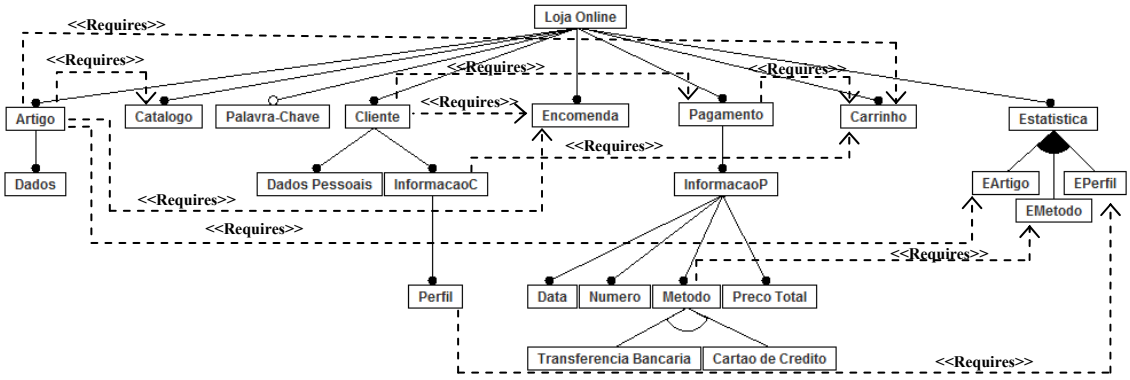


Figura 5.13 Terceira versão do modelo de *features*

Resumo das heurísticas. Com a conclusão do modelo de *features* podem-se apresentar as várias heurísticas que foram sendo descritas ao longo desta secção, a nível de engenharia de domínio, para se produzir o modelo de *features* recorrendo aos modelos i*. As heurísticas são:

H.1 As *features* podem ser extraídas através das propriedades que descrevem os elementos intencionais.

H.2 As *features* relacionadas com acções terão possivelmente sub-*features*.

H.3 A partir de um elemento intencional pode-se obter mais do que uma *feature*, extraindo-se uma *feature* da palavra Y que se relaciona com X na frase "X de Y".

H.4 Se nos elementos intencionais aparece "X de Y" e se obtêm as *features* X e Y, então possivelmente X será sub-*feature* de Y, ou vice-versa.

H.5 Se a partir de um elemento intencional se obtiver uma *feature* que é indicada com a cardinalidade [0..1], então essa *feature* será assinalada como opcional no modelo de *feature*.

H.6 As *features* que sejam obtidas a partir de elementos intencionais, com ligações de meio-fim para um objectivo, com a cardinalidade (j..k) serão agrupadas como alternativas *or*.

H.7 As *features* alternativas *xor* encontram-se nos elementos intencionais que têm ligações de meio-fim para um objectivo e com uma cardinalidade (1..1), representando que só se poderá escolher uma única opção daquele grupo.

H.8 As *features* obtidas através de elementos intencionais que omitem informação podem ter sub-*features* com os detalhes.

H.9 Duas *features* diferentes, obtidas através de elementos intencionais diferentes que se relacionam através de uma ligação meio-fim ou de decomposição de tarefa, estarão relacionadas hierarquicamente no modelo de *features*.

H.10 Duas *features* diferentes obtidas através de elementos intencionais diferentes, que se relacionam através de uma ligação de dependência, estarão relacionadas hierarquicamente no modelo de *features*.

H.11 Uma *feature* X requer outra Y quando se verifica, utilizando as heurísticas anteriores, que X necessita de Y como sua sub-*feature* mas Y é independente de X, assim, Y encontra-se no modelo de *features* num ramo diferente do de X.

Validação do modelo de *features*. A validação é feita através dos *stakeholders* que irão confirmar ou rectificar as várias opções do modelo de *feature*.

5.2.2 Procedimentos a Nível de Engenharia de Aplicação

A nível da engenharia de aplicação o processo apresentado na Figura 5.2 baseia-se no proposto pela engenharia de linhas de produto de *software*, para a configuração de produtos.

O modelo de *features* e os modelos da *framework* i* são configurados para um produto específico, e serão ambos configurados para que se possa guardar a informação que complementa o modelo de *features* para uma aplicação específica. Será inicialmente apresentada a configuração do modelo de *features* e depois o do modelo i*, portanto configura-se primeiro o modelo mais simples e posteriormente o modelo i* que contém mais

detalhes. Os modelos da *framework* i* sofrerão alterações devido aos objectivos e decisões para uma aplicação específica.

Por exemplo, para uma aplicação em que se quer ter a possibilidade de comprar e encomendar os artigos com cartão de crédito, procurar no catálogo, produzir estatísticas sobre artigos comprados e perfil dos clientes, serão apresentados os três modelos utilizados. Portanto, o modelo de *features*, o modelo SD e o modelo SR ficarão com as opções que aparecem na Figura 5.14, Figura 5.15 e Figura 5.16, respectivamente.

A. Configurar o modelo de *features* para uma aplicação específica. O modelo de *features* é configurado de acordo com as especificações da aplicação.

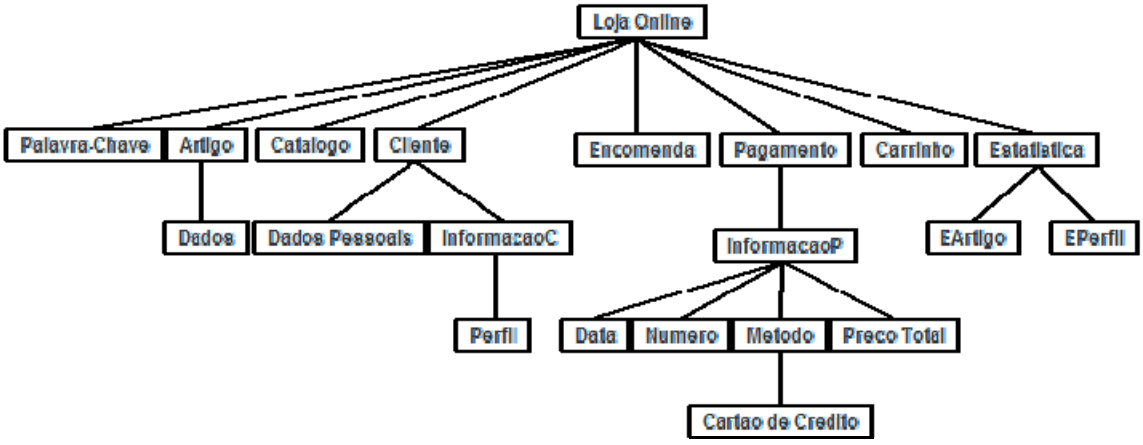


Figura 5.14 Modelo de *features* para a aplicação

B. Configurar os modelos i* para uma aplicação específica. No modelo SD apenas se elimina o recurso "Palavra-Chave", sendo essa a consequência da aplicação não suportar essa opção.

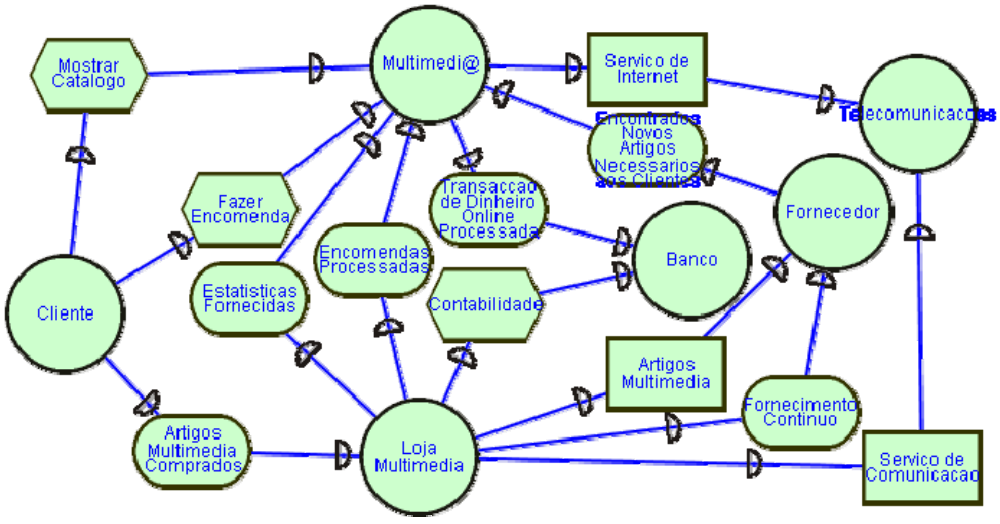


Figura 5.15 Modelo SD para a aplicação específica

O modelo SR para esta aplicação não tem a pesquisa por palavra-chave, nem os elementos intencionais associados a essa opção. Além disso, os objectivos "Artigos Procurados Listados" e "Obtidos Método de Pagamento Usados" que estavam associados às estatísticas e não são necessários para mais nenhuma situação foram eliminados, deixando apenas os objectivos "Artigos Comprados Listados" e "Perfil do Cliente Obtido" que são as opções válidas à aplicação que se está a configurar.

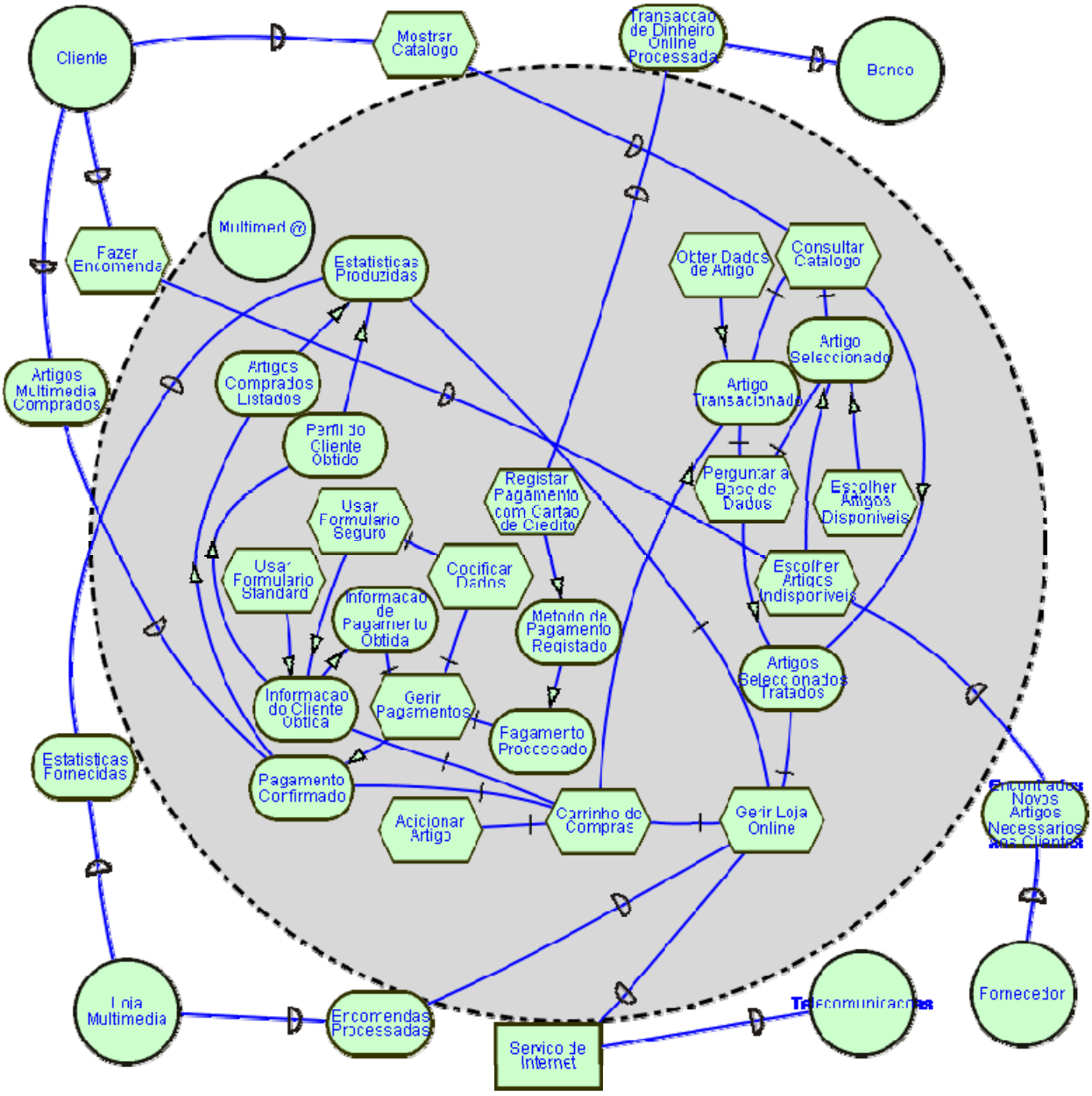


Figura 5.16 Modelo SR para a aplicação específica

Apresentou-se nesta secção o processo para a engenharia de domínio e o processo para a engenharia de aplicação adaptado para que a *framework* i* permita representar LPSs.

Na próxima secção serão apresentados os metamodelos do modelo de *features* e da *framework* i* separadamente, tal como o metamodelo da abordagem IStarLPS, que se propôs na secção 5.1 e, o qual fundamenta os processos.

5.3 Analisar os metamodelos e definir as relações entre eles.

O metamodelo é ele próprio um modelo que descreve a estrutura de modelos. Ele define os construtores de uma linguagem e as suas relações, como também as suas restrições e regras de modelação [37].

A adaptação da *framework* i* para facilitar o desenvolvimento do modelo de *features*, designada por abordagem IStarLPS, encontra-se de acordo com o metamodelo da secção 5.3.3. Esta adaptação permite ajudar a obter as *features* do modelo de *features* recorrendo à *framework* i*, facilitando a obtenção das *features* de uma forma semiautomática recorrendo às heurísticas definidas na secção 5.2. Portanto, na secção 5.3.1 apresenta-se o metamodelo da *framework* i* de acordo com Yu [Yu, 1995]. Na secção 5.3.2 encontra-se o metamodelo do modelo de *features* de acordo com Czarnecki *et al.* [27], ou seja, o modelo de *features* que suporta cardinalidade. E na secção 5.3.3 é apresentado o metamodelo da abordagem IStarLPS.

5.3.1 Metamodelo da *Framework* i*

Yu [11] originalmente especificou o metamodelo da *framework* i* usando Telos [38], mas posteriormente Alencar *et al.* [37] especificou-o em MOF (*Meta-Object Facility*) [39] com algumas alterações. Para apresentar o metamodelo conforme a descrição de Yu [11], adaptou-se o metamodelo de [37] removendo os conceitos introduzidos por Alencar *et al*.. Assim, na Figura 5.17 encontra-se o metamodelo da *framework* i* especificado em MOF e produzido com a ferramenta StarUML [40]. Este metamodelo mostra como os vários conceitos da *framework* i* se relacionam e para facilitar a compreensão é feita a sua descrição.

Na Figura 5.17, 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 (*Depender*), através de uma ligação (*DependerLink*), de um *Dependum*, e a satisfação deste último pode ser da responsabilidade de outro *DependebleNode* (*Dependee*) através de uma ligação *DependeeLink*. Um *Dependum* será definido por um tipo (*DependumKind*). No caso do *DependableNode*, este é especializado num actor (*Actor*) ou num elemento intencional (*IntentionElement*).

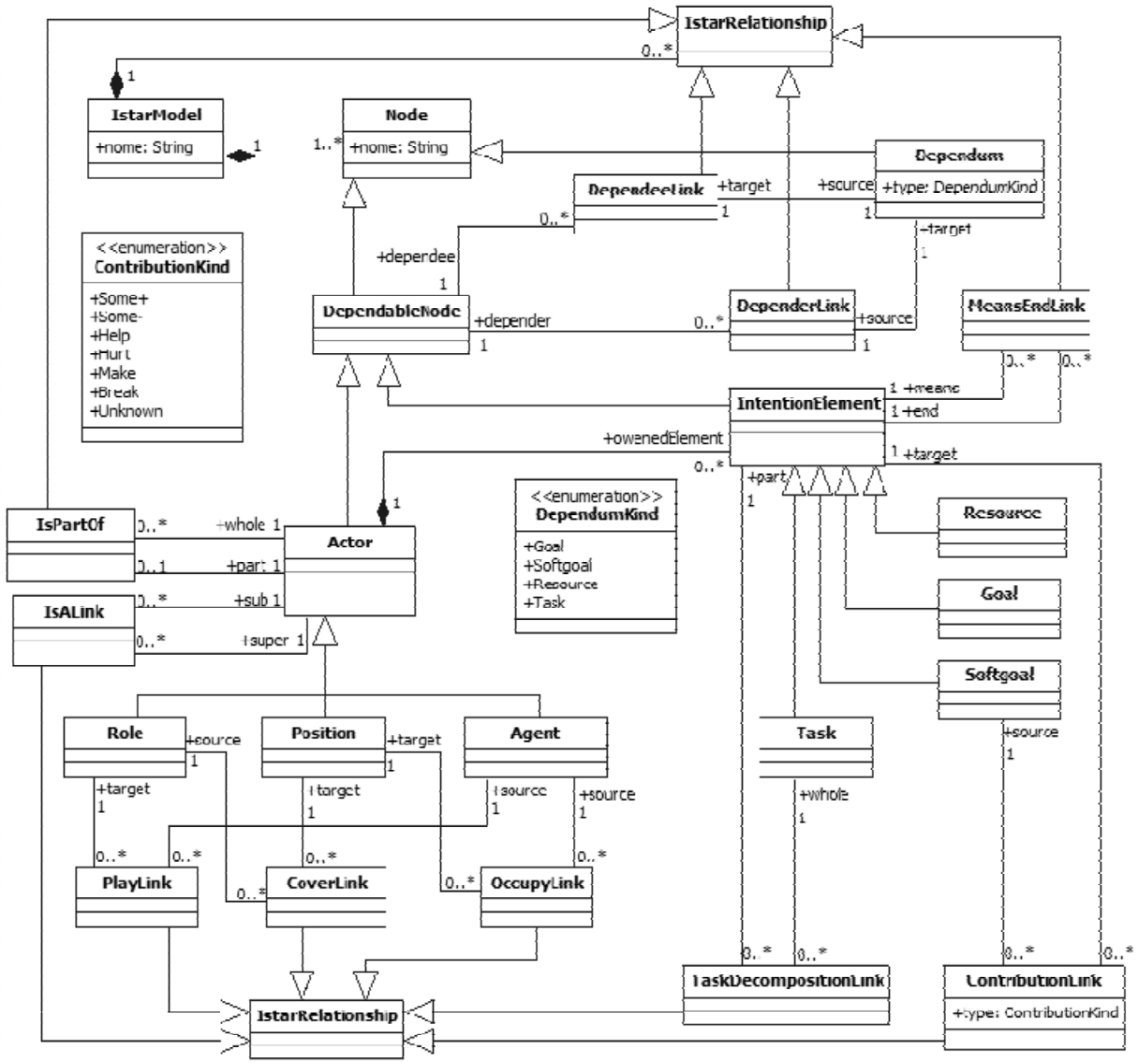


Figura 5.17 Metamodelo da *framework* i*

Um *IntentionElement* pode ser especializado em recurso (*Resource*), objectivo (*Goal*), *Softgoal* ou tarefa (*Task*). Um *IntentionElement* pode também ser um meio (*means*) para atingir outro *IntentionElement* (*end*) através de uma ligação de meio-fim (*MeansEndLink*). Um *Softgoal* pode ter contribuições de outros *IntentionElement* através da ligação de contribuição (*ContributionLink*). Uma *Task* poderá ainda ser decomposta em outras tarefas, através da ligação de decomposição *TaskDecompositioLink*.

Um *Actor* também pode conter zero ou mais *IntentionElement*, ser um subtipo de outro *Actor* através da ligação *IsALink*, fazer parte de outros *Actor* 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.

A Figura 5.18 ilustra alguns conceitos apresentados no metamodelo da *framework* i* da Figura 5.17, com o exemplo do modelo SR para o actor Loja Multimédia.

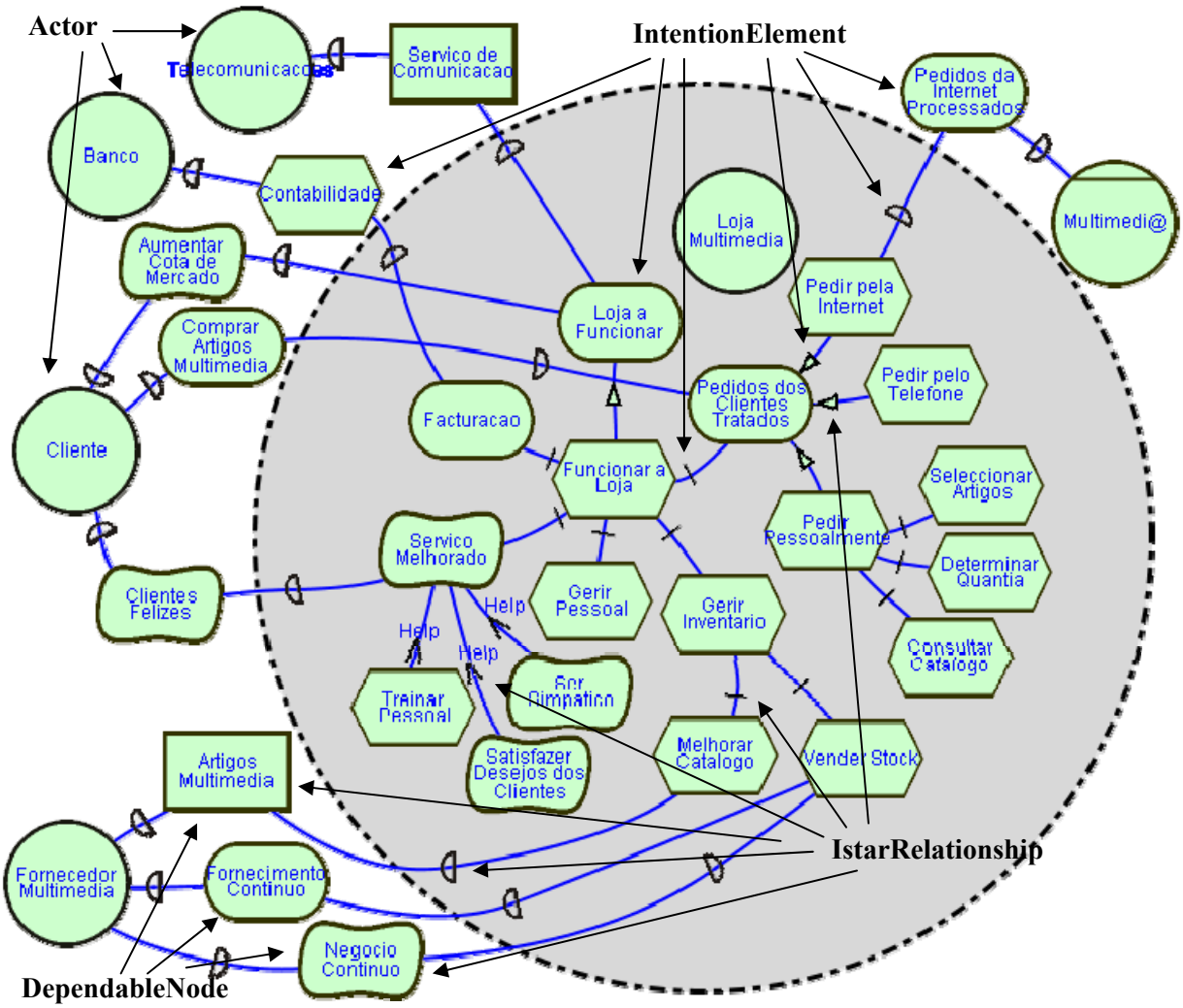


Figura 5.18 Identificação de alguns conceitos do metamodelo, num modelo SR

Este modelo indica os conceitos de *Actor*, *IntetionElement*, *DependableNode* e *IstarRelationship* do metamodelo, num modelo i*.

5.3.2 Metamodelo do Modelo de *Features*

A Figura 5.19 representa o metamodelo do modelo de *feature* baseado em cardinalidade, de acordo com Czarnecki *et al.* [27]. Esse metamodelo encontra-se representado num diagrama de classes UML [22] que foi produzido pela ferramenta StarUML [40].

O metamodelo do modelo de *features* encontra-se estruturado do modo descrito a seguir. O modelo de *features* (*FeatureModel*) consiste em pelo menos uma *feature* raiz (*RootFeature*), que representa o elemento raiz de vários diagramas de *features* no modelo. Além da *RootFeatures* existem as *features* agrupáveis (*GroupedFeature*) e as *features* unitárias (*SolitaryFeature*). Dentro de um conjunto apenas é possível agrupar *GroupedFeature*. Por definição, uma *SolitaryFeature* não permite pertencer a uma *GroupedFeature*. Normalmente num modelo de *features* existem muitas *SolitaryFeature*. As *features* podem ter um atributo opcional (*Attribute*) de um determinado tipo (*TypeValue*), e os *Attribute* podem ter um valor opcional. Neste modelo simplificado, apenas foram referidos os atributos palavras (*StringValue*) e inteiros (*IntValue*).

A *FDReference* representa a Referência do Diagrama de *Feature* (do inglês, *Feature Diagram Reference*) e, serve para referenciar uma *RootFeature*. Uma *RootFeature* pode ser referenciada por várias *FDReference*.

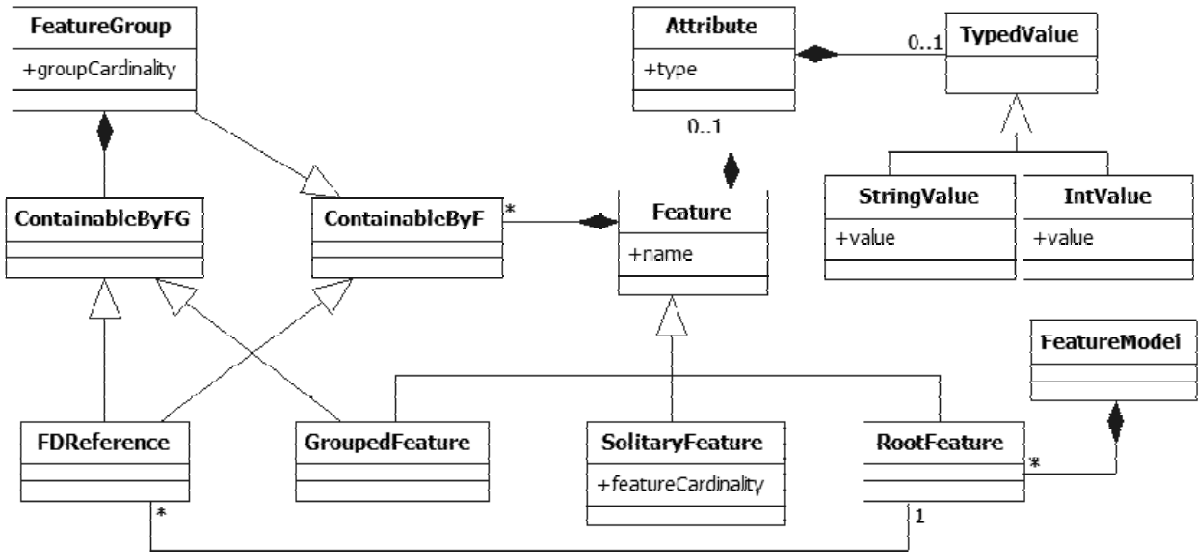


Figura 5.19 Metamodelo do modelo de *features* com cardinalidade

As classes abstractas *ContainableByFG* e *ContainableByF* referem-se a elementos que podem pertencer a uma *GroupedFeature* e a uma *Feature*, respectivamente. Uma *GroupedFeature* pode apenas conter *GroupedFeature* ou *FDReference*, enquanto uma *feature* pode conter *SolitaryFeature, GroupedFeature* e *FDReference*. Uma *sub-feature SolitaryFeature* é

qualificada por uma cardinalidade de *feature* e essa cardinalidade especifica com que frequência uma *sub-feature SolitaryFeature* pode ser duplicada.

Para ilustrar a informação sobre o metamodelo, a Figura 5.20 identifica exemplos de cada um dos conceitos explicados anteriormente.

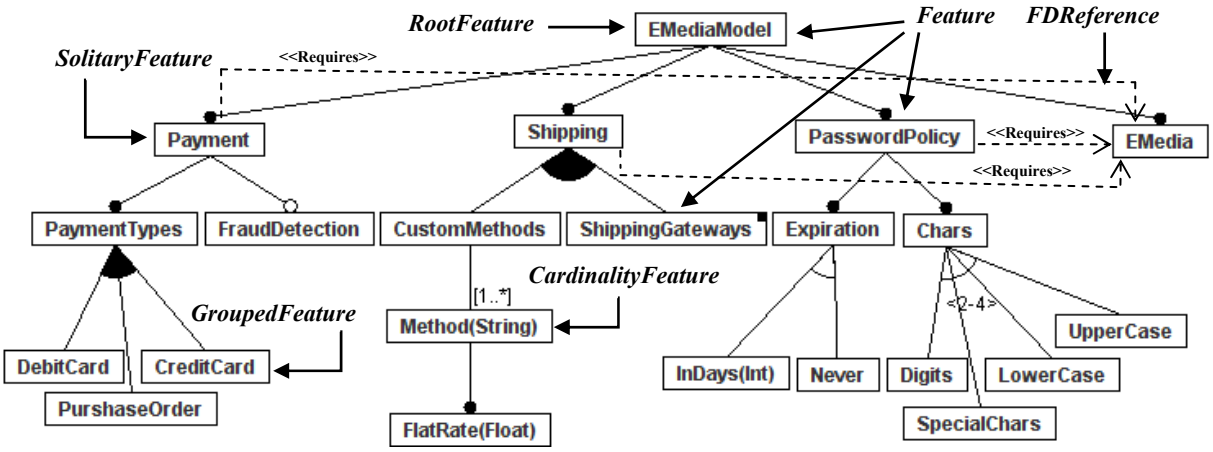


Figura 5.20 Identificação dos conceitos do modelo de *features* baseado em cardinalidade

Os conceitos do metamodelo do modelo de *features* identificados na Figura 5.20 são *RootFeatures*, *SolitaryFeature*, *GroupedFeatures*, *FDReference*, *CardinalityFeature* e *Feature*.

5.3.3 Metamodelo da Abordagem IStarLPS

Uma *feature* pode representar uma característica funcional ou não-funcional de um sistema, em qualquer nível de abstracção de *software* (ex. requisitos, arquitectura, etc.).

Os modelos da *framework* i* contêm informação que permite obter as *features* mais facilmente do que o método tradicional. Utilizando a *framework* i* pretende-se obter as *features* de um modelo de *features* de uma forma sistemática, fornecer informação complementar ao modelo de *features* e produzir o modelo de *features* mais cedo do que ao desenvolvê-lo pelo método tradicional.

O metamodelo da abordagem IStarLPS, ou seja, o relacionamento da *framework* i* com o modelo de *features*, é apresentado na Figura 5.21. O metamodelo apresentado está com três cores: azul, branco e rosa. Na zona azul encontra-se o metamodelo relativo ao modelo da *framework* i* de acordo com Yu [11]. Na zona branca encontram-se as ligações e cardinalidade adicionadas para esta dissertação, com o objectivo de se adaptar os modelos i* para linha de produto. Na zona rosa encontra-se o metamodelo do modelo de *features* de

acordo com Czarnecki [41]. Dividiu-se o metamodelo nestas três zonas de forma a facilitar o seu entendimento. Visto que as zonas em azul e em rosa já foram anteriormente explicadas, esta secção apenas explica os conceitos apresentados na zona branca, que são:

- Ligações entre a classe *Feature* (do metamodelo do modelo de *features*) e as classes *Resource, Goal, e Task* (do metamodelo do *framework* i*).
- A cardinalidade no modelo i* representada na classe *IntentionElement* pelo atributo *cardinality*.

Estes conceitos serão seguidamente explicados mais detalhadamente, recorrendo também ao exemplo da *Loja de Multimédia Online* que ilustra cada uma das situações.

Relativamente às ligações apresentadas no metamodelo da Figura 5.21, elas representam possíveis relacionamentos de:

a) 1 objectivo (*goal*) com 0, 1 ou mais *features*;

b) 1 *feature* com 0, 1 ou mais objectivos;

c) 1 recurso (*resource*) com 0, 1 ou mais *features*;

d) 1 *feature* com 0, 1 ou mais recursos;

e) 1 tarefa (*task*) com 0, 1 ou mais *features*;

f) 1 *feature* com 0, 1 ou mais tarefas.

E ainda:

g) Diferentes elementos i* poderão estar relacionados com 1 *feature*;

h) Um elemento intencional ter uma cardinalidade na sua descrição.

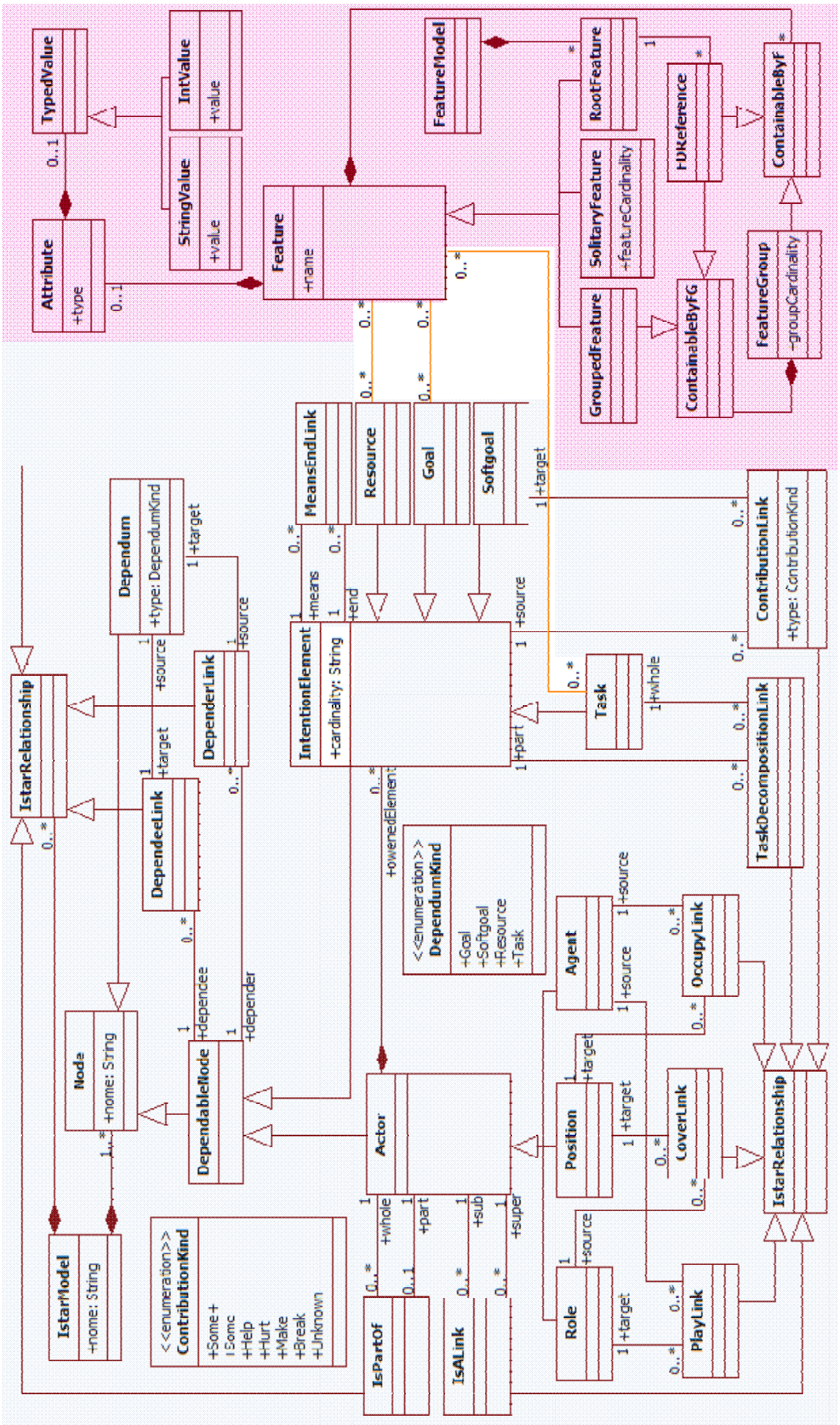


Figura 5.21 Metamodelo da abordagem IStarLPS

Assim, cada um destes pontos será ilustrado com um exemplo do caso de estudo da *Loja de Multimédia Online*, apresentado na secção 5.2.

a) Um objectivo poderá estar associado a nenhuma, uma ou várias *features*, porque cada *feature* representa uma característica relevante do sistema, e essa característica irá permitir a satisfação de zero ou mais objectivos. Logo existe uma relação entre estes dois conceitos. O objectivo será o conceito mais provável para se extrair qualquer um das especificações de *features*, ou seja, poderá ser uma *RootFeature*, raiz de um modelo de *features*, uma *GroupedFeature* ou uma *SolitaryFeature*.

***Exemplo*:** 1 objectivo – 2 *feature*

A partir do objectivo "Obtidos os Métodos de Pagamento Usados" podem-se obter as *features* designadas por "Método" e "Pagamento".

b) Uma *feature* poderá estar associada a nenhum, um ou vários objectivos, pela razão referida no ponto anterior, ou seja, uma *feature* contém uma propriedade relevante do sistema que irá permitir que um objectivo possa ser satisfeito.

***Exemplo*:** 1 *feature* – 2 objectivos

A *feature* "Pagamento" extraída dos objectivos "Pagamento Confirmado" e "Pagamento Processado".

c) Um recurso poderá estar associado a nenhuma, uma ou várias *features*, porque um recurso representa algo que pode ser transaccionado entre o sistema e os actores exteriores ao sistema, e poderá se extrair daí uma *feature* para o modelo de *features*.

***Exemplo*:** 1 recurso – 1 *feature*

O recurso "Palavra-chave" é uma informação que está relacionada com a *feature* "Palavra-chave".

d) Uma *feature* poderá estar associada a nenhum, um ou vários recursos, pela razão referida no ponto anterior, e pode-se verificar que uma *feature* poderá estar explícita ou implicitamente num recurso.

***Exemplo*:** 1 *feature* – 1 recurso

A *feature* "Artigo" relaciona-se com o recurso "Artigo Multimédia".

e) Uma tarefa poderá estar associada a nenhuma, uma ou várias *features*, porque as tarefas num modelo i* representam as acções necessárias a realizar para que um objectivo seja alcançado, sendo que, a partir dessas acções poderão ser obtidas *features*.

***Exemplo*:** 1 tarefa – 2 *feature*

Da tarefa "Obter Dados de Artigo" obtêm-se as *features* "Dados" e "Artigo".

f) Uma *feature* poderá estar associada a nenhuma, uma ou várias tarefas, pela razão referida no ponto anterior.

***Exemplo*:** 1 *feature* – 2 tarefa

A *feature* "Artigo" relaciona-se com as tarefas "Adicionar Artigo" e "Escolher Artigos Disponíveis".

g) Diferentes tipos de elementos i* poderão estar relacionados com uma mesma *feature*.

***Exemplo*:** 1 tarefa e 1 objectivo – 1 *feature*

A tarefa "Gerir Pagamentos" e o objectivo "Pagamento Processado" estão ambos relacionados com a mesma *feature* "Pagamento".

h) Um elemento intencional pode ter uma cardinalidade explícita na sua descrição.

***Exemplo*:** 1 recurso com cardinalidade

O recurso "[0..1] Palavra-Chave" tem a cardinalidade [0..1] indicando que é opcional numa aplicação.

Estes pontos referem-se à obtenção de *features* a partir dos conceitos dos modelos i*, como as *features* se podem relacionar com os conceitos i* e onde pode aparecer cardinalidade nos elementos i*.

No metamodelo proposto, é possível perceber como se relaciona um elemento intencional com uma *feature*. Portanto, o modelo i* será desenvolvido normalmente, sendo a produção do modelo de *features* de acordo com as *features* e ligações obtidas através dos modelos i*. Assim é possível desenvolver o modelo de *features* recorrendo a informação sobre os objectivos da LPS, como é o caso da *framework* i*.

5.4 Resumo

Neste capítulo apresentou-se a abordagem IStarLPS e usou-se o caso de estudo da *Loja de Multimédia Online* para a ilustrar.

Os metamodelos da *framework* i* e do modelo de *features* foram apresentados, tal como o metamodelo da abordagem IStarLPS. Esse metamodelo foi explicado e exemplificado de modo a tentar clarificar os seus conceitos e relações.

Através da utilização do processo exposto na secção 5.1 pode-se aproveitar os conhecimentos previamente adquiridos na utilização separada dos modelos i* e dos modelos de *features*. O processo apresentado baseia-se nos próprios processos da *framework* i* e da ELPS em separado, o que permite uma fácil adaptação ao uso desta abordagem. Para ajudar a confirmar as várias heurísticas descritas na secção 5.2 desenvolveu-se um caso de estudo, descrito no capítulo 6, sobre um *Observador Sanitário* (do inglês *Health-Watcher*). Neste caso de estudo é aplicada a abordagem IStarLPS de modo a mostrar como se utiliza e como funciona o processo.

6. Caso de Estudo e Comparação com Outras Abordagens

Neste capítulo é descrito um caso de estudo do *Observador Sanitário* (*Health-Watcher*) onde se aplicam heurísticas da abordagem IStarLPS. Este sistema controla queixas e notificações e ainda, fornece informação sobre as instituições de saúde aos cidadãos. Seguidamente é feita uma comparação entre a abordagem IStarLPS com outras abordagens.

6.1 Enunciado do Problema – Observador Sanitário

O sistema real do *Observador Sanitário* é adaptado de [42]. A adaptação consiste no desenvolvimento de uma linha de produtos de *Observador Sanitário*. Baseia-se num sistema real de registo e controlo de queixas e um acesso à informação sobre as instituições de saúde por parte dos cidadãos.

Com a distribuição do *Observador Sanitário* pretende-se melhorar a qualidade dos serviços prestados do Sistema de Saúde Pública e ter um controlo de queixas (denúncias e notificações). Como se trata de uma LPS nem todas as aplicações terão disponíveis as mesmas opções.

Os cidadãos podem aceder ao sistema para fazer a sua queixa ou, em alguns produtos é possível obter informação sobre os serviços de saúde. As pesquisas que poderão estar disponíveis são:

- Informação sobre doenças,
    - Descrição, sintomas, prevenção;
- Campanhas de vacinação,
- Informação sobre as suas queixas,
    - Nome e dados do queixoso, data da queixa, descrição e observações da queixa, situação (aberto, suspenso, fechado), análises técnicas, dados das análises, empregado que fez as análises;

- Quais as unidades de saúde que têm determinada especialidade,
- Quais as especialidades de uma determinada unidade de saúde.

Em caso de queixa, esta será registada no sistema e encaminhada para o departamento específico (representado por um assistente registado), o qual tomará conta do caso e dará uma resposta quando as análises estiverem concluídas. Esta solução será registada no sistema, ficando disponível para consulta. As aplicações disponíveis permitem registar 2 a 3 tipos de queixas que são:

- Queixas animais:
    - Casos de apreensão de um animal, controlo de animais patogénicos (roedores, escorpiões, morcegos, etc.), doenças relacionadas com mosquitos ou maus-tratos a animais.
    - Informação adicional necessária: tipo de animal, quantidade de animais, data das perturbações, dados do local das perturbações.
- Queixas alimentares:
    - Casos onde se suspeita de ingestão de comida contaminada.
    - Informação adicional necessária: nome da vítima, dados da vítima, quantidade de pessoas que comeram a comida, quantidade de doentes, quantidade de pessoas que foram hospitalizadas e quantidade de pessoas que faleceram, local de tratamento dos pacientes, refeição suspeita.
- Queixas diversas:
    - Casos relacionados com várias razões, que não estão mencionadas acima (falta de higiene em restaurantes, ruptura nos esgotos, camiões de transporte de água suspeitos, etc.)

Este sistema pode ser usado publicamente, após ter sido instalado em quiosques colocados em vários pontos estratégicos, nos quais os próprios cidadãos podem fazer as suas queixas e notificações.

O sistema deverá ainda permitir trocas de informação com o Sistema de Vigilância Sanitária. Finalmente, todos os meses será enviado, para o director de cada unidade de saúde, um relatório com as estatísticas das frequências dos vários tipos de queixas.

6.1.1 Procedimentos a Nível de Engenharia de Domínio

Como é proposto na abordagem IStarLPS apresentada no Capítulo 4, procede-se inicialmente à produção do modelo SD.

a. Identificar e modelar os objectivos da linha de produtos.

Definir actores. Identificaram-se os seguintes actores:

- Cidadão;
- Assistente de Departamento;
- Director de Unidade de Saúde;
- Sistema de Vigilância Sanitária;
- Observador Sanitário.

Identificar objectivos. Identificaram-se os seguintes objectivos:

- Queixa Registada;
- Informação Pesquisada;
- Consulta Efectuada sobre Queixa.

Definir o modelo SD. Estabelecem-se as ligações de dependência entre os vários actores. Também se identificam e modelam as dependências de tarefas, *softgoals* e recursos.

A Figura 6.1 representa o modelo SD da linha de produtos do *Observador Sanitário*. Um Cidadão pode registar uma queixa, consultar uma queixa ou pesquisar informação. Para um Cidadão poder ter o objectivo "Queixa Registada" satisfeito, tem de fornecer os dados da queixa ao Observador Sanitário. A consulta de uma queixa pode ser efectuada após o cidadão ter indicado o "Número da Queixa" que pretende consultar. O objectivo "Informação Pesquisada" é opcional porque tem cardinalidade [0..1], e para que um Cidadão possa pesquisar informação necessita de indicar o tipo de informação que pretende. O Observador

Sanitário trocará informações sobre as queixas com o Sistema de Vigilância Sanitária. Os Directores de Unidades de Saúde recebem um relatório com as estatísticas. Cada departamento tem assistentes registados, mas para isso têm de fornecer os seus dados ao Observador Sanitário. Os vários tipos de queixas vão sendo registados e redireccionados para os respectivos departamentos. Em cada departamento um assistente após receber a queixa, analisa-a e regista uma resposta a essa queixa.

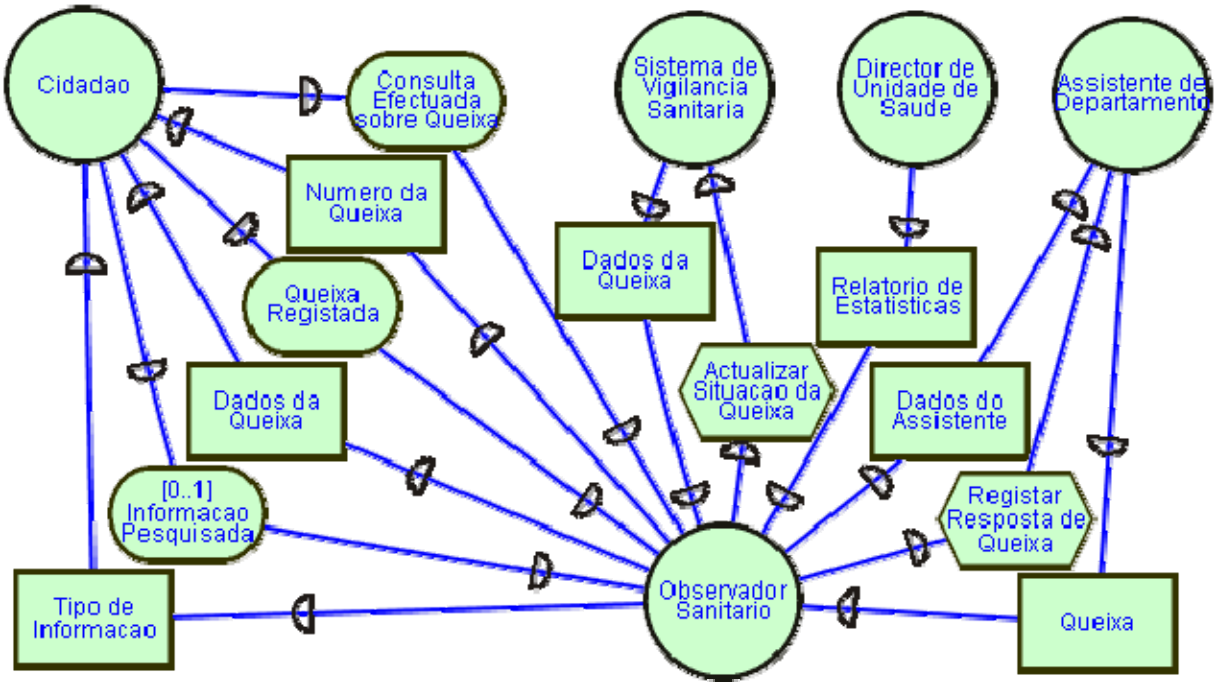


Figura 6.1 Modelo SD do Observador Sanitário

Validar objectivos e dependências. A validação do modelo é feita através dos *stakeholders*.

b. Obter as *features* através do modelo SD. Obtêm-se as primeiras candidatas a *features* recorrendo ao modelo SD e de acordo com a heurística **H.1** definida na abordagem IStarLPS.

A Tabela 6.1 regista todos os elementos intencionais do modelo SD que se relacionam directamente com o actor que representa a LPS, ou seja o Observador Sanitário, tal como as *features* que são obtidas a partir deles.

Tabela 6.1 Primeira obtenção de *features*

Elemento Intencional		***Features***	
Tipo	**Nome**		
Objectivo	Informação Pesquisada	Informação	
Objectivo	Queixa Registada	Queixa	
Objectivo	Consulta Efectuada sobre Queixa	Consulta	Queixa
Tarefa	Registar Resposta da Queixa	Resposta	Queixa
Tarefa	Actualizar Situação da Queixa	Situação	Queixa
Recurso	Dados da Queixa	Dados	Queixa
Recurso	Tipo de Informação	Tipo	Informação
Recurso	Dados da Queixa	Dados	Queixa
Recurso	Número da Queixa	Número	Queixa
Recurso	Relatório de Estatística	Relatório	Estatística
Recurso	Queixa	Queixa	
Recurso	Dados do Assistente	Dados	Assistente

c. Modelar a satisfação das dependências no modelo SR.

Desenvolver modelo SR. Apenas será expandido o actor que representa a LPS, ou seja o Observador Sanitário, mostrando como são satisfeitas as dependências a ele associadas.

A Figura 6.2 apresenta o modelo SR do actor Observador Sanitário. A tarefa principal designa-se por "Gerir Observador Sanitário" e necessita tratar a informação disponibilizada pelo sistema, gerir os assistentes registados, tratar das queixas e das estatísticas sobre as queixas.

Para que um cidadão possa ter a sua queixa registada é necessário fornecer os dados da queixa. Por sua vez, a tarefa "Tratar Queixa" tem de adicionar o queixoso, enviar os dados da queixa para o assistente e, quando este terminar as análises, registar a resposta da queixa e actualizar a queixa. Os dados da queixa serão enviados para o Sistema de Vigilância Sanitária e depois este actualizará a situação da queixa.

Um cidadão pode consultar uma queixa indicando o número da queixa. A tarefa "Procurar Queixa" necessita de obter os dados da queixa e do queixoso para satisfazer o objectivo "Consulta Efectuada sobre Queixa".

Um "Relatório de Estatísticas" é enviado aos Directores das Unidades de Saúde através da realização da tarefa "Produzir Estatísticas" que necessita do objectivo "Queixas Listadas" e este pode ser satisfeito através das tarefas "Obter Queixa Diversa", "Obter Queixa Alimentar"

ou “Obter Queixa Animal”. Estas 3 sub-tarefas têm uma cardinalidade (1..3) e uma ligação meio-fim para o objectivo “Tipos de Queixas Listadas”, por isso as *features* daí obtidas pertencerão a uma alternativa *or*.

O objectivo “Informação Pesquisada” tem uma cardinalidade [0..1] indicando que é opcional numa aplicação e, pode ser satisfeito através do objectivo “Informação Obtida”. Para um Cidadão pesquisar informação é necessário ele indicar que tipo de informação pretende. A informação a ser obtida pode ser sobre campanhas de vacinação, doenças, especialidades listadas por unidades de saúde ou unidades de saúde listadas por especialidades. Os sub-objectivos “Campanhas de Vacinação Listadas”, “Doenças Listadas”, “Especialidades Listadas por Unidades de Saúde” ou “Unidades de Saúde Listadas por Especialidades” têm uma cardinalidade (1..4) e uma ligação meio-fim para o objectivo “Informação Obtida”, o que indica que as *features* daí obtidas pertencerão a um grupo de alternativas *or*.

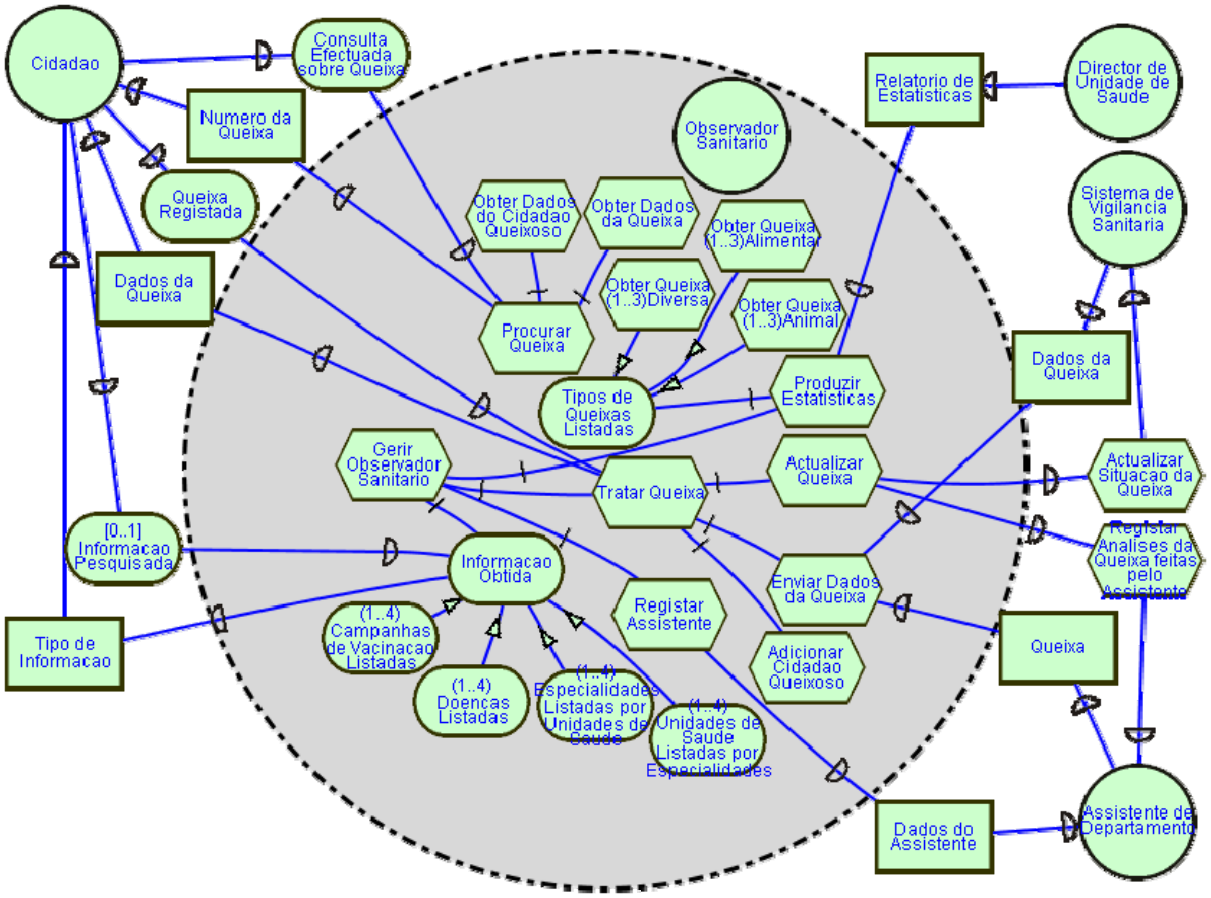


Figura 6.2 Modelo SR do Observador Sanitário

Validar as decisões do SR. A validação é feita através dos *stakeholders* que irão confirmar as várias decisões para a construção do modelo SR.

d. Modelar as *features* restantes.

Obter e analisar as *features*. As restantes *features* são obtidas recorrendo à heurística **H.1** e, a Tabela 6.2 reúne essas *features* com as da Tabela 6.1.

Tabela 6.2 Segunda obtenção de *features* recorrendo ao modelo SR

Elemento Intencional		***Features***		
Tipo	**Nome**			
Tarefa	Gerir Observador Sanitário	Observador Sanitário		
Objectivo	Informação Pesquisada	Informação		
Recurso	Tipo de Informação	Tipo	Informação	
Objectivo	Informação Obtida	Informação		
Objectivo	Campanhas de Vacinação Listadas	Campanha de Vacinação		
Objectivo	Doenças Listadas	Doença		
Objectivo	Especialidades Listadas por Unid. de Saúde	Especialidade	Unid. de Saúde	
Objectivo	Unid. de Saúde Listadas por Especialidades	Unid.de Saúde	Especialidade	
Tarefa	Registar Assistente	Assistente		
Recurso	Dados do Assistente	Dados	Assistente	
Objectivo	Queixa Registada	Queixa		
Tarefa	Tratar Queixa	Queixa		
Tarefa	Adicionar Cidadão Queixoso	Cidadão		
Tarefa	Enviar Dados da Queixa	Dados	Queixa	
Recurso	Dados da Queixa	Dados	Queixa	
Tarefa	Actualizar Queixa	Queixa		
Recurso	Dados da Queixa	Dados	Queixa	
Recurso	Queixa	Queixa		
Tarefa	Registar Análises da Queixa feitas pelo Assistente	Análise	Queixa	Assistente
Tarefa	Actualizar Situação da Queixa	Situação	Queixa	
Objectivo	Consulta Efectuada sobre Queixa	Consulta	Queixa	
Recurso	Número da Queixa	Número	Queixa	
Tarefa	Procurar Queixa	Queixa		
Tarefa	Obter Dados do Cidadão Queixoso	Dados	Cidadão	
Tarefa	Obter Dados da Queixa	Dados	Queixa	
Objectivo	Tipos de Queixas Listadas	Tipo	Queixa	
Tarefa	Obter Queixa Diversa	Diversa		
Tarefa	Obter Queixa Alimentar	Alimentar		
Tarefa	Obter Queixa Animal	Animal		
Recurso	Relatório de Estatística	Relatório	Estatística	
Tarefa	Produzir Estatísticas	Estatística		

A Figura 6.3 representa uma primeira versão do modelo de *features* recorrendo à Tabela 6.2 e ao modelo SR da Figura 6.2. As heurísticas serão mencionadas de acordo com a necessidade de utilização para este caso de estudo, não serão por isso usadas por sequência numérica.
Assim, segundo a heurística **H.5** a *feature* "Informação" é opcional. As *features* "Especialidade", "Campanha de Vacinação", "Doenças" e "Unidades de Saúde" pertencem a um grupo de alternativas *or* de acordo com a heurística **H.6**. As *features* "Animal", "Alimentar" e "Diversa" também se encontram agrupadas num grupo *or*, segundo a heurística **H.6**. As hierarquias nesta versão foram encontradas recorrendo às heurísticas **H.6, H.9** e **H.10**.

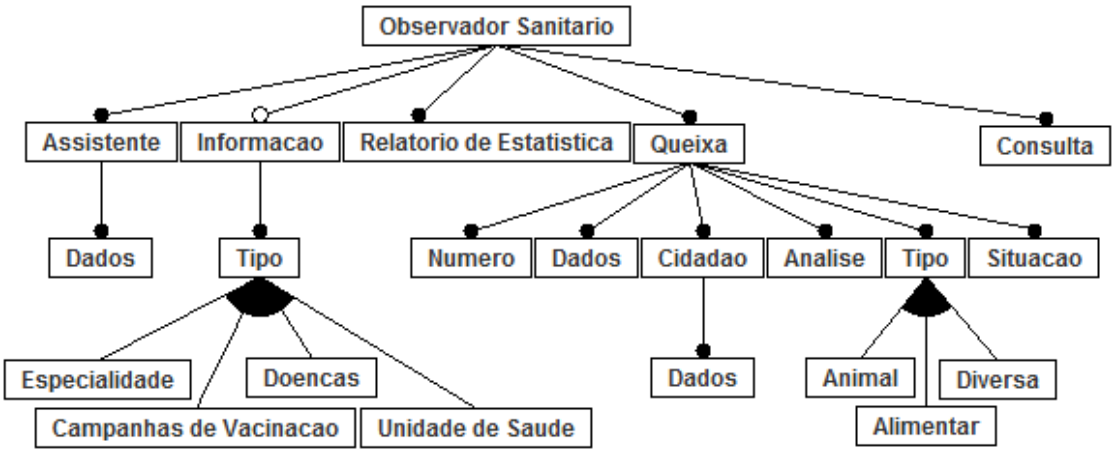


Figura 6.3 Primeira versão do modelo de *features*

A Figura 6.4 ilustra uma versão mais refinada do que a da Figura 6.4Figura 6.3. Nesta segunda versão podem-se encontrar *features* que requerem outras e, alterações para clarificar nomes de algumas *features*. Distinguem-se duas *features* diferentes com o mesmo nome "Tipo", portanto para as distinguir adaptou-se os seus nomes para "TipoI" e "TipoQ" relativo ao tipo de informação e tipo de queixa, respectivamente. Passaram a distinguir-se três tipos de dados, "DadosA" do assistente, "DadosC" do cidadão queixoso e "DadosQ" da queixa.

A *feature* "Relatório de Estatística" requer a *feature* "Queixa" segundo as heurísticas **H.10** e **H.11**. A *feature* "Análise" requer a *feature* "Assistente" de acordo com as heurísticas **H.3** e **H.11**. E a *feature* "Consulta" requer a *feature* "Queixa" segundo as heurísticas **H.3** e **H.11**. A *feature* "Especialidade" requer "Unidade de Saúde" e a "Unidade de Saúde" requer a "Especialidade" segunda as heurísticas **H.3** e **H.11**.

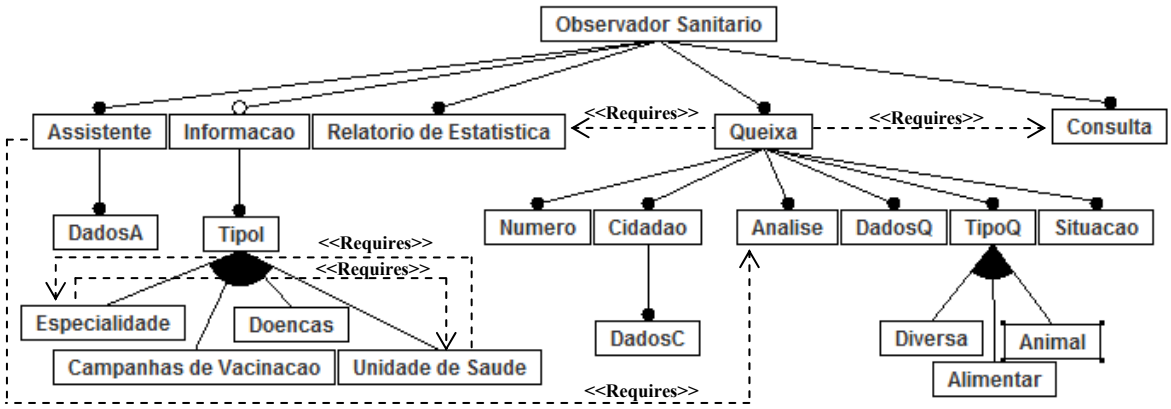


Figura 6.4 Segunda versão do modelo de *features*

Validação do modelo de *features*. A validação é feita através dos *stakeholders* que irão confirmar ou rectificar as várias opções do modelo de *feature*.

6.1.2 Procedimentos a Nível de Engenharia de Aplicação

Nesta secção apresenta-se os modelos de *features*, SD e SR para uma configuração de uma aplicação da LPS do *Observador Sanitário*. A aplicação permite aceder a informação sobre campanhas de vacinação e doenças, além de permitir que se registem queixas diversas e queixas sobre animais.

A. Configurar o modelo de *features* para uma aplicação específica. A Figura 6.5 representa o modelo de *features* para a configuração acima descrita.

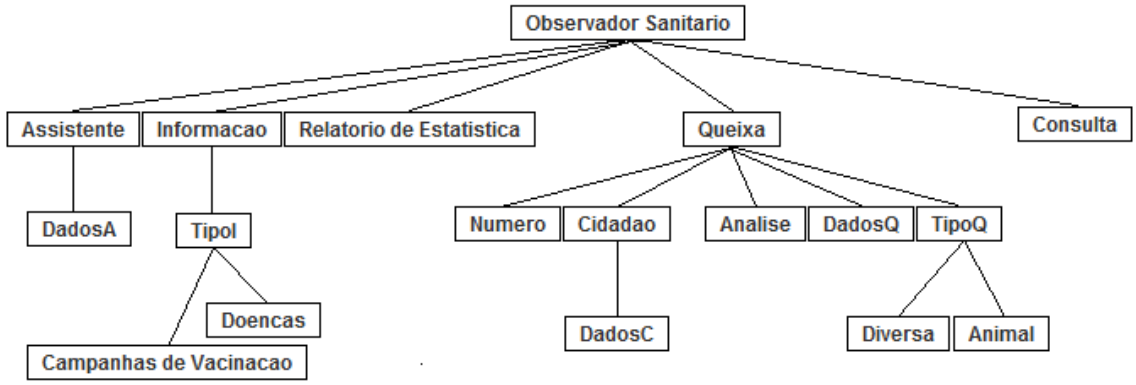


Figura 6.5 Configuração do modelo de *features* para uma aplicação específica

B. Configurar os modelos i* para uma aplicação específica. A Figura 6.6 representa o modelo SD e a Figura 6.7 representa o modelo SR para a configuração da aplicação descrita acima. Na configuração do modelo SR foram removidos os elementos intencionais que não são necessários para esta aplicação específica.

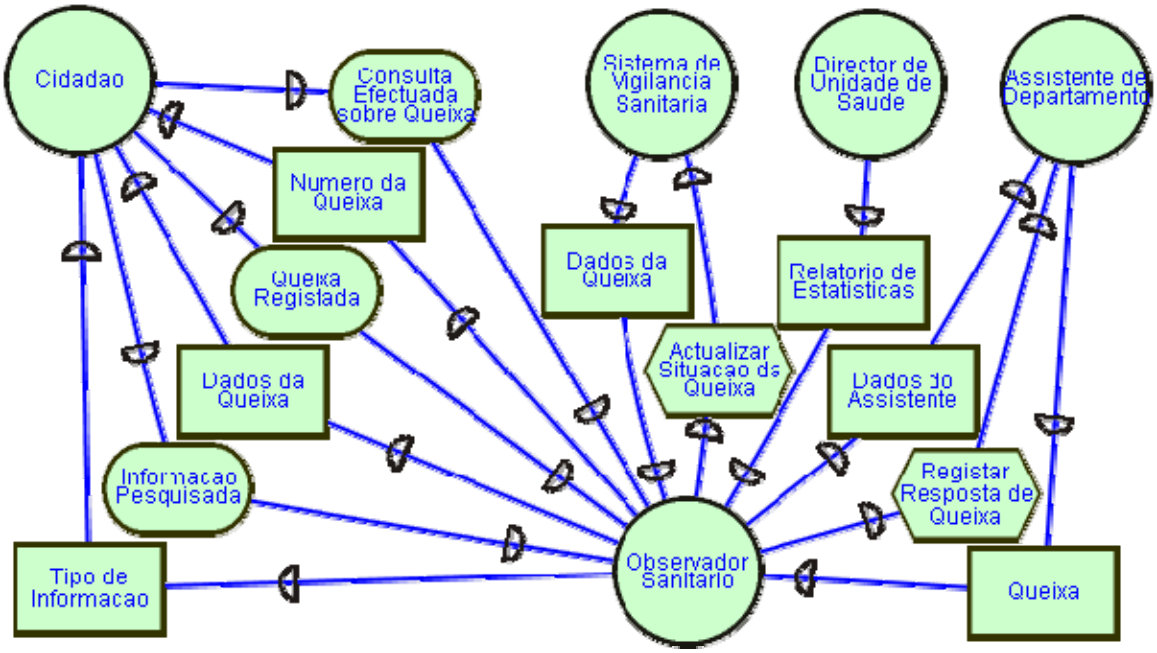


Figura 6.6 Configuração do modelo SD para uma aplicação específica

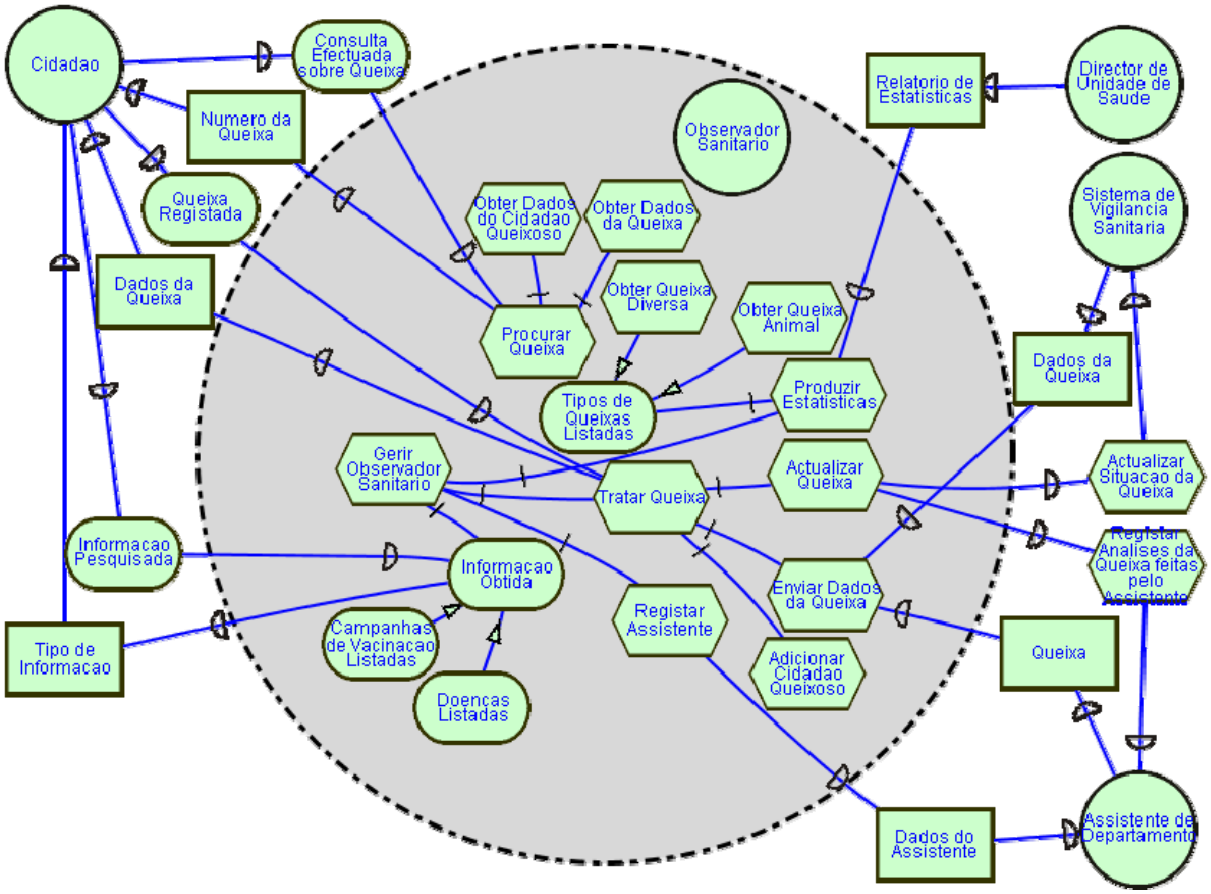


Figura 6.7 Configuração do modelo SR para uma aplicação específica

Aplicou-se a abordagem IStarLPS para o caso de estudo do *Observador Sanitário* e foi possível encontrar algumas situações que foram melhoradas. É útil ter em atenção o modo

como se descrevem os elementos intencionais e o modo como se relacionam no modelo i* pois vão influenciar o desenvolvimento do modelo de *features*.

6.2 Comparação da Abordagem IStarLPS com Outras Abordagens

Nesta secção compara-se a abordagem IStarLPS com as abordagens de linhas de produtos de *software* descritas no Capítulo 4. Serão apresentados os critérios de comparação e, posteriormente é feita a comparação entre as abordagens descritas nos trabalhos relacionados e a abordagem IStarLPS.

6.2.1 Critérios de Comparação

Para se proceder à comparação entre as abordagens utilizaram-se alguns critérios descritos no projecto AMPLE [43].

Utilizou-se 5 critérios de comparação que são relativos a linhas de produto de *software*. Os critérios são definidos da seguinte forma:

Modelação de requisitos: Consistem em actividades para capturar os requisitos funcionais e de arquitectura da linha de produtos. Também são definidas as dependências entre essas actividades. Pode incluir um mapeamento de restrições específicas dos requisitos.

Modelação dos pontos comuns e diferentes: Consiste em actividades de identificação dos pontos comuns e diferentes entre os requisitos. Inclui a separação dos requisitos que são válidos para todo o domínio, daqueles que são apenas válidos em casos especiais, e.g. para uma variante específica do produto. Esta actividade é fortemente relacionada com a modelação do domínio e modelação de *features*.

Modelação do domínio: Consiste em actividades que especificam o domínio. Também contempla as dependências entre essas actividades.

Modelação de features: Consiste em actividades para identificar, estudar e descrever as *features* relevantes para um dado domínio. O objectivo da modelação de *features* é expressar as relações entre *features*, propriedades das *features*, e/ou superstruturas de *features*. A possibilidade de ver os pontos comuns e diferentes da LPS é uma importante característica. Também é permitida a configuração e interacção de *features*. A modelação de *features* pretende ajudar a estruturar os requisitos e definir as variações permitidas numa linha de produtos.

Suporte para engenharia de aplicação: Permite identificar as variações entre os requisitos de uma instância de LPS e os requisitos da LPS.

6.2.2 Aplicação da Comparação

Para cada um dos critérios definidos será assinalada uma das seguintes opções:

Sim ou "*Tipo de Abordagem"*: A característica verifica-se na abordagem;

Não: A característica não se verifica na abordagem;

Não Especificado: A característica não se encontra especificada para a abordagem.

As abordagens a comparar são:

- IStarLPS proposta nesta dissertação;
- Casos de uso e LPS de Gomaa [32];
- i* aspectual e modelo de *features* de Silva *et al.* [33];
- EROO e modelo de *features* de Yu *et al.* [34];
- LPS e MATA de Jayaraman *et al.* [35].

A Tabela 6.3 regista a satisfação dos critérios definidos na em 6.2.1 para cada umas das abordagens.

Faz sentido proceder à comparação entre estas abordagens com estes critérios, pois cada uma delas foi ajustada para ser usada em linhas de produto de *software*, e estão sumariamente descritas no capítulo 4.

A abordagem orientada a casos de uso e LPS satisfaz praticamente todos os critérios de comparação, com a excepção de não ter suporte para a engenharia de aplicação. Por sua vez, a abordagem IStarLPS satisfaz todos os critérios, mas é importante realçar o facto destas abordagens se basearem em paradigmas distintos, um é orientado a objectivos e o outro orientado a casos de uso.

Segundo a Tabela 6.3, a abordagem i* aspectual e modelo de *features* satisfaz os critérios tal como a IStarLPS. Embora a abordagem i* aspectual e modelo de *features* satisfaça esses critérios, ela baseia-se na condição de que todas as *features*, que não sejam comuns às

aplicações, serão sempre modeladas como aspectos, e isso não será sempre verdade. De qualquer modo, a abordagem IStarLPS é uma abordagem orientada a objectivos que foi adaptada para suportar linhas de produtos, enquanto a i* aspectual e modelo de *features* é uma abordagem orientada a objectivos e aspectos que tentaram utilizar para representar os conceitos de um modelo de *features*.

Tabela 6.3 Comparação entre abordagens

Abordagens / **Critérios**	**IStarLPS**	**Casos de uso e LPS**	**i* aspectual e modelo de *features***	**EROO e modelo de *features***	**LPS e MATA**
Modelação de requisitos	Orientada a Objectivos	Orientada a Casos de Uso	Orientada a Aspectos e Objectivos	Orientada a Objectivos	Orientada a Aspectos
Modelação dos pontos comuns e variáveis	Sim	Sim	Sim	Sim	Sim
Modelação do domínio (Modelo de)	Objectivos i*	Casos de Uso	Objectivos i*	Objectivos Genérico	Objectos
Modelação de *features*	Sim	Sim	Sim	Sim	Sim
Suporte para engenharia de aplicação	Sim	Não	Sim	Não especificado	Não especificado

A abordagem EROO e modelo de *features* não especifica, na sua documentação, se tem suporte para a engenharia de aplicação, mas os restantes critérios encontram-se satisfeitos. Relativamente à comparação da abordagem IStarLPS com EROO e modelo de *features* é importante referir que a intenção das abordagens é diferente, pois a IStarLPS pretende desenvolver o modelo de *features* o mais completo possível, recorrendo ao modelo orientado

a objectivos, enquanto a EROO e modelo de *features* pretende obter apenas um modelo de *features* inicial.

Por fim, a abordagem LPS e MATA satisfaz quase todos os critérios com a excepção de não ter especificado, na sua documentação, o suporte para engenharia de aplicação. Ao comparar a abordagem IStarLPS com a LPS e MATA, é importante referir que a abordagem IStarLPS é orientada a objectivos e satisfaz todos os critérios, enquanto a LPS e MATA é orientada a objectos e aspectos, sendo por isso baseadas em diferentes paradigmas.

6.3 Resumo

Neste capítulo aplicou-se a abordagem IStarLPS ao caso de estudo do Observador Sanitário.

Comparou-se a abordagem IStarLPS com as abordagens: casos de uso e LPS, i* aspectual e modelo de *features*, EROO e modelo de *features*, LPS e MATA. Conclui-se com isto, que as várias abordagens apresentadas usam paradigmas diferentes que se relacionam com linhas de produtos de *software*. Enquanto isso, a abordagem IStarLPS além de satisfazer todos os critérios analisados, apresenta uma proposta mais completa que a abordagem EROO e modelo de *features*, mais fiável que a abordagem i* aspectual e modelo de *features* e, permite uma representação das *features* numa fase mais precoce do que as abordagens casos de uso e LPS ou LPS e MATA.

O próximo capítulo apresenta as contribuições e limitações desta dissertação, e também indica alguns trabalhos futuros.

7. Conclusão

Para se adaptar a *framework* i* com linhas de produto foi necessário investigar abordagens orientadas a objectivos e de LPS, dando maior relevo à abordagem *framework* i* e aos modelos de *features*, visto serem fundamentais para esta dissertação. A abordagem IStarLPS foi apresentada no Capítulo 5 através da descrição do processo que permite produzir um modelo de *features* recorrendo aos modelos i*. No Capítulo 6 desenvolveu-se o caso de estudo do *Observador Sanitário* (*Health Watcher System*) de acordo com a abordagem IStarLPS. Nas próximas secções são apresentadas as contribuições e limitações desta dissertação tal como trabalhos futuros.

7.1 Contribuições e Limitações do Trabalho

Esta dissertação apresenta como se pode desenvolver uma LPS, através da modelação dos seus objectivos e das várias maneiras de os satisfazer, recorrendo à *framework* i*. Por conseguinte, as principais contribuições desta dissertação são: a beneficiação da engenharia de requisitos da ELPS com a EROO e, a extensão da utilização da *framework* i* para as linhas de produtos.

Outra importante contribuição centra-se na extracção de *features* e obtenção de informação sobre a variabilidade e a interacção entre as *features*, de uma forma sistematizada e flexível.

A inserção de cardinalidade nos modelos i* que permite representar a variabilidade de uma linha de produtos também é outra contribuição importante para a utilização da *framework* i* em LPS. A cardinalidade nestes modelos torna mais fácil a extracção da informação para a produção do modelo de *features*.

Porém, de acordo com os objectivos desta dissertação podem-se apontar como limitações ao trabalho realizado, a falta de uma ferramenta de suporte à abordagem IStarLPS, pois a existência de uma ferramenta encoraja a utilização das abordagens. Também seria interessante que a abordagem IStarLPS permitisse encontrar *features* através da análise dos *softgoals*, mas de qualquer modo os modelos de *features* tratam principalmente os requisitos funcionais dos

sistemas. Contudo, estas limitações servem como motivação para trabalho futuro que será apresentada na secção seguinte.

7.2 Trabalho Futuro

Para complementar a abordagem IStarLPS seria interessante obter informação dos *softgoals* para melhorar os modelos de *features*. Também se pode realizar como trabalho futuro outros casos de estudo baseados em sistemas reais aplicando a abordagem IStarLPS de maneira a melhorá-la. Ainda, se poderia implementar uma ferramenta que suportasse a abordagem IStarLPS visando a promoção da utilização da abordagem. Estes são apenas alguns dos possíveis trabalhos que se poderão realizar futuramente e que poderão melhorar a abordagem IStarLPS e consequentemente a engenharia de requisitos das linhas de produto de *software*.

8. Bibliografia

1. Lamsweerde A. Goal-Oriented Requirements Engineering: A Guided Tour *In RE'01 - 5th IEEE Inter. Symposium on Requirements Engineering*: Toronto, 2001.
2. Sommerville I, Sawyer P. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.
3. Jacobson I, Christerson M, Jonsson P, Overgaard G. *Object-oriented Software Engineering: A Use Case Approach*. 1992.
4. Lamsweerde A. Requirements Engineering in the Year 00: A Research Perspective *Proc. of the 22nd Inter. Conference on Software engineering*: Ireland 2000;5-19.
5. Clements P, Northrop L. *Software Product Lines: Practices and Patterns*. Addison-Wesley: Boston, MA, USA, 2007.
6. Pohl K, Böckle G, Van Der Linder F. *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer, 2005.
7. Czarnecki K, Antkiewicz M. Mapping features to models: A template approach based on superimposed variants. In: Springer-Verlag (ed) *the 4th Inter. Conf. on Generative Programming and Component Engineering*, 2005; 422 – 437.
8. Rashid A, Sawyer P, Moreira A, Araújo J. Early aspects: a model for aspect-oriented requirements engineering *Requirements Engineering, 2002. Proc. IEEE Joint Inter. Conf. on*, 2002; 199-202.
9. Finkelstein A, Sommerville I. The Viewpoints FAQ. *BCS/IEE Software Engineering Journal* 1996; 11(1)
10. Lamsweerde A. KAOS Tutorial. Cediti, 2003.
11. Yu E. Modelling Strategic Relationships for Process Reengineering *Dept. Of Computer Science*. University of Toronto, 1995.
12. GRL web site, http://www.cs.toronto.edu/km/GRL/, Last access May 2008
13. Mussbacher G, Amoyt D, Araújo J, Moreira A, Weiss M. Visualizing Aspect-Oriented Goal Models with AoGRL *2nd Inter. Requirements Engineering Visualization*, 2007.
14. Chung L, Nixon B, Yu E, Mylopoulos J. Non-Functional Requirements in Software Engineering. Kluwer Academic: Boston, 2000.
15. Yu Y, Leite J, Mylopoulos J. From goals to aspects: discovering aspects from requirements goal models*Proc. of the 12th IEEE Inter. Requirements Engineering Conference*: Kyoto, Japan, 2004; September.
16. Lapouchnian A. Goal-Oriented Requirements Engineering: An Overview of the Current Research. University of Toronto, 2005.
17. Santander V, Castro J. Deriving use cases from organizational modeling *Requirements Engineering, 2002. Proc. IEEE Joint Inter. Conf. on*, 2002; 32-39.
18. Yu E. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering *Proc. RE-97 - 3rd Inter. Symposium on Requirements Engineering*: Annapolis, 1997; 226 - 235.
19. Castro J, Kolp M, Mylopoulos J. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems* 2002; 27(6): 365-389
20. OME 3.1, http://www.cin.ufpe.br/~eti907/ferramentas/ome310j.zip, tool website, Last access May 2008.
21. Objectiver web site, http://www.objectiver.com, Last access April 2008.

22. UML web site, http://www.uml.org/, Last access May 2008.
23. Liu L, Yu E. Designing information systems in social context: a goal and scenario modelling approach. *Information Systems* 2004; 29(2): 187-203
24. Silva L, Sampaio J. Generating Requirements Views: A Transformation-Driven Approach*In: Proc. of 3rd Workshop on Software Evolution through Transformations*: Rio de Janeiro, Brazil, 2006.
25. Nunes C, Nunes I. Uma linha de Produto para Sistemas Multi-Agente. Laboratório de Engenharia de Software, Dept. De Informática – PUC – Rio, 2007.
26. Metzger A, Pohl K. Variability Management in Software Product Line Engineering *Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th Inter. Conf. on*, 2007; 186-187.
27. Czarnecki K, Helsen S, Eisenecker U. Staged configuration through specialization and multi-level configuration of feature models. In: Practice SPIa (ed), 2005;143-169.
28. Kang K, Cohen S, Hess J, Novak W, Peterson A. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Carnegie Mellon University, Software Engineering Institute, 1990.
29. Czarnecki K, Eisenecker S. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice* 2005; 10(1): 7-29
30. Araújo J. Slides das aulas de ERDS. FCT-UNL-DI, 2008.
31. CaptionFeatures 1.0 web site, https://sourceforge.net/projects/captainfeature/, Last access June 2008.
32. Gomaa H. *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. Addison-Wesley, 2004.
33. Silva C, Alencar F, Araújo J, Moreira A, Castro J. Tailoring an Aspectual Goal-Oriented Approach to Model Features *The 20th Inter Conf. on Software Engineering and Knowledge Engineering* San Francisco Bay, USA, 2008.
34. Yu Y, Leite J, Lapouchnian A, Mylopoulos J. Configuring features with stakeholder goals*Proc. of the 2008 ACM Symposium on Applied computing,*. ACM: Brazil, 2008.
35. Jayaraman P, Whittle J, Elkhodary AM, Gomaa H. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis *Lecture notes in computer science*. Springer, 2007; 151.
36. Whittle J, Jayaraman P. MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation. *Workshop on Aspect Oriented Modeling at the Inter. MODELS Conf., Nashville, TN* 2007
37. Alencar F, Silva C, Lucena M, Castro J, Santos E, Ramos R. Improving the Understandability of i* Models *Proc. of the 10th Inter. Conf. on Enterprise Information Systems*: Barcelona, 2008.
38. Mylopoulos J, Borgida A, Jarke M, Koubarakis M. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems* 1990; 8(4): 325-362
39. Object Management Group (OMG), Meta-Object Facility (MOF), Available at http://www.omg.org/docs/formal/05-07-04.pdf, Last access July 2008.
40. StarUML Web site, http://staruml.sourceforge.net/en/, Last access June 2008.
41. Czarnecki K, Helsen S, Eisenecker U. Staged Configuration Using Feature Models *Software Product Lines: 3th Inter. Conf. SPLC 2004*: Boston MA USA, 2004; 266-283.
42. Soares S, Borba P, Laureano E. Distribution and persistence as aspects. *Softw. Pract. Exper.* 2006; 36(7): 711-759. DOI http://dx.doi.org/10.1002/spe.v36:7
43. Kovačević J, Aférez M, Kulesza U, Moreira A, Araújo J, Amaral V, Alves V, Rashid A, Chitchyan R. Survey of the state-of-the-art in Requirements Engineering for Software Product Line and Model-Driven Requirements Engineering, 2007.