Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado em Engenharia Informática
1º Semestre, 2009/2010

Adaptação da Abordagem Theme para Linhas de Produtos de Software

Por

Inês Carvalho Nunes Simão (27179)

Dissertação apresentada Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do Grau de Mestre em Engenharia Informática.

Orientador: Prof. Doutor João Baptista da Silva Araújo Júnior

Lisboa

2010

Nº do aluno: 27179
Nome: Inês Carvalho Nunes Simão

Título da dissertação:
Adaptação da Abordagem Theme para Linhas de Produtos de Software

Palavras-Chave:

- Engenharia de Requisitos Orientadas a Aspectos
- Abordagem Theme
- Linhas de Produto de Software
- Modelo de *Features*

Keywords:

- Aspect-Oriented Requiremenst Engineering
- Theme Approach
- Software Product Lines
- Feature Model

Agradecimentos

Quero agradecer a todos os que de uma maneira ou de outra me ajudaram na realização deste trabalho.

Primeiro quero agradecer ao meu orientador João Araújo pela sua orientação, sugestões e críticas que me ajudaram a desenvolver de melhor modo o trabalho, e também por todo o trabalho, paciência e disponibilidade demonstradas ao longo da dissertação. Agradeço também à minha colega e amiga Patrícia Varela, pelo apoio e ajuda durante esta dissertação. Quero também agradecer aos colegas João Santos, que me ajudou na criação do metamodelo para o modelo de *features*, e Vasco Sousa, que me ajudou no desenvolvimento da ferramenta criada para este trabalho, pelo tempo e paciência dispendidos.

Quero agradecer também ao meu namorado Tiago pela paciência, compreensão e carinho que teve nesta fase, que apesar de eu nem sempre estar disponível ele estava sempre lá para mim.

No final, mas não menos importante, agradeço também aos meus pais pelo apoio, força e incentivo na realização desta dissertação.

Obrigada a todos.

Resumo

Parte da investigação na área de requisitos para Linhas de Produtos de Software (LPS) tem sido realizada de modo a estudar a forma pela qual se podem definir e estruturar artefactos de requisitos da forma mais modularizada possível. O objectivo é que esses artefactos sejam capazes de servir como base para a derivação rentável de produtos e também a fim de facilitar a sua evolução.

Na especificação de LPS, a modelação de *features* é uma técnica chave para capturar pontos comuns e variáveis nas famílias de sistemas de linhas de produtos. Uma *feature* pode denotar qualquer característica funcional ou não funcional ao nível dos requisitos. Contudo, os modelos de *features* mostram uma perspectiva muito específica das linhas de produtos, sendo necessário ter uma abordagem que mostre outras perspectivas ao nível dos requisitos, onde a modularização se deve ter em conta.

Uma das abordagens de engenharia de requisitos que endereça de forma eficiente a modularização de requisitos é a abordagem Theme, uma vez que adopta o paradigma da orientação a aspectos que se caracteriza por identificar, modelar e compor requisitos transversais, facilitando assim a evolução de sistemas.

Contudo, o uso de Theme ainda não foi suficientemente explorado para descrever linhas de produtos. Os modelos de Theme, tal como outras abordagens de requisitos, oferecem uma forma natural de identificar pontos comuns e variáveis nas fases iniciais de requisitos, podendo ser combinados com o modelo de *features* e fornecendo uma abordagem mais expressiva para a engenharia de requisitos para LPS.

Deste modo, este trabalho visa desenvolver uma abordagem que investiga como o desenvolvimento de LPS pode ser beneficiado com a integração de uma abordagem orientada a aspectos, nomeadamente a abordagem Theme.

Abstract

Part of the research in requirements for software product lines (SPLs) has been done in order to study the form in which you can define and structure requirements artifacts as modularized as possible. The goal is that these artifacts are capable of being the basis for cost-effective derivation of products, and also to facilitate its evolution.

In SPL specification, feature modeling is a key technique for capturing commonalities and variabilities in system families and product lines. A feature may denote any functional or non-functional characteristic at the requirements level. However, feature models show a very specific perspective of a product line, so it is necessary to have an approach that shows other perspectives at requirements level, where modularization must be taken into account.

One of the requirements engineering approaches that address efficiently the modularization of requirements is the Theme approach, since it adopts the aspect-oriented paradigm that is characterized by identifying, modeling and composing crosscutting requirements, thus facilitating system's evolution.

However, the use of the Theme approach to describe software product lines has not been sufficiently explored yet. But Theme models, like other requirements approaches, offer a natural way to identify commonalities and variabilities at the early requirements stages, and can be combined with feature model to provide a more expressive approach for SPL requirements engineering.

Thus, this work aims to develop an approach that investigates how the development of SPL can be benefited from the integration of an aspect-oriented approach, namely, the Theme approach.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

Acrónimos

Acrónimo	Significado
AORA	**A**spect-**O**riented **R**equirements **A**nalysis
AORE	**A**spect-**O**riented **R**equirements **E**ngineering
ATL	**A**tlas **T**ransformation **L**anguage
DOM	**D**esenvolvimento **O**rientado a **M**odelos
DSOA	**D**esenvolvimento de **S**oftware **O**rientado pelos **A**spectos
ELPS	**E**ngenharia de **L**inhas de **P**rodutos de **S**oftware
EMF	**E**clipse **M**odeling **F**ramework
ER	**E**ngenharia de **R**equisitos
EROA	**E**ngenharia de **R**equisitos Orientada pelos Aspectos
FODA	**F**eature **O**riented **D**omain **A**nalysis
GME	**G**eneric **M**odeling **E**nvironment
GMF	Eclipse **G**raphical **M**odeling **F**ramework
KAOS	**K**eep **A**ll **O**bjectives **S**atisfied
LDE	**L**inguagens para **D**omínios **E**specíficos
LPS	**L**inhas de **P**rodutos de **S**oftware
M2M	**M**odel To **M**odel
MATA	**M**odeling **A**spects using a **T**ransformation **A**pproach
MDD	**M**odel **D**riven **D**evelopment
MOF	**M**eta **O**bject **F**acility
OCL	**O**bject **C**onstraint **L**anguage
OMG	The **O**bject **M**anagement **G**roup
QVT	**Q**uery / **V**iews / **T**ransformations
RDL	**R**equirements **D**escription **L**anguage
UML	**U**nified **M**odeling **L**anguage
UMLT	**U**nified **M**odeling **L**anguage **T**ransformation
XML	E**X**tensible **M**arkup **L**anguage

1. Introdução

1.1 Engenharia de Requisitos e Linhas de Produtos de Software

A Engenharia de Requisitos (**ER**) tem como finalidade cobrir todas as actividades envolvidas na descoberta, manutenção e gestão de um conjunto de requisitos para um determinado sistema (Kotonya e Sommerville, 1998). Requisitos são descrições de como o sistema se deve comportar, dos serviços oferecidos pelo sistema. São também conhecimento do domínio de aplicação e restrições sobre o desenvolvimento do sistema (Sommerville e Sawyer, 1997). Os requisitos reflectem as necessidades dos *stakeholders*, pelo que, a ER é um processo de descoberta das suas necessidades, documentando-as através das actividades de análise, documentação e implementação.

ER é, portanto, parte do processo de engenharia de software, estando envolvida com a ligação das actividades iniciais de desenvolvimento de um sistema, de modo que a sua adequação e a relação custo/eficácia da solução possa então ser analisada. O reconhecimento de um problema a ser resolvido para uma especificação detalhada faz parte também do processo de ER.

As actividades mais comuns da ER são (Kotonya e Sommerville, 1998):

- **Identificação de requisitos:** Efectuar o levantamento de requisitos de um sistema futuro. O objectivo é identificar, sobre o domínio da aplicação, quais os serviços que o sistema deve proporcionar, o desempenho solicitado do sistema e as restrições;
- **Análise de requisitos:** É efectuada para resolver problemas de identificação e alcançar um entendimento nas mudanças para os requisitos, uma vez que estes são identificados por diferentes *stakeholders*;

- **Especificação/documentação dos requisitos:** Para documentar requisitos são usados diferentes diagramas para descrever os requisitos em vários níveis de detalhe;
- **Validação dos requisitos:** Verifica o conjunto de requisitos definidos e tenta descobrir os possíveis problemas que ocorram. Este processo envolve os *stakeholders* do sistema e os engenheiros de requisitos;
- **Gestão de requisitos:** Permite gerir as mudanças nos requisitos durante o processo de engenharia de requisitos, e o processo de desenvolvimento de sistemas. Esta gestão assegura que a qualidade dos requisitos seja mantida.

Destas actividades, serão consideradas na abordagem proposta a identificação e análise de requisitos.

A ER pode ser aplicada para um único produto ou para uma Linha de Produtos de Software (LPS) (Clements e Northrop, 2002). Uma LPS é um conjunto ou grupo de produtos que têm uma maioria de propriedades comuns que apenas variam em certas propriedades específicas. Desenvolver um grupo de produtos que têm propriedades em comum, contribui muito para reutilização em todas as fases de desenvolvimento de um sistema (Clements e Northrop, 2002).

Uma LPS tem duas actividades importantes: desenvolvimento dos artefactos base (*core assets*) e desenvolvimento do produto (Clements e Northrop, 2002). Na actividade de desenvolvimento dos artefactos base, que também é conhecida como engenharia de domínio, requisitos da família de produtos e potenciais membros desta família são identificados, analisados e um *framework* de domínio reutilizável é desenvolvido. No desenvolvimento do produto, também designado como engenharia de aplicação, o *framework* de domínio é instanciado para desenvolver aplicações individuais.

Nesta dissertação utiliza-se o modelo de *features* (Czarnecki et al., 2004) que permite representar a variabilidade de uma família de produtos, utilizando-se outros modelos para o complementar, sendo possível dar maior semântica ao modelo de *features* e obter mais informação.

1.2 Motivação

A investigação de requisitos para Linhas de Produtos de Software (LPS) tem explorado formas pelas quais artefactos são capazes de servir como base, para a derivação de produtos para utilizadores individuais.

A modelação de *features* é uma técnica chave para capturar pontos comuns e variáveis em famílias de sistemas e linhas de produtos. Uma *feature* pode denotar qualquer característica funcional ou não funcional ao nível dos requisitos. Contudo, os modelos de *features* mostram uma perspectiva muito específica das linhas de produtos.

Uma preocupação, no desenvolvimento de LPS, é a modularização dos seus artefactos, incluindo os artefactos de requisitos, com o intuito de melhorar a reutilização. Técnicas específicas para garantir modelos mais modularizados são necessárias.

As abordagens de Engenharia de Requisitos Orientadas a Aspectos (EROA), por sua vez, têm a vantagem de identificar e modularizar os assuntos transversais de forma sistemática, de modo a facilitar o entendimento do seu impacto noutros conceitos do domínio do problema e a sua evolução (Baniassad et al., 2006). Uma das abordagens de EROA é a abordagem Theme (Clarke e Baniassad, 2005), que apresenta as seguintes vantagens: possibilidade de identificar os aspectos na fase inicial do desenvolvimento de software, permitindo mapeá-los em artefactos das fases posteriores do ciclo de vida; fornece ligações de rastreabilidade dos requisitos para o desenho; verifica a cobertura dos requisitos no desenho; e possibilita o aumento da escalabilidade.

No entanto esta abordagem não se encontra ainda muito explorada para LPS, pelo que se pretende combinar estas duas abordagens, definindo-se como estas podem ser utilizadas em conjunto de forma sistemática e sinergética.

1.3 Objectivos e Contribuições

O objectivo desta dissertação é adaptar a abordagem Theme, uma abordagem de Engenharia de Requisitos Orientada a Aspectos, às Linhas de Produtos de Software. Como os modelos em Theme têm a potencialidade de identificar a variabilidade nas fases

iniciais dos requisitos, será possível contribuir com maior expressividade no desenvolvimento de LPSs.

Este trabalho é importante para a área de engenharia de software, na medida em que a adaptação da abordagem Theme a LPS irá resultar numa abordagem mais expressiva para o desenvolvimento de linhas de produtos.

Outro dos objectivos do presente trabalho, para apoiar na definição da abordagem desenvolvida oferecendo suporte à mesma, é a construção e implementação de uma Linguagem de Domínio Específico (LDE wiki, 2009), assim como a definição e implementação de um conjunto de regras de transformação (Czarnecki e Helsen, 2003), para que seja possível converter modelos Theme em modelos de *features*. Após a implementação das regras de transformação será feita a validação deste processo com recurso a um caso de estudo de complexidade significativa.

1.4 Organização do Documento

Este documento está organizado do seguinte modo:

- Capítulo 2. Fornece algumas informações sobre diferentes técnicas e tecnologias usadas nesta dissertação apresentando a Engenharia de Requisitos Orientada a Aspectos. Destas abordagens, é dada mais ênfase à abordagem Theme.

- Capítulo 3. Aborda a área de Engenharia de Linhas de Produto de Software e o modelo de *features*.

- Capítulo 4. Apresenta alguns trabalhos relacionados de Engenharia de Requisitos, aspectual e não aspectual, com LPS.

- Capítulo 5. São descritos os processos de utilização da abordagem Theme-SPL.

- Capítulo 6. É apresentado um caso de estudo, *Smart Home*, para ilustrar a aplicação da abordagem Theme-SPL a um caso real. É também feita a comparação da abordagem Theme-SPL com abordagens orientadas a aspectos integradas com LPS.

- Capítulo 7. São apresentadas as Linguagens de Domínio Específico e as ferramentas que permitem a sua construção, assim como o Desenvolvimento Orientado a Modelos, a Transformação de Modelos e respectivas ferramentas.
- Capítulo 8. São descritos os passos tomados para a criação da ferramenta.
- Capítulo 9. São descritos quais os caminhos tomados para a validação da ferramenta criada e os respectivos resultados da validação.
- Capítulo 10. São apresentadas as conclusões e limitações desta dissertação, assim como um possível trabalho futuro.
- Anexo A: É apresentado o questionário feito aos utilizadores para validação da ferramenta.
- Anexo B: É apresentado o manual de utilizador, da ferramenta, para que futuros utilizadores possam usar a mesma.

2. Engenharia de Requisitos Orientadas a Aspectos

Dijkstra introduziu o conceito "separação de assuntos" para referir a capacidade de identificar, separar e manipular partes do software, que são fundamentais para uma determinada meta ou objectivo (Dijkstra, 1976).

O Desenvolvimento de Software Orientado a Aspectos (DSOA) tem em conta a separação de assuntos, dando ênfase ao modo como os assuntos transversais são encapsulados em módulos separados (aspectos). As abordagens de engenharia de requisitos orientada a aspectos (que estão associadas às fases iniciais do DSOA) preocupam-se com a separação de requisitos transversais. Portanto, estas abordagens aparecem como alternativas promissoras para a obtenção de documentos de requisitos melhor estruturados, pois têm em conta a identificação, separação, representação e composição de assuntos transversais (Rashid et al., 2003).

Neste capítulo são introduzidos os conceitos base do Desenvolvimento de Software Orientado a Aspectos (DSOA) e Engenharia de Requisitos Orientada pelos Aspectos (EROA). Também irão ser descritas algumas abordagens de EROA, em particular a abordagem Theme.

2.1 Desenvolvimento de Software Orientado pelos Aspectos (DSOA)

O principal objectivo do DSOA, do inglês *Aspect-Oriented Software Development*, é o desenvolvimento de métodos, técnicas e mecanismos para a identificação sistemática, modularização, representação e composição de assuntos transversais (ou aspectos) ao longo do ciclo de vida do software (Rashid et al., 2003), sendo um assunto (i.e., *concern*) uma área de interesse ou foco no sistema.

O DSOA foca em novas abstracções e mecanismos para identificar, especificar e representar assuntos transversais, tais como a segurança, tempo de resposta, mobilidade e

disponibilidade do sistema, e a sua modularização em unidades funcionais separadas, bem como a sua composição automatizada (Chitchyan et al., 2005).

Os métodos tradicionais (e.g. os métodos orientados a objectos) não são capazes de lidar bem com certos assuntos que não se alinham com os critérios de decomposição utilizados e, portanto, eles acabam espalhados em várias partes do sistema de software caracterizando o espalhamento (em inglês *scattering*), onde a especificação de uma propriedade não é encapsulada em uma única unidade, e o emaranhamento (do inglês *tangling*), em que cada requisito contém a descrição de várias propriedades.

A Programação Orientada a Aspectos (POA) (Kiczales et al., 1997) fornece apoio para separação de assuntos, introduzindo os aspectos para modularizar os assuntos transversais. POA reforça a abstracção, a compreensibilidade e a adaptabilidade do código. Minimiza o impacto de mudança propondo o encapsulamento de assuntos em módulos separados.

O AspectJ (AspectJ Project, 2007) é a linguagem POA mais desenvolvida e a mais usada. AspectJ é uma extensão orientada a aspectos para a linguagem Java que apoia a implementação de código transversal, ou seja, aspectos como módulos de aspectos separados. Os aspectos podem conter várias entidades, tais como: *joinpoints*, *pointcuts* e *advices*. Um *joinpoint* é um ponto no fluxo de execução de um programa. Um *pointcut* é um elemento que identifica os *joinpoints*. Um *advice* descreve uma determinada função, método ou procedimento que deve ser aplicado num determinado *joinpoint* de um programa.

O DSOA tem alguns benefícios, dos quais se destacam (Rashid et al., 2003): a melhoria da capacidade de raciocinar sobre o domínio do problema e a sua solução correspondente; a redução na aplicação do tamanho do código, custos de desenvolvimento e tempo de manutenção; uma melhoria na reutilização do código; reutilização de requisitos, arquitectura e nível de desenho; e melhores métodos de modularização.

2.2 Engenharia de Requisitos Orientada pelos Aspectos (EROA)

As abordagens de Engenharia de Requisitos Orientada a Aspectos (EROA), do inglês *Aspect-Oriented Requirements Engineering,* demonstram o interesse e a preocupação em

tratar assuntos transversais (i.e., *crosscutting concerns*) na fase de requisitos (Baniassad et al., 2006) sendo que assuntos transversais são aqueles que estão espalhados (*scattered*) e emaranhados (*tangled*) com outros assuntos. EROA tem como principal objectivo a gestão de assuntos transversais nas fases iniciais do desenvolvimento do software (Araújo et al., 2005), e.g., elicitação e análise de requisitos. A EROA não só visa melhorar o suporte para a separação de requisitos transversais, funcionais e não funcionais, durante a análise de requisitos, mas também para fornecer melhores meios de identificação e gestão de conflitos que podem surgir durante a análise (Brito et al., 2007) (Rashid et al., 2003).

A EROA é baseada no princípio de separação de requisitos com maior importância dada à identificação, modularização e composição de requisitos transversais, desde o início do ciclo de vida de desenvolvimento de software, assegurando a homogeneidade do mesmo.

Os principais objectivos da EROA (Rashid et al., 2003) são a separação de assuntos transversais (aspectos), e encontrar uma maneira de identificar, modularizar e compor assuntos aspectuais e não aspectuais, sendo assim possível compreender o efeito de aspectos sobre outros requisitos, facilitando a modelação de aspectos nas fases de desenho e implementação.

As abordagens de EROA focam em como modularizar, especificar e compor os assuntos transversais através de mecanismos apropriados de abstracção, representação e composição (Chitchyan, 2005).

Seguidamente apresenta-se a abordagem Theme, que será o foco desta dissertação, e outras abordagens de EROA.

2.3 Theme

A abordagem Theme (Clarke e Baniassad, 2005) é definida para análise (Theme/Doc) e desenho (Theme/UML) baseada no conceito de temas.

2.3.1 Conceitos

Temas são um "encapsulamento de um assunto" (Clarke e Baniassad, 2005) e podem ser relacionados uns com os outros. Existem duas formas pelas quais os temas se podem

relacionar: por conceito de partilha e por transversalidade. O conceito de partilha, Figura 2.1, é uma categoria onde diferentes temas têm elementos de desenho que representam os mesmos conceitos no domínio. O conceito de partilha é uma categoria de transversalidade, no paradigma de separação simétrico, uma vez que qualquer assunto, transversal ou não, pode ser encapsulado em um único tema.

Figura 2.1 Temas relacionados pelo conceito de partilha (Clarke e Baniassad, 2005)

A segunda categoria de relacionamento é o assimétrico transversal, onde o comportamento em um tema é desencadeado pelo comportamento em outro tema, como mostra a Figura 2.2. Os temas podem ser de dois tipos: base e transversais. Um tema transversal é aquele que partilha estruturas comportamento com outros temas. Um tema base é aquele que desencadeia um comportamento aspectual. Enquanto temas independentes e temas que partilham conceitos podem operar sem conhecimento de outros, temas transversais requerem um conhecimento abstracto dos temas base e não podem operar independentemente.

Temas transversais são também chamados de aspectos. Os temas não transversais são a base sobre a qual os aspectos operam.

Figura 2.2 Temas relacionados por transversalidade (Clarke e Baniassad, 2005)

A abordagem Theme fornece apoio para o desenvolvimento orientado a aspectos sendo dividida em dois níveis, para diferentes fases do ciclo de vida do software:

- **Theme/Doc** (Análise de Requisitos): Consiste na identificação dos temas no documento de requisitos. Fornece heurísticas para identificar quais dos temas são transversais, ou aspectos, e quais são a base.

- **Theme/UML** (Desenho dos Temas): Permite separar os modelos de desenho para cada um dos temas identificados nos requisitos, seguindo os conceitos do desenvolvimento de software orientado a aspectos, tais como modularização, relacionamento e composição.

Esta abordagem envolve identificar e desenhar temas separados que são depois combinados para fazer um único sistema, ou seja, a abordagem Theme desenha cada característica do sistema separadamente e depois oferece uma forma de as combinar. Temas podem ser "representados" em Análise, Desenho e Composição.

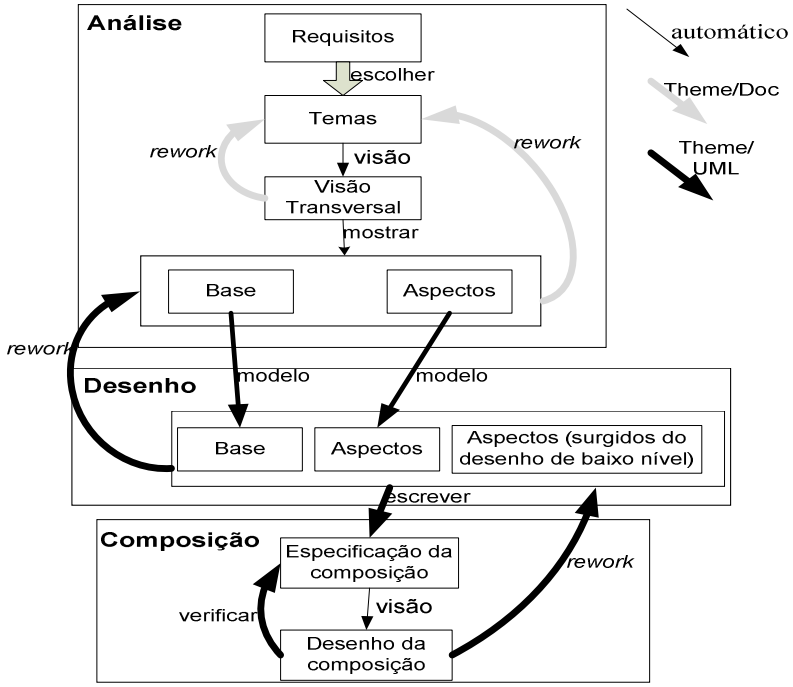

Figura 2.3 Processo da abordagem Theme (Clarke e Baniassad, 2005)

Na Figura 2.3 podemos observar que a representação Theme/Doc é usada para visualizar e analisar requisitos e a Theme/UML é usada na aplicação de projecto. De maneira geral, o processo da abordagem Theme é dividido em três actividades principais (Clarke e Baniassad, 2005):

Análise: O primeiro passo desta abordagem é realizar a análise dos requisitos para identificar temas. Isto envolve mapear requisitos para assuntos no sistema. O Theme/Doc mostra a relação entre comportamentos, em que estas relações expõem assuntos e levam a identificar aspectos.

Desenho: Usam-se os temas encontrados pelo Theme/Doc para identificar potenciais classes e métodos para, em seguida preencher os detalhes do desenho e fazer as alterações necessárias para o beneficiar.

Composição: Especifica como os modelos Theme/UML devem ser combinados. Em muitos casos algumas das visões, do inglês *views*, ajudam a determinar como os temas se relacionam uns com os outros, ou seja, se se sobrepõem ou se são transversais.

2.3.2 Theme/Doc aplicado a um exemplo

Os passos que compõe o Theme/Doc são apresentados com o exemplo abastecimento de veículos com Via Verde (Via Verde, 2005). Pretende-se desenvolver um sistema que utiliza identificadores "Via Verde" para efectuar o abastecimento de veículos em estações de serviço. Os identificadores obtêm-se através da sua adesão, em papel ou electrónico, em que o cliente tem que fornecer os seus dados pessoais, do seu cartão de débito e ainda do veículo a registar. É necessário fazer a activação do identificador numa caixa de multibanco associando este ao cartão de débito. Para abastecer basta que o identificador colado no pára-brisas do veículo seja válido. Um veículo aderente ao aproximar-se de uma das várias bombas de abastecimento automático activa o sistema que acende uma luz verde. Para abastecer o condutor deve inserir primeiro o código do cartão de débito associado ao identificador. O abastecimento termina quando o condutor coloca a mangueira de volta no suporte da bomba. O valor a pagar, que depende do tipo e quantidade de combustível seleccionado, é mostrado no visor da bomba. Uma vez terminada a operação o veículo pode seguir. O débito em conta é automático, portanto se a conta não tiver saldo suficiente o abastecimento não é autorizado.

2.3.2.1 Análise de Requisitos com Theme/Doc

O primeiro passo da abordagem Theme é examinar a documentação de requisitos do sistema (Tabela 2.1). Após este passo irá identificar-se os temas principais do sistema e os temas transversais (aspectos).

Tabela 2.1 Requisitos para Abastecimento de Veículos

R No.	Descrição dos Requisitos
R1	Cliente **adere** à via verde.
R2	A **adesão** à via verde inclui o cliente fornecer os seus dados pessoais, do cartão de débito e do veículo a registar.
R3	O cliente **activa** identificador no multibanco, se **adesão** for concluída com sucesso.
R4	Com a **adesão** concluída o cliente **recebe** um identificador.
R5	Para **abastecer** o veículo as seguintes acções são realizadas: Aproximar bomba; Validar Identificador; Validar PIN; Inserir PIN; Seleccionar quantia/combustível; Iniciar abastecimento/Retirar; Mostrar quantia; Concluir abastecimento/Colocar; Emitir extracto; Abandonar bomba.
R6	Veículo **aproxima-se** da bomba de gasolina.
R7	O identificador é **validado**.
R8	Se identificador válido **acende** luz verde.
R9	Se identificador inválido **acende** luz amarela.
R10	O Cliente deve **inserir** o PIN do cartão de débito associado ao identificador.
R11	O PIN deve ser **validado**.
R12	Se PIN errado, é **mostrado** no visor uma mensagem de erro indicando as tentativas restantes.
R13	Se PIN **válido** deve-se **verificar** dinheiro em conta.
R14	Se não houver dinheiro na conta é **mostrada** uma mensagem de erro.
R15	O Cliente deve **seleccionar** o montante e o tipo de combustível pretendido na bomba.
R16	Condutor **retira** a mangueira do suporte para encher o depósito.
R17	Quando a mangueira é **retirada** do suporte a bomba de gasolina coloca contadores de litros, preço e tipo de combustível a zero e **mostra** qual o tipo de combustível escolhido e qual o preço por litro.
R18	Condutor deve **colocar** a mangueira no suporte na bomba, após abastecer.
R19	*Display* **mostra** valor a pagar que depende do tipo de combustível e da quantidade do mesmo.
R20	Bomba de gasolina **emite** recibo ao cliente, se solicitado por este.
R21	Administrador do sistema tem que se **autenticar** para **registar** funcionário.
R22	Administrador do sistema **regista** funcionário no sistema.
R23	Funcionário tem que se **autenticar** para **registar** reclamação.
R24	O Cliente **efectua** uma reclamação ao funcionário.
R25	Funcionário **regista** reclamação no sistema.
R26	Veículo **abandona** a bomba de gasolina.
R27	Extracto mensal é **produzido** e enviado ao cliente.

2.3.2.2 Identificação de um Tema

A actividade de identificação de um tema envolve iteração sobre os temas até se obter um conjunto que faça sentido. Este processo envolve olhar para as responsabilidades de cada tema, para ver se representam um conjunto coerente de comportamentos.

Encontrar temas para o sistema tem quatro actividades principais, sendo elas: escolher o conjunto inicial de potenciais temas, refinar o conjunto de temas, identificar quais dos temas são aspectuais e preparar para o desenho.

Baniassad e Clarke (Baniassad e Clarke, 2005) indicam a escolha dos verbos de acção para o ponto inicial dos temas e a partir daí ir refinando para encontrar os temas exactos para o sistema.

Alteramos o Theme/Doc para representar temas optando-se por, para cada tema, escrever **Verbo+Objecto**, em vez de só verbo, como feito pelos autores da abordagem. Uma vez que poupa tempo para a análise, oferece uma maior semântica e também porque um verbo pode estar ligado a mais que um tema, podendo ser temas diferentes. Por exemplo, usando o verbo obter, não sabemos que tipo de informação se obtém. Quando são verbos intransitivos, como por exemplo parar, basta escrever apenas o verbo.

Os potenciais **Temas** são encontrados percorrendo os requisitos e seleccionando os termos chave. Foram identificados vinte e dois potenciais temas, descritos na Figura 2.4.

Abandonar bomba de gasolina	**Display mensagem de erro**	**Registar funcionário**
Abastecer veículo	**Emitir recibo**	**Registar reclamação**
Acender luz	**Fazer reclamação**	**Seleccionar quantia/combustível**
Activar identificador	**Iniciar abastecimento**	**Validar identificador**
Aderir via verde	**Inserir PIN**	**Validar PIN**
Aproximar bomba de gasolina	**Mostrar quantia/erro**	**Verificar saldo**
Autenticar	**Produzir extracto**	
Concluir abastecimento	**Receber identificador**	

Figura 2.4 Lista de Temas

Foram também identificadas, nos requisitos, onze **entidades-chave** que são entidades, pessoas ou outros sistemas que interagem com o sistema, ilustradas na Figura 2.5.

Banco	*Display*	Multibanco	Veículo
Bomba de gasolina	Funcionário	Semáforo	Via verde
Cliente	Identificador	Sistema	

Figura 2.5 Lista de Entidades

Na Figura 2.6 mostramos uma lista relacionando temas e requisitos.

Abandonar bomba de gasolina -> R5, R26	Iniciar abastecimento -> R5, R16, R17
Abastecer veículo -> R5	Inserir PIN -> R5, R10
Acender luz -> R8, R9	Mostrar quantia -> R5, R17, R19
Activar identificador -> R3	Produzir extracto -> R27
Aderir via verde -> R1,R2, R3, R4	Receber identificador -> R4
Aproximar bomba de gasolina -> R5, R6	Registar funcionário -> R21, R22
Autenticar -> R21, R23	Registar reclamação -> R23, R25
Concluir abastecimento -> R5, R18	Seleccionar quantia/combustível -> R5, R15
Diplay mensagem de erro -> R12, R14	Validar Identificador -> R5, R7, R8,R9
Emitir recibo -> R5, R20	Validar PIN -> R5, R11, R12, R13
Efectuar reclamação -> R24	Verificar saldo -> R13, R14

Figura 2.6 Lista de Relacionamento de Temas e Requisitos

Para visualizar a relação entre temas e requisitos, o Theme/Doc usa o diagrama Visão de Relação de Temas, do inglês *theme-relationship view* ou *relationship view*. As Visões de Relação são criadas dando o conjunto de requisitos e o conjunto de temas. Os requisitos são representados por caixas com os cantos arredondados e os temas por losangos. Se o nome de um tema é mencionado num requisito, há uma ligação a partir do requisito para aquele tema, na visão de relação.

Os temas identificados anteriormente resultam na Visão de Relação de Temas, mostrada na Figura 2.7.

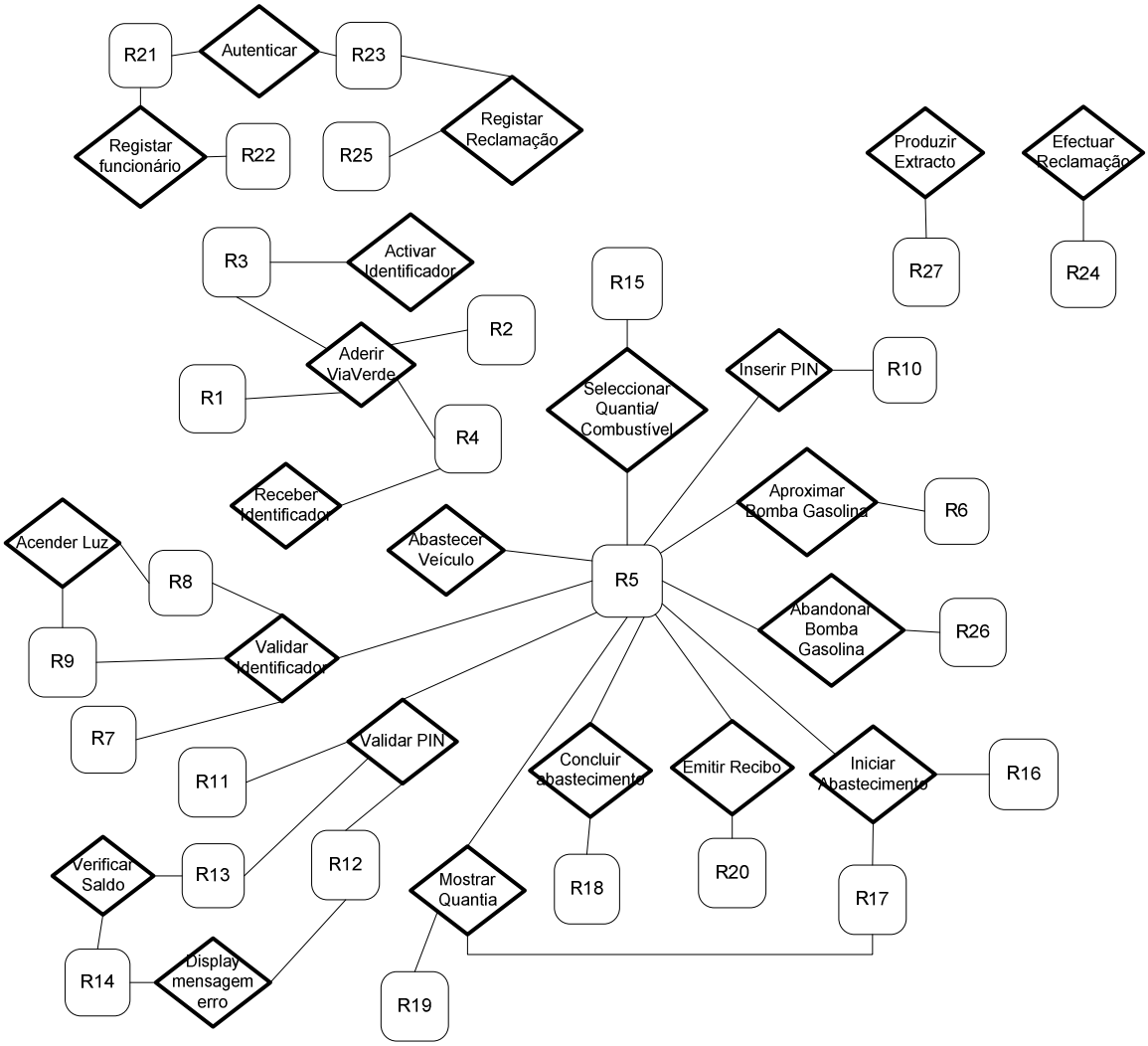

Figura 2.7 Visão de Relação de Temas Inicial

Pode haver temas diferentes que representam eventos ou comportamentos que acontecem sempre em conjunto. Nestes casos, estes mesmos temas, devem ser agrupados em um único tema que fica ligado aos requisitos de todos os temas que lhe deram origem.

No exemplo da via verde existem vários temas que podem ser agrupados, sendo eles: **Acender luz/Validar Identificador** (presentes nos Requisitos R5, R7, R8 e R9), pois a luz apenas acende na fase de validação, logo parece lógico agrupar estes dois temas; **Registar Reclamação/Efectuar reclamação** (presentes nos requisitos R23, R24 e R25) podem ser agrupados, uma vez que ambos os temas se referem a comportamentos intimamente relacionados, pelo que se irá efectuar um melhor desenho tendo os temas como um único tema; e os temas **Concluir Abastecimento** e **Emitir Recibo** (presentes nos requisitos R5, R18 e R20) também serão agrupados.

Com este agrupamento obtém-se a Visão de Relação, mostrada na Figura 2.8.

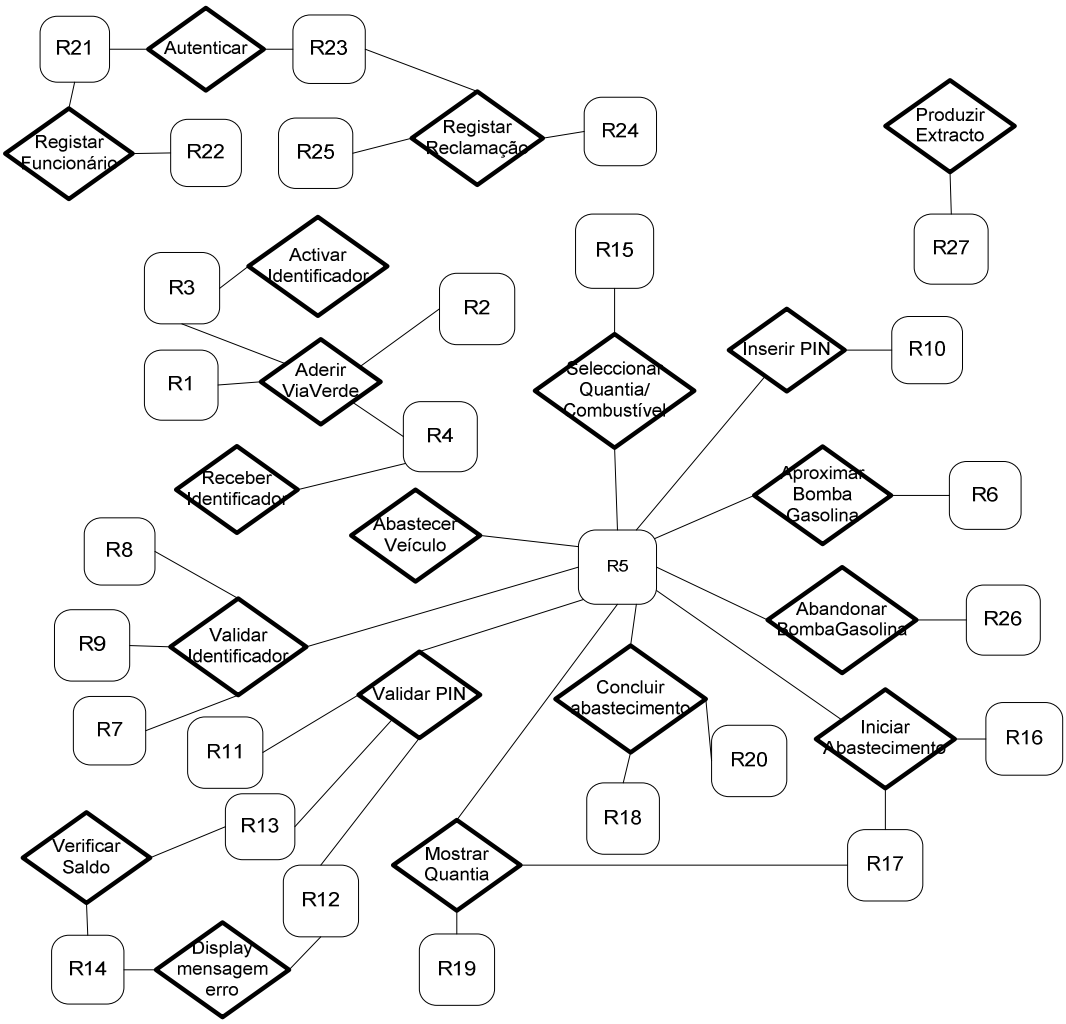

Figura 2.8 Visão de Relação com Temas Agrupados e Removidos

2.3.2.3 Identificação de Aspectos (Temas Transversais)

Segundo os autores (Clarke e Baniassad, 2005), aspectos são comportamentos que estão espalhados pelo sistema e no documento de requisitos, que se manifestam como descrições de comportamentos que estão interconectados e entrelaçados. Para identificar temas aspectuais utilizando a abordagem Theme deve-se olhar para os requisitos partilhados.

Algumas questões devem ser postas para a identificação de aspectos, sobre os requisitos partilhados (Clarke e Baniassad, 2005):

1. Pode o requisito ser dividido em temas isolados? Se puder, deve-se reescrever o requisito, para melhor dividir responsabilidades entre temas.

2. É um tema dominante no requisito? Se sim, o tema dominante, provavelmente, deve ser responsável por esse requisito e não pelo requisito a ser partilhado pelos outros temas.

3. É o comportamento do tema dominante desencadeado pelos outros temas mencionados nos requisitos? Se sim, identificou-se a relação desencadeada entre os temas.

4. É o tema dominante desencadeado em várias situações? Se, ao longo dos requisitos, o tema dominante é descrito como desencadeado em múltiplas situações, então é aspectual. O tema dominante torna-se o aspecto e os temas que o desencadeiam tornam-se a base.

Para decidir se uma relação de partilha representa uma relação aspecto-base, tem que se averiguar se as questões para identificar um aspecto, definidas acima, se verificam.

A Figura 2.8 mostra que se o requisito R21 é um requisito partilhado, então algum dos temas ligado a ele deve ser responsável por ele. Analisando a descrição do R21, verifica-se que a acção "Autenticar" é uma acção necessária quando se considera "Registar Funcionário", sendo assim o tema responsável pelo R21. O requisito R23 também é um requisito partilhado, e tal como no requisito anterior, verifica-se que a acção "Autenticar" é uma acção necessária quando se considera "Registar Reclamação", sendo assim o tema responsável pelo R23. Como o tema "Autenticar" é desencadeado nestas duas situações, então este tema é um tema aspectual.

Os requisitos R12 e R14 são requisitos partilhados e em ambos se verifica, que a acção "Display mensagem erro" é necessária para quando se considera "Validar PIN" e "Verificar Saldo" respectivamente, pelo que "Display mensagem erro" é um tema aspectual.

Todos os restantes temas são considerados temas base.

A Visão de Relação Transversal, do inglês *Crosscutting-relationship View*, é identificada através de uma seta cinzenta, que indica a relação de transversalidade a partir do tema aspectual para os temas base. Por exemplo, o tema autenticar apresenta uma relação de transversalidade com os outros dois temas, como referido anteriormente, que se representa associando os requisitos partilhados, R21 e R23, com ele e colocando uma seta cinzenta do tema autenticar (aspectual) para os dois temas base. A Figura 2.9 mostra a hierarquia transversal resultante para o sistema.

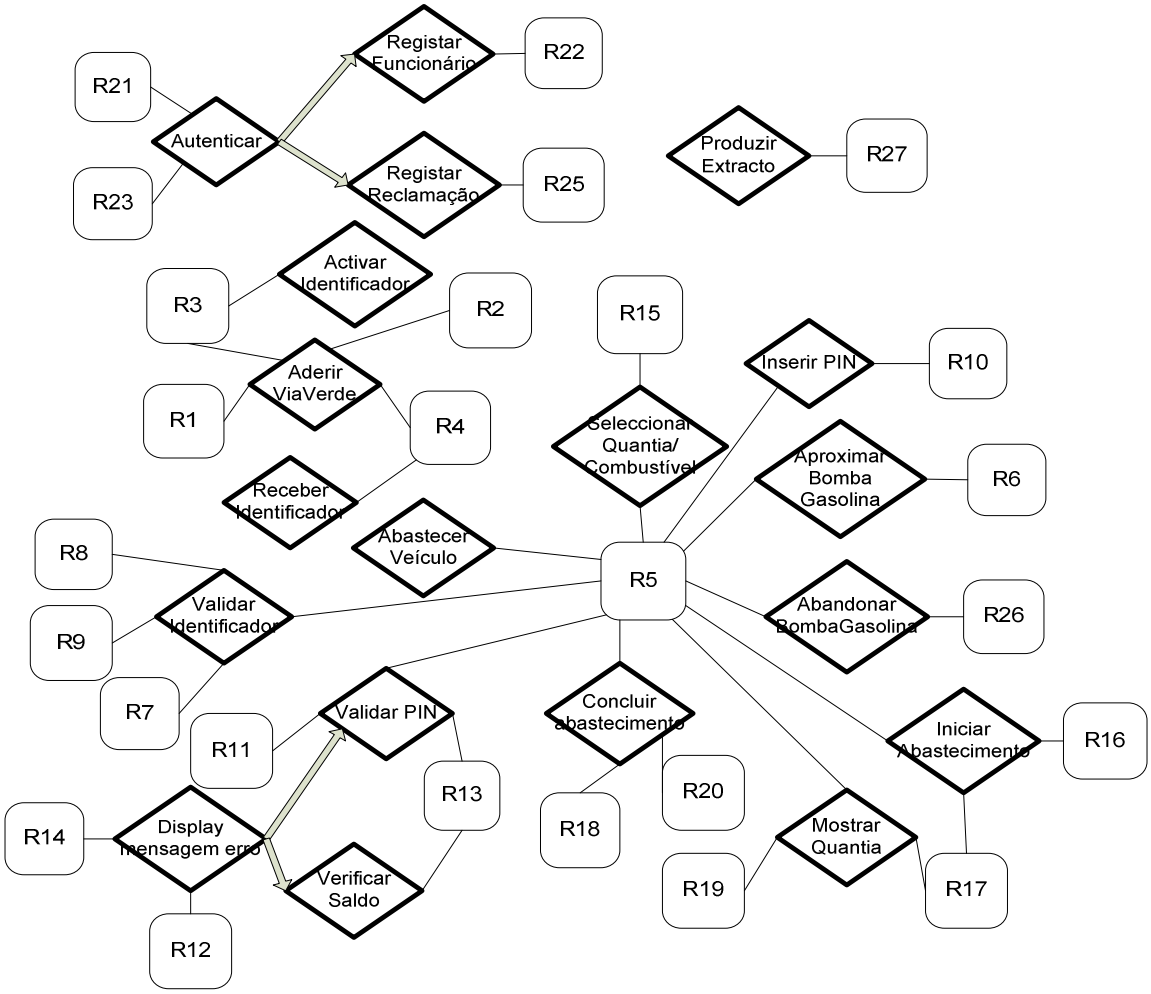

Figura 2.9 Visão Transversal

2.3.2.4 Preparação para o Desenho

A Visão Individual do Tema é usada para planear o desenho e a modelação dos temas identificados no passo anterior. Esta visão ajuda a determinar quais os objectos e comportamentos que devem ser modelados em Theme/UML, mostrando todos os

requisitos ligados ao tema de interesse, e é usada para planear as classes e métodos no Theme/UML. Ao modelarmos usamos as práticas de desenho orientado a objectos, para auxiliar na determinação de quais classes e métodos são descritos. Na maior parte dos casos, os métodos são as acções e as classes são as entidades. Porém, podemos acrescentar outras entidades e métodos como decisões de desenho que não foram contemplados na fase de análise. Na Figura 2.10, os requisitos R21 e R22 referem-se aos temas "Autenticar" e "Registar Funcionário" e às entidades "Sistema" e "Funcionário" e os requisitos R23 e R25 referem-se aos temas "Autenticar" e "Registar Reclamação" e às entidades "Sistema" e "Funcionário".

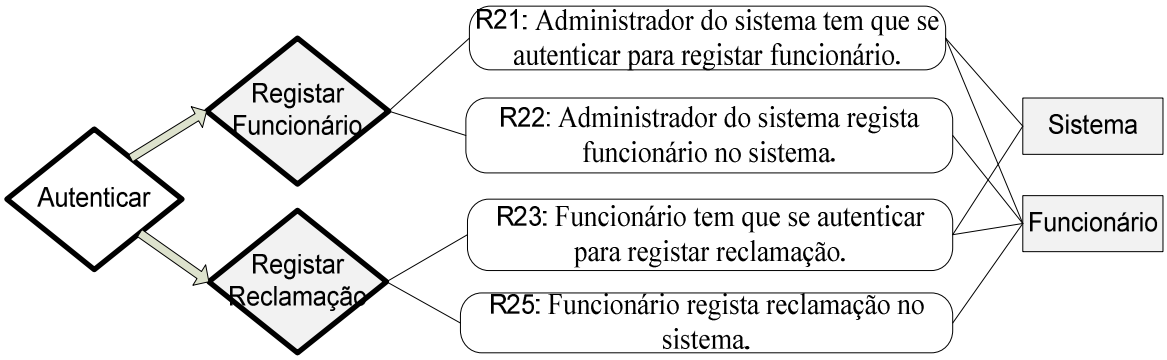

Figura 2.10 Visão Individual do Tema Aspectual Autenticar

2.3.3 Theme/UML

O principal objectivo do Theme/UML é fornecer meios para se considerar os assuntos de cada tema separadamente, o que leva a trabalhar com modelos de desenho separados para cada assunto.

2.3.3.1 Desenhar os Temas

O Theme/UML prescreve três altos níveis de actividade para adicionar ao processo de desenho (Clarke e Baniassad, 2005):

1. Desenhar separadamente os temas que foram identificados, usando o Theme/Doc;
2. Especificar a relação entre o desenho dos temas;
3. Compor os temas para efeitos de verificação.

O Theme/UML permite conceber modelos distintos, para cada um dos temas identificados nos requisitos. É fundamental em alguns passos importantes de desenvolvimento de software orientado a aspectos: modularizar, relacionar e compor.

A abordagem Theme aproxima-se da abordagem simétrica para a decomposição do sistema, uma vez que temas são assuntos individuais sem se preocupar se se tratam de aspectos ou assuntos separados. Contudo, os termos transversal ou aspecto são definidos como no paradigma assimétrico, ou seja, como comportamento que é desencadeado em várias situações.

Para o exemplo optou-se por fazer o desenho do tema Validar Identificador.

Validar Identificador

O tema "Validar Identificador" é um tema base, sendo que estes temas são compostos pelos requisitos associados, a lista de entidades associadas e todos os comportamentos associados com o tema. Os nós entidade que são específicos da visão individual do tema são mostrados nas caixas. Este tema é descrito nos requisitos R5, R7, R8 e R9 do conjunto, como ilustrado na Figura 2.11.

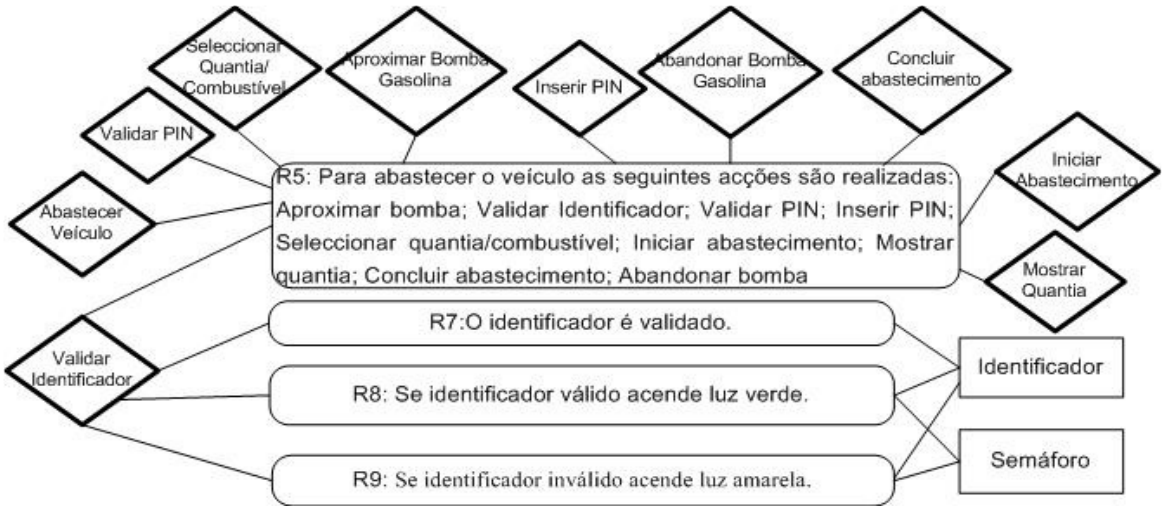

Figura 2.11 Visão Individual do Tema Validar Identificador

O diagrama de classes apresentado na Figura 2.12 mostra o impacto estrutural das decisões de desenho tomadas para cada requisito da visão individual do tema. Uma decisão que é necessária tomar em relação a uma entidade, da visão individual do tema, é se a mesma deve ser representada como uma classe ou como um atributo de uma classe.

Neste exemplo ambas as entidades são classes. Por norma, mas não é regra, as entidades só dão origem a classes no diagrama de classes.

Por exemplo, no requisito R7 é necessário verificar se o identificador é válido, pelo que é necessário o evento (i.e. operação) validarIdentificador (id). Esta decisão resulta numa classe de interface Bomba que necessita de uma relação com a classe Identificador.

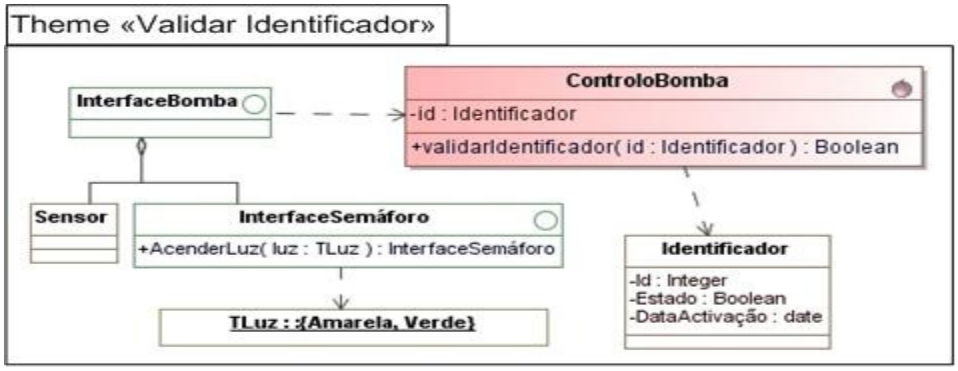

Figura 2.12 Diagrama de Classes - Validar Identificador

As mensagens e os objectos no diagrama de sequência, Figura 2.13, são obtidos pelos eventos (operações) e pelos objectos do diagrama de classes, respectivamente.

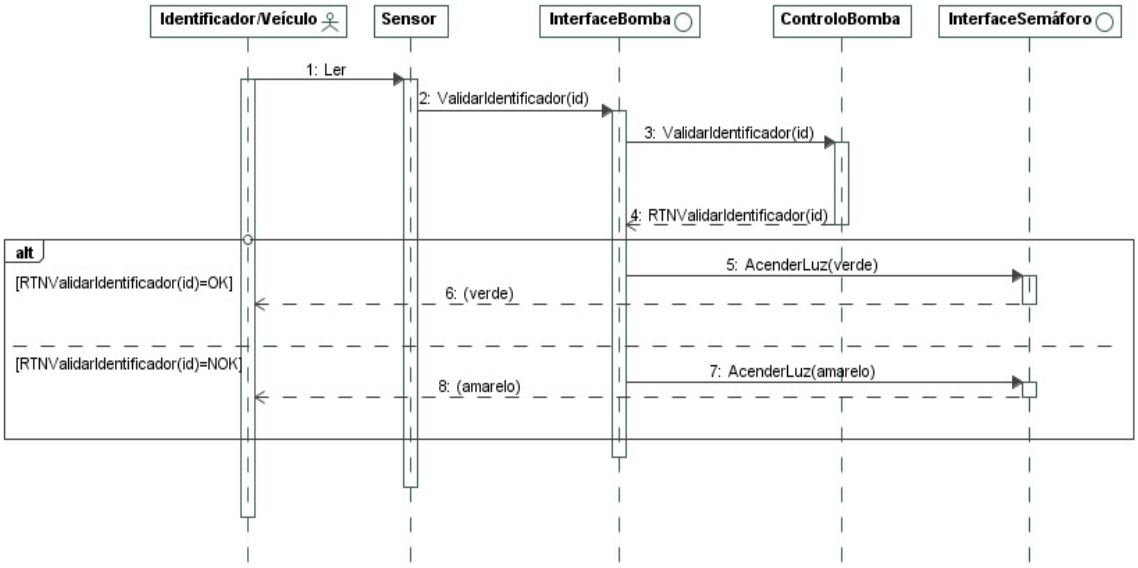

Figura 2.13 Diagrama de Sequência - Validar Identificador

2.3.3.2 Composição

Até aqui modularizámos o desenho separando assuntos, sempre que possível. O resultado é o conjunto de desenho de temas, cada um contendo todos e somente aqueles elementos

de desenho que se referem aos requisitos que o tema representa. Ou seja, não existe espalhamento ou emaranhamento no desenho do tema. Deve-se considerar agora a composição dos temas relacionando uns com os outros e definindo como eles interagem.

Conforme Clarke e Baniassad (Clarke e Baniassad, 2005), com a composição demonstra-se como novas funcionalidades podem ser adicionadas ao sistema sem fazer alterações ao desenho existente. Consegue-se fazer isto uma vez que cada novo assunto (como derivado a partir de novos requisitos) é considerado para ser um tema. Os novos temas podem ser desenhados separadamente e compostos com o desenho completo usando uma relação de composição.

Com a abordagem Theme simétrica para separação, existe um tema para cada assunto no domínio. Para especificar uma aplicação coerente, deve-se compor os temas que correspondem às necessidades da aplicação. O conjunto de temas para serem compostos para uma aplicação deve incluir todos os temas desenhados, ou diferentes aplicações podem requerer apenas diferentes subconjuntos desses temas. Em todos os casos, Theme/UML fornece um modo para especificar como os temas devem ser compostos. Em geral, os passos a seguir para compor temas são (Clarke e Baniassad, 2005):

1. Escolher os temas a serem compostos. Composição em Theme/UML é no nível de granularidade do tema.

2. Identificar os elementos de desenho dentro dos temas que correspondem. Até agora, descreveram-se os temas que podem ter especificações para o mesmo domínio, a partir daqui começa-se a trabalhar através da correspondência a esses conceitos.

3. Especificar como os temas devem ser integrados. O Theme/UML fornece dois tipos de integração – *merge* e *override*.

4. Especificar como os conflitos devem ser resolvidos entre elementos de desenho que correspondem. Quando temas são fundidos a resolução de conflitos pode ser necessária.

5. Identificar comportamentos transversais no tema base para ligar (*bind*) aos templates num tema aspectual.

A Figura 2.14 mostra os passos de composição de temas, referidos atrás.

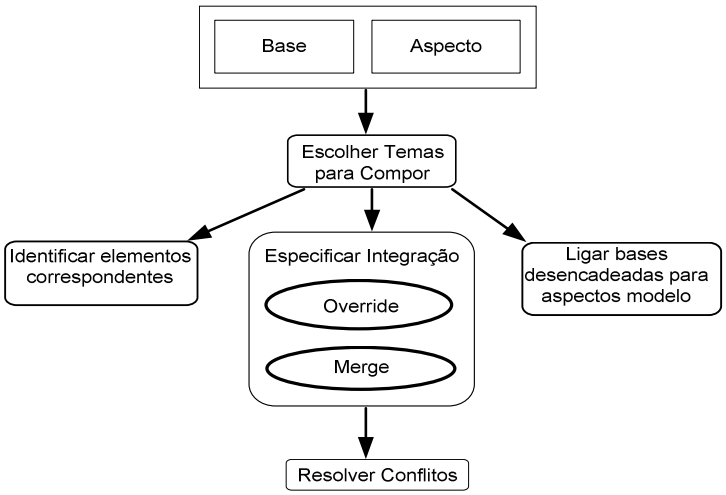

Figura 2.14 Visão geral do processo de composição (Clarke e Baniassad, 2005)

O Theme/UML define um novo tipo de relação, chamada relação de composição, que identifica sobreposições nos diferentes temas, especificando como resolver conflitos entre elementos sobrepostos e como os elementos sobrepostos, podem ser integrados num modelo composto. Em Theme temos a relação de composição transversal (orientado a aspectos) onde o comportamento dinâmico em um tema é desencadeado em conjunto com o comportamento de outros temas.

A composição de um tema deve ser feita em um diagrama de composição. As relações são representadas por linhas, ligando os temas. Todas as relações têm uma nota UML contendo as *tags* da composição (informação adicional sobre como os assuntos devem ser relacionados/compostos). Estas *tags* identificam os elementos correspondentes de desenho nos modelos relacionados, e especificam como os elementos correspondentes devem ser integrados. Para temas parametrizados a relação de composição contém *tags binding*, bind (params), para ligar os parâmetros aos valores concretos. De seguida ilustra-se a composição, utilizando o exemplo do posto de gasolina com via verde.

Para compor o comportamento "Autenticar" (aspecto) com os temas base, deve-se especificar onde, na base, o comportamento transversal deve ocorrer. Deste modo, no tema "Autenticar" um template deve ser colocado, especificando qual o parâmetro que deve ser substituído no desenho pelos elementos no tema aspectual. Sendo que este parâmetro é responsável por desencadear o comportamento transversal. O tema

Autenticar tem um parâmetro modelo chamado Autenticar.autenticarOp (..) que representa o método actual que desencadeia o comportamento de autenticar. Quando o tema **Autenticar** é composto com o seu tema base (Registar Reclamação, neste caso), irá ocorrer o desencadeamento a partir do tema base, para substituir as referências a autenticarOp (..). Para especificar essa substituição é usada uma *tag* "bind []", especificando os métodos a serem considerados na composição. Neste caso, queremos que ocorra autenticação antes de o funcionário registar uma reclamação. Estas operações são manuseadas pelo método Sistema.inserirReclamacao(), ou seja, a autenticação irá ocorrer antes de serem registadas reclamações.

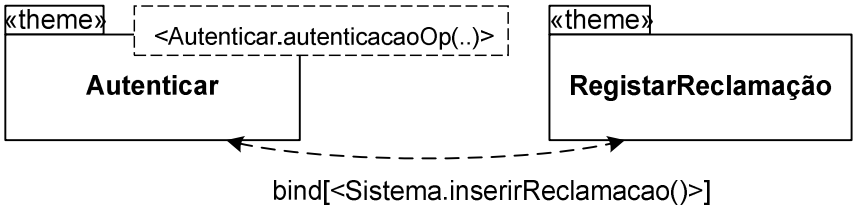

Figura 2.15 Relação de Composição: Autenticar/ Registar Reclamação

Com isto concluímos a descrição detalhada da abordagem Theme. A seguir será apresentada uma breve descrição de outras abordagens de EROA.

2.4 Outras Abordagens de Engenharia de Requisitos Orientada pelos Aspectos

2.4.1 *Aspect-Oriented Requirements Engineering* (AORE)

Em Rashid (Rashid et al, 2003) é apresentado um modelo baseado em *viewpoints* que começa pela identificação e especificação dos assuntos e requisitos dos *stakeholders*. Esta identificação é efectuada utilizando XML como linguagem para definir os requisitos, aspectos candidatos e regras de composição, sendo este último um benefício adicional de usar XML devido à facilidade com que se pode modificar e adicionar novas propriedades. Há um apoio explícito para estabelecer *trade-offs* aspectuais e negociações subsequentes. Estes *trade-offs* tentam encontrar a combinação que melhor satisfaz os objectivos dos *stakeholders*.

A separação antecipada de requisitos transversais ajuda nas fases posteriores, uma vez que simplifica a tarefa de determinação do seu mapeamento.

Depois da composição de aspectos candidatos e dos requisitos dos *stakeholders,* usando as regras de composição definidas, a identificação e resolução de conflitos deve ser tida em conta. Este passo é realizado construindo uma matriz de contribuições, negativas e positivas, para cada aspecto. Após esta atribuição de pesos aos aspectos em conflito deve-se, no final, resolver estes conflitos com os *stakeholders.*

A maior vantagem deste modelo é fornecer mecanismos para separar requisitos, assim como identificar e tratar conflitos.

2.4.2 Casos de Uso com Aspectos

Jacobson (Jacobson, 2003) apresenta uma abordagem de como integrar casos de uso com aspectos. Sugere que um caso de uso corresponde a um assunto, e é um assunto transversal desde que a implementação de cada caso de uso afecte diversos módulos. Este método abrange todo o ciclo de desenvolvimento de software, desde a identificação de requisitos até à implementação.

Com a ligação de aspectos a casos de uso (Jacobson, 2003) podem surgir novos tipos de módulos, módulos de casos de uso e módulos de componentes. Os primeiros irão ser transversais em relação aos módulos de componentes. Os módulos de componentes fornecem a estrutura estática do sistema, e os módulos de casos de uso fornecem o comportamento dinâmico adicionado a esta estrutura.

Os casos de uso podem particionar o sistema em "fatias" de casos de uso (*use case slices*), com elementos de cada modelo do ciclo de vida (e.g. casos de uso, classes). A separação de assuntos é efectuada por *slicing*, em que os *slices* são chamados de módulos de casos de uso, que apresentam uma relação de transversalidade com os módulos de componentes.

2.4.3 Modelação de Cenários com Aspectos

Esta abordagem define cenários como um traço de uma execução, através de um sistema existente ou de um sistema em desenvolvimento (Whittle e Araújo, 2004).

Este modelo centra-se em requisitos baseados em cenários, que são facilmente entendidos pelos *stakeholders*, logo são comummente utilizados em Engenharia de Requisitos. A origem deste modelo está relacionada com o facto de se poder desenvolver um conjunto de cenários, onde os cenários transversais (ou aspectuais), tais como, excepções ou casos de falha, são especificados, complementados com regras de composição com os cenários não aspectuais.

O modelo suporta cenários aspectuais e não aspectuais que são modelados independentemente, e irão fundir-se com os restantes através das regras de composição para validar o conjunto completo de cenários.

2.4.4 *Aspect-Oriented Requirements Analysis* (AORA)

A abordagem AORA, do inglês *Aspect-Oriented Requirements Analysis*, tem como objectivo identificar, especificar, modularizar e compor assuntos, especialmente assuntos transversais, durante a Engenharia de Requisitos (Brito e Moreira, 2004). O processo começa com a identificação de assuntos com base na extracção de requisitos e reutilização de catálogos. Após esta identificação os assuntos são especificados, identificando-se responsabilidades, contribuições, prioridades, assuntos necessários e construindo modelos de assuntos. O processo termina com a composição dos assuntos, que inclui a identificação de *match points*, assuntos transversais, resolução de conflitos e definição de regras de composição. Pode em qualquer fase do processo surgir novos assuntos.

Quando os conflitos são identificados, devido à composição de diferentes assuntos, a abordagem invoca um processo para a sua resolução, atribuindo diferentes prioridades aos assuntos que entram em conflito.

2.4.5 *Requirements Description Language* (RDL)

A RDL (Chitchyan et al., 2006) tira uma série de conceitos de sintaxe e semântica da linguagem natural, sendo a sua composição inteiramente baseada em semântica de interdependências de requisitos. A RDL é complementada por uma simples, mas intuitiva, representação visual para permitir a representação de composição, que é

particularmente útil para a validação da composição de requisitos com os *stakeholders,* que podem encontrar dificuldades de compreensão na composição especificada. Os conceitos principais da RDL são assuntos, requisitos, indivíduo, objecto, relação, composição e sequenciação. Os assuntos são identificados através de composição que detectam e resolvem conflitos e inconsistências entre eles.

Tal como outras abordagens orientadas a aspectos referenciadas anteriormente, a RDL utiliza a linguagem XML para escrever os requisitos. Embora o XML seja expressivo e fácil de alterar, assim como serve como um meio de comunicação eficaz entre os requisitos dos analistas, arquitectos e desenhadores, o XML não é entendido pelos utilizadores comuns, podendo tornar-se uma desvantagem na linguagem.

A RDL pode ser utilizada para revelar as influências mútuas de requisitos, utilizando os operadores de composição derivada, da própria semântica, das dependências dos requisitos.

2.4.6 VGraph

VGraph é uma abordagem orientada a objectivos onde são identificadas situações de transversalidade. O modelo é composto por *softgoals* (requisito não funcional), objectivos (requisito funcional) e tarefas (contribui para a satisfação de ambos os requisitos).

O V-Graph é um modelo de três níveis em forma de letra V e apresenta no topo dois vértices, *hard goals e softgoals*, que são respectivamente requisitos funcionais e não funcionais em termos de modelo de objectivos (Yu et al, 2004). Existe uma relação de contribuição entre Tarefas com Objectivos e *SoftGoals,* ou seja, uma tarefa contribui (de alguma forma) para que os requisitos, funcionais ou não funcionais, sejam satisfeitos. De forma similar existe uma correlação entre Objectivos e *SoftGoals.*

Este modelo permite a descrição de nós intencionais (objectivos e *softgoals*) e nós operacionais (tarefas). Cada elemento do V-Graph é composto por duas partes, um tipo e um tópico. O tipo descreve uma função genérica ou um requisito não funcional genérico. O tópico captura a informação contextual para o elemento. Existem dois tipos de ligações entre os requisitos e as tarefas, podendo ser ligações de contribuição (*and, or, make, help, unknown, hurt, break*) ou de correlação (*help, unknown, hurt, break*). As ligações *make* e

help indicam contribuição positiva; o *unknown* indica que há um relacionamento, mas é desconhecido se é positivo ou negativo; e o *hurt* e o *break* indicam contribuição negativa (Yu et al., 2004).

O processo termina quando todos os objectivos principais e todos os *softgoals* são satisfeitos. Nesta etapa estamos aptos a identificar aspectos candidatos, através das tarefas que possuem um alto número de módulos.

2.4.7 I* e Aspectos

Em (Alencar et al., 2006) é definida uma abordagem que incorpora aspectos no *framework* i*. Esta abordagem identifica e modulariza assuntos transversais, assim como define regras de composição entre aspectos.

Numa primeira etapa da abordagem é efectuada a identificação e representação de aspectos candidatos, seguidamente é efectuada a identificação da relação entre os aspectos candidatos, a composição, e por fim é feita a análise de *trade-offs*, o que pode iniciar uma nova iteração. Sendo a primeira e segunda fases efectuadas de forma paralela.

São identificados três *guidelines* para identificar assuntos transversais, utilizando os modelos SD e SR. O modelo SD descreve relações de dependência entre os actores. E por outro lado, o modelo SR explica como os actores atingem os seus objectivos.

A identificação de aspectos candidatos nos estados iniciais ajuda a reduzir a complexidade do modelo i*, promovendo a inteligibilidade e a modularização de assuntos que são *scattered* e *tangled,* na especificação do sistema.

2.5 Theme e Outras Abordagens

A abordagem Theme será alvo de comparação com as abordagens referidas na secção 2.4. Para tal, foi definido um conjunto de critérios baseado em (Blair et al., 2004), sendo eles:

Apoio para conceitos aspectuais: A abordagem apoia directamente os conceitos AOSD?

Actividade do ciclo de vida: Quais as actividades do processo de desenvolvimento que são suportadas?

Regras de composição: Fornece regras explícitas para compor aspectos com requisitos?

Resolução de conflitos: A abordagem fornece algum método para identificar e resolver conflitos?

Mapeamento para desenho: A abordagem prevê o modo como os requisitos aspectuais são mapeados para os artefactos de desenho e implementação?

Ferramenta de apoio: Existe uma ferramenta de apoio para a abordagem?

Ainda incluímos o critério **publicação em livros** uma vez que é importante que esteja documentada em detalhe a descrição do método num documento que possa ser comercializado largamente, com vários casos de estudo.

A Tabela 2.2 regista esta comparação, através dos critérios definidos.

Tabela 2.2 Comparação de abordagens

	Critério						
Abordagem	Apoio de conceitos aspectuais	Actividades do ciclo de vida	Regras de composição	Resolução de conflitos	Publicação em livros	Mapeamento para desenho	Ferramenta de apoio
AORE	Sim	Elicitação e análise	Sim	Sim	Não	Limitado	Sim (protótipo descontinuado)
Casos de Uso com Aspectos	Sim	Análise	Sim	Não	Não	Limitado	Não
Cenários com Aspectos	Sim	Análise e Especificação	Não	Não	Não	Sim	Não
AORA	Sim	Elicitação e análise	Sim	Sim	Não	Sim	Sim (protótipo)
RDL	Sim	Elicitação e Análise	Sim	Sim	Não	Não	Sim
VGraph	Sim	Elicitação e análise	Não	Não	Não	Não	Não
I* e Aspectos	Sim	Elicitação e análise	Sim	Não	Não	Não	Não
Theme	Sim	Elicitação, Análise, Desenho	Sim	Sim	Sim	Sim	Sim (comercial)

De entre as abordagens especificadas, a abordagem Theme foi escolhida para a realização deste trabalho, uma vez que a abordagem, através do Theme/Doc apoia a elicitação e

análise de requisitos, identificando os comportamentos base e transversais através de um conjunto de heurísticas de simples aplicação, além de oferecer um mecanismo sistemático para mapear os modelos de análise para desenho. Além disso é a única abordagem que tem um livro publicado a descrevê-la onde são fornecidos vários casos de estudo de domínios de aplicação diferentes, o que ajuda o seu entendimento. Por fim é uma das abordagens de EROA mais referenciada na literatura.

Tendo os requisitos partilhados, através dos temas, ajuda a identificar com maior facilidade os comportamentos que podem ser transversais. Através de regras de composição permite-se identificar e especificar como os conflitos devem ser resolvidos, embora isto não esteja descrito explicitamente no método. Permite-se também o mapeamento de requisitos do desenho Theme/UML para a implementação de modo eficiente utilizando diferentes abordagens de programação orientadas a aspectos.

2.6 Sumário

Neste capítulo ofereceu-se uma visão geral sobre as várias abordagens orientadas a aspectos. Começou-se por introduzir o Desenvolvimento de Software Orientado pelos Aspectos. Foi apresentada a abordagem Theme (Clarke e Baniassad, 2005). A abordagem é simétrica uma vez que qualquer assunto, transversal ou não, pode ser encapsulado em um tema. Os temas podem ser de dois tipos, base e aspectuais, sendo os segundos desencadeados a partir do comportamento dos primeiros. O processo da abordagem é composto por três passos essenciais, Análise, Desenho e Composição. Para mais bem demonstrar o processo, desde o levantamento de requisitos até à composição, utilizou-se o exemplo da Via Verde. Foram apresentadas algumas das abordagens EROA, para se poder mostrar outros tipos de modelação de abordagens orientadas a aspectos.

Uma vez que o objectivo da presente dissertação é adaptar a abordagem Theme às Linhas de Produto de Software, no Capítulo 3 será abordado o tema de LPS onde são apresentados os conceitos básicos, as actividades envolvidas na engenharia de Linhas de Produto de Software e o Modelo de *Features*.

3. Linhas de Produtos de Software (LPS)

3.1 Definição

Uma LPS é um conjunto de sistemas de software intensivos que partilham e gerem um conjunto de propriedades comuns, satisfazendo as necessidades específicas de um segmento de mercado particular e que são desenvolvidos para um conjunto comum de artefactos base em um modo prescrito (Clements e Northrop, 2002).

A prática de Linhas de Produtos de Software consiste no uso sistemático de artefactos base para montar, instanciar ou gerar os vários produtos que constituem uma LPS, envolvendo estratégias de grande reutilização.

As LPSs emergem rapidamente, como uma solução viável e um importante paradigma do desenvolvimento do software, que permite às empresas realizar melhorias na ordem de grandeza no *time to market*, custos, produtividade, qualidade, etc. Uma LPS também pode permitir uma rápida entrada no mercado e uma resposta flexível fornecendo uma capacidade de personalização em massa.

As LPSs podem trazer muitas vantagens para os envolvidos no processo, assim como para o cliente. Os componentes comuns são reutilizados muitas vezes e, além disso, reparos e correcções de defeitos em um produto podem rapidamente ser propagados para outros membros da linha de produtos pelo uso de componentes partilhados. As vantagens das LPS são (Clements e Northrop, 2002): atingir ganhos de produtividade em grande escala; melhorar o tempo de mercado; manutenção da presença do mercado; manter o crescimento; melhorar a qualidade dos produtos; aumento da satisfação dos clientes; alcançar objectivos reutilizáveis; possibilitar personalização em massa e compensação por dificuldades em contratar engenheiros de software.

Por outro lado, se a escolha da abordagem não for a correcta para o desenvolvimento de linhas de produtos, pode levar a problemas, tais como (Clements e Northrop, 2002):

inconsistência entre os componentes desenvolvidos e necessidades dos produtos; componentes mal projectados levando à perda de desempenho; dificuldades para a construção das interfaces dos componentes; dificuldades organizacionais com atribuição de papéis e responsabilidades; administração incorrecta do conhecimento; evolução dos componentes sem modificar as interfaces, e artefactos mal desenvolvidos não podem apoiar mais que um produto, nem acomodar a variação entre os mesmos.

Os artefactos base são artefactos reutilizáveis e recursos, que formam a base para a LPS. Estes incluem, mas não estão limitados a, arquitectura, componentes de software reutilizáveis, modelos de domínio, declarações de requisitos, documentação, especificações, modelos de desempenho, cronogramas, orçamentos, planos de teste, casos de teste, planos de trabalho e descrições do processo.

3.2 Processos da Engenharia de Linha de Produtos de Software

A Engenharia de Linhas de Produtos de Software (ELPS) encontra-se dividida em dois processos importantes (Pohl et al., 2005):

- A Engenharia de Domínio é o processo responsável por estabelecer uma plataforma reutilizável e por definir pontos comuns e variáveis das linhas de produtos. A plataforma consiste em todos os tipos de artefactos de software (requisitos, desenho, realização e testes) sendo que a ligação entre estes artefactos facilita a reutilização sistemática e consistente.

- A Engenharia de Aplicação é o processo responsável por derivar aplicações de linhas de produtos, a partir da plataforma estabelecida na engenharia de domínio, explorar a variabilidade de linhas de produtos e assegurar a ligação correcta das variabilidades de acordo com as necessidades específicas da aplicação.

Na Figura 3.1 é ilustrado o relacionamento entre as três actividades da abordagem para linhas de produtos. A direcção de rotação das setas indica que o desenvolvimento do núcleo é usado para desenvolver produtos, mas também pode ser usado para evoluir fora do desenvolvimento do produto. A figura mostra que todas as actividades são essenciais e estão ligadas, além de procurar demonstrar que todas as três actividades são inter-relacionadas e altamente iterativas, não existindo uma “Primeira” actividade. Em alguns

contextos os produtos existentes são extraídos para artefactos base, em outros os artefactos base podem ser desenvolvidos ou adquiridos para uso futuro.

Figura 3.1 Desenvolvimento de linha de produtos (SEI/CMU, 2006)

Na Figura 3.2 encontra-se representado o *framework* da ELPS segundo Pohl (Pohl et al., 2005). A Engenharia de Domínio e a Engenharia de Aplicação são complementares interagindo em processos paralelos que compõem um modelo base, ou seja, um sistema de produção de software orientado à reutilização. Um processo de engenharia de aplicação desenvolve produtos de software, a partir de artefactos de software criados por um processo de engenharia de domínio.

A Engenharia de Domínio simboliza o princípio do desenho para reutilização, enquanto a Engenharia de Aplicação simboliza o princípio do desenho com reutilização.

O Processo de Engenharia de Linhas de Produtos de Software (PELPS) é um modelo de processo de software que evolui, por meio de várias iterações. Assim os sistemas desenvolvidos são capazes de se adaptar a mudanças nos requisitos durante cada iteração. As secções 3.2.1 e 3.2.2 apresentam os dois processos da ELPS, assim como os respectivos sub-processos.

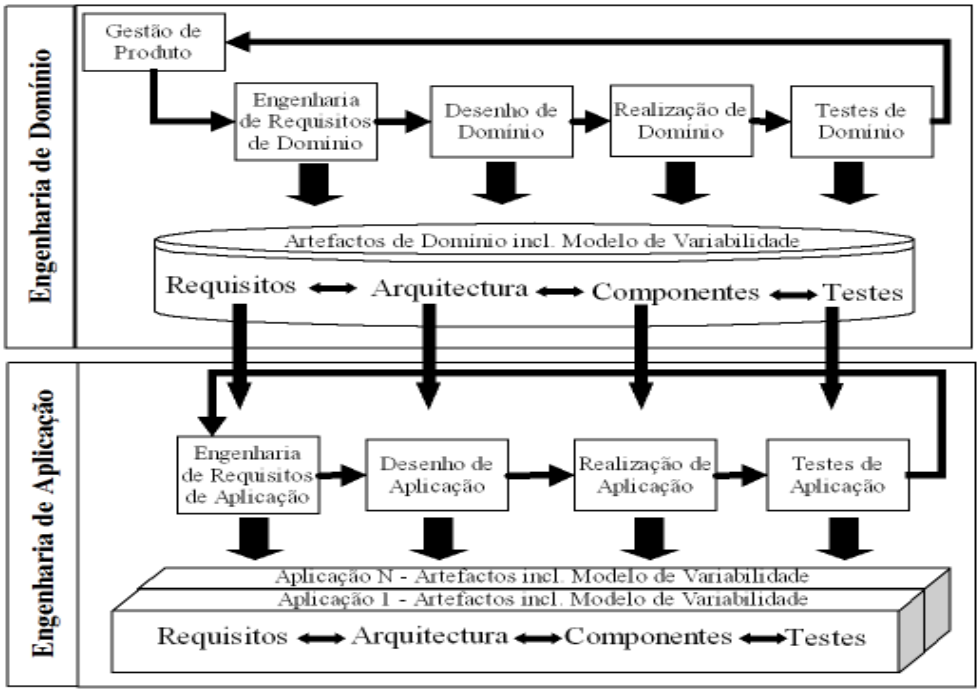

Figura 3.2 *Framework* de engenharia de linha de produtos de software (Pohl et al., 2005)

3.2.1 Engenharia de Domínio

Durante a fase de Engenharia de Domínio as semelhanças e variabilidades nas linhas de produtos são analisadas do ponto de vista geral dos requisitos. Este processo é composto por cinco sub-processos, sendo eles (Pohl et al., 2005):

- Gestão do produto: Este sub-processo permite lidar com os aspectos económicos de uma linha de produtos de software e em particular com a estratégia de mercado. A principal função é a gestão da lista de documentos de um produto da companhia ou unidade de negócio. A ELPS permite definir o que está dentro da área da linha de produtos e o que está fora.

- Engenharia de Requisitos de Domínio: Engloba todas as actividades para obter e documentar os requisitos comuns e variáveis das linhas de produtos.

- Desenho de Domínio: Engloba todas as actividades para definir a arquitectura de referência das linhas de produtos.

- Realização de Domínio: Lida com o detalhe do desenho e a implementação de produtos de software reutilizáveis.

- Testes de Domínio: São responsáveis pela verificação e validação de componentes reutilizáveis. Testam os componentes contra a sua especificação como é o caso de requisitos, arquitectura e desenho de artefactos. Além disso, desenvolvem testes de artefactos reutilizáveis para reduzir o esforço de teste da aplicação.

3.2.2 Engenharia de Aplicação

Durante a fase de Engenharia de Aplicação um membro da linha de produtos é desenvolvido, fazendo uso da reutilização dos artefactos de domínio e explorando a variabilidade das linhas de produto. A Engenharia de Aplicação é composta pelos sub-processos (Pohl et al., 2005):

- Engenharia de Requisitos de Aplicação: Identifica os requisitos dos *stakeholders* para a aplicação, em que é necessário fazer um mapeamento entre os requisitos da aplicação e os requisitos do processo de engenharia de domínio.

- Desenho de Aplicação: Engloba as actividades para produzir a aplicação de arquitectura. Usa a arquitectura de referência para instanciar a arquitectura de aplicação. Selecciona e configura as partes necessárias da arquitectura de referência e incorpora adaptações específicas da aplicação.

- Realização de Aplicação: Os principais objectivos são a selecção e configuração de componentes de software reutilizáveis, assim como a realização de artefactos de aplicação específica.

- Testes de Aplicação: Compreende as actividades necessárias para validar e verificar uma aplicação, contra a sua especificação.

3.3 Variabilidade

Segundo Pohl (Pohl et al., 2005) em engenharia de linhas de produtos de software, a variabilidade é uma propriedade essencial nos artefactos de domínio, o que faz com que esta seja utilizada para capturar a variabilidade dos requisitos do domínio, a arquitectura, os componentes e os testes. A variabilidade é introduzida durante a gestão do produto em sub-processos da engenharia de domínio, quando as características comuns e variáveis das aplicações da LPS são identificadas.

Definir e suportar variabilidade durante os diferentes estados do ciclo de vida da engenharia de linhas de produtos é apoiado pelo conceito de gestão de variabilidade, cujo conceito engloba as seguintes questões: suporta actividades envolvidas na definição de variabilidade; gere artefactos variáveis; apoia actividades envolvidas com resolução de variabilidade; recolhe, armazena e gere informação necessária para preencher estas tarefas.

Existem várias abordagens para explicitamente documentar a variabilidade de uma LPS, como é o caso da abordagem pioneira e mais referenciada, *Feature-Oriented Domain Analysis* (FODA) (Kang et al., 1990). O Modelo de *Features* (*Features Model*) dessa abordagem foi refinado por (Czarnecki et al., 2004), que inclui elementos adicionais. Para este trabalho será utilizado o modelo de *features* de Czarnecki, uma vez que é uma abordagem chave para capturar variabilidades de um sistema, de forma mais completa.

3.4 *Feature Oriented Domain Analysis* (FODA)

A abordagem FODA (Kang et al., 1990) é um método para descobrir e representar semelhanças entre sistemas de software relacionados. A abordagem permite a reutilização, pois trata de maneira sistemática as semelhanças entre os sistemas de um domínio. Estas semelhanças dão origem a *features,* que são utilizadas em vários produtos do domínio. O modelo FODA engloba os seguintes elementos: Diagrama de *Features,* que mostra uma decomposição hierárquica de *features*, podendo estas ser obrigatórias (devem ter), alternativas (selecção de muitos) e opcionais (pode ou não ter) relações; Definições da *Feature* que descreve todas as *features*; Regras de composição que indicam quais as combinações que são válidas e quais as que não são; Análise da *feature* para a escolha, ou não, de uma determinada *feature* indicando o *trade-offs*.

O modelo fornece uma visão detalhada dos problemas resolvidos por software, em um dado domínio.

3.5 Modelo de *Features*

Uma *Feature* é uma característica de um produto que utilizadores e clientes consideram importante, na descrição e distinção de membros de uma família de produtos. Pode ser um requisito específico ou uma selecção entre os requisitos específicos e alternativos.

Uma *feature* é, também, uma propriedade do sistema usada para distinguir os requisitos comuns dos requisitos variáveis de uma LPS. Pode estar relacionada com certas características do produto como funcionalidade, usabilidade e desempenho, assim como também pode estar relacionada às características de implementação como o tamanho, a plataforma de execução ou compatibilidade com certos padrões.

O modelo de *features* foi introduzido como parte do método FODA, e representa uma hierarquia de propriedades de conceitos do domínio. É uma das técnicas mais bem sucedidas para facilitar a reutilização de artefactos de software. O modelo de *features* é uma representação hierárquica, que visa captar os relacionamentos estruturais entre as *features* de um domínio de aplicação. O modelo também representa as *features* comuns e variáveis de instâncias de conceitos (por exemplo, sistemas de software) e dependências entre as *features* variáveis. Um modelo de *features* consiste em um diagrama composto de *features* e alguma informação adicional, tais como descrições semânticas de cada *feature*, pontos variáveis, motivos (i.e. *rationale*) para cada *feature*, prioridades e regras de dependência.

No contexto das linhas de produto de software, um modelo de *features* representa a própria linha de produtos.

Uma *feature* pode ser de um dos seguintes tipos:

- Obrigatória: A *feature* tem de estar presente em todos os membros da linha de produtos.
- Opcional: A *feature* pode ou não estar presente em um membro da linha de produtos.
- Alternativa: É uma *feature* que é composta de um conjunto de *features* das quais se escolhe uma ou mais, devendo-se indicar se é necessário escolher apenas uma ou se se pode escolher mais que uma. Nestas *features* é necessário fazer a distinção de alternativas *OR,* que permite mais do que uma *feature*, e *XOR*, que mostra a exclusão mútua.

Uma *feature* obrigatória é representada por uma aresta terminada por um círculo preenchido a preto. Uma *feature* opcional é representada por uma aresta terminada por

um círculo vazio. As *features* alternativas são representadas por arestas que estão ligadas e conectadas por um arco. Se o arco for vazio deve-se escolher apenas uma das alternativas (*XOR*), se for preenchido é permitido escolher mais de uma alternativa (*OR*) (notação de Van Deursen e Klint, 2002).

O modelo de *features* também introduz duas regras de decomposição: "*requires*", que permite adicionar uma determinada *feature* a uma instância solicitada, e "*mutex-with*", que permite fazer o inverso.

Czarnecki (Czarnecki et al., 2004) propõe colocar cardinalidade nas *features* para poder remover ambiguidades e representar a informação mais facilmente, sendo as diferentes cardinalidades assim definidas:

- 0..1 – Pode-se escolher uma ou nenhuma *feature* do conjunto de *sub-features*.
- 1 – Exactamente uma *feature* tem de ser escolhida de um conjunto de *sub-features*.
- 0..* - Um número arbitrário de *features* (ou nenhuma) tem de ser seleccionado do conjunto de *sub-features*.
- 1..* - Pelo menos uma *feature* tem de ser seleccionada do conjunto de *sub-features*.

Para o caso de estudo apresentado, através do modelo de *features,* podemos representar as principais funcionalidades do mesmo. A Figura 3.3 mostra uma possível representação para o modelo de *features* do Sistema de abastecimento de veículos, através de Via Verde, utilizando as notações referidas anteriormente.

Neste modelo são *features* obrigatórias o Pagamento e a Bomba, as *features* Reclamação, Identificador, Extracto e Adesão são *features* opcionais. A *feature* Adesão possui três *sub-features* obrigatórias, sendo elas: Formulário, Activação Multibanco e Recepção Identificador. A *feature* Extracto contém um grupo de *sub-features* numa alternativa *XOR*, que indica que apenas uma delas é escolhida para a aplicação. No caso da *feature* Pagamento, esta é composta por quatro *sub-features* numa alternativa OR, mas com cardinalidade, que indica que entre uma a quatro *sub-features* terão de ser usadas na

aplicação. Para o caso do pagamento com Via Verde, este encontra-se ligado com "*requires*" à *feature* Adesão, uma vez que o pagamento com via verde requer a adesão. A linha a tracejado da *feature* Activação Multibanco até à *feature* Recepção Identificador indica que a recepção do identificador está dependente da activação do mesmo no multibanco por parte do aderente, cuja ligação se denomina de *"requires*". O mesmo sucede com as *features* Identificador e Recepção Identificador, uma vez que a *feature* Identificador está dependente da *feature* Recepção do Identificador.

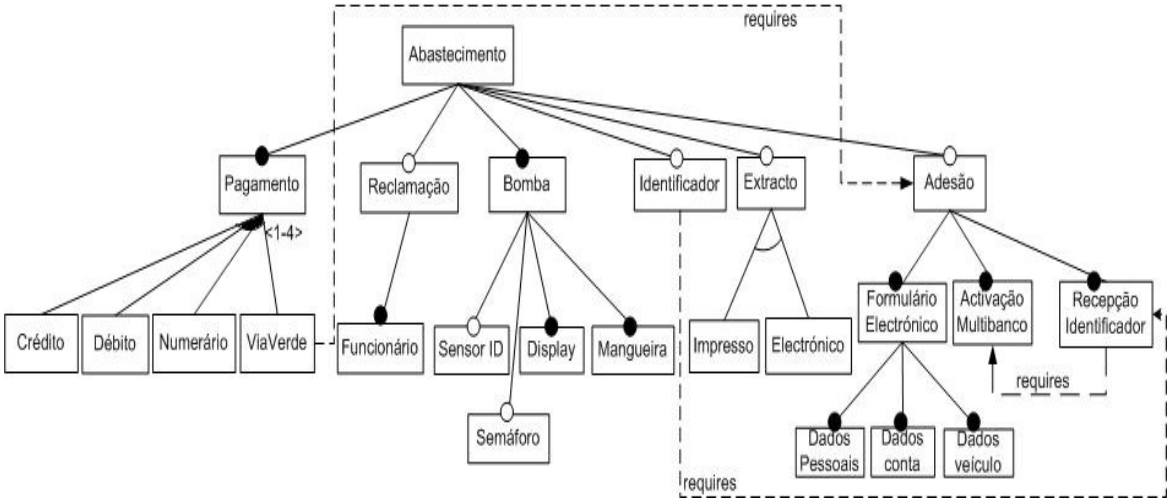

Figura 3.3 Modelo de *Features* do Sistema de Abastecimento

3.6 Sumário

Neste capítulo foram apresentados os principais conceitos de ELPS. Primeiramente foi apresentada uma breve introdução às LPS, assim como algumas das suas vantagens e desvantagens. A ELPS é constituída por três actividades essenciais, sendo elas, o desenvolvimento de artefactos base, o desenvolvimento do produto e a actividade de gestão. A actividade de desenvolvimento de artefactos base é conhecida também por Engenharia de Domínio e a actividade de desenvolvimento do produto é conhecida por Engenharia de Aplicação. Cada um destes processos é composto por sub-processos. Os artefactos base formam a base das LPS e são os blocos de construção para que novos produtos sejam concebidos. O desenvolvimento de um produto específico é o principal propósito de uma LPS. Por fim, as actividades de gestão são cruciais para adoptar uma abordagem da LPS, uma vez que todos os recursos necessários são determinados nestas actividades.

Sendo a variabilidade uma propriedade central da ELPS, foi também apresentado o seu conceito nesta secção. Foram também apresentados os conceitos do modelo de *features* para documentar a variabilidade de uma LPS, sendo ilustrados através do modelo para o exemplo de abastecimento.

No Capítulo 4 serão apresentadas algumas das abordagens de engenharia de requisitos, aspectuais e não aspectuais, integradas com LPS.

4. Abordagens de Engenharia de Requisitos com LPS

As Linhas de Produtos de Software e os Modelos de *Features* são cada vez mais utilizados, levando a que várias abordagens se tentem adaptar a estes modelos. Neste capítulo irão ser apresentadas abordagens de ER, aspectuais e não aspectuais, com LPS.

4.1 Trabalhos Relacionados de ER Orientada a Aspectos para LPS

Nesta subsecção serão apresentados trabalhos de engenharia de requisitos aspectuais com LPS.

4.1.1 MATA e LPS

Em (Jayaraman et al., 2007) é descrita uma abordagem baseada em UML para manter a separação de *features* durante a modelação, usando uma linguagem de composição baseada em transformações de grafos (o que leva a uma mais fácil reutilização do modelo de *features*) e uma abordagem para detecção de interacções estruturais indesejáveis entre modelos de *features* diferentes. A linguagem pode ser usada para compor modelos de LPS para um dado conjunto de *features*. As *features* base são expressas em termos de diagramas UML, tais como, diagramas de classe, sequência e estados. As *features* variáveis são especificadas em UMLT (*Unifield Modeling Language Transformation*), ou seja, numa representação UML de transformação de grafos que indica como é feita a modificação dos modelos base.

A técnica utilizada é a técnica MATA (*Modeling Aspects using a Transformation Approach*) que usa transformações de grafos para especificar e compor aspectos, mas para linhas de produtos, pois podem-se adicionar novas *features* ou combinar *features* facilmente.

4.1.2 I* aspectual e LPS

Silva (Silva et al., 2008) apresenta uma abordagem que liga o i* aspectual a linhas de produto de software, suportando variabilidade. A selecção de características específicas e a sua composição para um produto individual, com os recursos mais importantes de LPS, é facilitado devido à orientação a aspectos. Combinar o i* aspectual com abordagens onde extraem mais cedo a variabilidade no ciclo de vida do desenvolvimento de uma LPS enriquece a variabilidade capturada pelo i* aspectual. Esta variabilidade pode facilitar a relação entre vários tipos de requisitos (por exemplo, requisitos funcionais e não funcionais) em um mesmo modelo. Os autores propõem um conjunto de heurísticas para criar modelos i* aspectuais a partir do modelo de *features*, ilustrando como uma estratégia orientada a objectivos pode ser usada para representar variabilidade em modelos de requisitos, sendo as heurísticas as seguintes: separação de *features* em modelos i*; tipos de *features* e relacionamento nos modelos; verificação da exactidão das relações e o mapeamento dos nomes das *features* para os nomes dos elementos do modelo i* aspectual. A abordagem considera que cada *feature* opcional ou alternativa irá ser mapeada num aspecto, o que nem sempre acontece.

Existe pouca capacidade na gestão de complexidade dos modelos, pelo que a representação de todas as variabilidades torna a sua compreensão difícil.

4.1.3 Casos de Uso e Modelo de *Features*

Em (Bonifácio e Borba, 2009) é proposta uma abordagem para gerir a variabilidade em casos de uso, permitindo uma melhor separação dos assuntos entre as linguagens usadas para gerir variabilidades e linguagens utilizadas para especificar cenários de casos de uso. Bonifácio e Borba (Bonifácio e Borba, 2009) vão além das questões de composição de cenários e consideram uma noção mais abrangente de gestão de variabilidade, incluindo artefactos tais como, modelos de *features* e configuração de conhecimento, explicando isto como um fenómeno transversal. A abordagem apresenta as seguintes contribuições: caracterização da linguagem de gestão de variabilidade como um assunto transversal, propondo uma abordagem em que os assuntos de variabilidade são separados de outros assuntos; um *framework* para modelar o processo de composição de mecanismos de

variabilidade de cenários. Este *framework* oferece: uma base para descrever a variabilidade como mecanismo transversal; a especificação de três fontes de variabilidade para cenários de caso de uso, sendo elas, a variabilidade de função, a variabilidade de dados e a variabilidade no controlo de fluxo.

A abordagem (Bonifácio e Borba, 2009) apresenta um mecanismo de separação para mapear assuntos de LPS, assim como também apresenta uma forma para reduzir o número de alternativas nos cenários base da LPS para que possa evoluir, por meio de introdução de novos cenários.

4.2 Trabalho Relacionados de ER Não Aspectuais para LPS

Nesta subsecção serão apresentados trabalhos de engenharia de requisitos não aspectual, nomeadamente orientados a casos de uso e a objectivos, com LPS.

4.2.1 Casos de uso e LPS

Gomaa (Gomaa, 2004) define uma abordagem que combina casos de uso com linhas de produtos de software. Para os casos de uso permitirem a sua ligação a linhas de produtos usa estereótipos UML em diagramas de casos de uso, tais como, "kernel", opcionais e alternativos, assim como modelação de casos de uso como *features* no modelo de *features*.

Nesta abordagem o conceito de transversalidade é usado apenas dentro de uma única especificação de caso de uso, descrevendo os pontos de variação que modelam a variabilidade dentro do caso de uso. Para esses pontos de variação foi necessário adaptar os estereótipos *«extends»* e *«includes»* para suportar variabilidade.

Gomaa utilizou o modelo de *features* para integrar casos de uso com linhas de produtos, podendo os casos de uso ser mapeados para *features*. Uma *feature* pode corresponder a um único caso de uso, a um grupo de casos de uso ou a um ponto de variação dentro de um caso de uso, de onde se pode verificar que os casos de uso e as *features* se complementam. Considerando as propriedades de reutilização de casos de uso e de *features*, poderá haver mapeamento entre ambos. Gomaa (Gomaa, 2004) considera que a modelação de um grupo de casos de uso reutilizáveis será mapeado para uma *feature* e

representado através de um pacote de casos de uso. A modelação de *features* poderá corresponder a uma metaclasse e a relação de casos de uso com *features* irá apresentar uma forma tabular, em que as *features* são representadas em uma tabela. Na abordagem apenas os requisitos funcionais de uma LPS são modelados.

4.2.2 *Problem Frames* e Linhas de Produtos

Classen (Classen et al., 2007) mostra que os diagramas de *features* e a abordagem *problem frames* podem ser utilizados como técnicas complementares. A análise dos *problem frames* poderia servir como uma primeira análise de requisitos, no desenvolvimento das linhas de produtos. Com a abordagem de *problem frames* pode-se colocar os diagramas de *features* no contexto de domínio no mundo real, contrariamente ao que acontece quando se liga o modelo de *features* com abordagens de casos de uso e modelos orientados a objectivos. A complementaridade entre as abordagens é mostrada quando são indicados diferentes pontos fracos dos diagramas de *features* e estes podem ser resolvidos através dos *problem frames*. A abordagem proposta (Classen et al., 2007) dá maior relevância aos seguintes pontos: os diagramas de *features* existentes serão complementados por uma análise subjacente da complexidade do problema; notações e semântica dos diagramas de *features* e *problem frames* não são alterados ou estendidos; não há alterações para o processo nem metodologias são sugeridas; os diagramas de *features* podem ser ligados aos *problem frames*.

4.2.3 EROO e Modelo de *Features*

Em Yu et al. (Yu et al., 2008) é proposta uma abordagem dirigida por modelos, em que através de um modelo de objectivos, que captura as necessidades dos *stakeholders,* se obtém um modelo de *features* que representa a variabilidade do sistema. O objectivo da abordagem é combinar as vantagens dos objectivos e das *features* de modo a traçar e configurar o sistema de *features,* de acordo com os objectivos dos *stakeholders*.

Yu et al. (Yu et al., 2008) introduzem a ferramenta OpenOME de apoio à abordagem, através da manutenção de rastreabilidade entre objectivos e *features*. Esta ferramenta gera um modelo de *features* inicial através dos objectivos dos *stakeholders* descritos em modelos i*. Yu et al. afirmam que a ferramenta mantém a rastreabilidade entre os

objectivos e as *features*, permitindo também a configuração de *features* de uma aplicação através do uso de algoritmos de raciocínio sobre objectivos. Contudo, a abordagem não apoia uma estruturação apropriada para lidar com as várias alternativas disponíveis de linhas de produto de software.

4.2.4 IStarLPS

António (António, 2009) (António et al., 2009) apresenta uma abordagem que adapta o *framework* i* com linhas de produtos. Tal como na engenharia de linhas de produtos, o processo da abordagem é também composto por duas fases. Na primeira fase, Engenharia de Domínio, inicialmente são identificados e modelados os objectivos da linha de produtos, de seguida é produzido o modelo SD e depois o modelo SR (como acontece no *framework* i*). Por fim é efectuada a extracção de *features* e obtenção de informação sobre a variabilidade e a interacção entre as *features*, de uma forma sistematizada e flexível. Na segunda fase, Engenharia de Aplicação, configura-se primeiro o modelo de *features* (onde são seleccionadas as *features* de um determinado produto) e depois os modelos i*. A autora propõe um conjunto de heurísticas para ir construindo o modelo à medida que se vão encontrando as *features.*

Esta abordagem insere a cardinalidade nos modelos i* que permite representar a variabilidade, de uma linha de produtos. A cardinalidade nestes modelos torna mais fácil a extracção da informação, para a produção do modelo de *features*.

4.2.5 LPS-KAOS

A abordagem descrita por Baptista (Baptista, 2009) (Batista et al., 2009) orientada a objectivos, permite especificar as linhas de produto de software, em que foram definidas um conjunto de heurísticas que permitem a sistematização da identificação das *features* para um sistema e também permitem, metodicamente, o desenvolvimento do modelo de *features.* A abordagem é, tal como nas linhas de produtos de software, composta por dois processos: Engenharia de Domínio e Engenharia de Aplicação. Na Engenharia de Domínio são definidos procedimentos para a obtenção dos objectivos de uma LPS, usando a abordagem KAOS, que vão fazer parte do modelo de *features*. Para além da identificação destes objectivos e *features* de um sistema os modelos KAOS e de *features*

que serão produzidos, vão permitir dar uma visão clara do sistema, em termos de diagramas. Na Engenharia da Aplicação são utilizados os modelos da LPS especificados a nível de Domínio para a configuração de uma aplicação específica.

4.3 Sumário

Este capítulo mostrou algumas abordagens que integram Linhas de Produtos de Software e abordagens de Engenharia de Requisitos, aspectuais e não aspectuais, e permitem a modelação da variabilidade de uma linha de produtos no desenvolvimento de um software.

No próximo capítulo será apresentada uma abordagem que permite adaptar a abordagem Theme para Linhas de Produtos de Software, que se designou por Theme-SPL.

5. Abordagem Theme-SPL

Neste capítulo será apresentado o processo que descreve a abordagem a ser desenvolvida, designada por Theme-SPL, que adapta a abordagem Theme para Linhas de Produtos de Software. Este processo mostra como desenvolver um modelo de *features* recorrendo à informação de modelos Theme. O processo é ilustrado de dois modos. No primeiro (secção 5.2), analisando a abordagem Theme (Clarke e Baniassad, 2005), é definido um conjunto de heurísticas para elaborar o modelo de *features* através dos modelos de Theme.

No segundo modo (secção 5.3) são definidas um conjunto de heurísticas para estender a abordagem Theme com conceitos de LPS para usar Desenvolvimento Orientado a Modelos, de modo a derivar automaticamente um modelo de *features* a partir dos modelos Theme. Este segundo conjunto de heurísticas é elaborado tendo em conta as heurísticas definidas na secção 5.2, e verificando o que se precisa dos requisitos para se poder gerar o modelo de *features*. Com esta análise, é então adicionada a informação nos modelos Theme para que a geração do modelo de *features* dependa mais dos modelos Theme do que da listagem de requisitos. Isto irá ajudar no desenvolvimento de uma Linguagem de Domínio Específico para a abordagem, e nas transformações de modelos Theme para modelos de *features*.

Do modo que a abordagem Theme está definida a implementação das heurísticas iria envolver processamento de linguagem natural, que não é o pretendido para a presente dissertação.

5.1 Processo de Utilização da Abordagem Theme-SPL

Como foi visto no Capítulo 3, a engenharia de linha de produtos de software é composta por duas fases, Engenharia de Domínio e Engenharia de Aplicação. Do mesmo modo, o processo desta abordagem é composto por estas mesmas duas fases. Primeiramente é

obtido o processo a nível de Engenharia de Domínio para a linha de produtos, como ilustra a Figura 5.1, e de seguida configura-se um produto a nível de Engenharia de Aplicação, como mostra a Figura 5.2.

Na Figura 5.1 o processo encontra-se divido em três partes. Inicialmente é efectuada a elicitação e análise de temas e requisitos, em que é examinada a documentação de requisitos do sistema; seguidamente são identificados os principais temas, assim como as entidades, e é efectuado o relacionamento entre temas e requisitos para se definir a visão de relação de temas; no final os temas e requisitos são validados pelos *stakeholders*. Após esta análise é então construído o modelo de *features*, em que primeiramente são identificadas as *features* e analisada a variabilidade recorrendo à visão de relação de temas, definida no passo anterior, sendo então o modelo de *features* definido e posteriormente validado. Para a validação ser realizada é necessária a participação dos *stakeholders* a fim de detectar possíveis inconsistências e omissões. Apesar de elaborada, a validação não é o foco deste trabalho, sendo efectuada para avaliar a usabilidade da Linguagem de Domínio Específico desenvolvida. Na última fase do processo irá refinar-se o modelo de *features*, em que serão identificados os temas aspectuais, é construída a visão transversal, e com base nesta é então refinado o modelo de *features* tendo como base os temas aspectuais, sendo o mesmo validado no final.

Na Figura 5.2 o processo encontra-se dividido em duas fases, em que o modelo de *features* e os modelos de Theme são analisados e configurados para uma dada instância da linha de produtos. Tanto o modelo de *features* como os modelos de Theme são configurados para o produto pretendido.

O processo apresentado para a engenharia de domínio, na Figura 5.1, e para a engenharia de aplicação, na Figura 5.2, pretende facilitar a obtenção do modelo de *features* através da abordagem Theme. Assim, os processos que se apresentam nesta secção tiram partido dos conhecimentos previamente adquiridos da utilização das abordagens de engenharia de LPS e da abordagem Theme, em separado.

Nas secções 5.2 e 5.3 será ilustrado cada passo do processo, com o exemplo de Software de Telemóveis para ajudar na compreensão da abordagem, cujos requisitos são estabelecidos da seguinte forma (Moodle, 2008):

"É preciso desenvolver componentes de software para telemóveis que suportem as seguintes funcionalidades: realizar chamadas, permitir chamadas em espera, manter a lista de contactos, marcar eventos numa agenda, tirar fotografias e enviá-las por MMS, enviar uma mensagem para vários utilizadores, enviar e receber e-mails e SMS, configurar alarme e transferir dados entre dois telemóveis.

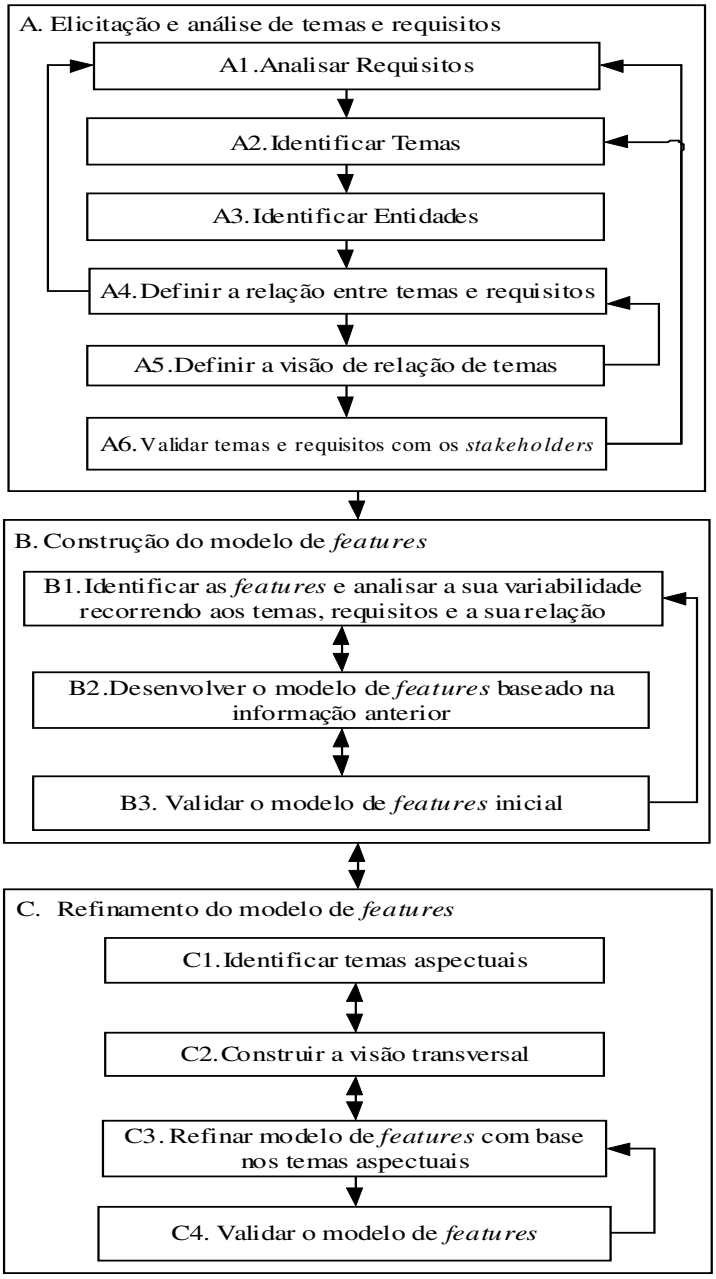

Figura 5.1 Processo a nível de Engenharia de Domínio

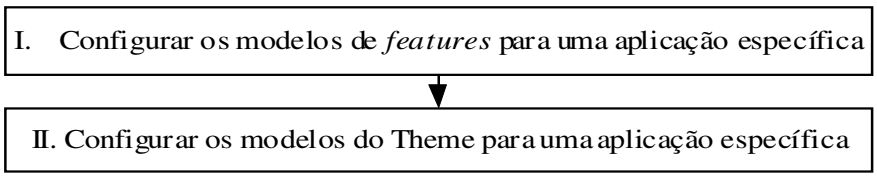

Figura 5.2 Processo a nível de Engenharia de Aplicação

Para realizar chamadas o procedimento é o habitual, mas deve ser permitida a chamada em espera tanto para quem realiza a chamada, como para quem já está a realizar uma chamada enquanto outra está a ser atendida.

É preciso também manter uma lista de contactos onde se deve guardar o nome, o número do telemóvel, o endereço de e-mail, o tipo do contacto (e.g. empresa, particular), o número de fax e a data de aniversário.

A agenda deve fornecer opções de escolha. Portanto, para marcar um evento na agenda o utilizador deve indicar um período em que o evento possa ser marcado, ou seja, a data de início e a data de fim, assim como da duração do evento.

Com relação às mensagens SMS, além de enviar deve-se guardar a mensagem a ser enviada. O envio pode ser para um só telemóvel ou para vários.

Para configurar o alarme pode-se ter a opção de activá-lo todos os dias àquela hora e quantas repetições são desejadas.

No que diz respeito à transferência de dados entre telemóveis, deve-se primeiro activar o mecanismo de transferência (e.g. bluetooth), seleccionar o tipo de dados a enviar (e.g. fotografia, SMS), assim como seleccionar o ficheiro propriamente dito.

O carregamento dos telemóveis pode ser feito através de multibanco ou do site do banco onde o cliente tem conta. O valor carregado vai sendo debitado de acordo com algumas utilizações do aparelho (e.g. chamadas, envio de mensagens SMS)."

5.2 Aplicação da Abordagem Theme-SPL sem suporte a Desenvolvimento Orientado a Modelos

Como visto na secção 5.1, o processo está dividido em engenharia de domínio e engenharia de aplicação, tal como na ELPS. 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 de Telemóveis para ilustrar os vários passos do processo da abordagem Theme-SPL.

A. Elicitação e análise de temas e requisitos.

Para modelar este sistema será produzida a visão de relação de temas da abordagem Theme. Para tal começa-se por analisar os requisitos do sistema, seguidamente identificam-se quais os temas e as entidades pertinentes, e define-se a relação entre os temas e requisitos, tal como no desenvolvimento dos modelos da abordagem Theme para um único sistema. A visão de relação de temas será construída tendo em conta os requisitos e os temas, sendo para isso elaborada a lista de relacionamento de temas e requisitos.

A1. Analisar Requisitos. Os requisitos identificados para o sistema de telemóveis estão descritos na Tabela 5.1.

Tabela 5.1 Requisitos de desenvolvimento de componentes de software para telemóveis

R No.	Descrição dos Requisitos
R1	Os **componentes** de **software** do telemóvel suportam as seguintes funcionalidades: **realizar chamadas**, **receber chamadas**, **permitir chamadas em espera**, pode **gerir** a **lista de contactos**, **marcar eventos** na **agenda**, **tirar fotografias**, **enviar SMS** **(informação)**, **receber SMS** **(informação)**, pode **enviar MMS** **(informação)**, pode **receber MMS** **(informação)**, pode **enviar e-mails** **(informação)**, pode **receber e-mails** **(informação)**, **configurar alarme** e pode **transferir dados**.
R2	De entre os **componentes** de **software** existentes no telemóvel o utilizador selecciona um item para começar.
R3	O utilizador deve manter a **lista de contactos** actualizada, podendo introduzir novos contactos na aplicação, sendo que estes têm que conter o nome, número de telemóvel, qual o tipo de contacto, e podem ou não conter o número de fax, e-mail e a data de aniversário.
R4	Para **realizar** uma **chamada** o utilizador pode procurar o contacto do destinatário, na aplicação **contactos**, e **premir** o **botão** de chamada através do teclado do telemóvel.

R5	Para **realizar** uma **chamada** é preciso **verificar** se há **saldo** no telemóvel.
R6	Se não há saldo no telemóvel é **mostrada** uma mensagem ”Sem saldo”.
R7	Se há saldo no telemóvel **a chamada é realizada.**
R8	Quando o utilizador está a **realizar** uma **chamada**, a sua **chamada** pode ficar **em espera**, sendo mostrado no visor do telemóvel uma mensagem “Chamada em espera”.
R9	Ao **receber** uma **chamada** o utilizador, através do teclado, clica no botão pretendido.
R10	Se pretende atender a chamada prime o botão verde de chamada.
R11	Se não pretende atender a chamada então o utilizador prime o botão vermelho “Rejeitar Chamada”.
R12	O utilizador utiliza a **agenda** para marcar eventos importantes, indicando a data e a duração do mesmo.
R13	Para **enviar** uma **MMS**, o utilizador, utilizando a máquina fotográfica do telemóvel, **tira** uma **fotografia** e envia-a a outro utilizador.
R14	Para **enviar** uma **MMS**, o utilizador terá que anexar a fotografia à mensagem, pode escrever o texto referente à fotografia, se assim o pretender, e pode procurar o contacto na **lista** de **contactos.**
R15	Para **enviar MMS** é preciso **verificar** se há **saldo** no telemóvel.
R16	Se não há saldo no telemóvel é **mostrada** uma mensagem no visor ”Sem saldo”.
R17	Se há saldo no telemóvel **mostra** uma mensagem no visor “Mensagem multimédia enviada”.
R18	Ao **receber** uma **MMS** irá ser apresentada uma mensagem “Nova mensagem de multimédia recebida” no visor do telemóvel.
R19	Para ler a nova **MMS recebida** o utilizador **prime** o **botão** do teclado “Abrir mensagem”.
R20	Para **enviar** uma **SMS**, o utilizador terá que escrever a mensagem e pode procurar o contacto na **lista** de **contactos**, podendo a mesma ser enviada a um ou mais destinatários.
R21	A **mensagem enviada** pode ser guardada nos rascunhos, caso o utilizador o pretenda.
R22	Para **enviar SMS** é preciso **verificar** se há **saldo** no telemóvel.
R23	Se não há saldo no telemóvel é **mostrada** uma mensagem no visor ”Sem saldo”.
R24	Se há saldo no telemóvel a mensagem é enviada e **mostra** uma mensagem no visor “Mensagem enviada”.
R25	Ao **receber** uma **SMS** irá ser apresentada uma mensagem “Nova SMS Recebida” no visor do telemóvel.
R26	Para ler a nova **SMS recebida** o utilizador **prime** o **botão** do teclado “Abrir mensagem”.
R27	Para **enviar** um **e-mail** o utilizador pode seleccionar o (s) contacto (s), na **lista** de **contactos**, escrever o assunto e escrever a mensagem de e-mail.

R28	Para **enviar** o **e-mail** é preciso **verificar** se há **saldo** no telemóvel.
R29	Se não há saldo no telemóvel é **mostrada** uma mensagem no visor ”Sem saldo”.
R30	Se há saldo no telemóvel **mostra** uma mensagem no visor “E-mail enviado”.
R31	Ao **receber** uma nova **mensagem de correio electrónico** irá ser apresentada uma mensagem “Novo e-mail Recebido” no visor do telemóvel.
R32	Para ler o novo **E-mail recebido** o utilizador **prime** o **botão,** no teclado, “Abrir correio electrónico”.
R33	O utilizador **configura** o **alarme** tendo a opção de activá-lo todos os dias àquela hora e quantas repetições desejar.
R34	É permitido ao utilizador fazer **transferência de dados** entre telemóveis por pelo menos um dos mecanismos: bluetooth, infravermelhos ou USB.
R35	Para efectuar a **transferência de dados** o utilizador tem que activar o mecanismo de transferência, seleccionar o tipo de dados a enviar, fotografia ou SMS, e seleccionar o ficheiro.
R36	O utilizador tem que efectuar o **carregamento** do **telemóvel** para utilizar o mesmo, através do método pretendido, tendo que fornecer o seu número de telemóvel, pode fornecer a referência dependente de qual a rede telefónica de que é cliente, e a quantia.
R37	O método para **carregamento** do **telemóvel** apenas pode ser efectuado por multibanco ou do site do banco na internet.

A2. Identificar Temas. De seguida percorrem-se os requisitos seleccionando os termos chave, que serão os potenciais temas. Foram identificados dezanove potenciais temas, como ilustra a Figura 5.3.

Carregar Telemóvel	**Mostrar Mensagem Visor**	**Receber SMS**
Configurar Alarme	**Permitir Chamada em Espera**	**Software Telemóvel**
Enviar E-mail	**Premir Botão**	**Tirar Fotografias**
Enviar MMS	**Realizar Chamada**	**Transferir Dados**
Enviar SMS	**Receber Chamada**	**Verificar Saldo**
Gerir Lista Contactos	**Receber E-mail**	
Marcar Eventos Agenda	**Receber MMS**	

Figura 5.3 Lista de Temas

A3. Identificar Entidades. Foram identificadas, nos requisitos, nove entidades mostradas na Figura 5.4.

Bluetooth	Máquina Fotográfica	Telemóvel
Infravermelhos	Multibanco	Visor
Internet	Teclado	USB

Figura 5.4 Lista de Entidades

A4. Definir a relação entre temas e requisitos. Na Figura 5.5 mostramos uma lista relacionando os temas com os requisitos.

Carregar Telemóvel -> R36, R37	Realizar Chamada -> R1, R4, R5, R6, R7, R8
Configurar Alarme -> R1, R33	Receber Chamada -> R9, R10, R11
Enviar E-mail -> R1, R27, R28, R29, R30	Receber E-mail -> R1, R31, R32
Enviar MMS -> R1, R13, R14, R15, R16, R17	Receber MMS -> R1, R18, R19
Enviar SMS -> R1, R20, R21, R22, R23, R24	Receber SMS -> R1, R25, R26
Gerir Lista Contactos -> R1, R3, R4, R14, R20, R27	Software Telemóvel -> R1, R2
Marcar Eventos Agenda -> R1, R12	Tirar Fotografia -> R1, R13
Mostrar Mensagem Visor -> R5, R6, R8, R16, R17, R18, R19, R23, R24, R25, R29, R30, R31	Transferir Dados -> R1, R34, R35
Permitir Chamada em Espera -> R1, R8	Verificar Saldo -> R5, R6, R7, R15, R16, R17, R22, R23, R24, R28, R29, R30
Premir Botão -> R4, R19, R26, R32	

Figura 5.5 Lista de Relacionamentos de Temas e Requisitos

A5. Definir a Visão de Relação de Temas. A visão de relação de temas é criada através da lista de relacionamento de temas e requisitos, isto é, cada tema encontra-se ligado aos requisitos que o contém. Se o nome de um tema é mencionado num requisito, então existe uma ligação a partir do requisito para esse tema na visão de relação. A Figura 5.6 mostra a Visão de Relação para este sistema.

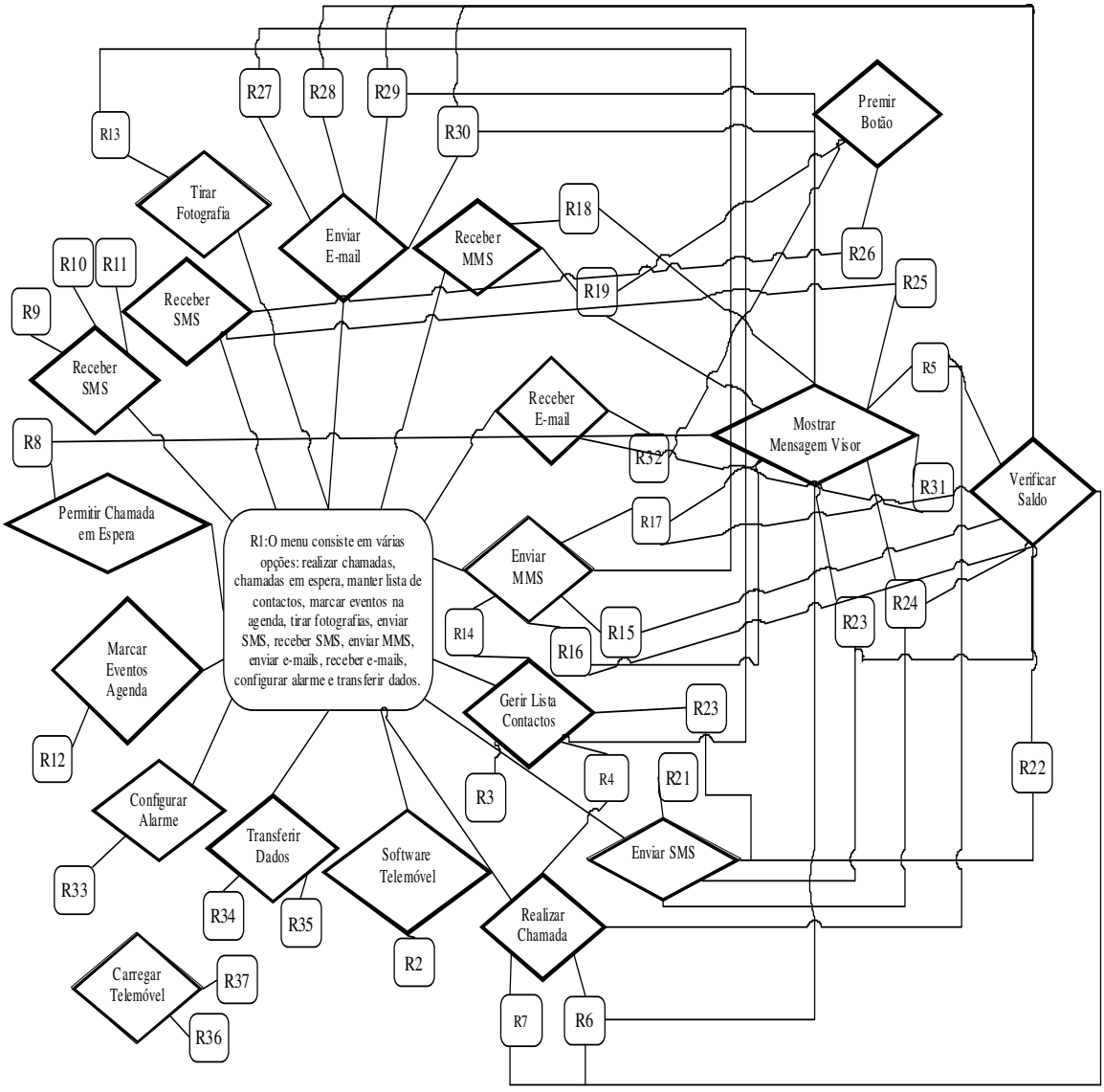

Figura 5.6 Visão de Relação de Temas

A6. Validar temas e requisitos com os *stakeholders*. A validação dos temas e requisitos é feita ao longo do desenvolvimento do relacionamento de temas e requisitos, através dos *stakeholders* que irão rever as decisões tomadas para a construção da visão de relação de temas.

B. Construção do modelo de *features*.

Após a elaboração da visão de relação de temas é possível obter algumas das principais *features*.

B1. Identificar as *features* e analisar a sua variabilidade recorrendo aos temas, requisitos e à sua relação. Através da relação entre temas e requisitos podem ser identificadas as *features* inicias, assim como através das entidades. Deste modo, as *features* serão obtidas através dos temas, requisitos e entidades, definidos na fase anterior.

Na secção 3.5 definiu-se *feature* como sendo uma característica de um produto, que utilizadores e clientes consideram importante na descrição e distinção de membros de uma família de produtos. Como referido atrás, a identificação das *features* obtém-se através dos temas, requisitos e entidades compondo as seguintes heurísticas para a obtenção das mesmas:

> **H.1 (Identificação das *features* através dos temas)**: As *features* são extraídas dos temas, pelo que, a cada tema corresponderá uma *feature*. Por exemplo, no tema "Lista Contactos" a *feature* será "Lista Contactos".
>
> **H.2 (Identificação das *features* através das entidades)**: Identificação das *features* através das entidades, sendo que estas estão relacionadas com os temas. Por exemplo, a entidade "Máquina Fotográfica" é necessária uma vez que o tema "Tirar Fotografia" necessita que o telemóvel possua máquina fotográfica, de onde se obtém a *feature* "Máquina Fotográfica".
>
> **H.3 (Identificação das *features* através do relacionamento entre temas e requisitos):** Possivelmente, uma *feature* X terá pelo menos uma *sub-feature* Y quando o tema que dá origem a X se encontra relacionado com um ou mais requisitos. Por exemplo, o tema "Configurar Alarme" poderá ser mais detalhado com *sub-features* onde neste caso, olhando para a lista de relacionamento de temas e requisitos, a acção "Configurar Alarme" encontra-se relacionada com o requisito "O utilizador configura o alarme tendo a opção de activá-lo todos os dias àquela hora e quantas repetições desejar", pelo que após a análise deste se conclui que as *sub-features* serão "Dia", "Hora" e "Repetições".
>
> **H.4 (Identificação de *features* que podem ser modularizadas)**: Se dois ou mais temas começam pelo mesmo verbo e são do mesmo tipo, então cria-se uma nova *feature*, Verbo+Substantivo, que tem como *sub-features* os temas que lhe deram origem. Por exemplo, os temas "Enviar SMS", "Enviar MMS" e "Enviar E-mail",

começam todos pelo verbo Enviar, e os três são relativos ao envio de informação, pelo que é criada uma *feature* "Enviar Dados", que terá como *sub-features* o "Enviar SMS", "Enviar MMS" e "Enviar E-mail".

Do mesmo modo, pela heurística H.4, é também criada uma nova *feature* "Receber Mensagem" que tem como *sub-features* "Receber MMS", "Receber SMS" e "Receber E-mail", que lhe deram origem.

H.5 (Identificação de hierarquia e variabilidade de *features*): Pode-se obter informação sobre a hierarquia de *features* quando, num requisito, aparece uma expressão do tipo "quando X pode (acontecer) Y", e obtêm-se as *features* X e Y, então possivelmente Y será *sub-feature* de X, e Y será opcional. Como no exemplo "Quando realizar uma chamada a mesma **pode** ficar em espera", obtêm-se as *features* "Realizar Chamada" e "Chamada em espera", que estão relacionadas hierarquicamente, onde "Chamada em espera" é relativa a "Realizar Chamada", e por isso será *sub-feature* de "Realizar Chamada", e "Chamada em Espera" é opcional.

A *sub-feature* "Texto" de "Enviar MMS" é opcional, uma vez que, pela heurística H.5, no requisito aparece "para enviar MMS **pode** escrever o texto".

A Figura 5.7 representa a primeira versão do modelo de *features* com as *features* obtidas anteriormente da Lista de Temas da Figura 5.3, da Lista de Entidades da Figura 5.4, da Lista de Relacionamento de Temas e Requisitos da Figura 5.5 e da Visão de Relação de Temas, aplicando as heurísticas de H.1 a H.5.

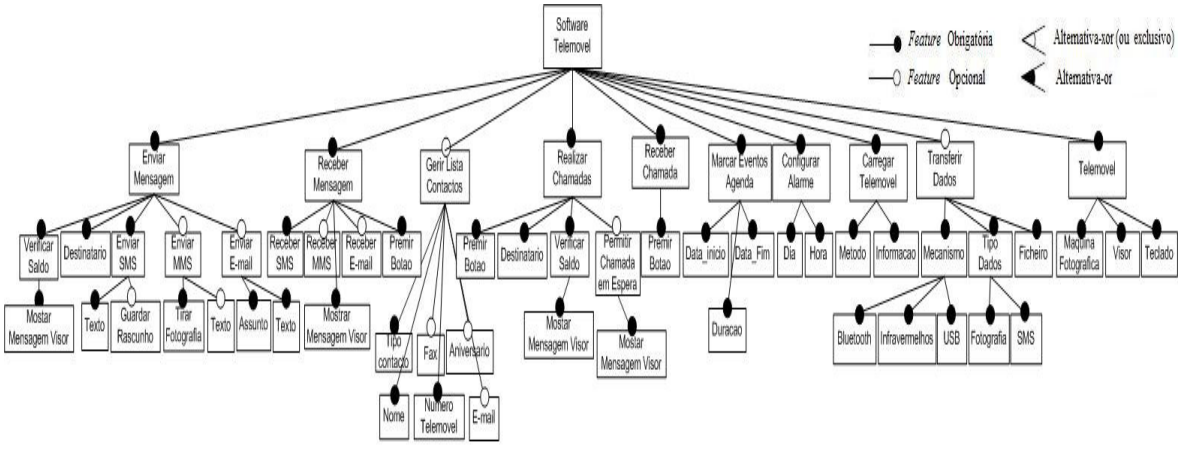

Figura 5.7 Primeira versão do modelo de *features*

H.6 (Identificação de alternativas XOR do modelo de *features*): Através das alternativas no documento de requisitos são definidas as variabilidades que irão fazer parte do sistema. Se nos requisitos aparecerem várias alternativas para escolher uma entre várias, então estamos perante um "XOR" (ou exclusivo). Como por exemplo, o "Método" de pagamento só terá uma das *sub-features* "Multibanco" ou "Internet" pelo que as *features* pertencem a um grupo "XOR", onde só uma das alternativas pode aparecer na aplicação.

H.7 (Identificação de alternativas OR do modelo de *features*): Se nos requisitos aparece uma expressão do tipo "pelo menos um" de várias alternativas, então estamos perante uma alternativa "OR". Por exemplo, o "Mecanismo" de transferência pode ser "por pelo menos um dos mecanismos: bluetooth, infravermelhos ou USB", pelo que é uma alternativa "OR", pois permite que o telemóvel apresente mais do que um mecanismo de transferência.

O "Tipo Dados" a transferir permite enviar "Fotografia" ou "SMS" pelo que as *sub-features* pertencem a um grupo "OR", uma vez que ambos os tipos são permitidos.

A Figura 5.8 representa uma segunda versão do modelo de *features*, após aplicação das heurísticas H.6 e H.7, à versão anterior, onde são ilustradas variabilidades.

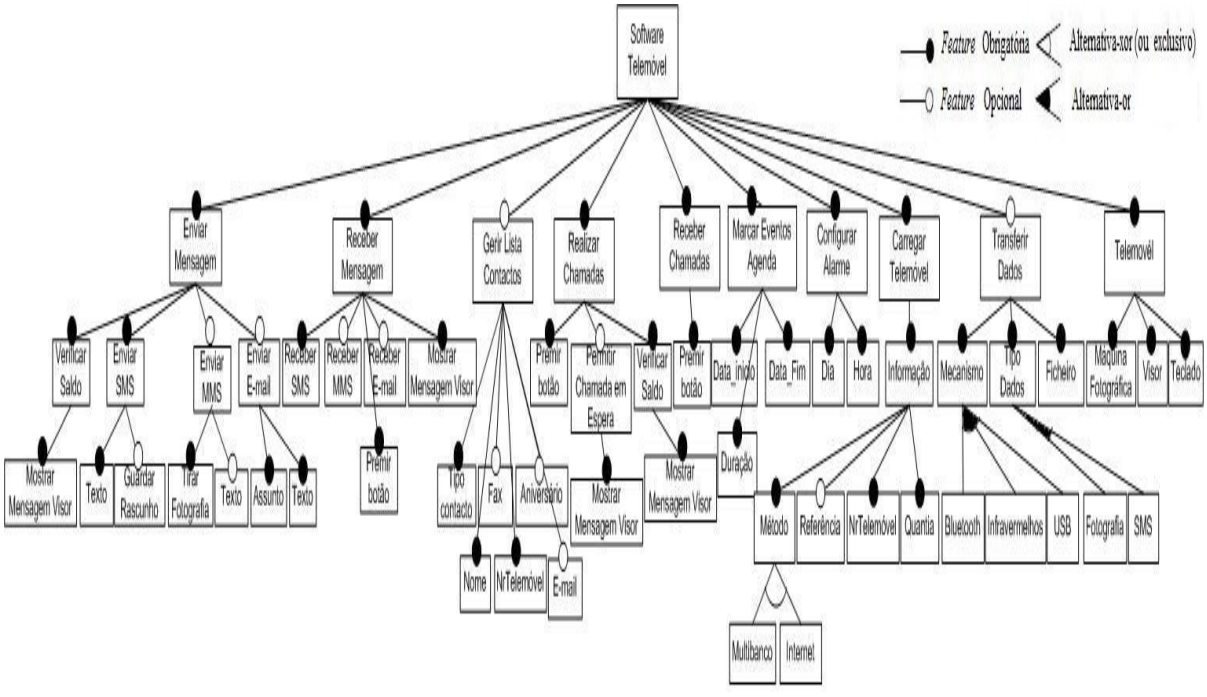

Figura 5.8 Segunda versão do modelo de *features*

B3. Validar modelo de *features* inicial. A validação é feita pelos *stakeholders* que irão efectuar uma revisão pormenorizada sobre as várias opções do modelo de *features*, e se correspondem com as especificações pretendidas.

C. Refinamento do modelo de *features*.

Após desenhar o modelo de *features* inicial, o mesmo é refinado através da visão transversal, uma vez que esta oferece informação adicional para completar o modelo.

C1. Identificar temas aspectuais. Como referido na secção 2.3.2.3, para identificar temas aspectuais e temas base devem ser colocadas algumas questões para identificação dos mesmos. Tendo em conta essas questões, e olhando para o requisito R14, verifica-se que o mesmo é partilhado pelos temas "Gerir Lista Contactos" e "Enviar MMS". Analisando a descrição do mesmo verifica-se que a acção "Gerir Lista Contactos" é necessária quando se considera "Enviar MMS", sendo assim o tema responsável pelo R14. Os requisitos R20 e R27 são também requisitos partilhados, e tal como no requisito anterior verifica-se que a acção "Gerir Lista Contactos" é uma acção necessária quando se considera "Enviar SMS" e "Enviar e-mail", respectivamente, sendo este o tema dominante em ambos os requisitos. No caso do requisito R4, verifica-se que para realizar uma chamada é também necessária a acção "Gerir Lista Contactos". Como o tema "Gerir Lista Contactos" é desencadeado em mais de uma situação, verifica-se que o mesmo é um tema aspectual cujas bases são "Realizar Chamadas", "Enviar SMS", "Enviar MMS" e "Enviar e-mail".

Os requisitos R5, R6, R7, R15, R16, R17, R22, R23, R24, R28, R29 e R30 são partilhados, e em todos eles se verifica que a acção "Verificar Saldo" é necessária para quando se considera "Realizar Chamadas", "Enviar SMS", "Enviar MMS" e "Enviar e-mail", pelo que "Verificar Saldo" é um tema aspectual.

Todos os restantes temas presentes nos requisitos são temas base.

C2. Construir a Visão Transversal. Com base nos temas aspectuais e base é elaborada a visão transversal, de acordo com os conhecimentos adquiridos sobre a abordagem Theme, como ilustrado na Figura 5.9.

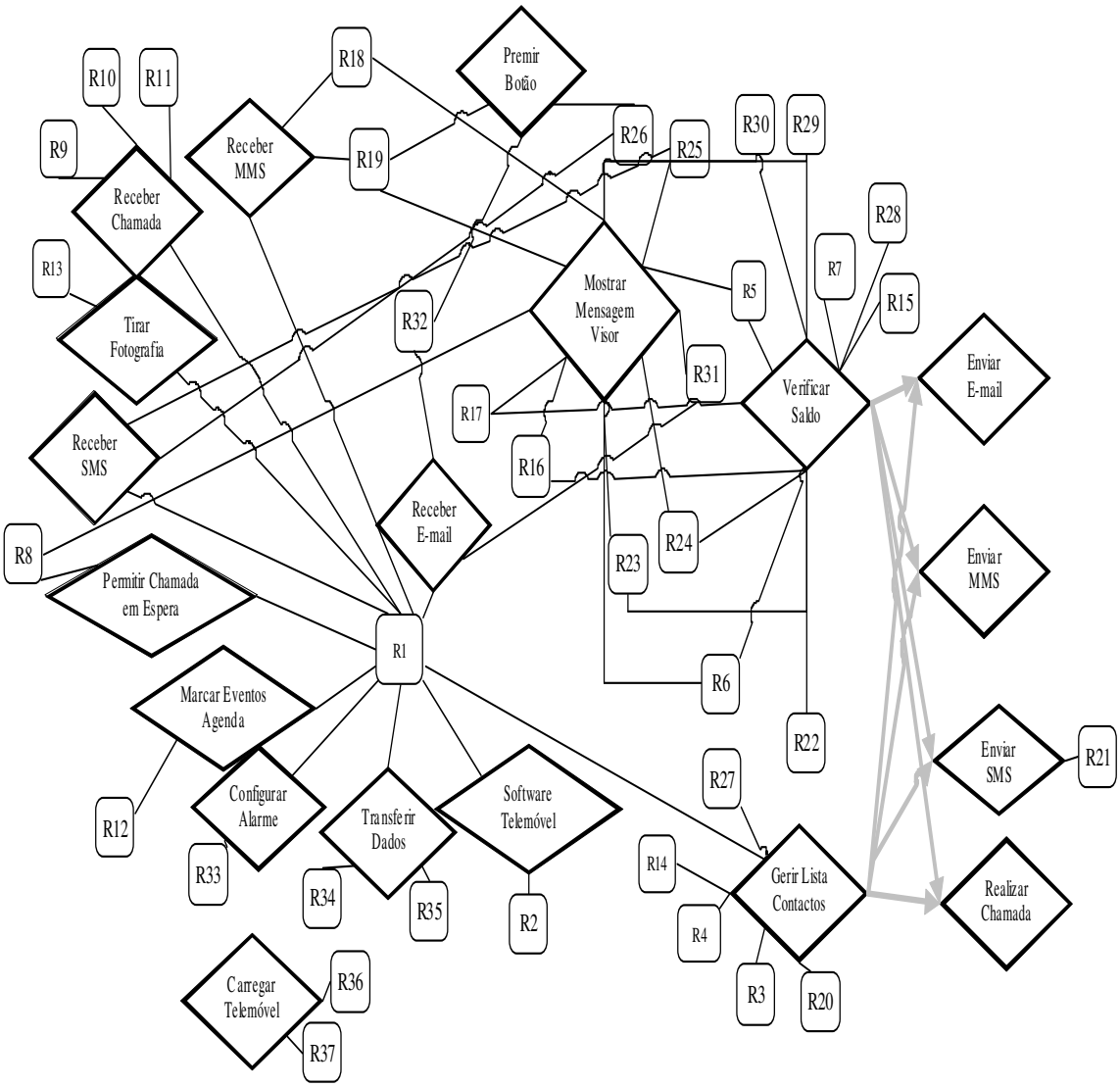

Figura 5.9 Visão Transversal do sistema de telemóveis

C3. Refinar o modelo de *features* com base nos temas aspectuais. A visão transversal oferece informação complementar para completar o modelo de *features*, melhorando-o, uma vez que permite modularizar as *features* repetidas.

> **H.8 (Identificação de *features* através de temas aspectuais)**: Se uma *feature* corresponde a um tema aspectual então a mesma deverá ficar ligada por decomposição às *features* que são afectadas por esta *feature* aspectual. Por exemplo, a *feature* "Gerir Lista Contactos", é extraída do tema aspectual "Gerir Lista Contactos", e será *sub-feature* de "Enviar Mensagem" e "Realizar Chamadas", tendo apenas uma *feature* "Gerir Lista Contactos" ligada a ambas as *features*.

A *feature* "Verificar Saldo" será *sub-feature* de "Enviar Mensagem" e "Realizar Chamadas", de acordo com a heurística H.8.

A Figura 5.10 ilustra uma terceira versão do modelo de *features*, com as *features* obtidas da Visão Transversal (Figura 5.9), aplicando a heurística H.8.

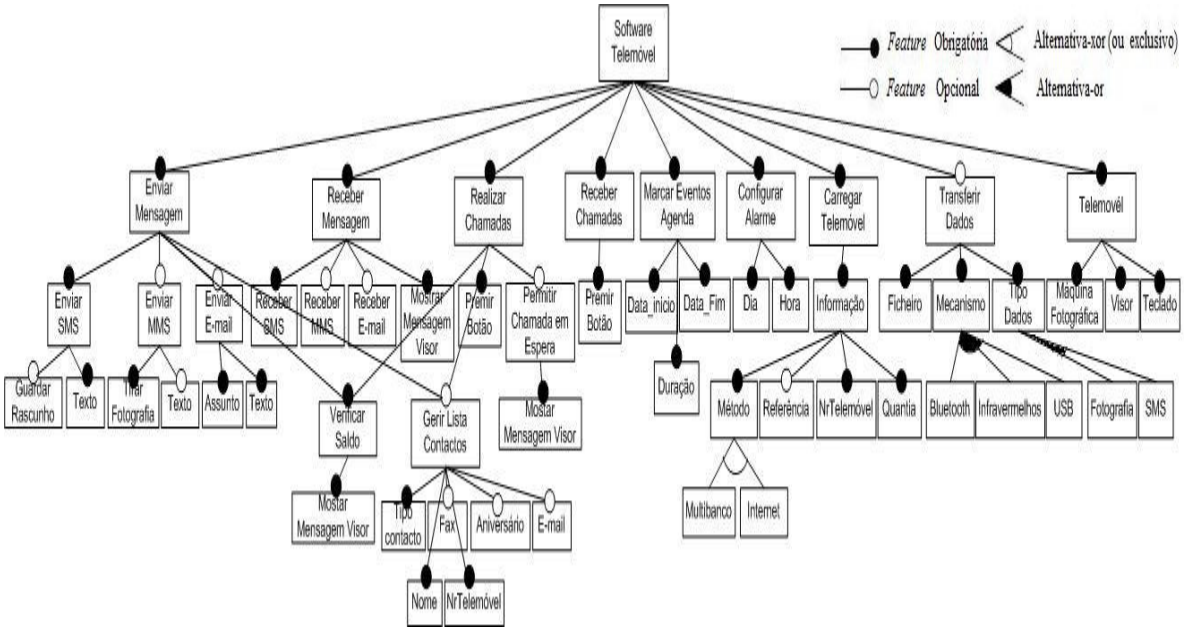

Figura 5.10 Terceira versão do modelo de *features*

> **H.9 (Identificação de relacionamentos de "*requires*" entre *features*):** Quando uma *feature* necessita de outra para satisfazer um requisito, sendo as duas *features* independentes uma da outra, então utiliza-se um relacionamento de "*requires*" entre as duas *features*. Por exemplo, a *feature* "Mensagem Visor" requer a *feature* "Visor", pois como se verifica no requisito R18, ao receber dados irá ser apresentada uma mensagem no visor do telemóvel, pelo que terá que haver um relacionamento de "*requires*" da *feature* "Mensagem Visor" com a *feature* "Visor".

A *sub-feature* "Fotografia" de "Enviar MMS" requer a *feature* "Máquina Fotográfica", através do requisito R13, e segundo a heurística H.9.

A notação utilizada para representar as ligações de "*requires*" é a utilizada em linhas de produtos de software, como foi referida na secção 3.5, ou seja, as *features* requeridas encontram-se assinaladas com uma seta a tracejado.

Na Figura 5.11 encontra-se ilustrada a última versão do modelo de *features*. O modelo apresenta referências a *features*, ou seja, uma representação onde se indica que uma *feature* requer outra, obtidas através da heurística H.9.

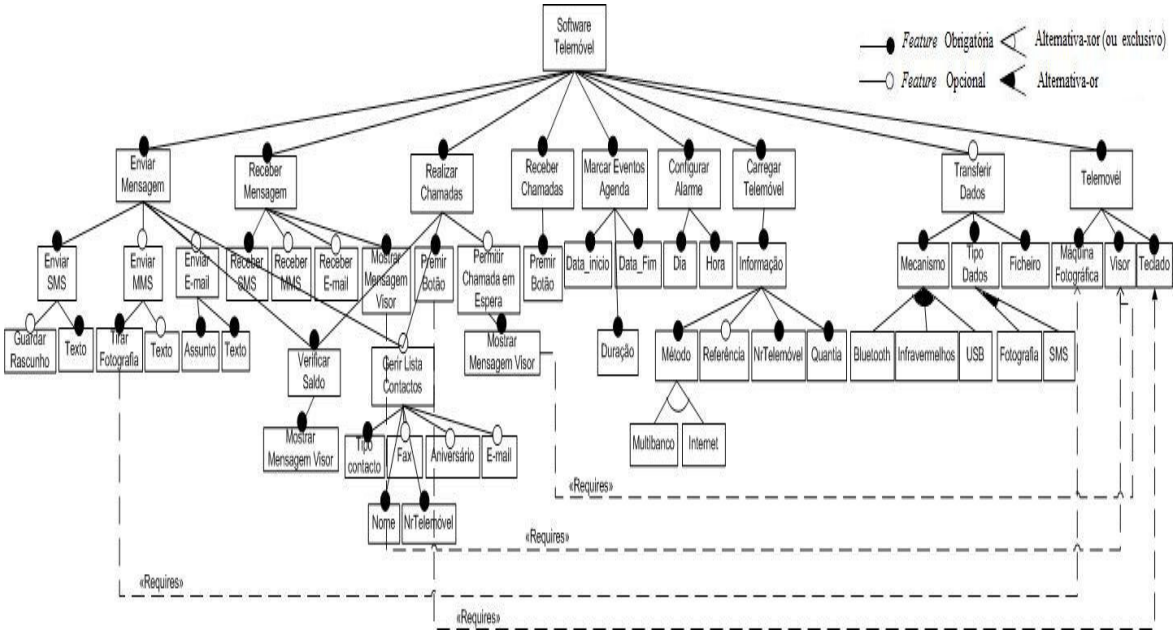

Figura 5.11 Quarta versão do modelo de *features*

C4. Validação do modelo de *features*. A validação é feita pelos *stakeholders* que irão confirmar ou rectificar as várias opções do modelo de *features*.

5.2.2 Processo a Nível de Engenharia de Aplicação

A nível de Engenharia de Aplicação, como ilustrado na Figura 5.2, o processo baseia-se na engenharia de linhas de produtos de software para a configuração de produtos.

Tanto o modelo de *features* como os modelos de Theme serão configurados para um produto específico, para que possam guardar informação que irá complementar o modelo de *features* para uma aplicação específica. Primeiramente é configurado o modelo de *features* e só posteriormente os modelos Theme. Apenas serão apresentados os modelos Theme que sofrem alterações para uma aplicação específica.

Por exemplo, para uma aplicação pretende-se: ter a possibilidade de enviar e receber SMS, MMS e E-mail, sendo que as MMS são enviadas apenas com fotografia; carregar o telemóvel através da internet, não sendo necessário referência; transferir fotografias e

SMS por bluetooth; realizar chamadas; receber chamadas; configurar o alarme; e marcar eventos na agenda.

O modelo de *features*, o documento de requisitos e a lista de entidades ficarão com as opções que aparecem na Figura 5.12, Tabela 5.2. e Figura 5.13, respectivamente.

I. Configurar os modelos de *features* para uma aplicação específica

A Figura 5.12 ilustra a configuração para o modelo de *features* do sistema.

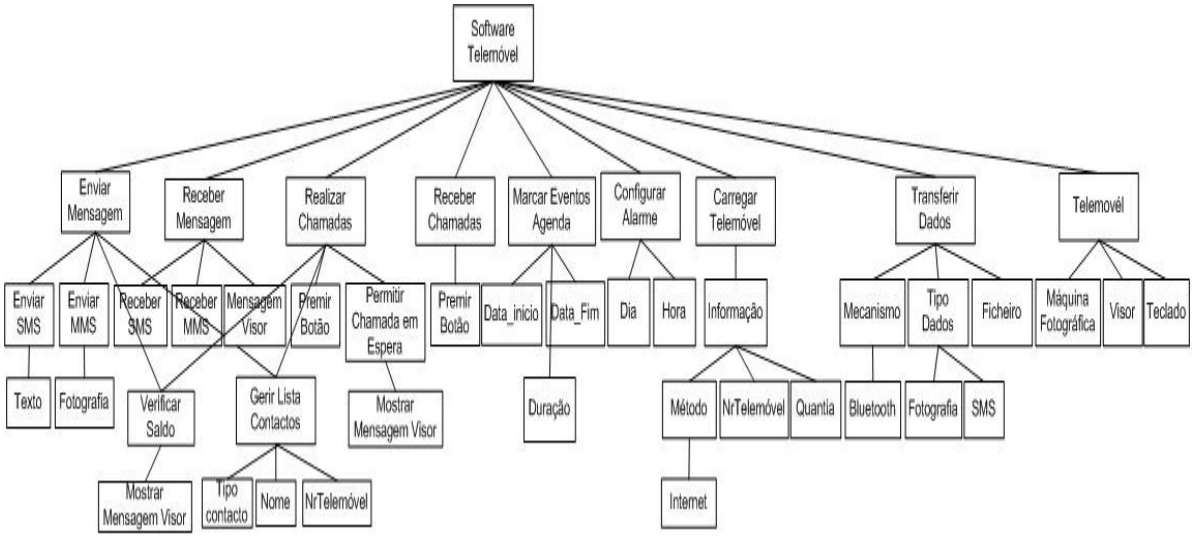

Figura 5.12 Modelo de *features* para a aplicação

II. Configurar os modelos de Theme para uma aplicação específica

Na lista de requisitos elimina-se o recurso "Multibanco", assim como nos diferentes mecanismos de transferência de dados, apenas existe bluetooth, as MMS são enviadas apenas com a fotografia, não escrevendo texto associado à mesma (Tabela 5.2).

Tabela 5.2 Requisitos para a aplicação específica

R No.	Descrição dos Requisitos
R1	Os **componentes** de **software** do telemóvel suportam as seguintes funcionalidades: **realizar chamadas**, **receber chamadas**, **chamadas em espera**, **gerir lista de contactos**, **marcar eventos** na **agenda**, **tirar fotografias**, **enviar SMS** (informação), **receber SMS** (informação), **enviar MMS** (informação), **receber MMS** (informação), **configurar alarme** e **transferir dados**.
R2	De entre os **componentes** de **software** existentes no telemóvel o utilizador pode seleccionar um item para começar.
R3	O utilizador pode manter a **lista de contactos** actualizada introduzindo novos contactos na aplicação, sendo que estes têm que conter o nome, número de

	telemóvel, fax, e-mail, qual o tipo de contacto e a data de aniversário.
R4	Para **realizar** uma **chamada** o utilizador deve procurar o contacto do destinatário, na **lista de contactos**, e **premir** o **botão** de chamada.
R5	Para **realizar** uma **chamada** é preciso **verificar** se há **saldo** no telemóvel.
R6	Se não há saldo no telemóvel é **mostrada** uma mensagem ”Sem saldo”.
R7	Se há saldo no telemóvel **a chamada é realizada.**
R8	Quando o utilizador está a **realizar** uma **chamada**, a sua **chamada** pode ficar **em espera**, sendo mostrado no visor uma mensagem “Chamada em espera”.
R9	O utilizador utiliza a **agenda** para marcar eventos importantes, indicando a data e a duração do mesmo.
R10	O utilizador ao **receber** uma **chamada**, através do teclado, clica no botão pretendido.
R11	Se pretende atender a chamada, prime o botão verde de chamada.
R12	Se não pretende atender a chamada então o utilizador prime o botão vermelho “Rejeitar Chamada”.
R13	O utilizador, utilizando a máquina fotográfica do telemóvel, **tira** uma **fotografia** e **envia-a** por **MMS** a outro utilizador.
R14	Para **enviar** uma **MMS**, o utilizador terá que anexar a fotografia à mensagem e procurar o contacto na **lista** de **contactos.**
R15	Para **enviar MMS** é preciso **verificar** se há **saldo** no telemóvel.
R16	Se não há saldo no telemóvel é **mostrada** uma mensagem ”Sem saldo”.
R17	Se há saldo no telemóvel **mostra** uma mensagem “Mensagem multimédia enviada”.
R18	Ao **receber** uma **MMS** irá ser apresentada uma mensagem “Nova mensagem de multimédia Recebida” no visor do telemóvel.
R19	Para ler a nova **MMS recebida** o utilizador **prime** o **botão** “Abrir mensagem”.
R20	Para **enviar** uma **SMS** o utilizador terá que escrever a mensagem e procurar o contacto, na **lista** de **contactos**, podendo a mesma ser enviada a um ou mais destinatários.
R21	A **mensagem enviada** é guardada nos rascunhos.
R22	Para **enviar SMS** é preciso **verificar** se há **saldo** no telemóvel.
R23	Se não há saldo no telemóvel é **mostrada** uma mensagem ”Sem saldo”.
R24	Se há saldo no telemóvel a mensagem é enviada e **mostra** uma mensagem “Mensagem enviada”
R25	Ao **receber** uma **SMS** irá ser apresentada uma mensagem “Nova SMS Recebida” no visor do telemóvel.
R26	Para ler a nova **SMS recebida** o utilizador **prime** o **botão** “Abrir mensagem”.
R33	O utilizador **configura** o **alarme** tendo a opção de activá-lo todos os dias àquela hora e quantas repetições desejar.
R34	É permitido ao utilizador fazer **transferência de dados** entre telemóveis por bluetooth.
R35	Para efectuar a **transferência de dados** o utilizador tem que activar o

	mecanismo de transferência, seleccionar o tipo de dados a enviar, fotografia ou SMS, e seleccionar o ficheiro.
R36	O utilizador tem que efectuar o **carregamento** do **telemóvel** para utilizar o mesmo, tendo que fornecer o seu número de telemóvel e a quantia.
R37	O **carregamento** do **telemóvel** é efectuado através do site do banco na internet.

Para a aplicação apenas se eliminam as entidades "Multibanco", "Infravermelhos" e "USB", assim como os temas "Enviar e-mail" e "Receber e-mail", como se verifica na Figura 5.13.

Bluetooth	Máquina Fotográfica	Telemóvel
Internet	Teclado	Visor

Figura 5.13 Lista de Entidades para a aplicação

Nesta secção foi apresentado o processo para a engenharia de domínio e de aplicação, adaptado para a abordagem Theme-SPL. Na próxima secção é apresentado o processo para ambas as engenharias, domínio e aplicação, para estender a abordagem Theme com conceitos LPS, de modo a usar desenvolvimento orientado a modelos.

5.3 Estender a abordagem Theme com conceitos de LPS para usar Desenvolvimento Orientado a Modelos

Tal como na secção 5.2, o domínio da linha de produtos pode ser definido com as várias opções e, posteriormente irá configurar-se os modelos para uma aplicação concreta.

5.3.1 Processo a Nível de Engenharia de Domínio

Iremos proceder à adaptação da abordagem Theme para Linhas de Produtos de Software para suportar o desenvolvimento orientado a modelos se assim for desejado. O processo utilizado é essencialmente o mesmo que o apresentado na secção 5.2. A diferença entre o processo utilizado na secção 5.2 e o que irá ser utilizado nesta secção reside na fase A.5, uma vez que é aqui que os modelos Theme são estendidos para conter relações específicas de LPS. Em geral, vamos obter modelos Theme estendidos antes de se gerar o modelo de *features* através de regras de transformação (descritas na secção 8.6). Para tal,

são redefinidas o conjunto de heurísticas, que substituem as descritas na secção 5.2, para adaptar a abordagem. As heurísticas definidas são:

G.1 Identificar a raiz: Encontrar um requisito que diz em que o sistema consiste. Modelar as partes (possibilidades) que compõe a raiz, *RootTheme*, e cada componente dará origem a um tema. O tipo de ligação está explicado nas heurísticas G.2 e G.3.

G.2 Identificar opcionalidade: Se num requisito aparecem descrições que indiquem opcionalidade, como por exemplo "quando X pode, ou não, executar Y" obtêm-se os temas X e Y, sendo que Y será opcional e é representado por uma ligação com o estereótipo «alternative», que indica que quando o tema X é executado o tema Y não tem que ser necessariamente executado (opcional). Como o caso do subtema "Inserir Referência" relacionado com o tema "Carregar Telemóvel".

G.3 Identificar obrigatoriedade: Se num requisito aparecem opções que indiquem obrigatoriedade, como por exemplo "quando X tem que ter (existir) Y", "quando X deve existir Y", "X obriga Y" obtém-se que o tema Y é obrigatório e é representado por uma ligação com o estereótipo «obligatory», que indica que quando o tema X é executado o tema Y tem que ser executado obrigatoriamente.

G.4 Identificar composição de temas: Através dos requisitos relacionados com um tema identificam-se quais os temas que compõe este, sendo a relação apresentada por uma ligação com o estereótipo «part_of». Pelo que, na visão de relação de temas, o tema "Configurar Alarme" fica relacionado com "Definir Dia", "Definir Hora" e "Definir Repetições".

G.5 Identificar temas que podem ser agrupados: Se existem dois ou mais temas do mesmo tipo, então é criado um novo tema com uma agregação para os temas que lhe deram origem. Os temas "Enviar SMS", "Enviar MMS" e "Enviar E-mail" são relativos ao envio de informação, pelo que é criado um novo tema "Enviar Mensagem" cujos subtemas serão os que lhe deram origem.

G.6 Identificar alternativas XOR: Se um tema se encontra relacionado com um requisito, em que aparece "apenas pode seleccionar uma" de entre várias alternativas, então esse tema será um *ThemeGroup* e os temas que representam as alternativas serão *GroupedThemes*, que se encontram ligados ao *ThemeGroup* com o estereótipo «alternative_xor». Por exemplo, o *ThemeGroup* "Escolher Método" terá como *GroupedThemes* "Multibanco" e "Internet".

G.7 Identificar alternativas OR: Se num requisito, relacionado com um tema, aparecem várias alternativas em que se "selecciona pelo menos uma entre várias", então o tema será um *ThemeGroup*, que contém um número mínimo e máximo de alternativas, e as alternativas serão um *GroupedTheme* cuja ligação apresenta o estereótipo «alternative_or». Por exemplo, o *ThemeGroup* "Escolher Mecanismo" permite escolher entre uma e três *GroupedThemes* ("Bluetooth", "Infravermelhos" e "USB").

G.8 Identificar relações entre temas e entidades: Se num requisito está presente um tema que dá uma (s) entidade (s) então essa (s) entidade (s) vai ser parte daquele tema.

Definir a Visão de Relação de Temas. Tendo em conta as heurísticas de G.1 a G.8 é então elaborada a visão de relação de temas estendida com conceitos de LPS, como mostrado na Figura 5.14.

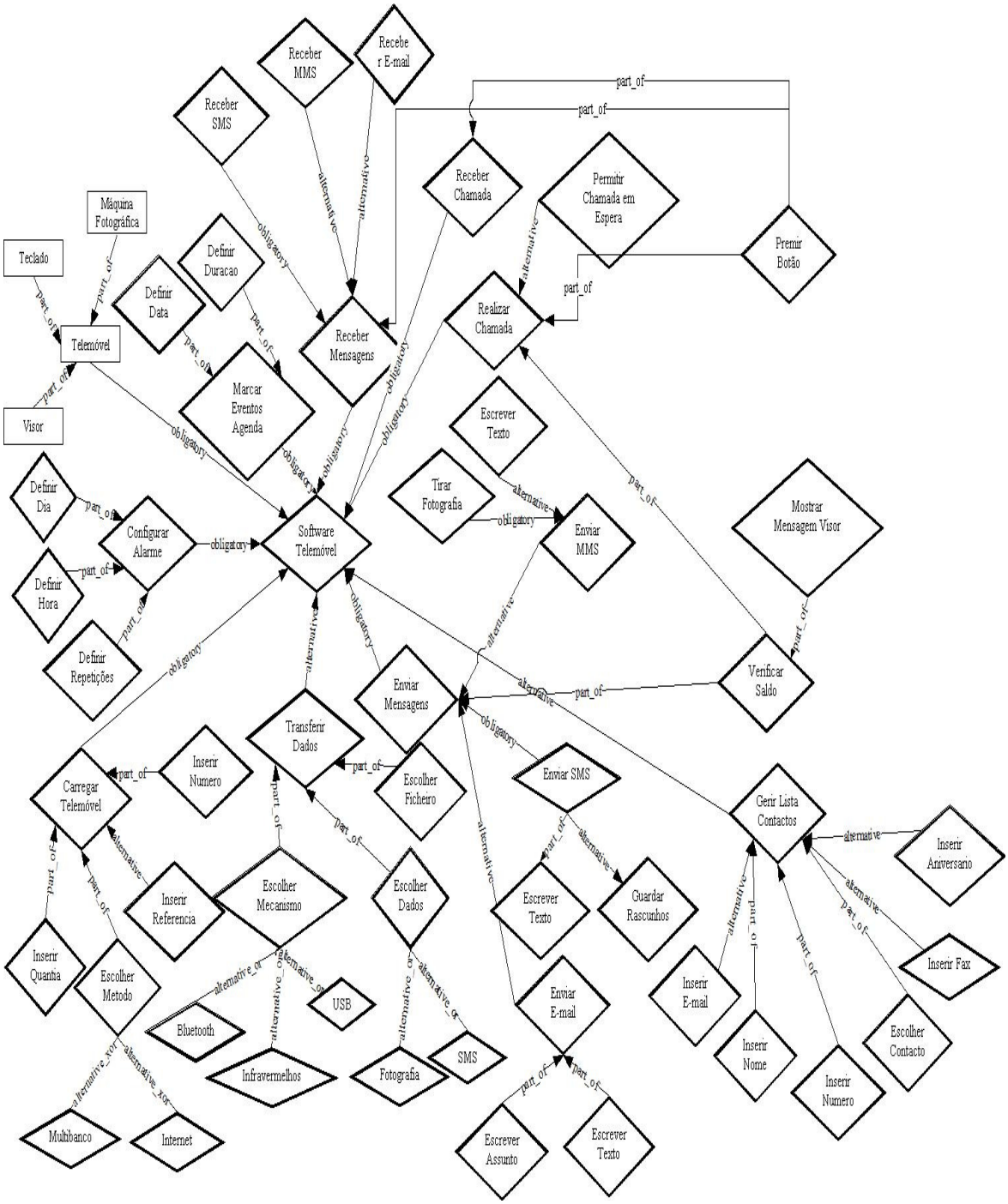

Figura 5.14 Visão de Relação de Temas após aplicação das heurísticas G.1 a G.8

Validar a visão de relação de temas com os *stakeholders*. A validação é feita através dos *stakeholders*.

B. Construção do modelo de *features.* Após a elaboração da visão de relação de temas, é efectuado o seu mapeamento para o modelo de *features*. O mapeamento é efectuado tendo em conta as seguintes heurísticas:

HM.1 A *RootFeature* é extraída da *RootTheme*. Por exemplo, a raiz "Software Telemóveis" dará origem à raiz do modelo de *features* com o mesmo nome.

HM.2 As *features* são extraídas dos temas e entidades, pelo que a cada tema ou entidade corresponderá uma *feature*. Por exemplo, no tema "Gerir Lista Contactos" a *feature* será "Gerir Lista Contactos" e na entidade "Máquina Fotográfica" a *feature* será "Máquina Fotográfica".

HM.3 As *FeatureGroup* são extraídas dos *ThemeGroup*, pelo que a cada *ThemeGroup* corresponde uma *FeatureGroup*, e o número máximo e mínimo de alternativas escolhidas no *ThemeGroup* serão representadas como cardinalidade na *FeatureGroup*. Por exemplo, o *ThemeGroup* "Escolher Mecanismo" será uma *FeatureGroup* "Escolher Mecanismo" e apresentará uma cardinalidade <1..3>.

HM.4 As *GroupedFeatures* são extraídas dos *GroupedThemes*, pelo que a cada *GroupedTheme* corresponde um *GroupedFeatures*. Por exemplo, o *GroupedTheme* "Internet" será o *GroupedFeature* "Internet".

HM.5 Se existe uma ligação de «part_of» ou «obligatory» entre dois temas, então dará origem a uma *feature mandatory*.

HM.6 Se existe uma ligação de «alternative» entre dois temas, então dará origem a uma *feature optional*. A ligação «alternative» entre os temas "Inserir Referência" e "Carregar Telemóvel" irá dar origem à ligação opcional entre as *features* "Inserir Referência" e "Carregar Telemóvel".

HM.7 Se existe uma ligação com o estereótipo «alternative_or» entre um *GroupedTheme* e um *ThemeGroup*, irá dar origem a uma alternativa OR.

HM.8 Se existe uma ligação com o estereótipo «alternative_xor» entre um *GroupedTheme* e um *ThemeGroup*, irá dar origem a uma alternativa XOR (ou exclusivo).

Aplicando as heurísticas de mapeamento de HM.1 a HM.8 à visão de relação de temas da Figura 5.14 obtém-se a primeira versão do modelo de *features.* O modelo obtido é idêntico ao modelo ilustrado na Figura 5.8.

Validar modelo de *features* inicial. A validação é feita pelos *stakeholders* que irão efectuar uma revisão pormenorizada sobre as várias opções do modelo de *features*, e se correspondem com as especificações pretendidas.

C. Refinamento do modelo de *features.* Após desenhar o modelo de *features* inicial, o mesmo é refinado através da visão transversal, uma vez que esta oferece informação adicional para completar o modelo.

Identificar temas aspectuais. Os temas aspectuais são identificados através da heurística seguinte:

> **G.9 Identificar temas aspectuais**: Os temas aspectuais são identificados tal como na abordagem Theme, apresentando o mesmo tipo de relação do tema aspectual para o tema base com o estereótipo «crosscuting», sendo definido qual o tipo de ligação, obrigatória ou alternativa, que o tema aspectual tem com o tema base. Como verificado na fase C1. da secção 5.2, os temas aspectuais serão os temas "Gerir Lista Contactos" e "Verificar Saldo".

Construir a Visão Transversal. De acordo com a heurística G.9 e G.10, definida em seguida, é elaborada a visão transversal como ilustrado na Figura 5.15.

> **G.10 Identificar relações entre temas independentes**: Se num requisito estão presentes temas independentes mas se um tema precisa de outro em que aparece "X utilizando Y" ou "X através de Y" então existe uma ligação entre os dois. Essa ligação é representada por uma seta a tracejado do tema X para o tema Y com o estereótipo «requires». Por exemplo, o tema "Tirar Fotografia" requer a entidade "Máquina Fotográfica", pois como se verifica no requisito "utilizando a máquina fotográfica do telemóvel, tira uma fotografia", terá que haver um relacionamento de «requires» do tema "Tirar Fotografia" para a entidade "Máquina Fotográfica".

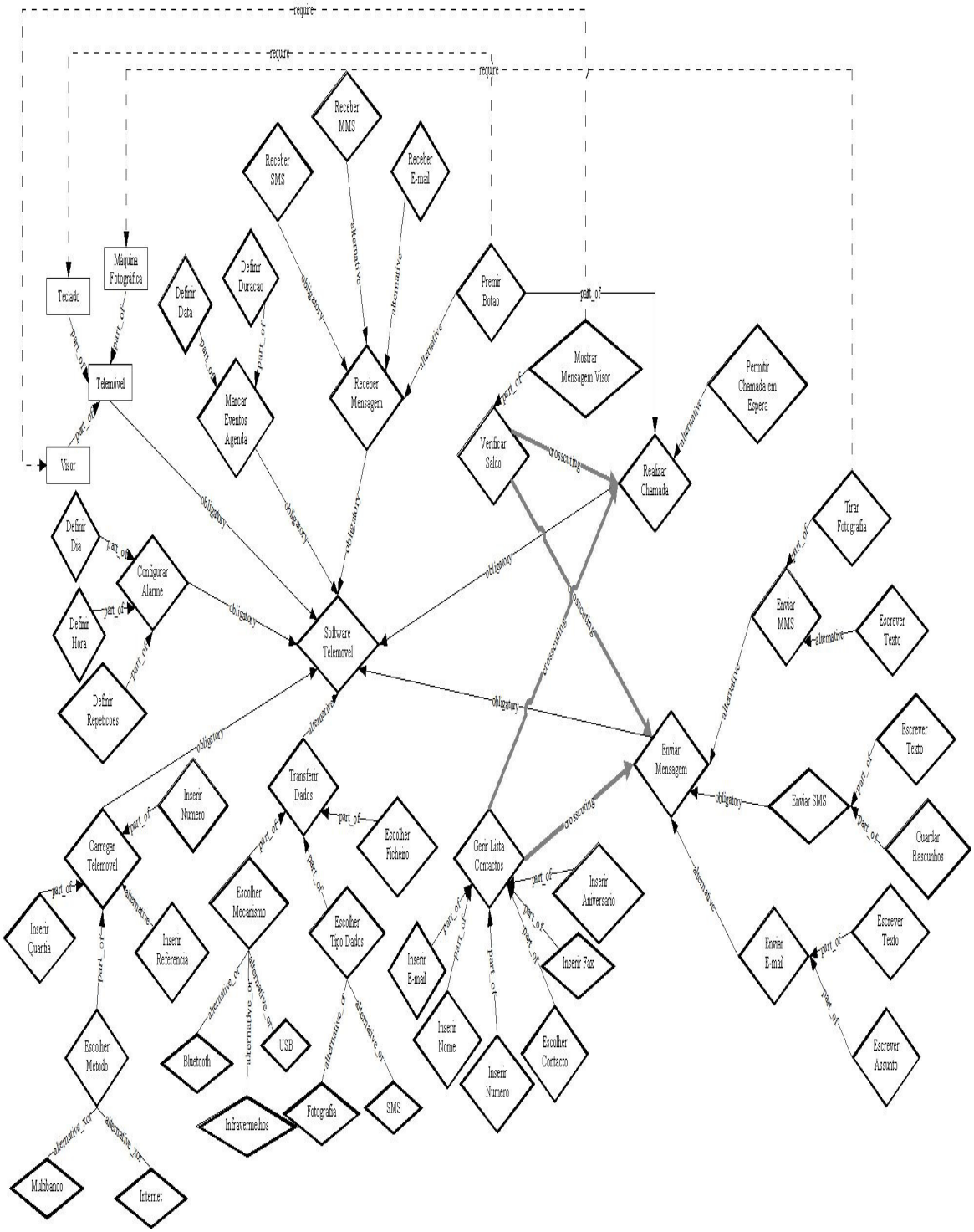

Figura 5.15 Visão Transversal

Refinar o modelo de *features* com base nos temas aspectuais. Após elaborada a visão transversal é efectuado o mapeamento de Theme para o modelo de *features,* tendo em conta as seguintes heurísticas de mapeamento:

HM.9 Numa ligação de «crosscuting» se o tipo da ligação é "*obligatory*" dará origem a uma *feature mandatory*. Caso o tipo ligação seja "*alternative*" dará origem a uma *feature optional*.

HM.10 As ligações com o estereótipo «requires» irão dar origem a ligações de "*requires*" nos modelos de *features*. E as ligações com estereótipo «*excludes*» darão origem a ligações "*excludes*" no modelo de *features*.

Aplicando as heurísticas de mapeamento HM.9 e HM.10 à visão transversal, apresentada na Figura 5.15, obtém-se um modelo de *features* idêntico ao modelo ilustrado na Figura 5.11.

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

5.3.2 Processo a Nível de Engenharia de Aplicação

Nesta secção apresenta-se o modelo de *features*, a visão de relação de temas (Figura 5.16) e a visão transversal (Figura 5.17) para uma configuração de software de telemóveis, sendo utilizada a mesma configuração descrita na secção 5.2.2. Não é apresentada a lista de requisitos uma vez que fica semelhante à apresentada na Tabela 5.2.

I. Configurar os modelos de *features* para uma aplicação específica

O modelo de *features* do produto resultante é semelhante ao apresentado na Figura 5.12.

II. Configurar os modelos de Theme para uma aplicação específica

Em ambas as visões se elimina a entidade "Multibanco", nos diferentes mecanismos de transferência de dados apenas existe bluetooth, as MMS são enviadas apenas com a fotografia não escrevendo texto associado à mesma, assim como os temas "Enviar e-mail" e "Receber e-mail" serão eliminados. A Figura 5.16 mostra a Visão de Relação de Temas para a aplicação definida e a Figura 5.17 mostra a Visão Transversal para a mesma aplicação.

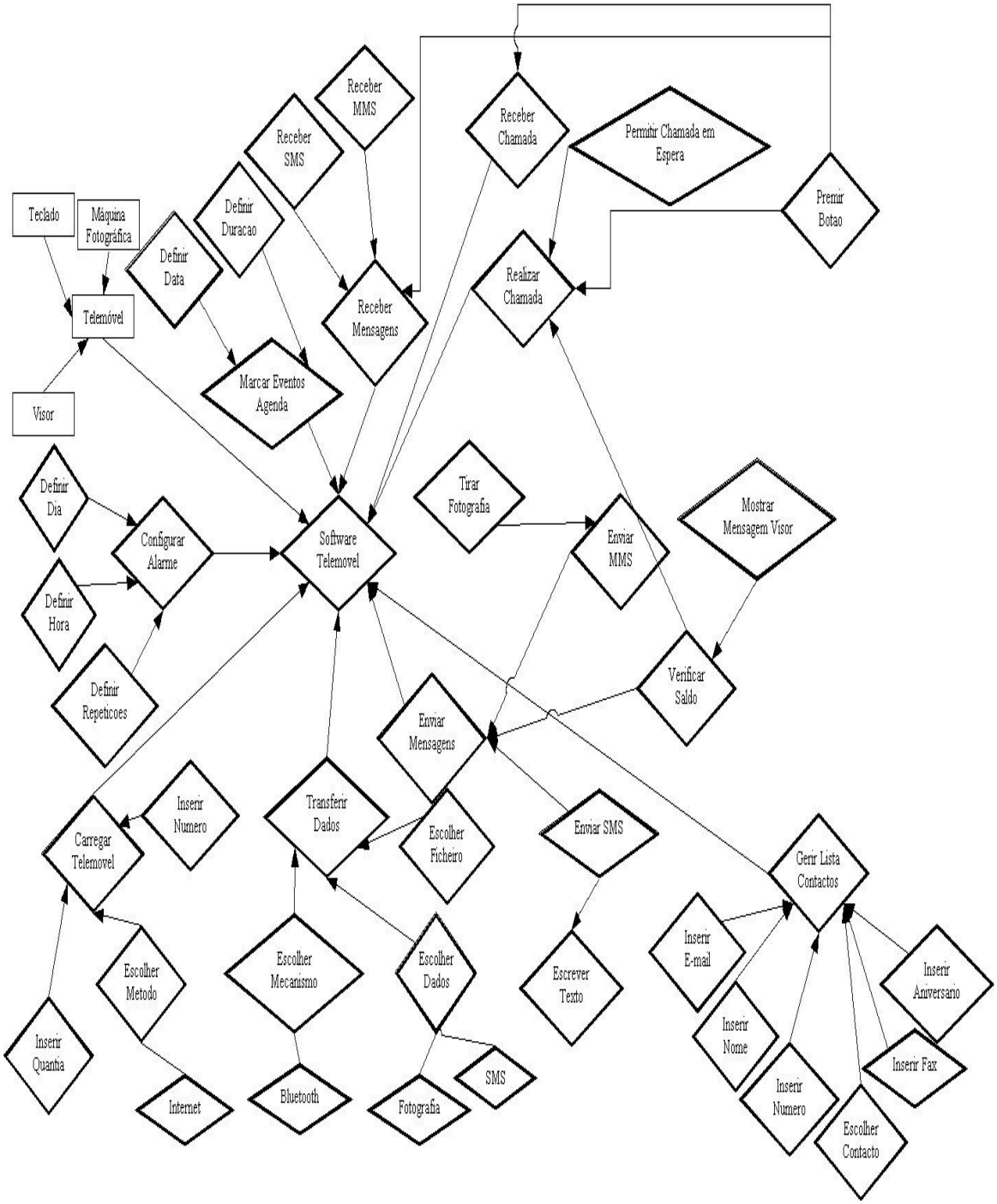

Figura 5.16 Visão de Relação de Temas para a aplicação

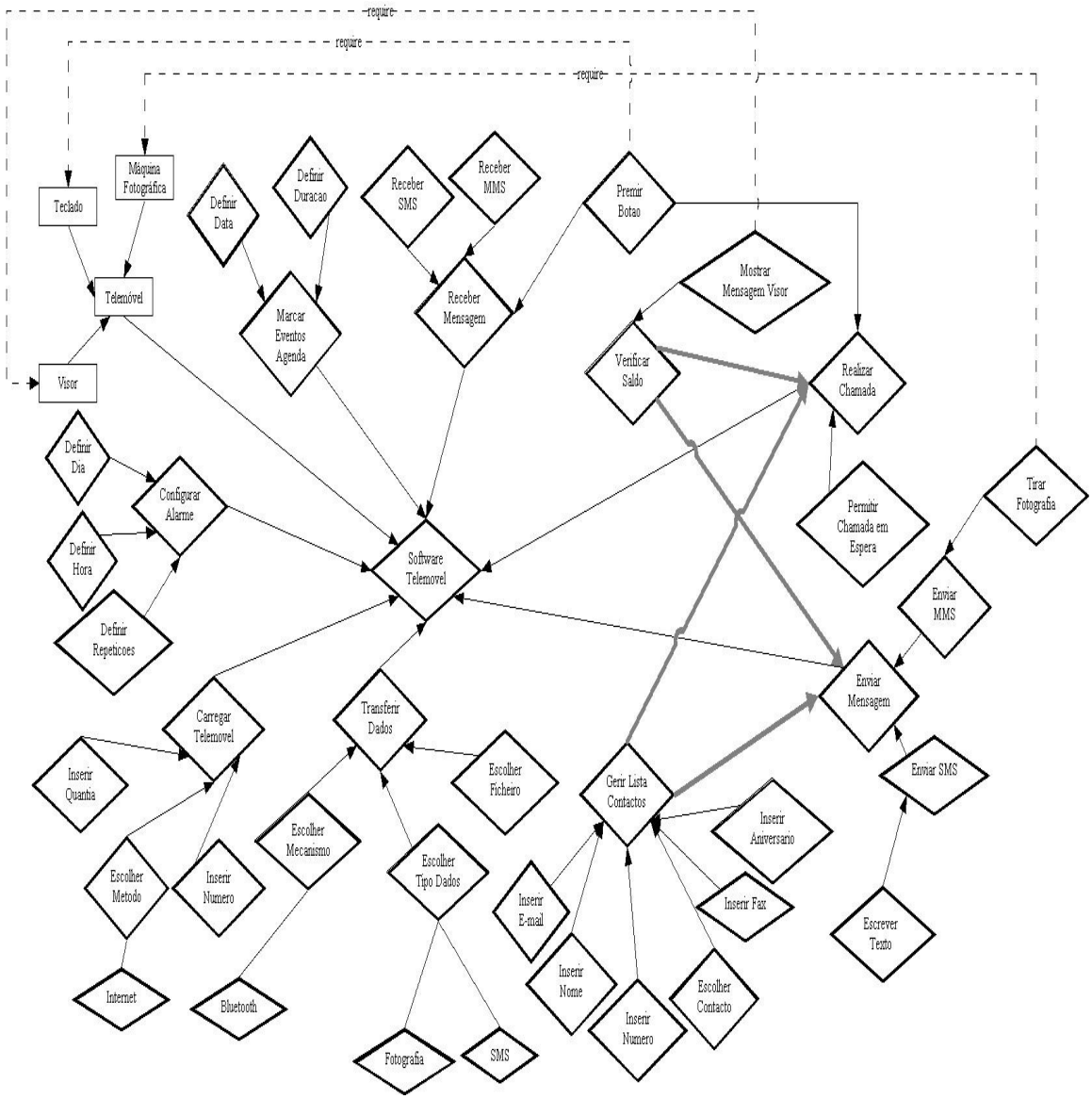

Figura 5.17 Visão Transversal para a aplicação

Nesta secção foi apresentado o processo para a engenharia de domínio e de aplicação, para estender os modelos Theme com conceitos LPS, de modo a se poder obter modelos de *features* através de modelos Theme usando desenvolvimento orientado a modelos.

5.4 Sumário

Neste capítulo apresentou-se a abordagem Theme-SPL, que é composta, tal como a ELPS, por dois processos: Engenharia de Domínio e Engenharia de Aplicação. Para melhor se demonstrar os processos, utilizou-se o exemplo de desenvolvimento de componentes de software de telemóveis.

A engenharia de domínio encontra-se dividida em três fases, tal como mostra a Figura 5.1, sendo elas: Elicitação e análise de temas e requisitos, Construção do modelo de *features* e Refinamento do modelo de *features*. Por outro lado, a nível da engenharia de aplicação, Figura 5.2, o processo está dividido em duas fases: Configurar os modelos de *features* para uma aplicação específica e Configurar os modelos de Theme para uma aplicação específica.

O processo foi aplicado de dois modos distintos: analisar a abordagem Theme para elaborar modelos de *features* através de modelos Theme, e estender a abordagem Theme com conceitos de LPS para usar o desenvolvimento orientado a modelos, de modo a obter modelos de *features*, de forma sistemática, através de modelos Theme.

No Capítulo 6 será apresentado um caso de estudo, *Smart Home*, onde será aplicada a abordagem Theme-SPL estendida com conceitos de LPS, de modo a mostrar como se utiliza e funciona o processo e para validar as várias heurísticas apresentadas neste capítulo de modo a se obter o modelo de *features*.

6. Caso de Estudo e Comparação com Outras Abordagens

6.1 Enunciado do Problema – *Smart Home*

O caso de estudo *Smart Home* (AMPLE, 2009) investiga uma casa inteligente, a sua configuração, e os serviços necessários para a sua automatização.

Nas casas europeias, normalmente encontra-se uma grande variedade de dispositivos eléctricos e electrónicos: equipamentos como luzes, termóstatos, persianas eléctricas, sensores de fogo e de vidros partidos, produtos como máquinas de lavar, secadores e máquinas de lavar louça, equipamentos de entretenimento como televisão, rádio e dispositivos para reproduzir música ou filmes, e dispositivos de comunicação como telefones (inteligentes) e computadores. Sensores são dispositivos para medir as propriedades físicas do meio ambiente e torná-los disponíveis para a casa inteligente (*Smart Home*). Actuadores activam dispositivos cujo estado pode ser monitorizado e alterado.

O objectivo do projecto *Smart Home* é ligar em rede os dispositivos e permitir aos habitantes de uma casa acompanhá-la e controlá-la a partir de várias interfaces utilizador. Os habitantes devem ser capazes de monitorar facilmente, configurar e controlar os dispositivos instalados na casa utilizando o controlo remoto dentro da casa. O estado de todos os sensores deve estar visível em algum monitor dentro da casa.

Deve ser possível equipar facilmente uma casa com todos os sensores e actuadores adquiridos por um cliente. Os tipos de dispositivos que devem ser suportados são, no mínimo: Luzes e interruptores; Aberturas de janelas, sensores de janelas, cortinas e sensores de vidros partidos; Ar condicionado, aquecedores e termóstatos; Sensores de portas e aberturas de portas; e Sensores de Fogo.

O sistema *Smart Home* deve oferecer funcionalidades de mais alto nível em que vários sensores e actuadores trabalham juntos. Essas funções e capacidades são descritas em detalhe como se segue:

- Controle de Energia: As luzes podem ser ligadas e desligadas em determinados momentos do dia. As cortinas podem ser abertas ou fechadas também, quando conveniente. O morador da casa pode configurar o ar condicionado (temperatura) para ser ligado e desligado na altura desejada do dia (por exemplo, ligar o ar condicionado antes de chegar a casa ou desligá-lo depois de sair da casa).

- Sistema de controlo do clima: Aquecimento, termóstatos, cortinas e janelas devem ser conjugados para manter a temperatura pretendida nos quartos da casa. As janelas automáticas abrem-se se a temperatura num quarto ultrapassa um certo limite, e fecham-se se a temperatura desce um certo limite. O sensor de temperatura exterior adiciona a capacidade de medir a temperatura exterior e tem-no em conta no controlo de temperatura inteligente.

- Sistema de Segurança: Sensores de vidros partidos, sensores de portas e detectores de movimento devem ser usados para detectar se as pessoas que não estão autorizadas a entrar na casa estão a tentar fazê-lo. Se a casa detectar intrusões, deve activar um alarme e a polícia deve ser chamada. A detecção de assaltantes deve dissuadir os assaltantes de invadir a casa, fornecendo diferentes formas de dispositivos de alarme que o cliente pode escolher. Os clientes podem instalar uma sirene, um sino, ou uma luz, ou uma combinação dos três. Além disso, a simulação de luz periodicamente liga as luzes da casa. A finalidade é simular habitantes dentro da casa caso estes estejam de férias. Este recurso deve dissuadir os assaltantes de invadir a casa uma vez que parece ocupada.

- Sistemas de manipulação de fogo e fumo: Detectores de fogo e fumo, sistemas de aspersão, sensores de janelas e de portas e aberturas/fechos, dispositivos de alarme e dispositivos de comunicação devem trabalhar em conjunto para impedir danos humanos principalmente em caso de incêndio e fumo. Além disso, o fogo deve ser extinto e os bombeiros devem ser chamados. O sistema deve ser inteligente o

suficiente para não começar a extinção de incêndio se há apenas pessoas a fumar na casa.

6.1.1 Procedimentos a Nível de Engenharia de Domínio

Como proposto na abordagem Theme-SPL para usar Desenvolvimento Orientado a Modelos, apresentada na secção 5.3, procede-se inicialmente à produção da visão de relação de temas. Como ilustrado na Figura 5.1, secção 5.1, inicialmente é efectuada a elicitação e análise de temas e requisitos, em seguida é elaborado o modelo de *features* e por fim efectua-se o refinamento do modelo. Estas três fases são demonstradas em seguida.

A. Elicitação e análise de temas e requisitos. Tal como na abordagem Theme, inicialmente são analisados os requisitos, identificados os temas e as entidades.

Analisar Requisitos. Como referido anteriormente, o primeiro passo da abordagem é examinar a documentação de requisitos do sistema, Tabela 6.1.

Tabela 6.1 Requisitos para o sistema *Smart Home*

R No.	Descrição dos requisitos
R1	Ao **aderir** ao sistema ***smart home***, este oferece as seguintes funcionalidades: terá que se **configurar** o **sistema** de **comunicação**, terá que **permitir chamadas** de **emergência**, pode **controlar controlo** de **energia**, pode **controlar controlo** de **clima**, pode **controlar controlo** de **segurança**, pode **controlar controlo** de **fumo/fogo**. Para utilizar o mesmo o utilizador terá que **efectuar** o **registo** no **sistema**, e **adquirir** os **sensores** pretendidos. Permite também ao utilizador **alterar** a sua **configuração** no sistema, assim como **registar** um **novo utilizador**.
R2	Os tipos de **sensores** que podem ser adquiridos são: luz, janelas, cortinas, vidros partidos, portas, fumo/fogo, movimento, temperatura e alarme.
R3	Os sensores de temperatura podem ser de temperatura exterior ou interior.
R4	É permitido **configurar** o **sistema** de **comunicação**, por pelo menos um dos seguintes sistemas, telemóvel ou computador, tendo que, em ambos os casos, **efectuar** uma **ligação** ao **controlo** remoto.
R5	É permitido **registar** um **novo utilizador** no sistema, em que este terá que introduzir o nome de utilizador, o PIN, e seguidamente confirmar as opções escolhidas.
R6	Para **efectuar** o **registo** no **sistema**, o utilizador tem que **efectuar** a **autenticação** introduzindo o nome de utilizador e o PIN.
R7	Após introduzir os dados da autenticação estes são **validados**.
R8	Se os dados foram introduzidos correctamente, então o utilizador tem acesso ao sistema.
R9	Se os dados inseridos são inválidos é **mostrada** no visor uma **mensagem** de **erro**, indicando as tentativas restantes.
R10	Para **alterar** a **configuração** o utilizador tem que **efectuar** a **autenticação** no sistema.

R11	Para **alterar** a sua **configuração** terá que introduzir os novos dados, podendo alterar um dos seguintes, ou o PIN ou o nome de utilizador, e confirmar opções escolhidas.
R12	O utilizador pode **controlar** o **controlo** de **energia** podendo o mesmo ser **ajustado** para os actuadores de **AC** utilizando o termóstato, **luzes** utilizando o sensor de luz, ou **cortinas** utilizando o sensor de cortinas.
R13	O utilizador pode **controlar** o **controlo** de **energia** tendo que inserir a data e hora a que pretende activar e desactivar o actuador pretendido, assim como as divisões da casa.
R14	O utilizador pode **controlar** o **controlo** de **clima** sendo que o mesmo pode ser **ajustado** para os actuadores de **AC** utilizando o termóstato, **janelas** utilizando o sensor de janelas, ou **cortinas** utilizando o sensor de cortinas.
R15	Para **controlar** o **controlo** de **clima**, o utilizador, terá que escolher um dos seguintes métodos: activar a opção de regulação da temperatura exterior ou definir a temperatura mínima e máxima utilizando em ambos os casos o sensor de temperatura. Terá também que inserir as divisões da casa em que este deve ser ligado.
R16	Para **controlar** o **controlo** de **segurança** o mesmo pode ser configurado para simular presença e para detectar intrusos.
R17	Para simular presença o utilizador tem que **ajustar** o actuador de **luzes**, definir qual a hora e a data de inicio/fim da simulação, qual a duração e frequência da simulação, utilizando o sensor de luz.
R18	Para detectar intrusos o utilizador pode **ajustar** o actuador de **portas** utilizando o sensor de portas, pode activar as funcionalidades de detecção de quebra de vidros, utilizando o sensor de quebra de vidros, e de detecção de movimento utilizando o sensor de movimento, pode **ajustar** o **alarme** utilizando o sensor de alarme, e pode activar/desactivar o sistema de chamadas de emergência para a polícia.
R19	Ao configurar o alarme, o mesmo pode ser de pelo menos um dos seguintes tipos: sirene, luz ou campainha.
R20	Para **controlar** o **controlo** de **segurança** o utilizador terá que inserir as divisões da casa em que este pode ser activado.
R21	O utilizador pode **controlar** o **controlo** de **fumo/fogo**, sendo que o mesmo pode ser **ajustado** para os actuadores **janelas**, **portas**, que activa/desactiva a opção de portas anti-fogo, **cortinas**, e **alarme** que activa/desactiva o alarme, utilizando os sensores de fumo/fogo e alarme.
R22	Para **controlar** o **controlo** de **fumo/fogo** o utilizador terá inserir as divisões da casa em que este deve ser ligado e pode activar/desactivar as chamadas de emergência para os bombeiros.

Identificar Temas. Percorrendo a lista de requisitos e seleccionando os termos chave identificam-se os potenciais temas. Os mesmos são ilustrados na Figura 6.1.

Aderir *Smart Home*	Ajustar Luzes	Configurar Sistema Comunicação
Adquirir Sensores	Alterar Configuração	Efectuar Autenticação
Ajustar AC	Controlar Alarme	Efectuar Ligação
Ajustar Alarme	Controlar Controlo Clima	Efectuar Registo
Ajustar Portas	Controlar Controlo Energia	Mostrar Mensagem Erro
Ajustar Janelas	Controlar Controlo Fumo/fogo	Permitir Chamada Emergência
Ajustar Cortinas	Controlar Controlo Segurança	Validar Dados

Figura 6.1 Lista de Temas do Smart Home

Identificar Entidades. Foram identificadas, nos requisitos, quinze entidades. A Figura 6.2 ilustra as entidades obtidas.

Cliente	Sensor Luz	Sensor Temperatura Interior
Sensor Alarme	Sensor Movimento	Sensor Vidros Partidos
Sensor Cortinas	Sensor Porta	Temperatura Exterior
Sensor Fumo/Fogo	Sensor Temperatura	Temperatura Interior
Sensor Janelas	Sensor Temperatura Exterior	Termóstato

Figura 6.2 Lista de Entidades do Smart Home

Definir a Visão de Relação de Temas. Obtém-se a Visão de Relação de Temas, de acordo com as heurísticas de G.1 a G.8 definidas na secção 5.3, para estender a abordagem Theme com conceitos LPS, como ilustra a Figura 6.3.

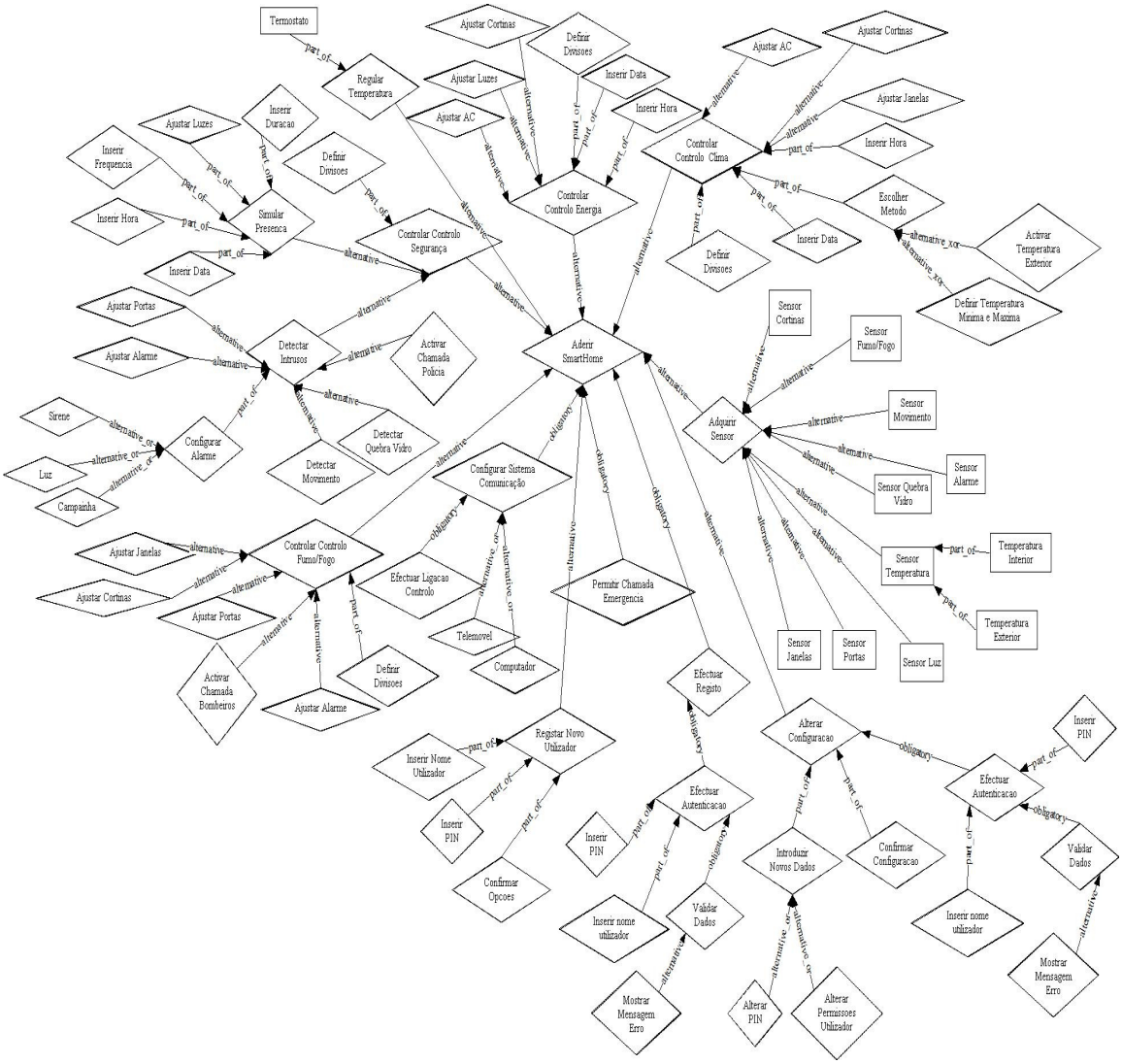

Figura 6.3 Visão de Relação de Temas para o sistema *Smart Home*

Validar temas e requisitos com os *stakeholders*. A validação é feita através dos *stakeholders*.

B. Construção do modelo de *features*. Após obter a visão de relação de temas é feito o seu mapeamento para o modelo de *features*. Considerando as heurísticas de mapeamento de **HM.1** a **HM.8** (secção 5.3), obtém-se o modelo ilustrado na Figura 6.4.

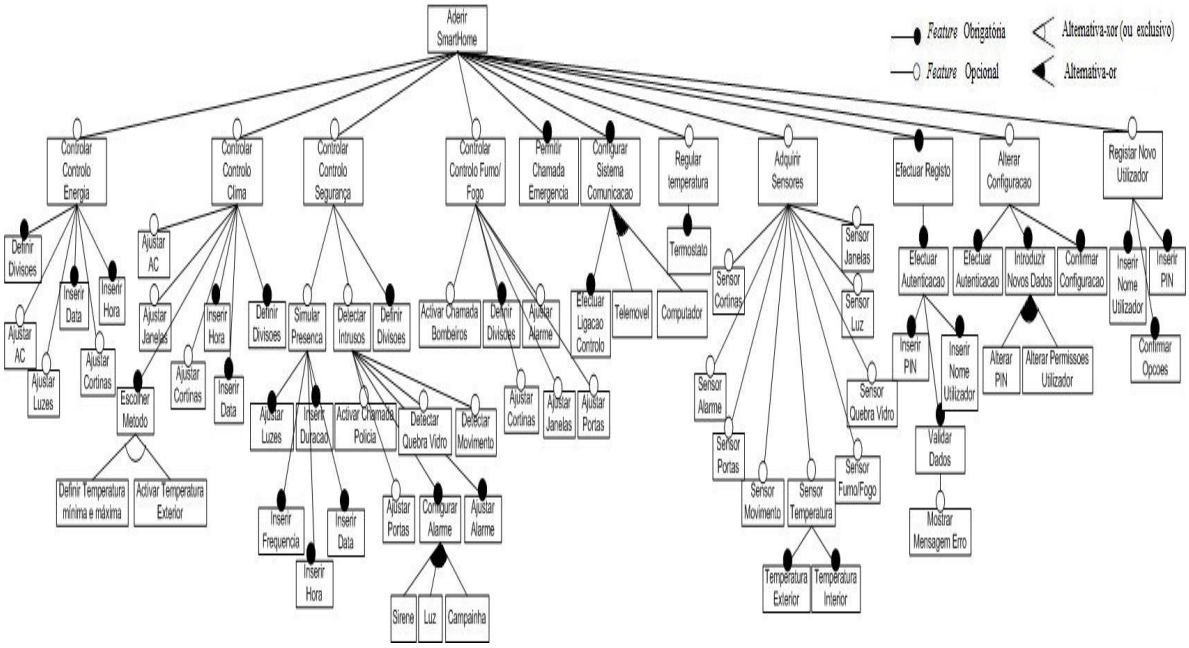

Figura 6.4 Primeira versão do modelo de *features*

Validar modelo de *features* inicial. A validação é feita pelos *stakeholders* que irão efectuar uma revisão pormenorizada sobre as várias opções do modelo de *features*, e se correspondem com as especificações pretendidas.

C. Refinamento do modelo de *features.* Após desenhar o modelo de *features* inicial, o mesmo é refinado através da visão transversal, uma vez que esta oferece informação adicional para completar o modelo.

Identificar temas aspectuais. Os temas aspectuais são identificados, tal como na abordagem Theme, através de um conjunto de questões (secção 2.3.2.3) que podem ser efectuadas olhando para a lista de requisitos. Tendo em conta essas questões e analisando a visão de relação de temas, verifica-se que o requisito R6 é partilhado pelos temas "Efectuar Registo Sistema" e "Efectuar Autenticação", assim como o requisito R10 é partilhado pelos mesmos temas "Efectuar Autenticação" e "Alterar Configuração".

Verifica-se que em ambos os requisitos o tema “Efectuar Autenticação” é uma acção necessária sendo o tema responsável pelos requisitos, logo é um tema aspectual.

O requisito R12 é partilhado pelos temas “Controlar Controlo Energia”, “Ajustar AC”, “Ajustar Luzes” e “Ajustar Cortinas”, sendo os três últimos responsáveis pelo requisito uma vez que para controlar o controlo de energia é necessário que um ou mais dispositivos sejam configurados. Estes três temas, juntamente com os temas “Ajustar Portas”, “Ajustar Janelas” e “Ajustar Alarme” são também temas responsáveis pelos requisitos R14, R17, R18 e R21, pelo que todos estes temas são considerados temas aspectuais.

Construir a Visão Transversal. Com base nos temas aspectuais e aplicando as heurísticas G.9 e G.10 (secção 5.3), obtém-se a Visão Transversal ilustrada na Figura 6.5.

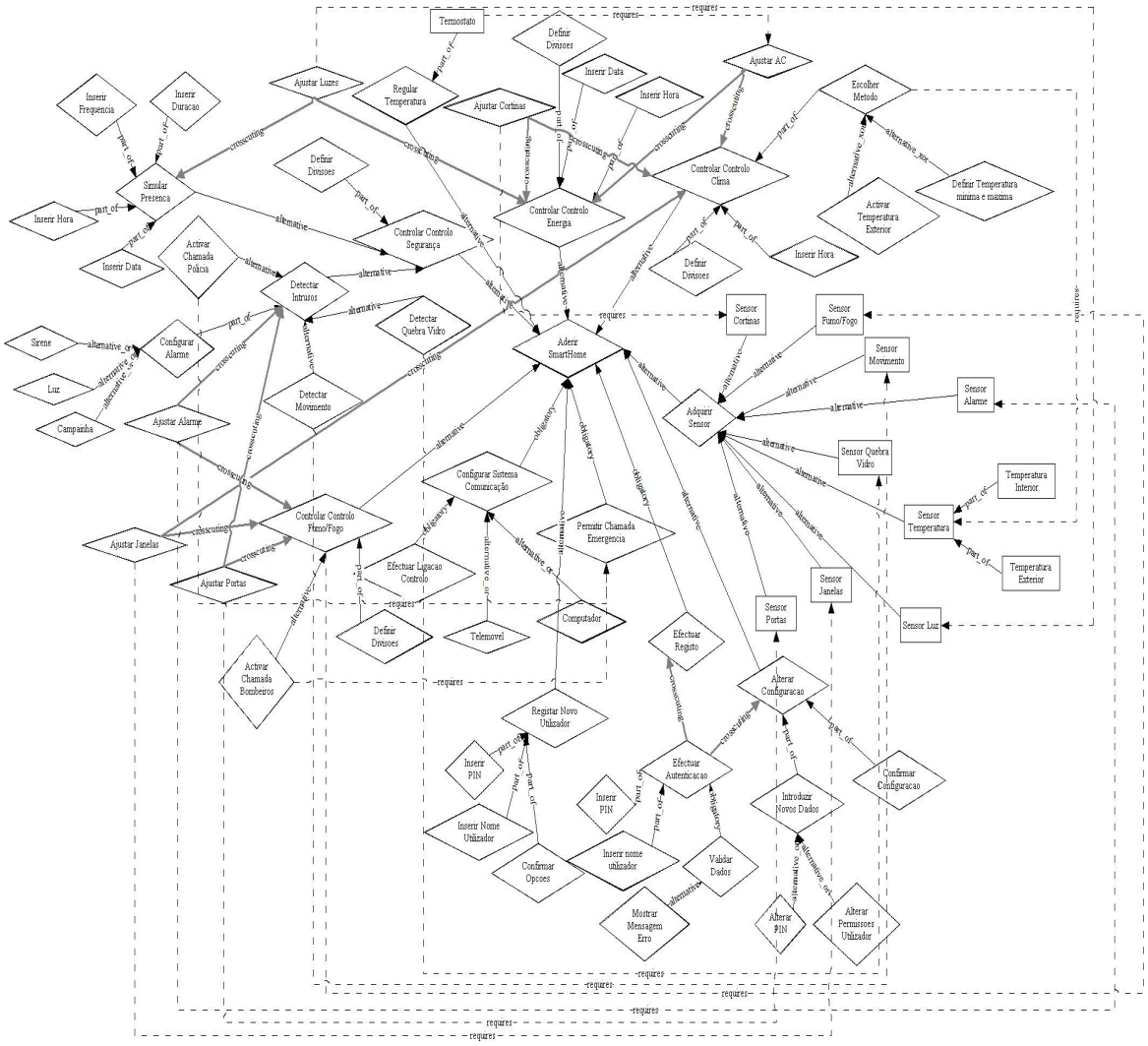

Figura 6.5 Visão Transversal para o sistema *Smart Home*

Refinar o modelo de *features* com base nos temas aspectuais. Em seguida é efectuado o mapeamento da visão transversal para o modelo de *features*. Segundo as heurísticas de mapeamento **HM.9** e **HM.10** (secção 5.3) obtém-se o modelo de *features* apresentado na Figura 6.6.

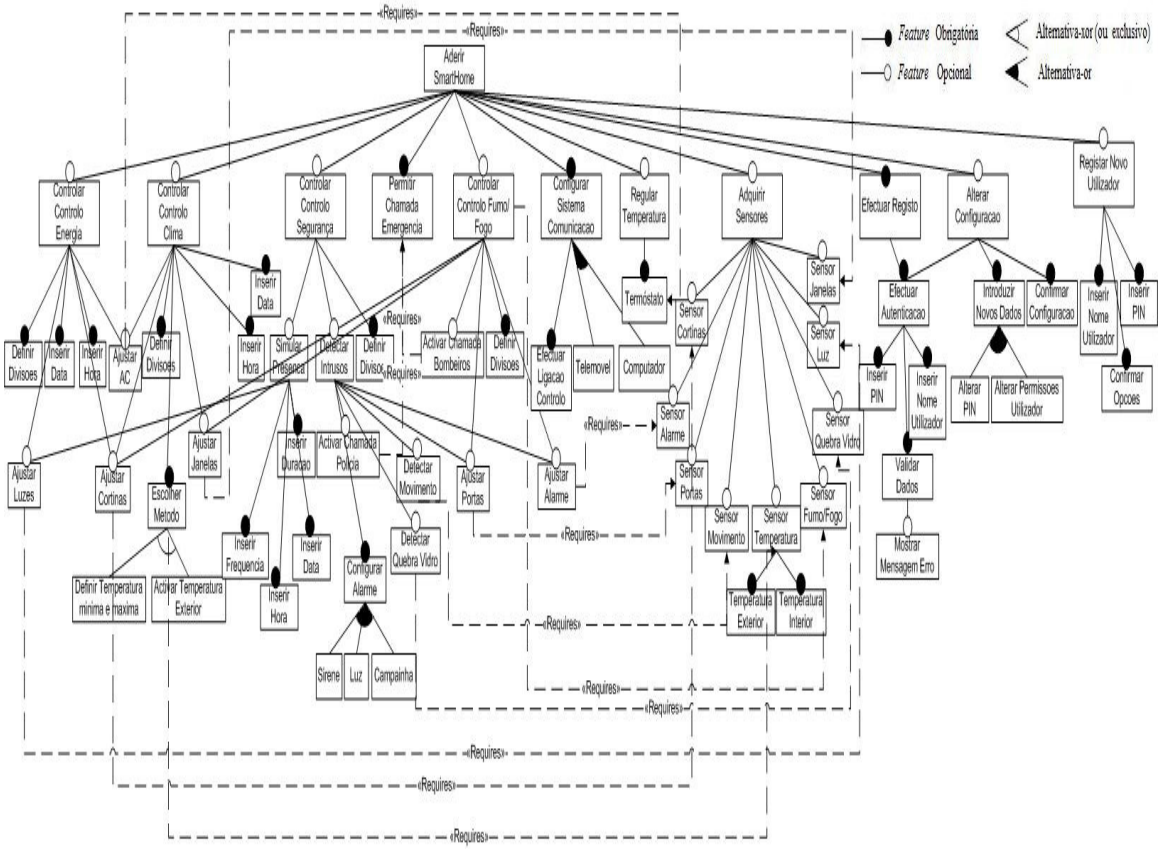

Figura 6.6 Versão final do modelo de *features*

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

6.1.2 Procedimentos a Nível de Engenharia de Aplicação

Nesta secção apresentam-se os modelos de *features*, a visão de relação de temas e a visão transversal para uma configuração do sistema *Smart Home*. A aplicação *Smart Home* permite registar um novo utilizador e o mesmo efectuar o seu registo no sistema. Permite controlar o controlo de energia e o controlo de segurança. O controlo de segurança apenas será configurado para simular presença. O sistema de comunicação é configurado apenas para o telemóvel.

I. Configurar os modelos de *features* para uma aplicação específica. A Figura 6.7 representa o modelo de *features* para a configuração acima descrita.

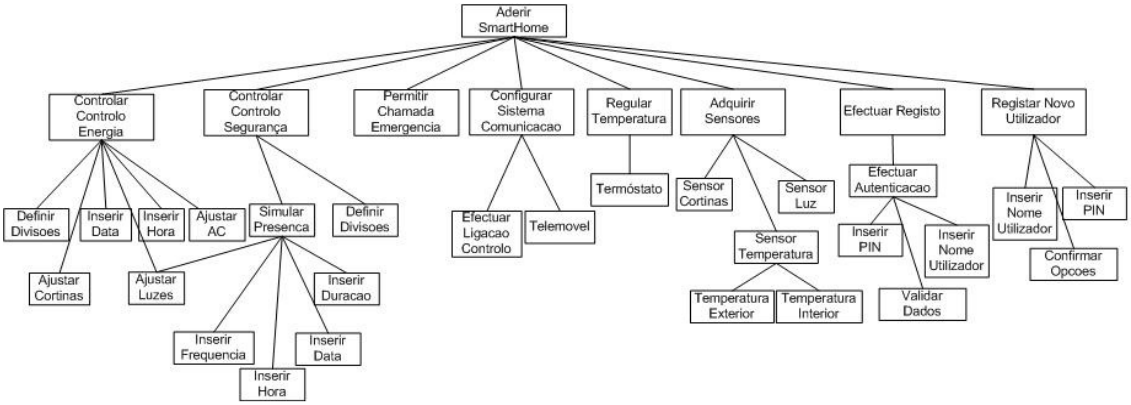

Figura 6.7 Configuração do modelo de *features* para uma aplicação específica

II. Configurar os modelos de Theme para uma aplicação específica. A Figura 6.8 representa a Visão de Relação de Temas e a Figura 6.9 representa a Visão Transversal para a configuração da aplicação descrita acima. Em ambos os modelos removeram-se os elementos intencionais que não são necessários para esta aplicação específica.

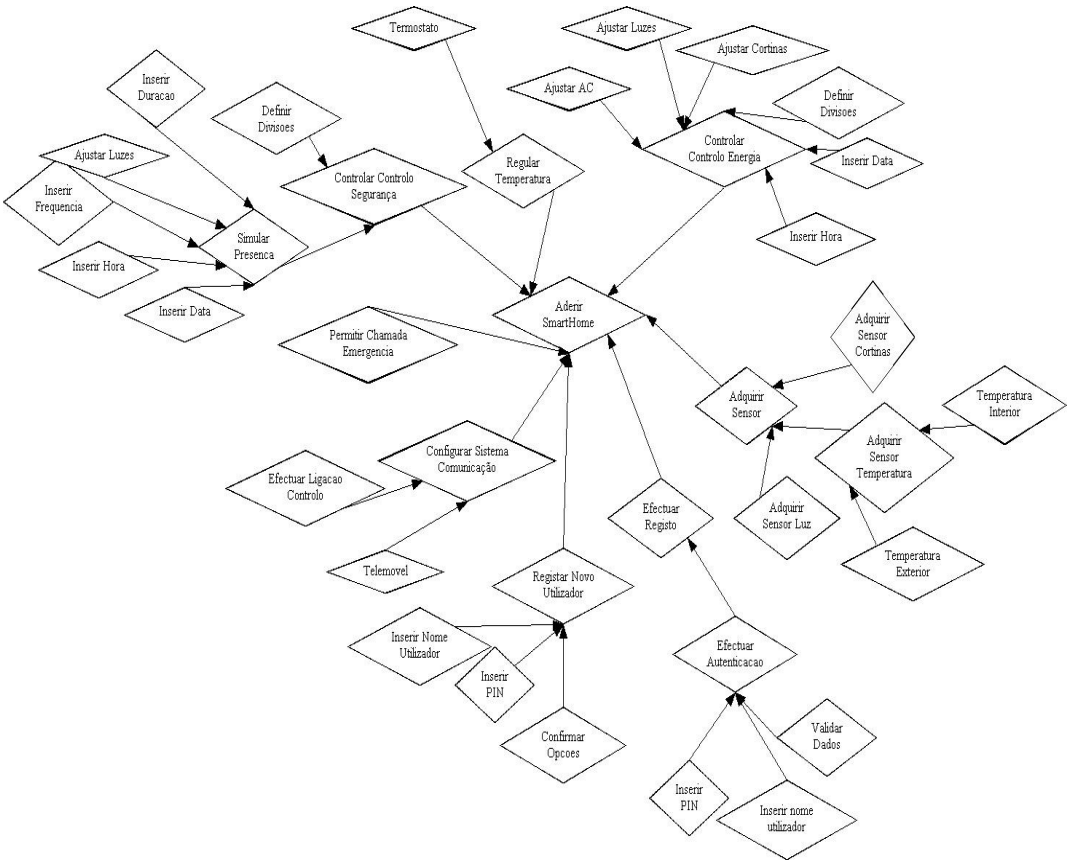

Figura 6.8 Configuração da Visão de Relação de Temas para uma aplicação específica

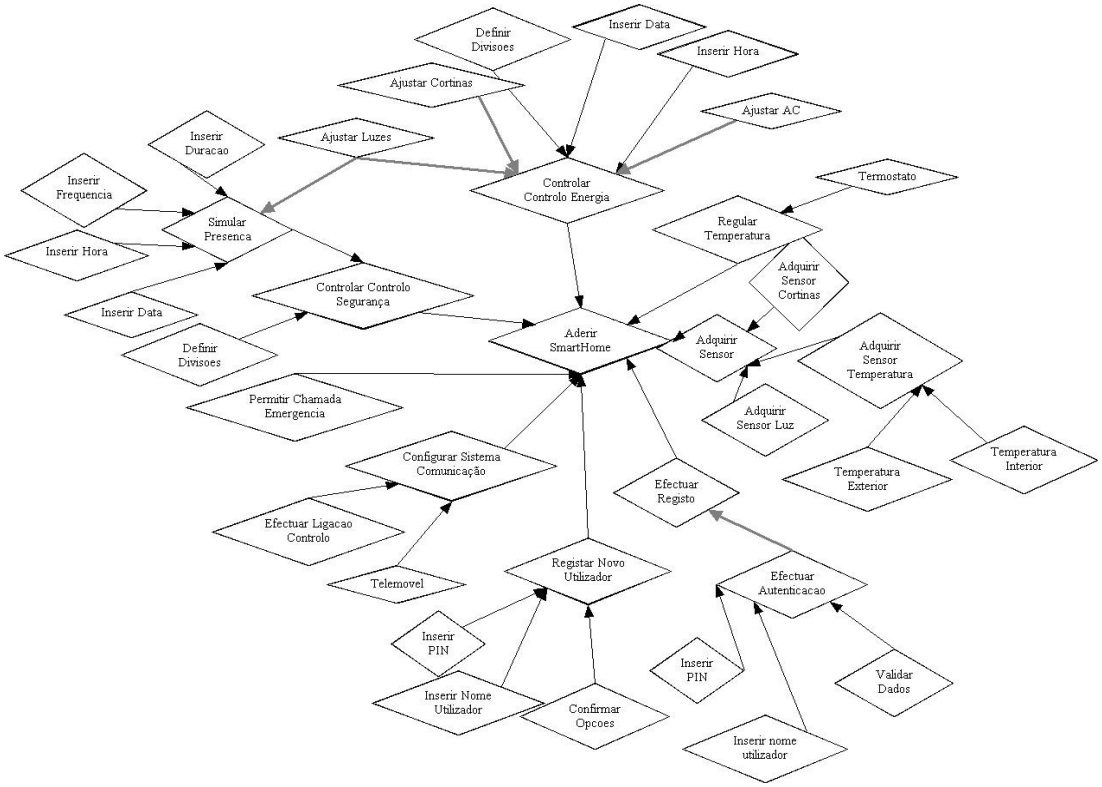

Figura 6.9 Configuração da Visão Transversal para uma aplicação específica

Nesta secção foi aplicada a abordagem Theme-SPL ao caso de estudo *Smart Home* para ajudar a confirmar as várias heurísticas apresentadas na secção 5.3, a fim de se obter o modelo de *features* através dos modelos de Theme.

Em seguida é efectuada uma comparação da abordagem Theme-SPL com outras abordagens de engenharia de requisitos orientada a aspectos integradas com linhas de produtos de software.

6.2 Comparação da Abordagem Theme-SPL com Outras Abordagens

Nesta secção comparam-se as abordagens de linhas de produtos de software orientadas a aspectos, descritas na secção 4.1. Apenas se irá fazer comparação entre abordagens orientadas a aspectos, uma vez que estas abordagens, quando integradas com LPS, podem trazer mais vantagens pois possuem mecanismos eficientes de abstracção, modularização e composição antes de iniciar qualquer mapeamento para o modelo de *features*, de modo a que haja uma estruturação apropriada para lidar com as várias alternativas disponíveis de uma LPS. Os modelos de *features* mostram uma perspectiva muito específica das linhas de produtos, sendo necessário ter uma abordagem que mostre outras perspectivas

ao nível dos requisitos, onde a modularização se deve ter em conta, o que é conseguido pelas abordagens aspectuais.

Serão apresentados os critérios de comparação, e posteriormente, é feita a comparação entre as abordagens.

6.2.1 Critérios de Comparação

A comparação entre as abordagens foi efectuada tendo em conta os seguintes critérios de comparação definidos no projecto da AMPLE (Kovačević et al., 2007):

Heurísticas para construção do modelo de *features*: Consiste num conjunto de passos que visam favorecer o acesso a novos desenvolvimentos teóricos ou descobertas.

Ferramenta para suporte da linguagem: A abordagem apresenta uma ferramenta de apoio.

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.

Modelação de *features*: Consiste em actividades para identificar, estudar e descrever as *features* relevantes para um dado domínio.

Modelação de cenários: Actividades para descrever e modelar o comportamento do tempo de execução, dos membros de família do sistema. Isto não só inclui a funcionalidade dos sistemas e das suas interacções com os utilizadores, mas também aspectos como a segurança, confiabilidade e desempenho.

Composição: Analisa a relação de composição das *features*.

Interacção entre *features*: Ocorre quando a integração de duas *features* modifica o comportamento de uma ou das duas.

De acordo com estes critérios, irá efectuar-se a comparação das abordagens de linhas de produtos de software orientadas a aspectos, definidas na secção 4.1, com a abordagem desenvolvida, Theme-SPL.

6.2.2 Aplicação da Comparação

A Tabela 6.2 regista a comparação das abordagens através dos critérios definidos na secção anterior. As abordagens a comparar são: Theme-SPL proposta nesta dissertação, MATA e LPS de Jayaraman (Jayaraman et al, 2007), I* aspectual e LPS de Silva (Silva et al., 2008), e Casos de Uso e Modelo de *Features* de Bonifácio (Bonifácio e Borba, 2009).

Na Tabela 6.2 o critério **Não** é usado quando o mesmo não se verifica na abordagem.

Tabela 6.2 Comparação entre abordagens orientadas a aspectos integradas com LPS

Abordagens / **Critérios**	**Theme -SPL**	**I* aspectual e LPS (Silva et al., 2008)**	**MATA e LPS (Jayaraman et al., 2007)**	**Casos de Uso e Modelo de *Features* (Bonifácio e Borba, 2009)**
Heurísticas para construção do modelo de *features*	Para obter de modo sistemático modelos de *features* através de modelos Theme.	Para reduzir a complexidade do modelo e identificar os assuntos transversais.	Não	Não
Ferramenta para suporte da linguagem	Linguagem de Domínio Específico que, através de regras de transformação, permite obter modelos de *features* através de modelos Theme.	Não	Permite, de forma automática, a composição de modelos UML de *features* e detecção de interacção de *features.*	Ferramenta de composição de modelos de cenários variáveis com modelos de *features*, configuração de produto e de conhecimento.
Modelação de requisitos	Orientada a aspectos e objectos (UML).	Orientada a aspectos e objectivos.	Orientada a aspectos e objectos (UML).	Orientada a casos de uso.
Modelação de *features*	Captura pontos comuns e pontos variáveis.	Captura pontos comuns e pontos variáveis.	Captura pontos comuns e pontos variáveis.	Captura pontos comuns e pontos variáveis.

Modelação de cenários	Através de diagramas de sequência (Theme/UML).	Não	Através de diagramas de sequência.	Através de casos de uso.
Composição	Associada ao uso do paradigma de orientação a aspectos.	Apoia a composição de modo a tentar reduzir a complexidade dos modelos i*.	Apoia composição de diagramas de classes, sequência e estado através de transformações de grafos.	Suportada por mecanismos de variabilidade de cenários.
Interacção entre *features*	Identifica relações de dependência *requires* e *excludes* no modelo de *features*.	Não	Pode ser verificada para coerência com as relações capturadas no diagrama de dependência de *features.*	Não

Em resumo, de acordo com a Tabela 6.2 a nossa abordagem, comparada com as outras abordagens, apresenta as seguintes vantagens: fornece um conjunto de heurísticas detalhado, segundo as quais é possível obter, de modo sistemático, modelos de *features* através de modelos Theme, e apresenta uma ferramenta que oferece uma forma sistemática e automática de transformar um modelo de Engenharia de Requisitos Orientado a Aspectos (no caso, modelos Theme) em um modelo de *features*.

6.3 Sumário

Neste capítulo foi aplicada a abordagem Theme-SPL ao caso de estudo *Smart Home*. Foi também efectuado um estudo comparativo da abordagem desenvolvida com as abordagens aspectuais I* aspectual e LPS, MATA e LPS e Casos de Uso e Modelo de *Features*.

O próximo capítulo apresenta as Linguagens de Domínio Específico, assim como quais as principais ferramentas que permitem a construção de uma LDE, e a Transformação de Modelos e as principais ferramentas de transformação destes. Todos estes conceitos são úteis para o entendimento do Capítulo 8 que descreve a LDE para Theme-SPL.

7. Linguagem de Domínio Específico

Uma Linguagem de Domínio Específico (LDE) (LDE wiki, 2009), do inglês *Domain-Specific Language* (DSL), é uma linguagem de programação que tem como objectivo especificar a solução de um problema específico numa área de domínio particular. Apenas se focam em expressar problemas de um domínio específico, usando conceitos desse domínio para expressar problemas a um nível de abstracção mais elevado.

A distância conceitual e semântica entre o espaço do problema e a linguagem usada é reduzida, uma vez que os conceitos presentes na LDE são a partir do domínio de destino e não termos genéricos a partir do domínio de solução.

Para especificar a solução de um domínio específico num domínio particular é usado o Modelo de Domínio Específico (MDE) (Kelly and Tolvanen, 2008), do inglês *Domain-Specific Model* (DSM), que consiste em modelar e desenvolver sistemas usando uma LDE, em vez de uma abordagem de uma linguagem mais geral, cujo objectivo é o de especificar a sintaxe do modelo para resolver o problema em questão.

Os MDEs apresentam um nível de abstracção elevado permitindo uma melhor produtividade durante o desenvolvimento e manutenção. Uma vez que a linguagem de modelação inclui regras de domínio é mais difícil criar especificações erradas e os erros de desenvolvimento são detectados e evitados cedo no processo de desenvolvimento.

O uso das LDEs oferecem uma programação mais simples, fácil e confiável. Também reduzem a quantidade e complexidade dos artefactos produzidos o que se traduz em uma melhor produtividade e em custos de manutenção menores. Permitem uma validação e optimização ao nível do domínio, conservação e reutilização do conhecimento do domínio (Deursen et al., 2000). Em alguns casos as LDEs são consideradas linguagens *end-user* uma vez que podem ser utilizadas por não programadores.

Quando comparadas com Linguagens de Domínio Geral, as LDEs são normalmente declarativas e fornecem um pequeno conjunto de notações e abstracções.

Alguns exemplos de LDEs são o SQL (SQL, 2009), HTML (W3C HTML, 2009), LATEX (LaTex Project, 2009), GraphViz (GraphViz, 2009) e os *shell scripts* do sistema Unix.

As vantagens de utilizar LDEs consistem na possibilidade de se poder expressar a solução na linguagem desejada e ao nível de abstracção do domínio do problema que se pretende resolver (LDE wiki, 2009). Deste modo os especialistas do domínio podem compreender, validar, modificar e desenvolver programas com a LDE. As LDEs também aumentam a qualidade, produtividade, fiabilidade, portabilidade e reutilização quando aplicadas a um domínio específico. Como as LDEs incorporam conhecimento sobre o domínio vão permitir a conservação e reutilização desse conhecimento. Pelo que os programas de LDE são concisos, auto-documentados e podem ser reutilizados. As LDEs permitem a geração de código comentado para o domínio especificado, sendo ainda possível validar uma LDE ao nível de domínio.

A maior desvantagem de utilizar LDEs é o custo de desenho, implementação e manutenção (LDE wiki, 2009). Os custos de educar os utilizadores da LDE são também consideráveis, apesar do facto de que a LDE usa os conceitos do domínio. Alguns riscos decorrentes de encontrar a LDE adequada para a aplicação e o equilíbrio entre a especificação do domínio e os conceitos das linguagens de programação de domínio geral. Outra desvantagem é a potencial perda de eficiência do código gerado quando comparado com código efectuado à mão, mas este inconveniente tende a desaparecer com o aumento de energia do computador.

Apesar das desvantagens, considera-se que as vantagens das LDEs conseguem superar largamente os seus pontos mais fracos, daí estarem com mais frequência a ser implementadas e com sucesso nos mais variados domínios.

7.1 Ferramentas para a Linguagem de Domínio Específico

Seguidamente são apresentadas as ferramentas mais relevantes, a nível de utilização, para a construção de uma linguagem de domínio específico.

7.1.1 GME

O Ambiente de Modelação Genérico, do inglês *Generic Modeling Environment* (GME), é uma ferramenta configurável para criar modelos de domínios específicos desenvolvida pela Universidade de Vanderbilt (GME, 2009). A configuração é realizada através de metamodelos especificando o paradigma de modelação do domínio de aplicação. A linguagem de metamodelação é baseada nos diagramas de classes UML e em restrições OCL. Os metamodelos, especificando o paradigma de modelação, são usados para gerar automaticamente o ambiente alvo específico do domínio. O ambiente específico do domínio gerado é então usado para construir modelos do domínio que são armazenados em modelos de bases de dados ou em formato XML. Estes modelos são usados para gerar automaticamente as aplicações ou para sintetizar entrada para diferentes ferramentas de análise.

7.1.2 DSL Tools

A DSL Tools é uma ferramenta para criar e modelar domínios específicos desenvolvida pela Microsoft, sendo um *plugin* para a plataforma Visual Studio (Bézivin et al., 2005) (DSL Tools, 2009). É um conjunto de ferramentas para criar, editar e visualizar domínios específicos, com o objectivo de automatizar o processo de desenvolvimento de software. A ferramenta está relacionada com o conceito de software *factories*, isto é, uma linha de produtos de software que une ferramentas de desenvolvimento extensíveis com padrões ou LDEs, baseada em receitas para construir tipos específicos de aplicações. Permite definir um modelo do domínio com um *designer* e um gerador de artefactos textuais. Através de um editor gráfico permite editar e definir modelos de domínio. Os dados são armazenados em formato XML. Além de gerar o código do modelo permite também a validação desse mesmo código.

7.1.3 EMF/GMF

O Eclipse é um *framework* de programação em Java que suporta um grande número de *plugins,* dos quais se destaca o GMF, do inglês *Graphical Modeling Framework*, e o EMF, do inglês *Eclipse Modeling Framework*, que são apresentados em seguida.

O EMF é um *framework* de modelação e geração de código para construção de ferramentas e outras aplicações, baseado num modelo de dados estruturado ou metamodelo (Ecore) (Eclipse/GMF wiki, 2009). O metamodelo pode ser construído utilizando uma interface gráfica ou recorrendo a um editor textual em XML.

Uma vez construído o metamodelo do domínio específico que se pretende implementar, o EMF permite a geração e validação do código específico do modelo construído, gerando um conjunto de classes Java, sendo cada uma destas classes uma classe ou uma relação entre classes expressa no metamodelo.

O GMF por sua vez utiliza o *Domain Model* (Ecore) que foi definido na fase relativa ao EMF. Procura disponibilizar componentes e infra-estruturas que permitam o desenvolvimento de editores gráficos, baseados no EMF (Eclipse/GMF wiki, 2009). Com estes editores torna-se possível definir toda a componente gráfica, tanto a nível dos elementos do modelo como a nível das ferramentas que permitem criar esses mesmos elementos. No GMF define-se quais as ferramentas responsáveis por cada elemento e como é que os elementos se relacionam entre si.

7.2 Desenvolvimento Orientado a Modelos

O Desenvolvimento Orientado a Modelos (Völter e Stahl, 2006), do inglês *Model Driven Development*, é uma abordagem que permite a utilização sistemática de modelos na implementação e especificação de um sistema.

Esta abordagem tem como conceitos básicos a noção de modelos, metamodelos, meta-metamodelos e transformações de modelos. Um modelo é a conceptualização do sistema representando uma visão completa do mesmo, ou parte dele, através de vistas particulares. Cada modelo deve estar em conformidade com o seu metamodelo, isto é, deve estar escrito na linguagem do metamodelo. Representa a sintaxe abstracta da linguagem e descreve todos os conceitos que podem ser utilizados nessa linguagem. Em geral, podemos ver o metamodelo como um modelo que descreve outro modelo. Por sua vez, o metamodelo deve estar em conformidade com o seu meta-metamodelo. Estas relações estão organizadas em quatro níveis de abstracção diferentes (Figura 7.1) sistema (M0), modelo do sistema (M1), metamodelo (M2) e meta-metamodelo (M3). O nível

correspondente ao sistema é uma instância do modelo sendo o resultado da sua representação. No que diz respeito às transformações, estas são especificadas entre modelos, concretamente entre um modelo origem e um modelo destino, com o intuito de refinar gradualmente o sistema.

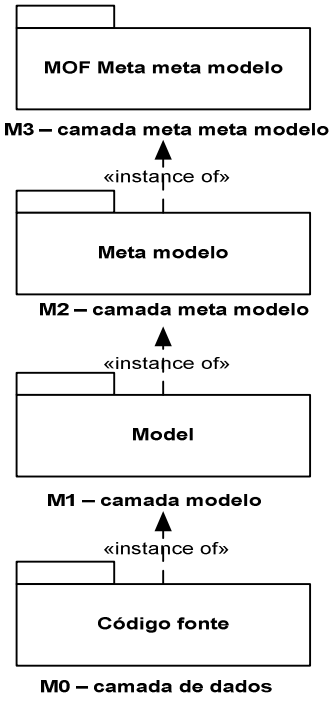

Figura 7.1 Arquitectura de quatro camadas (Bézivin, 2005)

Foram definidos vários *frameworks* para suportar este paradigma orientado a modelos, sendo o *Model-Driven Architecture* (MDA) (MDA, 2009) o *framework* de referência. Este *framework* resultou de uma iniciativa da OMG (*The Object Management Group*) (OMG, 2009) e tem como objectivo especificar um sistema independente da plataforma que irá suportá-lo.

7.3 Transformação de Modelos

A transformação de modelos (Czarnecki e Helsen, 2003), no contexto de Desenvolvimento Orientado a Modelos, é um processo a partir do qual modelos que estão de acordo com um conjunto de regras são transformados em outros, que por sua vez, estão em conformidade com um novo conjunto de regras. Nesta dissertação, essas regras são expressas através de metamodelos. Este processo envolve como principais elementos, metamodelos de origem e de destino, assim como regras de transformação. Estas últimas,

perante um determinado padrão (no metamodelo de origem), determinam as alterações necessárias a efectuar para que o modelo de destino esteja de acordo com o seu metamodelo. A Figura 7.2 ajuda a compreender melhor as relações entre os elementos envolvidos. Tanto os modelos de origem e destino, como as regras de transformação, estão de acordo com os seus metamodelos. Por sua vez, todos eles partilham o mesmo meta-metamodelo. Como se pode observar na Figura 7.2 tem-se um modelo de origem, designado A, ao qual se aplicam regras de transformação (com base em reconhecimento de padrões) e que o transformam noutro modelo (destino), designado de B. O facto de a aplicação das regras ter sempre em conta os metamodelos dos modelos envolvidos garante a geração de modelos sintacticamente correctos.

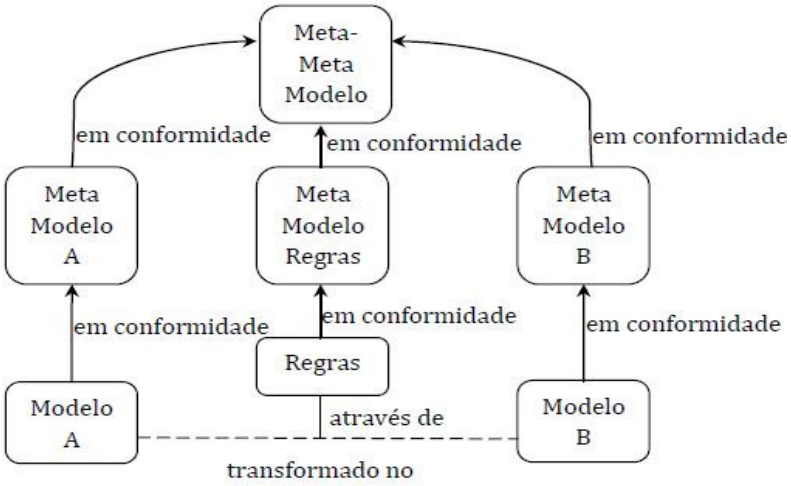

Figura 7.2 Processo de transformação de modelos (retirado de Sousa, 2009)

7.3.1 Ferramentas de Transformação de Modelos

A transformação de modelos apresenta uma grande variedade de ferramentas, contudo nesta dissertação apenas iremos focar aquelas que se encontram integradas com a plataforma Eclipse, uma vez que o *framework* considerado nesta dissertação irá ser desenvolvido nesta plataforma por ser mais acessível e bastante utilizada no meio académico. Pelo que, as ferramentas de transformação de modelos que se encontram integradas com o eclipse são (Eclipse model-to-model, 2009):

- **QVT Operacional** (Operational QVT, 2009): é um componente do projecto *Model To Model* (M2M) (Eclipse model-to-model, 2009) do Eclipse, cuja

implementação é baseada no QVT (*Query/Views/Transformations*) *standard* da OMG. Assenta num paradigma imperativo e foi desenhado para transformações cujos modelos de destino têm uma estrutura complexa. Este paradigma é útil para os casos em que não existe uma correspondência directa entre os elementos dos modelos de origem e destino. Neste contexto, ao contrário do paradigma declarativo, este problema é ultrapassado recorrendo à definição de procedimentos. O processo de transformação, com esta ferramenta, converte um ou mais modelos de origem em um ou mais modelos de destino e é baseado nas relações entre os seus elementos. O cenário típico consiste na conversão de um modelo origem (Mo), em conformidade com o seu metamodelo (MMo), num modelo de destino (Md), em conformidade com o seu metamodelo (MMd).

- **QVT Declarativo** (Declarative QVT, 2009): é um componente do projecto Eclipse (M2M), tal como o QVT Operacional, a diferença entre ambos consiste no paradigma utilizado. Em vez de adoptar por um paradigma imperativo opta por um declarativo. É uma ferramenta relativamente recente (foi lançada em Outubro de 2008) pelo que conta com pouca documentação.

- **ATL** (ATL, 2009) (ATL User Guide, 2009): do inglês *Atlas Transformation Language,* encontra-se integrado no projecto M2M da plataforma eclipse, e é a resposta do grupo de desenvolvimento ATLAS ao QVT da OMG. Apresenta uma linguagem híbrida permitindo especificar regras de forma declarativa e imperativa. O modo de transformação preferencial é o modo declarativo uma vez que permite expressar mapeamentos entre os elementos dos modelos de origem e de destino, de forma simples. Contudo, o ATL também disponibiliza construtores imperativos de forma a facilitar a especificação de mapeamentos entre elementos, que são difíceis de expressar de forma declarativa. A sintaxe dos elementos ATL, bem como as suas expressões são baseadas em OCL. As transformações entre modelos são definidas por meio de módulos ATL. Este tipo de unidades permite gerar um conjunto de modelos destino a partir de modelos origem, em que todos os modelos envolvidos estão de acordo com o seu próprio metamodelo. Um módulo ATL é composto por:

- ***Header Section***: define o nome do módulo e o nome das variáveis correspondentes aos modelos de origem e destino;
- ***Import Section***: secção opcional que permite importar bibliotecas ATL;
- ***Helpers***: podem ser vistos como equivalentes a métodos Java e são utilizados para factorizar código;
- ***Rules***: existem três tipos de regras: (i) *matched rules*, "*core"* da transformação declarativa; (ii) *lazy rules,* são semelhantes às anteriores mas só são aplicadas quando chamadas por outras regras; (iii) c*alled rules*, "*core"* da transformação imperativa.

Das três ferramentas de modelação descritas, o ATL, no contexto desta dissertação, reúne as melhores características para ser escolhido. Em relação às outras duas tem um suporte mais alargado no que respeita à documentação, com a inclusão de bastantes exemplos ilustrativos. Para além de estar de acordo com os standards da OMG, ao derivar do QVT permite especificar as regras de transformação de duas formas distintas: imperativa e declarativa, algo que não é possível nas outras duas ferramentas.

7.4 Sumário

Neste capítulo introduziram-se os principais conceitos das LDEs e quais as suas vantagens e desvantagens. Mostraram-se quais as ferramentas mais utilizadas para o desenvolvimento de LDEs de modo a determinar qual a melhor para o desenvolvimento da ferramenta desta dissertação. Foi apresentada a área de Desenvolvimento Orientado a Modelos fazendo referência aos seus principais conceitos. Por último foi descrita a transformação de modelos e as ferramentas que possibilitam implementar as transformações.

O próximo capítulo mostra os passos necessários para a construção de uma LDE com base nos metamodelos criados para a abordagem Theme-SPL e para o Modelo de *Features*.

8. Especificação da LDE para a Abordagem Theme-SPL

Para a construção da LDE, que implementa a abordagem Theme-SPL, foi escolhida a plataforma Eclipse com os plugins *Eclipse Modeling Framework* (EMF) e *Graphical Modeling Framework* (GMF) (Eclipse/GMF wiki, 2009).

De entre as ferramentas apresentadas na secção 7.1, esta foi escolhida pois em termos de definição de interface visual e de possibilidades de modelação, oferece os melhores métodos, tendo um vasto número de opções disponíveis para se fazer uma modelação correcta e completa do domínio em análise; suporta geração de código, pois ao gerar o modelo no editor é também criado um ficheiro XML com a descrição do modelo criado; o facto de este *framework* ser *freeware* e já existir há algum tempo, foi outro dos motivos da sua escolha, uma vez que foi criado de uma comunidade à volta do mesmo existindo muita informação disponível.

Os passos tomados na criação da ferramenta estão descritos nas próximas secções.

8.1 Criação do Modelo Ecore para Theme-SPL

Para a criação da LDE irão especificar-se os dois modelos Ecore (Eclipse/GMF wiki, 2009) para especificar a linguagem a ser construída.

O modelo Ecore consiste num metamodelo específico de uma LDE. Através deste modelo vão ser criadas todas as regras da LDE que se pretende construir, sendo por vezes necessário acrescentar algumas regras sintácticas recorrendo à linguagem *Object Constraint Language* (OCL) (OMG, 2005). O uso de OCL por vezes é necessário devido à incapacidade de se conseguir expressar todas as regras sintácticas de uma linguagem através do seu metamodelo.

Foi criado um modelo Ecore para a abordagem Theme-SPL e um modelo Ecore para o modelo de *features*, ambos adaptados para suportar Desenvolvimento Orientado a Modelos.

O modelo Ecore para a abordagem Theme-SPL contém um nó raiz, chamado *ThemeApproach*, ao qual estão ligadas as classes que representam os nós e as ligações entre os nós. Foram identificados os nós *RootTheme*, *Themes*, *Aspect*, *Entity*, *ThemeGroup* e *GroupedTheme*, e as ligações *Obligatory*, *Alternative*, *Part_of*, *Crosscuting*, *Alternative_or, Alternative_xor*, *Require* e *Extend*. Este modelo é apresentado na Figura 8.1.

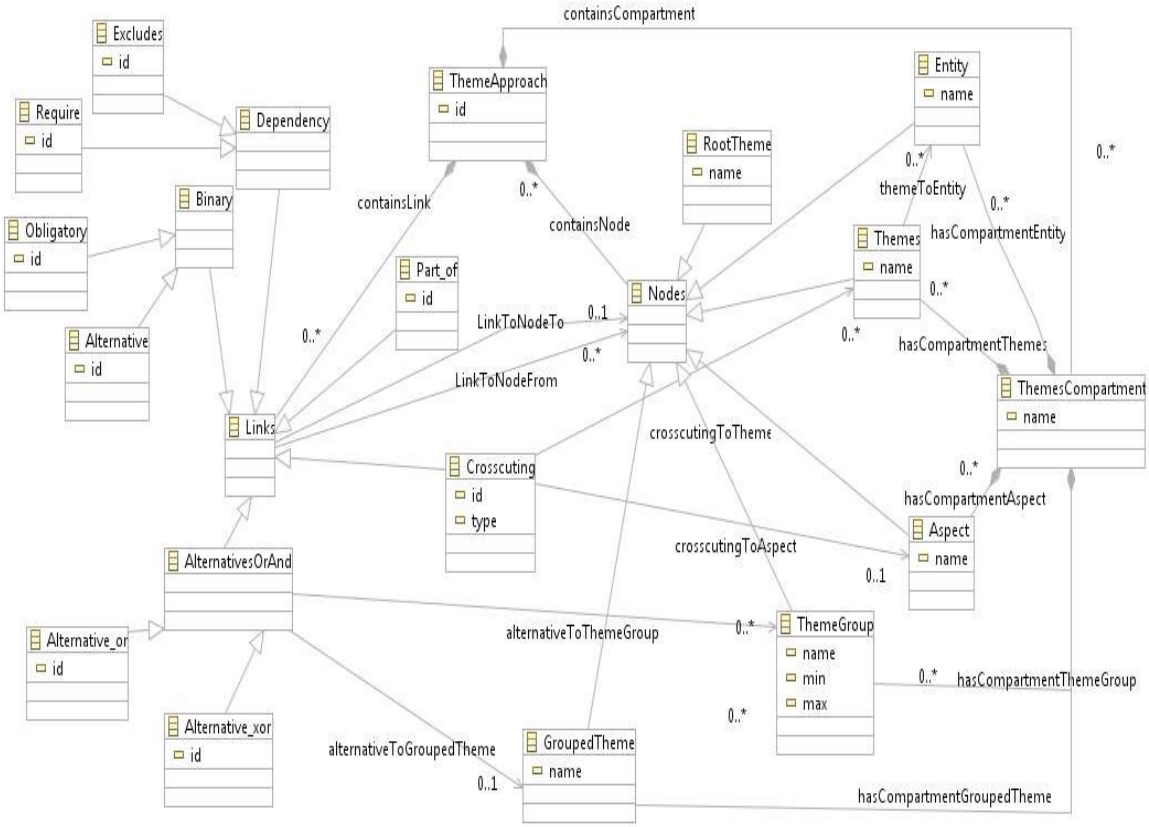

Figura 8.1 Modelo Ecore da abordagem Theme-SPL

Seguidamente são apresentados todos os conceitos existentes no modelo Ecore, o seu objectivo dentro da linguagem, qual o conceito de Theme que lhe está associado e as respectivas relações entre eles:

- *ThemeApproach*: este elemento não representa um conceito de Theme, mas sim um "diagrama", isto é, todos os elementos que lhe estão ligados são elementos possíveis de criar no editor da ferramenta Theme-SPL. Este elemento tem as seguintes ligações associadas:
  - *containsNode*: liga os elementos de Theme-SPL e *Nodes.* Esta ligação está relacionada com a criação de nós com a ferramenta.
  - *containsLink*: liga os elementos de Theme-SPL e *Links.* Esta ligação está relacionada com a criação de ligações com a ferramenta.
  - *containsCompartment*: liga os elementos de Theme-SPL e *ThemesCompartment*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar *Themes*, *Entity*, *Aspect*, *ThemeGroup* e *GroupedTheme*.
- *Nodes*: Representa todos os nós que podem ser construídos com a ferramenta.
- *RootTheme*: Representa o tema raiz, ou seja, contém as partes em que o sistema consiste. Tem um atributo *String name* que representa o nome da raiz.
- *Themes*: Representa um Tema da abordagem Theme. Tem um atributo *String name* que representa o nome do tema.
- *Aspect*: Representa um Aspecto da abordagem Theme. Tem um atributo *String name* que representa o nome do aspecto.
- *Entity*: Representa uma Entidade da abordagem Theme. Tem um atributo *String name* que representa o nome da entidade.
- *ThemeGroup*: Tema que permite agrupar dois ou mais *GroupedTheme*. Tem um atributo *String name* que representa o nome do *ThemeGroup, String min* que representa o número mínimo de *GroupedThemes* que o produto permite e *String max* que representa o número máximo de *GroupedThemes* que o produto permite.
- *GroupedTheme*: Representa um conjunto de temas agrupáveis. Tem um atributo *String name* que representa o nome do *GroupedTheme*.

- *ThemesCompartments*: Este compartimento destina-se a guardar *Themes*, *Entity*, *Aspect*, *ThemeGroup* e *GroupedThemes* da abordagem Theme-SPL. Tem um atributo *name*, do tipo *String*, para guardar o nome. Tem associadas as seguintes ligações:
  - *hasCompartmentEntity*: liga o compartimento e o elemento *Entity*. Esta ligação representa a possibilidade de colocar *Entity* no compartimento.
  - *hasCompartmentThemes:* liga o compartimento e o elemento *Themes*. Esta ligação representa a possibilidade de colocar *Themes* no compartimento.
  - *hasCompartmentAspect*: liga o compartimento e o elemento *Aspect*. Esta ligação representa a possibilidade de colocar *Aspects* no compartimento.
  - *hasCompartmentThemeGroup*: liga o compartimento e o elemento *ThemeGroup*. Esta ligação representa a possibilidade de colocar *ThemeGroups* no compartimento.
  - *hasCompartmentGroupedTheme*: liga o compartimento e o elemento *GroupedTheme*. Esta ligação representa a possibilidade de colocar *GroupedThemes* no compartimento.
- *Links*: Representa todas as ligações que podem ser construídas com a ferramenta. Tem associado as seguintes ligações:
  - *linkToNodeFrom* e *linkToNodeTo*: permitem relações entre todos os *Nodes.*
- *Binary*: Representa as ligações *Obligatory* e *Alternative*.
- *Obligatory*: Representa uma relação de obrigatoriedade entre dois ou mais temas. Tem um atributo *String id* que representa o nome da ligação.
- *Alternative*: Representa uma relação de opcionalidade entre dois ou mais temas. Tem um atributo *String id* que representa o nome da ligação.

- *Part_of:* Representa uma relação entre um tema e os temas que o compõe. Tem um atributo *String id* que representa o nome da ligação.

- *Crosscuting:* Representa o conceito de transversalidade entre aspectos e temas na abordagem Theme. Tem dois atributos, *String id* que representa o nome da ligação, e *String type* que indica qual o tipo de relação entre o aspecto e o tema (*obligatory* ou *alternative*). Tem associado as seguintes ligações:

  - *crosscutingToAspect:* permite que as relações de *crosscuting* apenas tenham origem no *Aspect*;

  - *crosscutingToTheme:* apenas permite que uma ligação de *crosscuting* tenha como classe de destino o *Themes*.

- *AlternativesOrAnd:* Representa as ligações *Alternative_or* e *Alternative_xor*. Tem associado as seguintes ligações:

  - *alternativeToGroupedTheme*: permite que tanto as relações *Alternative_or* como *Alternative_xor* apenas tenham como origem a classe *GroupedTheme*;

  - *alternativeToThemeGroup* permite que as ligações *Alternative_or* e *Alternative_xor* apenas tenham como destino a classe *ThemeGroup*.

- *Alternative_or:* Representa uma relação OR entre *GroupedTheme* e *ThemeGroup*. Tem um atributo *String id* que representa o nome da ligação.

- *Alternative_xor:* Representa uma relação XOR (ou exclusivo) entre *GroupedTheme* e *ThemeGroup*. Tem um atributo *String id* que representa o nome da ligação.

- *Dependency*: Representa as ligações *Requires* e *Excludes*.

- *Requires*: Representa uma relação entre dois temas independentes em que um dos temas precisa do outro. Tem um atributo *String id* que representa o nome da ligação.

- *Excludes*: Representa uma relação entre dois temas independentes em que se um deles está presente no produto o outro não pode estar. Tem um atributo *String id* que representa o nome da ligação.

As relações de *obligatory*, *alternative*, *part_of*, *requires* e *excludes* devem ter como destino qualquer uma das classes (*RootTheme*, *Themes*, *Aspect*, *Entity* e *ThemeGroup*), mas como origem apenas a classe *Themes, Entity* e *ThemeGroup*. Para colocar esta restrição no modelo o mesmo ficaria complicado e poderia perder a legibilidade, então a mesma foi feita com regras OCL, como explicado na secção 8.4.

8.2 Criação do Modelo Ecore para Modelo de *Features*

O modelo Ecore para o modelo de *features* contém um nó raiz de nome *FeatureModel*, e tal como no modelo da abordagem Theme-SPL, à raiz estão ligadas as classes que representam os nós e as ligações entre os nós. Foram identificados os nós *RootFeature*, *Feature*, *FeatureGroup* e *GroupedFeature*, e as ligações *Optional*, *Mandatory*, *Alternative_or*, *Alternative_xor*, *Require* e *Exclude*. O metamodelo é apresentado na Figura 8.2.

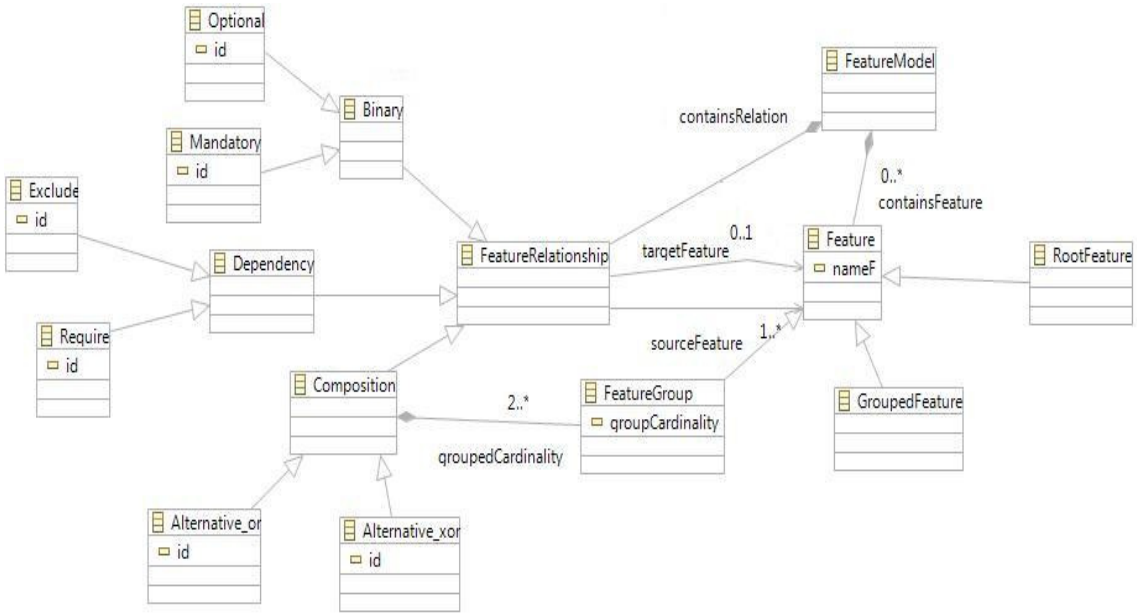

Figura 8.2 Modelo Ecore do modelo de *features* (adaptado de Czarnecki et al., 2004 e Kang et al., 1990)

Em seguida são apresentados todos os conceitos existentes no modelo Ecore, o seu objectivo dentro da linguagem, qual o conceito do modelo de *features* que lhe está associado e as respectivas relações entre eles:

- *FeatureModel:* Um modelo de *features* consiste em pelo menos uma raiz (*RootFeature*) que forma a raiz dos diferentes diagramas de *features* no modelo. Existem três tipos de *Features*: *RootFeature*, *GroupedFeature* e *FeatureGroup.* Este elemento tem as seguintes ligações associadas:
  - *containsFeature:* liga os elementos do modelo de *features* e *Features.* Esta ligação está relacionada com a criação de nós com a ferramenta.
  - *containsRelation:* liga os elementos do modelo de *features* e relações. Esta ligação está relacionada com a criação de ligações com a ferramenta.
- *Feature:* Representa uma *Feature* do modelo de *features*, e todos os nós que podem ser construídos com a ferramenta. Tem um atributo *String nameF* que representa o nome da *Feature*.
- *RootFeature:* Representa uma *RootFeature* do modelo de *features*. Tem um atributo *String nameF* que representa o nome da *RootFeature*.
- *GroupedFeature:* Representa uma *GroupedFeature* do modelo de *features*. Tem um atributo *String nameF* que representa o nome da *GroupedFeature*.
- *FeatureGroup:* Representa uma *GroupedFeature* do modelo de *features*. Tem um atributo *String nameF* que representa o nome da *GroupedFeature*, e uma *String groupCardinality* que representa a cardinalidade da *feature*.
  - *groupedCardinality:* indica que uma *FeatureGroup* pode conter duas ou mais relações de *Composition*.
- *FeatureRelationship:* Representa todas as ligações que podem ser construídas com a ferramenta.
- *Binary:* Representa as relações *mandatory* e *optional* do modelo de *features*.

- *Dependency:* Representa as relações *excludes* e *requires* do modelo de *features*.
- *Composition:* Representa as relações *alternative-or* e *alternative-xor* do modelo de *features*.

8.3 GMFGraph e GMFTool

O GMFGraph é o componente responsável pelos gráficos do modelo que se pretende criar e o GMFTool é o componente responsável pela apresentação visual da caixa de ferramentas a ser usada no editor, pelo que é importante que se dê especial atenção a esta fase de modelação da LDE, pois dela depende o retorno visual para o utilizador. Para se conseguir criar tanto a componente GMFGraph como a GMFTool usou-se o *dashboard* do GMF, ilustrado na Figura 8.3.

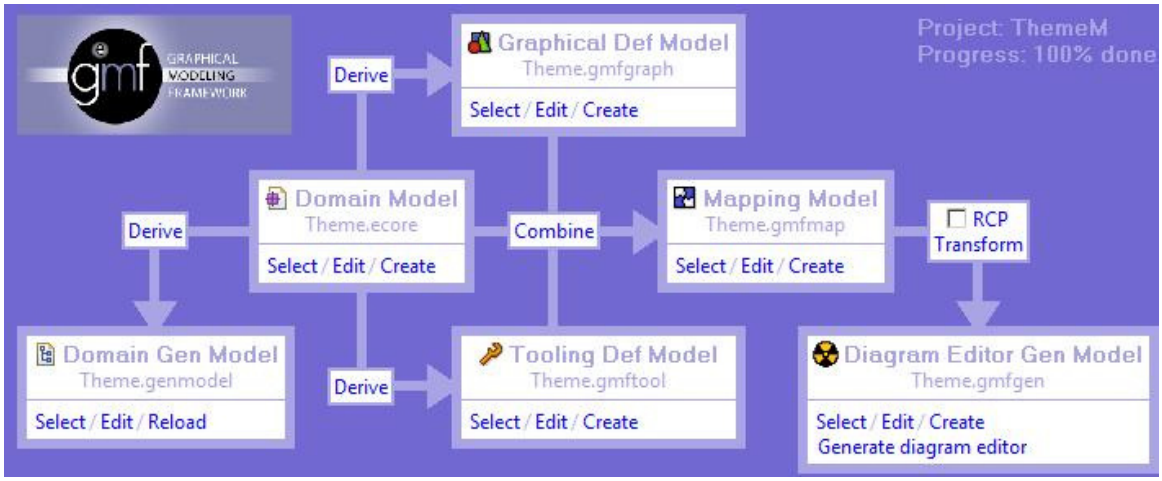

Figura 8.3 ***Dashboard***

Como se pode visualizar na Figura 8.3, a partir do modelo Ecore definido anteriormente (ilustrado na caixa Domain Model), pode-se derivar as componentes GMFGraph (na caixa Graphical Def Model) e GMFTool (na caixa Tooling Def Model), seleccionando quais as classes, as ligações e os atributos que irão aparecer na LDE final. Após estas duas componentes estarem correctamente criadas, as mesmas são combinadas, de modo a criar a componente GMFMap (na caixa Mapping Model) especificando mais uma vez quais as classes do modelo Ecore que serão os nós e quais serão as ligações. Finalmente, com essa componente correctamente criada gerar-se-á o editor da LDE que se pretende (através da caixa RCP Transform e Diagram Editor Gen Model), sendo neste caso um

editor para a abordagem Theme-SPL. Os mesmos passos são efectuados para o editor do modelo de *features*.

8.4 GMFMap

O GMFMap é o componente responsável pela parte lógica do editor gerado pelo GMF. É neste componente que se vai definir fisicamente os nós e as ligações do modelo. Caso seja necessário impor algumas restrições extra ao modelo é nesta fase que essas restrições (OCL) são incluídas.

Num nó existem propriedades para definir qual a sua origem no modelo Ecore e quais as relações com os nós criados pelos componentes GMFGraph e GMFTool. Indica-se qual a classe que representa o elemento que se quer criar, qual o nó definido no GMFGraph que representa o elemento e qual o nó no GMFTool que representa o elemento. Como mostrado na Figura 8.4.

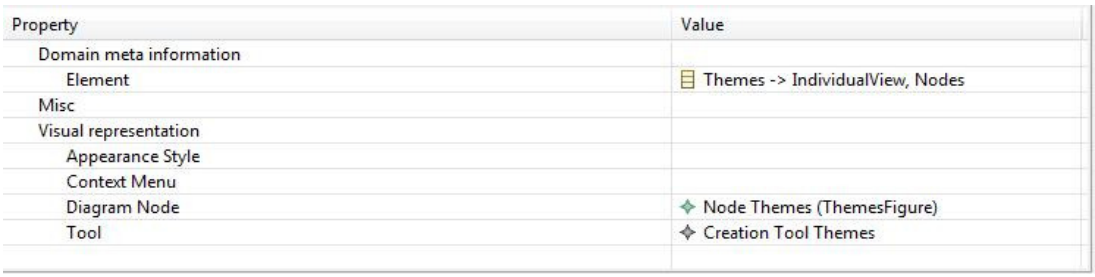

Figura 8.4 Propriedades de um nó na LDE Theme-SPL

Numa ligação existem propriedades para definir qual a sua origem no modelo Ecore, qual a relação com os nós criados pelos componentes GMFGraph e GMFTool e quais os seus nós de origem e destino. Indica-se qual a ligação correspondente no modelo Ecore que referencia a ligação que se deseja criar, qual a classe que representa o elemento que se quer criar, quais os nós de origem e destino que a ligação vai conter, qual o nó definido no GMFGraph que representa a ligação e qual o nó no GMFTool que representa a ligação. Como ilustra a Figura 8.5.

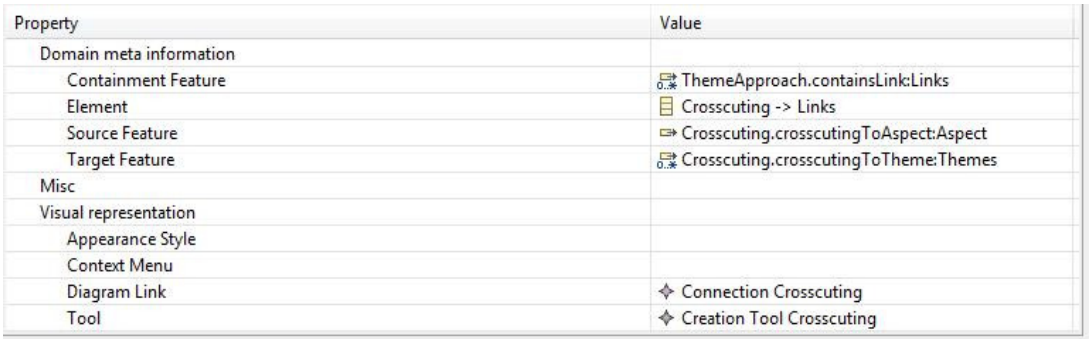

Figura 8.5 Propriedades de uma ligação na LDE Theme-SPL

Tendo em conta Suen e Baniassad (Suen e Baniassad, 2005), que definem clusters de requisitos para ter uma coesão mais lógica, foram definidos compartimentos de modo a melhorar também a escalabilidade do modelo. Estes compartimentos são também modelados no GMFMap (Figura 8.6), sendo criado um nó que contenha o *compartment*. Definem-se os elementos que podem ser contidos dentro do nó *compartment*, criando dentro do nó pai, nós *ChildReference* que vão definir um nó filho que pode ser contido dentro do *compartment*. Deste modo consegue-se definir quais os nós que são *compartments* e quais os nós que podem ser contidos dentro desses mesmos *compartments*.

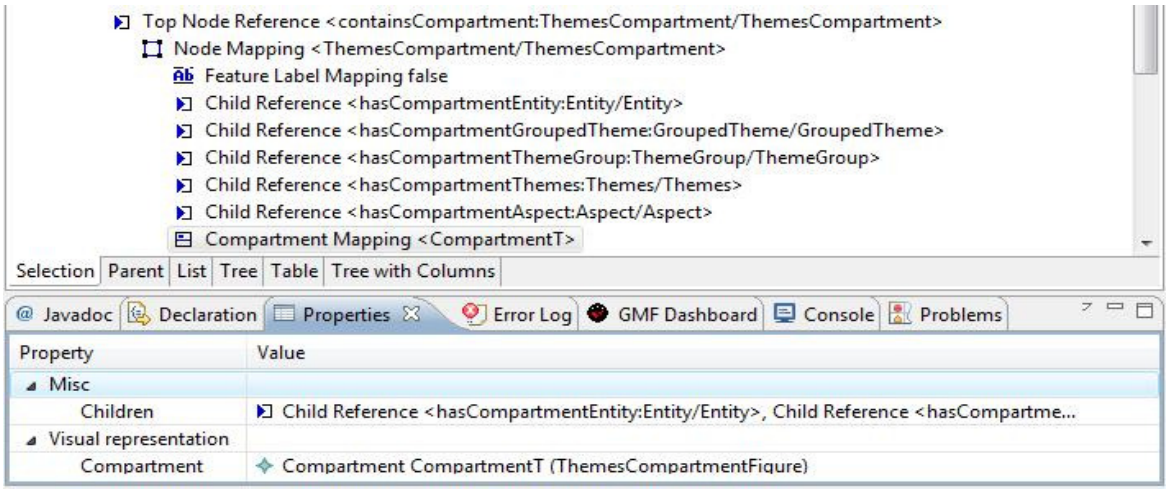

Figura 8.6 Propriedades de um *Compartment*

O compartimento é colapsável e, na sua forma colapsada, esconde o que está no seu interior o que pode ser útil quando se pretende analisar apenas porções do modelo em estudo, além de melhorar a complexidade visual dos mesmos.

Para uma melhor performance da LDE são criadas algumas restrições OCL ao nível das ligações. Foi introduzida uma restrição para evitar ligações de um elemento para ele próprio, uma vez que estas não faziam sentido. Para isso, foi usada a expressão OCL **self <> oppositeEnd**, isto é, o ponto inicial e o ponto final de uma ligação não podem coincidir no mesmo elemento. Esta restrição foi feita em todas as ligações.

Na LDE Theme-SPL, foi também introduzida a restrição **not oclIsTypeOf (RootTheme) and not oclIsTypeOf (GroupedTheme)**, na origem da relação, nas ligações *Part_of*, *Alternative* e *Obligatory,* uma vez que na raiz apenas podem entrar ligações, ou seja, não pode haver ligações a começar na raiz, e um *GroupedTheme* não tem filhos. Nas ligações *Requires* e *Excludes*, é introduzida a restrição **not oclIsTypeOf (RootTheme)**, uma vez que apenas estão ligados à raiz os componentes que a compõem, pelo que esta só suporta ligações de *Part_of*, *Alternative* e *Obligatory.*

Na LDE para o modelo de *features* foram também criadas algumas restrições OCL para as ligações. Nas ligações *mandatory* e *optional* foram criadas as seguintes restrições para a origem e para o destino das mesmas: **not oclIsTypeOf (GroupedFeature)** e **not oclIsTypeOf (RootFeature) and not oclIsTypeOf (GroupedFeature)**, respectivamente. Para as ligações *alternative_or* e *alternative_xor*, foram também criadas restrições OCL para a origem e para o destino das ligações: **not oclIsTypeOf (RootFeature) and not oclIsTypeOf (Feature) and not oclIsTypeOf (GroupedFeature)** e **not oclIsTypeOf (RootFeature) and not oclIsTypeOf (Feature) and not oclIsTypeOf (FeatureGroup)**, respectivamente.

A Figura 8.7 ilustra as propriedades de uma restrição OCL para um nó.

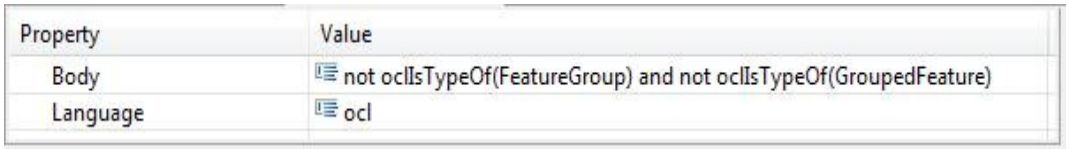

Figura 8.7 Propriedades de uma restrição OCL

8.5 Editor da LDE

As Figuras 8.8 e 8.9 mostram o editor da ferramenta Theme-SPL, e a Figura 8.10 mostra o editor da LDE para o modelo de *features*, onde se pode visualizar, à direita, a palete

responsável por conter todos os elementos e ligações que é possível utilizar na LDE e o resto da área para editar os modelos que se pretende construir. A palete da LDE Theme-SPL está dividida em três partes: Compartimento (*Compartment*), nós (*Themes*) e ligações (*Links*). Por outro lado, a palete da LDE para o modelo de *features* encontra-se dividida em duas partes, em que na primeira se encontram os nós (*Features*) e na segunda parte estão presentes as ligações (*Links*).

Para se criar um modelo com a ferramenta, deve-se colocar ou um compartimento (no caso da LDE Theme-SPL) ou um nó no espaço de edição. Dentro do compartimento colocam-se os conceitos desejados, sendo que qualquer elemento pode ser colocado dentro do compartimento. De seguida, criam-se as ligações desejadas entre elementos. É possível ligar elementos entre diferentes compartimentos, e entre elementos que não estejam em nenhum compartimento, desde que sejam ligações sintacticamente correctas de acordo com o metamodelo definido.

A distribuição dos elementos dentro dos compartimentos fica ao critério do utilizador, sendo o objectivo destes esconder a escalabilidade do modelo, através da colapsação das caixas, e ajudar a estruturar os modelos desenhados.

A Figura 8.9 representa o mesmo diagrama da Figura 8.8, com a diferença de ter os compartimentos colapsados.

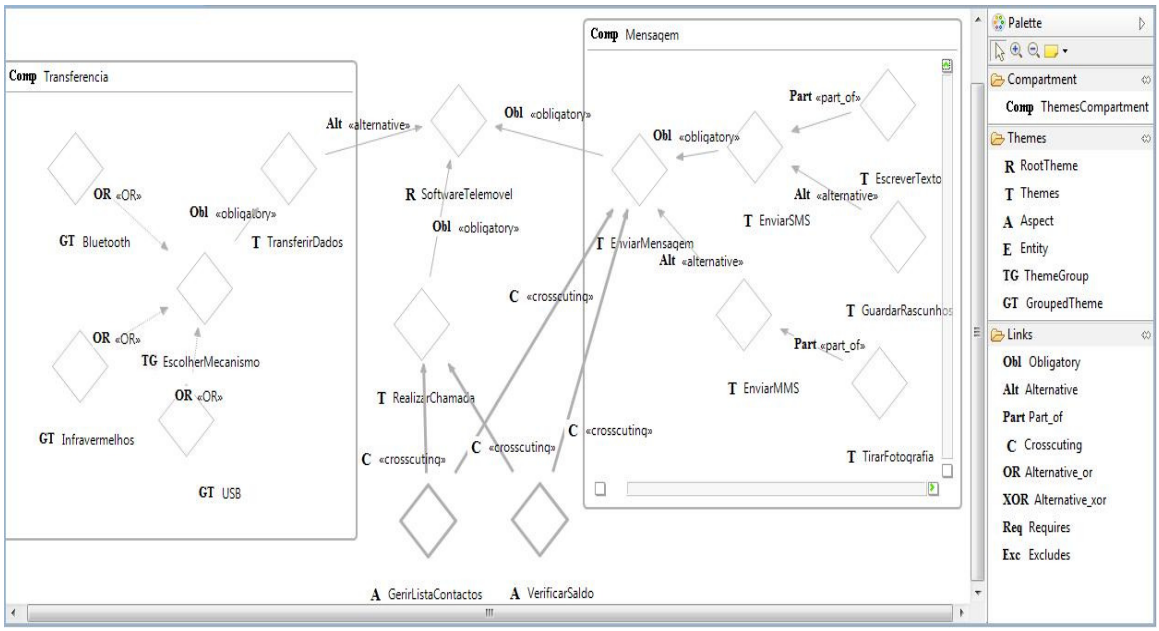

Figura 8.8 Visão Transversal realizada com a LDE Theme-SPL

Após o modelo Theme-SPL elaborado, são aplicadas regras de transformação (secção 8.6), definidas através das heurísticas de mapeamento (secção 5.3), de modelos Theme-SPL para o modelo de *features*. A Figura 8.10 mostra o resultado dessa transformação, aplicado ao modelo apresentado na Figura 8.8. Para mais informações sobre como utilizar a ferramenta, ver o Anexo B.

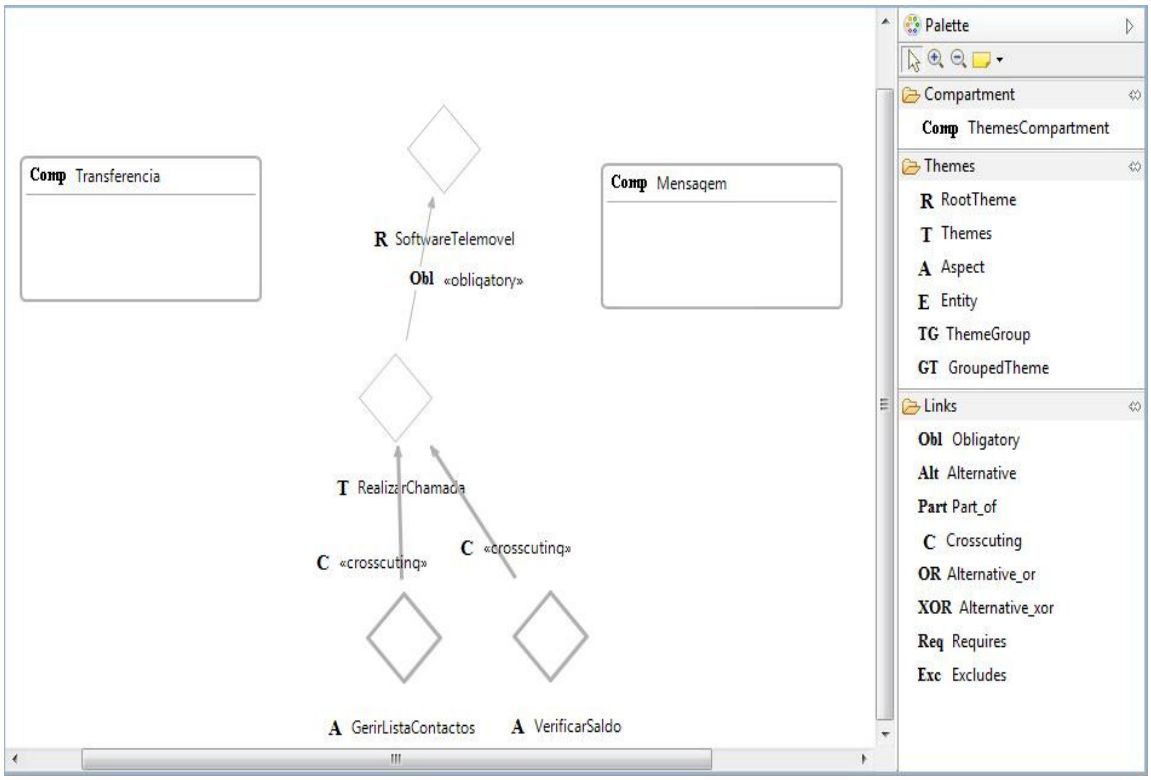

Figura 8.9 Visão Transversal com os compartimentos colapsados realizada com a LDE Theme-SPL

Na Figura 8.10 podemos verificar que a *subfeature* aspectual GerirListaContactos apresentação uma ligação opcional com ambas as *features* e a *subfeature* aspectual VerificarSaldo apresenta uma relação obrigatória com ambas as *features*. Nos casos em que a *subfeature* aspectual apresenta uma relação obrigatória com uma *feature* e opcional com outra, então a *feature* aspectual apresenta um círculo a branco e outro a preto de modo a indicar que uma relação é opcional e que outra é obrigatória.

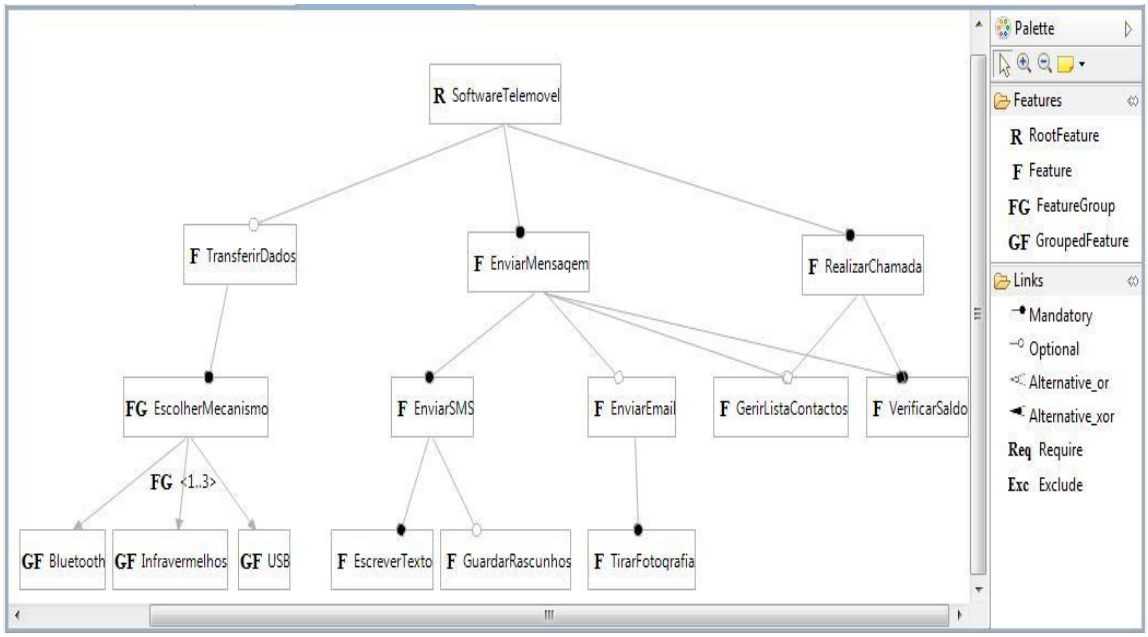

Figura 8.10 Editor da LDE criada para o modelo de *features*

8.6 Transformações ATL

Após criadas as duas LDEs são definidas regras de transformação, cujo objectivo é gerar, através dos modelos da abordagem Theme-SPL, uma visualização do modelo de *features*. Para efectuar a transformação recorreu-se ao metamodelo utilizado para a LDE gráfica, ou seja, o ficheiro Theme-SPL.ecore (Figura 8.1) e através de regras de transformação, utilizando o metamodelo FeatureModel.ecore (Figura 8.2), gera-se o editor gráfico para o modelo de *features*.

O código ATL apresentado em seguida consiste num *Header Section* e em dezasseis *Rules*. O primeiro contém o nome do módulo (Theme-SPL) e os metamodelos de origem (Theme-SPL) e destino (FM). Em cada regra, para cada instância de Theme-SPL cria uma instância do modelo de *features* (FM). As heurísticas de mapeamento referidas são as definidas na secção 5.3.

```
module ThemeSPL; -- Module Template
create OUT : FM from IN : Theme-SPL;
```

A regra para cada instância da abordagem Theme-SPL cria uma instância do modelo de *features* no qual, a ligação *containsFeature* do modelo de *features* corresponde à união de todos os nós representados em Theme-SPL, e a ligação *containsRelation* do modelo de *features* corresponde à união de todos os *links* representados em Theme-SPL.

```
rule FeatureModel{
      from
            p:Theme-SPL!ThemeApproach
      to
      out: FM!FeatureModel(
      containsFeature<-Set Theme!Entity.allInstancesFrom('IN')}
            ->union (Set {Theme!Themes.allInstancesFrom('IN')})
            ->union (Set {Theme!Aspect.allInstancesFrom('IN')})
            ->union (Set {Theme!ThemeGroup.allInstancesFrom('IN')})
            ->union (Set {Theme!GroupedTheme.allInstancesFrom('IN')})
            ->union (Set {Theme!RootTheme.allInstancesFrom('IN')}),
      containsRelation<-Set {Theme!Obligatory.allInstancesFrom('IN')}
             ->union (Set {Theme!Alternative.allInstancesFrom('IN')})
             ->union (Set {Theme!Part_of.allInstancesFrom('IN')})
             ->union (Set {Theme!Crosscuting.allInstancesFrom('IN')})
             ->union (Set {Theme!Alternative_or.allInstancesFrom('IN')})
             ->union (Set {Theme!Alternative_xor.allInstancesFrom('IN')})
             ->union (Set {Theme!Excludes.allInstancesFrom('IN')})
             ->union (Set {Theme!Require.allInstancesFrom('IN')}))}
```

A regra seguinte é obtida através da heurística de mapeamento **HM.1**. Para cada instância *RootTheme* do metamodelo de origem, cria uma instância *RootFeature* do metamodelo de destino, na qual o nome da raiz do modelo de *features* corresponde ao nome da raiz de Theme-SPL.

```
rule RootThemeToRootFeature {
      from
            p: Theme-SPL!RootTheme
      to
            out: FM!RootFeature (nameF <- p.name)}
```

A regra seguinte é obtida através da heurística de mapeamento **HM.2**. A regra, para cada instância *Aspect* do metamodelo de origem, cria uma instância *Feature* do metamodelo de destino, na qual o nome da *feature* corresponde ao nome do aspecto.

```
rule AspectToFeature {
      from
            p: Theme-SPL!Aspect
      to
            out: FM!Feature ( nameF <- p.name)}
```

Esta regra é obtida através da heurística de mapeamento **HM.2**. A regra, para cada instância *Entity* do metamodelo de origem, cria uma instância *Feature* do metamodelo de destino, na qual o nome da *feature* corresponde ao nome da entidade.

```
rule EntityToFeature {
      from
            p: Theme-SPL!Entity
      to
            out: FM!Feature ( nameF <- p.name)}
```

Esta regra é obtida através da heurística de mapeamento **HM.2**. A regra, para cada instância *Themes* do metamodelo de origem, cria uma instância *Feature* do metamodelo de destino, na qual o nome da *feature* corresponde ao nome do tema.

```
rule ThemesToFeature {
      from
            p: Theme-SPL!Themes
      to
            out: FM!Feature ( nameF <- p.name)}
```

A seguinte regra é obtida através da heurística de mapeamento **HM.3**. A regra, para cada instância *ThemeGroup* do metamodelo de origem, cria uma instância *FeatureGroup* do metamodelo de destino, na qual o nome da *feature group* corresponde ao nome do *theme group* e a cardinalidade do grupo é uma S*tring* que contém o número mínimo e máximo de temas que o grupo pode ter, separados por '...'. É necessário que o número mínimo seja sempre menor que o máximo para apresentar a cardinalidade ao utilizador.

```
rule ThemeGroupToFeatureGroup {
      from
            p: Theme-SPL!ThemeGroup
      to
            out: FM!FeatureGroup (
                  nameF <- p.name,
                  groupCardinality<-  if(p.min <=p.max)
                        then p.min+'..'+p.max
                  else
                        ''
                  endif)}
```

Esta regra é obtida através da heurística de mapeamento **HM.4**. A regra, para cada instância *GroupedTheme* do metamodelo de origem, cria uma instância *GroupedFeature* do metamodelo de destino, na qual o nome da *grouped feature* corresponde ao nome do *grouped theme*.

```
rule GroupedThemeToGroupedFeature {
      from
            p: Theme-SPL!GroupedTheme
      to
            out: FM!GroupedFeature (nameF <- p.name)}
```

A regra seguinte é obtida através da heurística de mapeamento **HM.5**. A regra, para cada instância *Part_of* do metamodelo de origem, cria uma instância *Mandatory* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *LinkToNodeFrom*, e à ligação destino, *target feature*, corresponde a ligação *LinkToNodeTo*. A correspondência entre ligações é efectuada de acordo com as

propriedades *Souce Feature* e *Target Feature* definidas no GMFMAP, como é exemplo a Figura 8.5.

```
rule Part_of{
      from
            p: Theme-SPL!Part_of
      to
            out:FM!Mandatory( sourceFeature <-p. LinkToNodeFrom,
                              targetFeature <- p. LinkToNodeTo)}
```

Esta regra é obtida através da heurística de mapeamento **HM.5**. A regra, para cada instância *Obligatory* do metamodelo de origem, cria uma instância *Mandatory* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *LinkToNodeFrom*, e à ligação destino, *target feature*, corresponde a ligação *LinkToNodeTo*.

```
rule Obligatory{
      from
            p: Theme-SPL!Obligatory
      to
            out:FM!Mandatory( sourceFeature <-p. LinkToNodeFrom,
                              targetFeature <- p. LinkToNodeTo)}
```

Esta regra é obtida através da heurística de mapeamento **HM.6**. A regra, para cada instância *Alternative* do metamodelo de origem, cria uma instância *Optional* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *LinkToNodeFrom*, e a ligação destino, *target feature*, corresponde à ligação *LinkToNodeTo*.

```
rule Alternative{
      from
            p: Theme-SPL!Alternative
      to
            out:FM!Optional(  sourceFeature <-p. LinkToNodeFrom,
                              targetFeature <- p. LinkToNodeTo)}
```

Esta regra é obtida através da heurística de mapeamento **HM.7**. A regra, para cada instância *Alternative_or* do metamodelo de origem, cria uma instância *Alternative_or* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *alternativeToThemeGroup*, e a ligação destino, *target feature*, corresponde à ligação *alternativeToGroupedTheme*.

```
rule Alternative_or{
      from
            p: Theme-SPL!Alternative_or
```

```
      to
      out:FM!Alternative_or(sourceFeature <-p.alternativeToThemeGroup,
                        targetFeature <- p.alternativeToGroupedTheme )}
```

Esta regra é obtida através da heurística de mapeamento **HM.8**. A regra, para cada instância *Alternative_xor* do metamodelo de origem, cria uma instância *Alternative_xor* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *alternativeToThemeGroup*, e a ligação destino, *target feature*, corresponde à ligação *alternativeToGroupedTheme*.

```
rule Alternative_xor{
      from
            p: Theme-SPL!Alternative_xor
      to
      out:FM!Alternative_xor( sourceFeature <-p.alternativeToThemeGroup,
                        targetFeature <- p.alternativeToGroupedTheme)}
```

Esta regra é obtida através da heurística de mapeamento **HM.9**. A regra, para cada instância *Crosscuting* do metamodelo de origem, cria uma instância *Mandatory* do metamodelo de destino, se o tipo de ligação de *crosscuting* for "*Obligatory*". A ligação origem, *source feature*, corresponde à ligação *crosscutingToTheme*, e a ligação destino, *target feature*, corresponde à ligação *crosscutingToAspect*.

```
rule CrosscutingObligatory{
      from
            p: Theme-SPL!Crosscuting
            (p.type = 'Obligatory')
      to
            out:FM!Mandatory( sourceFeature <-p.crosscutingToTheme,
                              targetFeature <- p. crosscutingToAspect)}
```

Esta regra é obtida através da heurística de mapeamento **HM.9**. Esta regra é obtida através da heurística de mapeamento **HM.9**. A regra, para cada instância *Crosscuting* do metamodelo de origem, cria uma instância *Optional* do metamodelo de destino, se o tipo de ligação de *crosscuting* for "*Alternative*". A ligação origem, *source feature*, corresponde à ligação *crosscutingToTheme*, e a ligação destino, *target feature*, corresponde à ligação *crosscutingToAspect*.

```
rule CrosscutingAlternative{
      from
            p: Theme-SPL!Crosscuting
            (p.type = 'Alternative')
      to
            out:FM!Optional(  sourceFeature <-p.crosscutingToTheme,
                              targetFeature <- p.crosscutingToAspect)}
```

Esta regra é obtida através da heurística de mapeamento **HM.10**. A regra, para cada instância *Require* do metamodelo de origem, cria uma instância *Require* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *LinkToNodeFrom*, e a ligação destino, *target feature*, corresponde à ligação *LinkToNodeTo*, e o *id* do modelo de *features* corresponde ao *id* de Theme-SPL.

```
rule Requires{
      from
            p: Theme-SPL!Require
      to
            out:FM!Require(   sourceFeature <- p. LinkToNodeFrom,
                              targetFeature <- p. LinkToNodeTo,
                              id <- p.id)}
```

Esta regra é obtida através da heurística de mapeamento **HM.10**. A regra, para cada instância *Excludes* do metamodelo de origem, cria uma instância *Exclude* do metamodelo de destino, na qual a ligação origem, *source feature*, corresponde à ligação *LinkToNodeFrom*, e a ligação destino, *target feature*, corresponde à ligação *LinkToNodeTo*, e o *id* do modelo de *features* corresponde ao *id* de Theme-SPL.

```
rule Excludes{
      from
            p: Theme-SPL!Excludes
      to
            out:FM!Exclude(   sourceFeature <-p. LinkToNodeFrom,
                              targetFeature <- p. LinkToNodeTo,
                              id <- p.id)}
```

8.7 Sumário

Neste capítulo foram descritos os passos de criação da ferramenta, nomeadamente qual a estratégia utilizada para abordar o problema, o desenho de ambos os modelos Ecore e quais os resultados obtidos, mais concretamente os editores criados e quais os conceitos que implementam. Foi também apresentado o código ATL usado para a transformação dos modelos Theme-SPL para o modelo de *features*.

No próximo capítulo vai ser descrito como foi feita a validação dos resultados.

9. Avaliação da Linguagem de Domínio Específico para Theme-SPL

A validação efectuada teve como principais objectivos avaliar que a ferramenta Theme-SPL é útil e cumpre os seus objectivos. A avaliação foi efectuada através de um inquérito com questões sobre a sintaxe da linguagem, facilidade de utilização da ferramenta e níveis de satisfação com a ferramenta. O inquérito foi feito a um grupo de dez alunos com conhecimento sobre abordagens de engenharia de requisitos e sobre linguagens para domínios específicos, que tenham experimentado numa cadeira da área de Engenharia de Software, utilizando o caso de estudo Observador Sanitário (Anexo A). Foi também validada a expressividade da LDE utilizando o caso de estudo *Smart Home* (referido na secção 6.1) para verificar se a sintaxe abstracta definida no metamodelo se mantinha nos modelos criados com a ferramenta.

9.1 Questionário

Para validar a ferramenta foi efectuado um questionário, baseado em (Norman et al., 1998). O questionário foi efectuado a um grupo de 10 indivíduos, com experiência em LDEs e conhecimento do modelo de *features*. O questionário encontra-se dividido em duas partes. Inicialmente é feita uma pequena introdução ao utilizador a explicar a abordagem Theme-SPL, mostrando o seu modelo para o caso de estudo, e é pedido, tendo em conta os conhecimentos adquiridos nas cadeiras de mestrado "Engenharia de Requisitos e Desenvolvimento de Software" e de "Linguagens para Domínios Específicos", para elaborar o modelo de *features* de acordo com o modelo Theme-SPL mostrado e com o caso de estudo em questão. O caso de estudo utilizado foi o caso do Observador Sanitário, onde o utilizador pode efectuar queixas e consultar informação sobre as mesmas. Na segunda parte é pedido para responderem a um conjunto de questões escalares sobre a ferramenta. É também pedido para responderem a um conjunto de questões de resposta aberta onde os utilizadores podem dar a sua opinião sobre os pontos fortes/fracos da ferramenta testada. Após os utilizadores elaborarem na LDE o

modelo de *features*, o mesmo é gerado a partir do modelo Theme-SPL e são analisadas as diferenças entre o modelo gerado e o modelo elaborado pelo utilizador. O questionário detalhado pode ser consultado no Anexo A.

9.2 Resultados da Validação

O resultado das questões foi bastante positivo. Os utilizadores consideraram a ferramenta intuitiva, de fácil aprendizagem e fácil de utilizar. Referiram também que após aprender o modelo Theme-SPL se torna simples elaborar o modelo de *features*, e que a transformação de um modelo orientado a aspectos para o modelo de *features* é uma mais-valia.

Em relação às questões de resposta escalar, as respostas são apresentadas de forma gráfica nas Figuras de 9.1 a 9.10. Na escala usada para representar as respostas dos utilizadores, "1" representa o valor pior (por exemplo, "Mais Difícil") e "5" representa o valor melhor (por exemplo, "Mais Fácil") da escala.

A Figura 9.1 mostra se os utilizadores compreenderam o modelo Theme-SPL. A resposta foi unânime, tendo todos os utilizadores respondido "Sim".

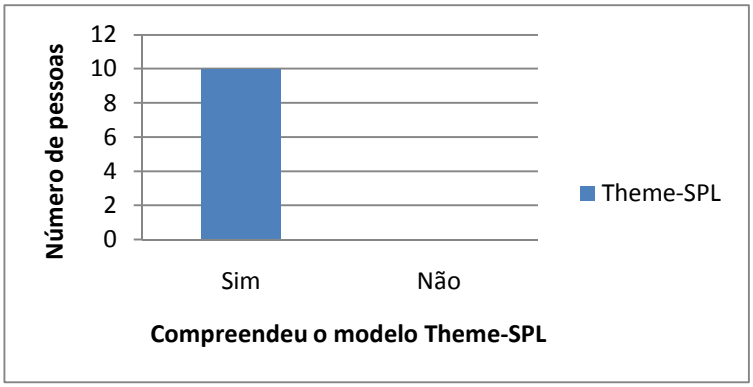

Figura 9.1 Compreensão do modelo Theme-SPL

Em relação ao grau de dificuldade em identificar as *features*, Figura 9.2, não houve grande dificuldade por parte dos utilizadores em identificar as mesmas.

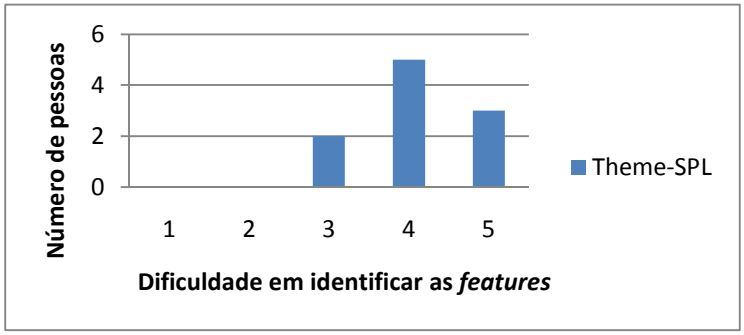

Figura 9.2 Grau de dificuldade em identificar as *features*

A Figura 9.3 mostra que a maioria dos utilizadores achou que o modelo Theme-SPL contribuiu bastante para ajudar a identificar as *features*, assim como para construir o modelo de *features* (Figura 9.4), tornando-se, de uma maneira geral, "mais fácil" construir o modelo de *features* com base no modelo Theme-SPL, como ilustra a Figura 9.5.

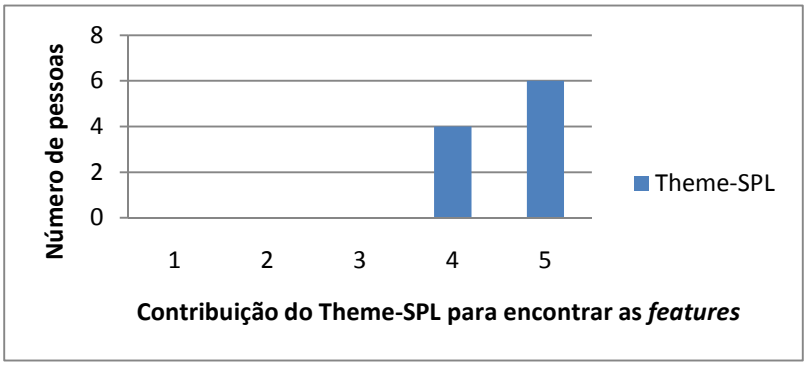

Figura 9.3 Contribuição do modelo Theme-SPL para identificar as *features*

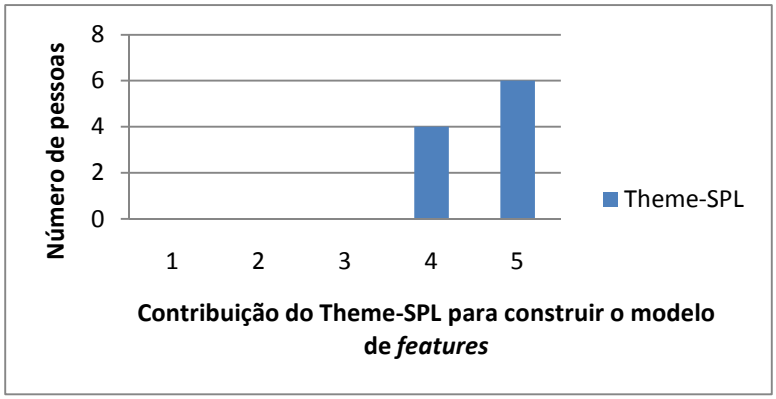

Figura 9.4 Contribuição do modelo Theme-SPL para a construção do modelo de *features*

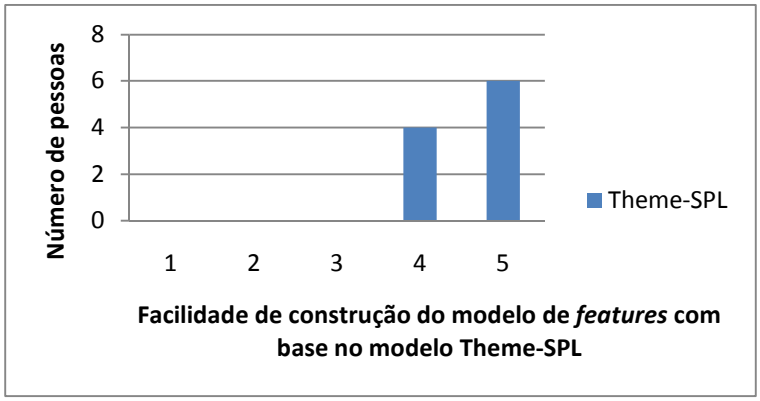

Figura 9.5 Facilidade na criação do modelo de *features* com base no modelo Theme-SPL

Após os utilizadores elaborarem o modelo de *features*, é efectuada a transformação, com a ferramenta, do modelo Theme-SPL para o modelo de *features*, e a maioria dos utilizadores obteve um modelo muito semelhante ao modelo gerado, como mostra a Figura 9.6.

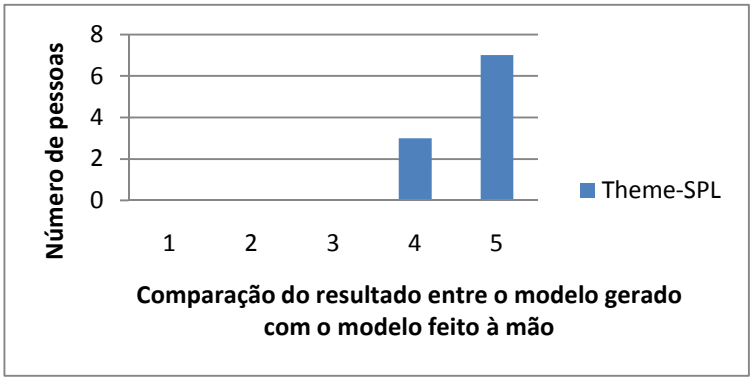

Figura 9.6 Comparação do resultado entre o modelo gerado e o modelo desenhado pelo utilizador

Foi pedido aos utilizadores, para então compararem o modelo gerado com o modelo elaborado por eles, e identificarem o que acharam que o modelo gerado fez melhor do que o elaborado. Os utilizadores referiram que o modelo gerado apresenta um melhor nível de organização, uma vez que está organizado visualmente por níveis, permite validação sintáctica, o tempo dispendido é muito menor e está menos sujeito a erros.

Relativamente à questão "O que fez o modelo gerado pior que o modelo elaborado?" as respostas foram unânimes e todos responderam que não houve nada que o modelo gerado tivesse feito pior.

Na Figura 9.7 são apresentados os resultados em relação à utilidade da ferramenta e, em geral, todos os utilizadores consideraram a ferramenta muito útil.

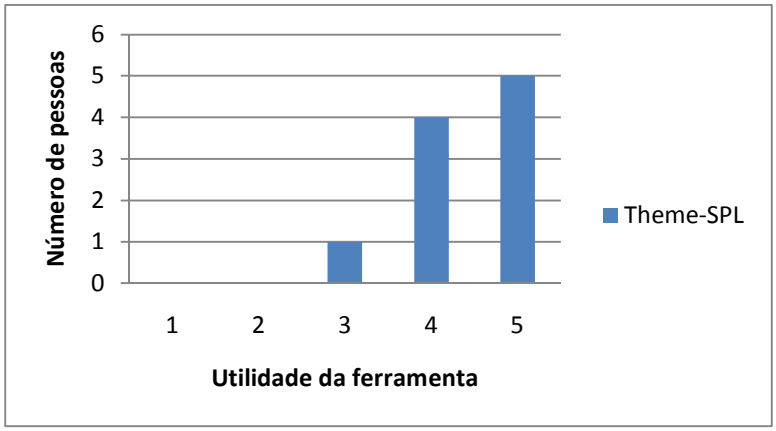

Figura 9.7 Utilidade da ferramenta

Relativamente à questão sobre a dificuldade em utilizar a ferramenta não houve, por parte dos utilizadores, dificuldades em utilizar a mesma, como mostra a Figura 9.8.

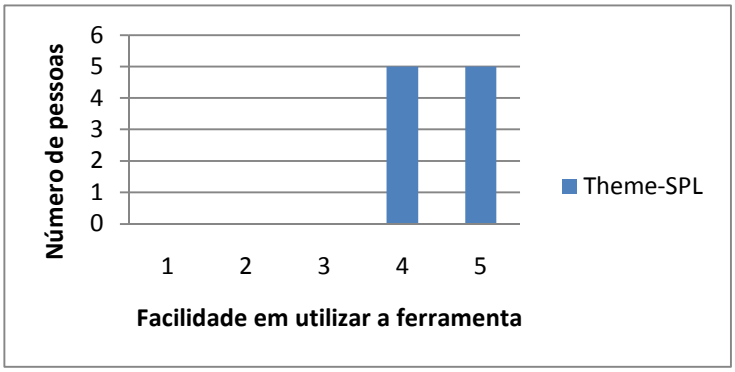

Figura 9.8 Dificuldade em utilizar a ferramenta

Em relação ao que os utilizadores acharam sobre o modo como a ferramenta lida com o problema de escalabilidade dos modelos Theme-SPL, consideraram que lida bem com o problema uma vez que permite colapsar as partes do modelo que o utilizador não pretende ver no momento, como ilustra a Figura 9.9.

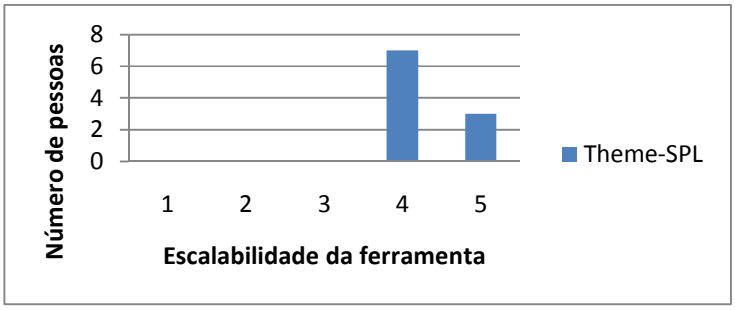

Figura 9.9 Escalabilidade nos modelos Theme-SPL

Foi colocada a questão para saber qual a opinião dos utilizadores sobre se os mecanismos de desenvolvimento orientado a aspectos são vantajosos para a construção do modelo de *features*, em que todos os utilizadores deram resposta positiva, Figura 9.10, tendo referido que identifica atempadamente os aspectos, o que trás uma grande contribuição ao modelo uma vez que estes devem ser identificados o mais cedo possível pois irão influenciar as posteriores fases do ciclo de vida do software, inclusive em LPS. Foi também referido que permitem estabelecer *features* iguais em vários pontos do modelo sendo uma grande vantagem uma vez que não há necessidade de "redesenhar" partes do modelo de *features*.

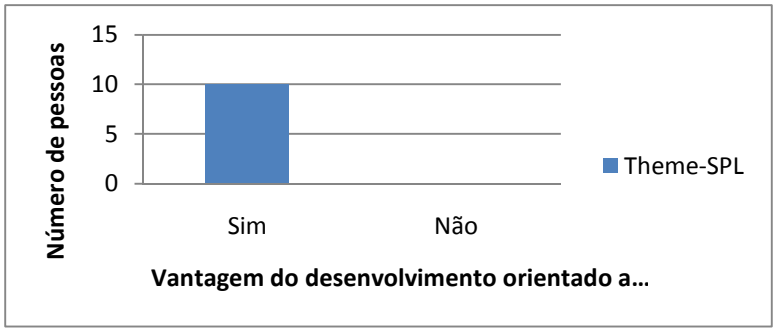

Figura 9.10 Vantagem do desenvolvimento orientado a aspectos para construção do modelo de *features*

Opinião dos utilizadores

Um grande número de comentários e sugestões foram feitos nas questões de resposta aberta. Para a questão "Quais os pontos fortes desta ferramenta?", as respostas foram:

- Faz a ponte entre dois modelos;
- Faz a ponte para uma abordagem conhecida;

- Obtenção do modelo de *features* em apenas dois cliques, e através de um modelo orientado a aspectos;
- A transformação é muito útil;
- Possibilidade de expandir e colapsar compartimentos permitindo uma maior escalabilidade dos modelos;
- Os ícones das *labels* são intuitivos.

Para as questões "Quais os pontos fracos da ferramenta?" e "O que sugere para que a ferramenta possa ser melhorada?", os detalhes de interface que foram sugeridos para serem reforçados foram:

i. Na palete da LDE, ao passar o rato pelas diversas opções deveria mostrar um texto a explicar a sua utilização;

ii. A raiz da abordagem Theme-SPL deveria ter uma cor que a diferencie das outras;

iii. Colocar os nomes dos nós que não se encontram dentro da figura mais perto da mesma, de modo a saber a que figura pertence cada nó;

iv. Colocar uma *String* "Input" nas *features* que são mapeadas de entidades;

v. Alterar a notação das alternativas OR e XOR para conter o arco a ligar as *features*.

Após a análise das sugestões dadas e dos pontos negativos, foram efectuadas alterações na LDE, tendo em conta as sugestões dos utilizadores.

A sugestão (i) foi considerada para ambas as LDEs, Theme-SPL e modelo de *features*, sendo que, ao passar o rato nos diversos *links* da palete é mostrado um texto relativo aos nós entre os quais o *link* permite efectuar uma ligação. A Figura 9.11 ilustra a palete da LDE para o modelo de *features* com a respectiva alteração.

A cor da raiz na abordagem Theme-SPL foi alterada, sugestão (ii), uma vez que os utilizadores sentiram alguma dificuldade em encontrá-la por ter nomenclatura semelhante aos temas, apenas tinha a *label* R a diferenciá-la, então colocou-se o fundo da raiz a cinzento. A Figura 9.12 ilustra a sugestão referida.

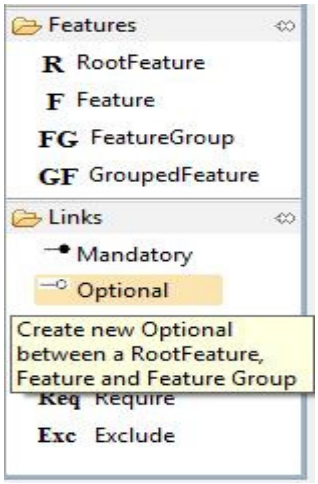

Figura 9.11 Palete da LDE para o modelo de *features* após efectuada a alteração (i)

Foi também efectuada a sugestão de colocar os nomes dos nós mais perto das figuras, sugestão (iii), tendo para isso que se alterar o código Java gerado, correspondente a cada figura. Para cada figura cujo nome está fora da mesma foi alterado o código de:

```
locator.setBorderItemOffset(new Dimension(-20, -20)); para:
locator.setBorderItemOffset(new Dimension(-1, -1));
```

A Figura 9.12 ilustra as alterações (ii) e (iii) efectuadas no modelo Theme-SPL após as sugestões dos utilizadores.

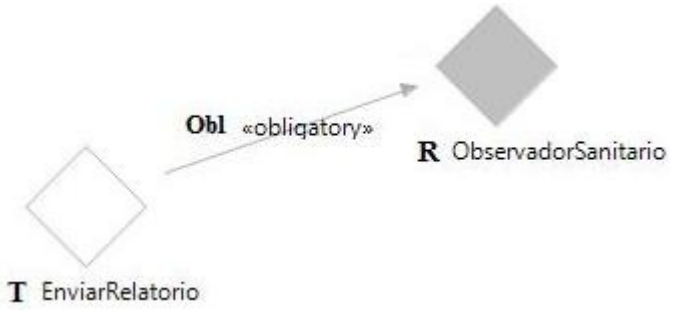

Figura 9.12 Modelo Theme-SPL após efectuadas as alterações (ii) e (iii)

Uma outra sugestão que foi considerada foi a sugestão (iv). Por exemplo, no caso de estudo do Observador Sanitário (Anexo A) para o tema "Inserir Queixa" em Theme-SPL é necessário estar presente a entidade "Queixoso" uma vez que é este que irá interagir com o sistema inserindo a queixa, mas para o modelo de *features*, o que existe é o input do queixoso e não o queixoso. Esta alteração foi efectuada, sendo que ao mapear

entidades, do modelo Theme-SPL para *features,* é inserida a S*tring* "Input" antes do nome da respectiva *feature*. Sendo esta mesma alteração ilustrada na Figura 9.13.

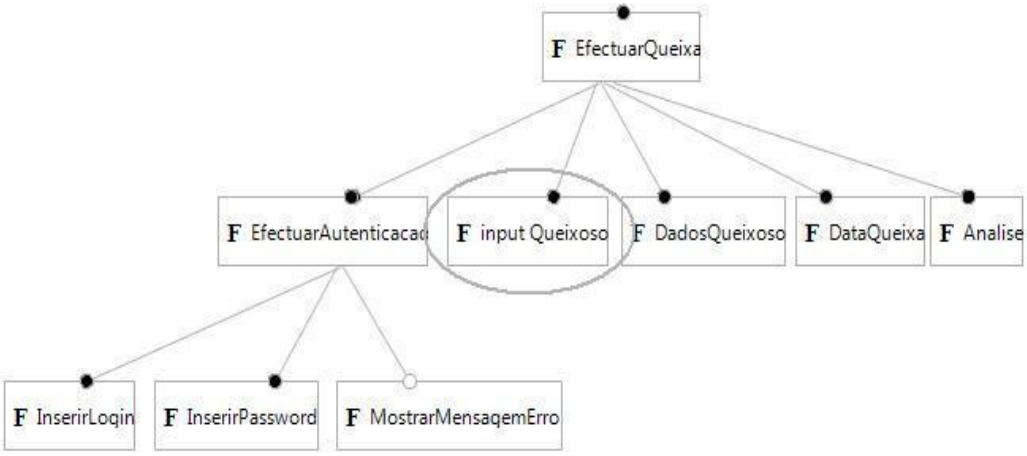

Figura 9.13 Modelo de *features* após efectuada a alteração iv.

Relativamente à sugestão para alterar a notação das alternativas OR e XOR para conter o arco a ligar as *features*, não é possível efectuar a mesma, uma vez que é uma limitação da plataforma eclipse.

9.3 Caso de estudo *Smart Home*

Para validação adicional da ferramenta, foi escolhido o caso de estudo *Smart Home*. Foi retirado do projecto Europeu AMPLE (AMPLE, 2009), sendo o exemplo de um caso industrial realizado na vida real. Este caso de estudo tem como foco analisar as configurações e serviços necessários numa casa inteligente, como foi descrito na secção 6.1.

9.3.1 Modelação com Theme-SPL

Para o caso de estudo foi modelada a Visão de Relação de Temas obtida na secção 6.1 (Figura 6.3), na LDE criada, sendo o resultado apresentado na Figura 9.14.

Aplicando as regras de transformação, descritas na secção 8.6, iremos obter o modelo de *features* correspondente à visão de relação elaborada na Figura 9.14, que como se pode verificar é bastante semelhante ao modelo obtido na Figura 6.4, secção 6.1. A Figura 9.15 mostra o modelo de *features* resultante da transformação aplicada.

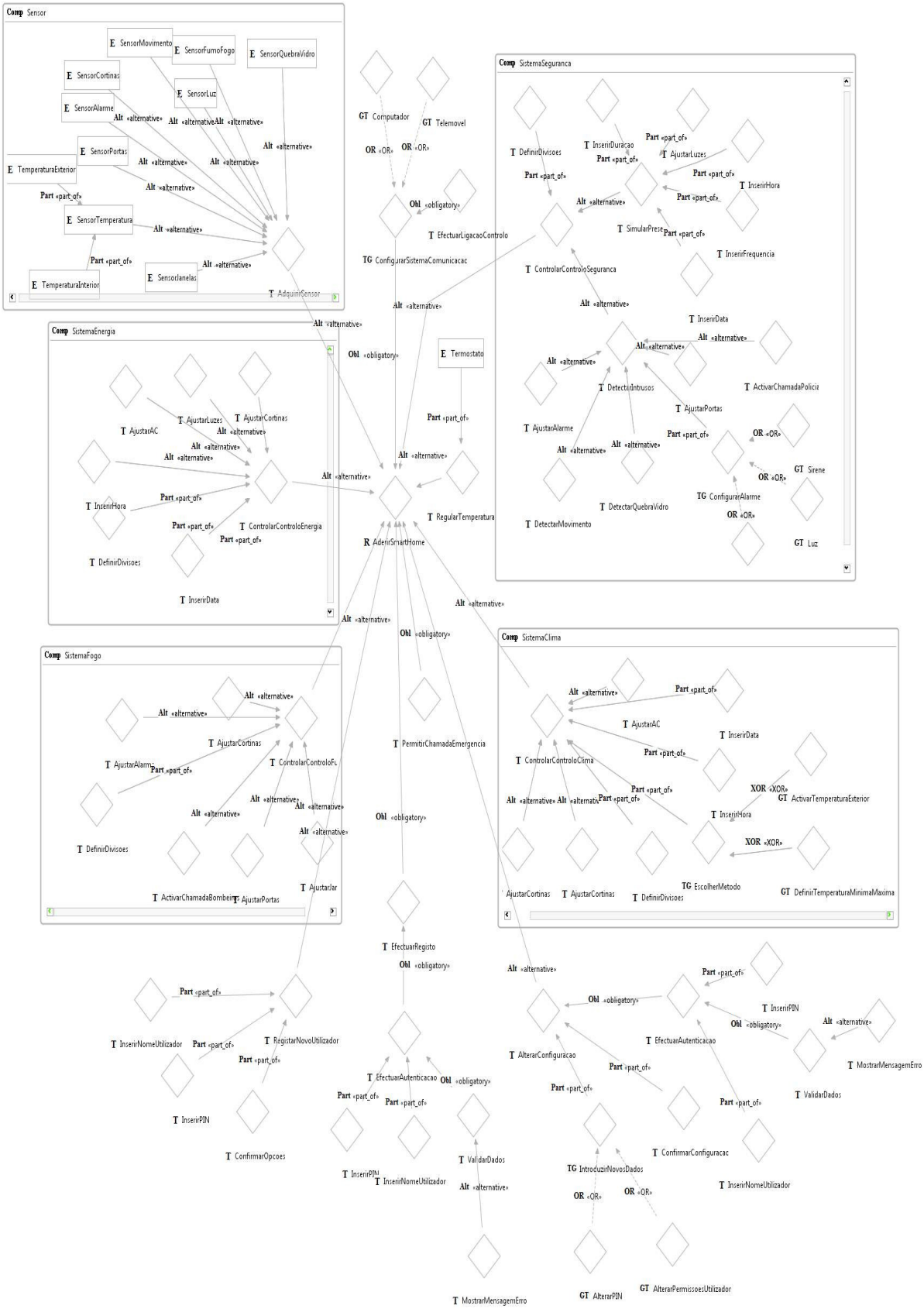

Figura 9.14 Visão de Relação de Temas para o caso de estudo *Smart Home* usando a LDE Theme-SPL

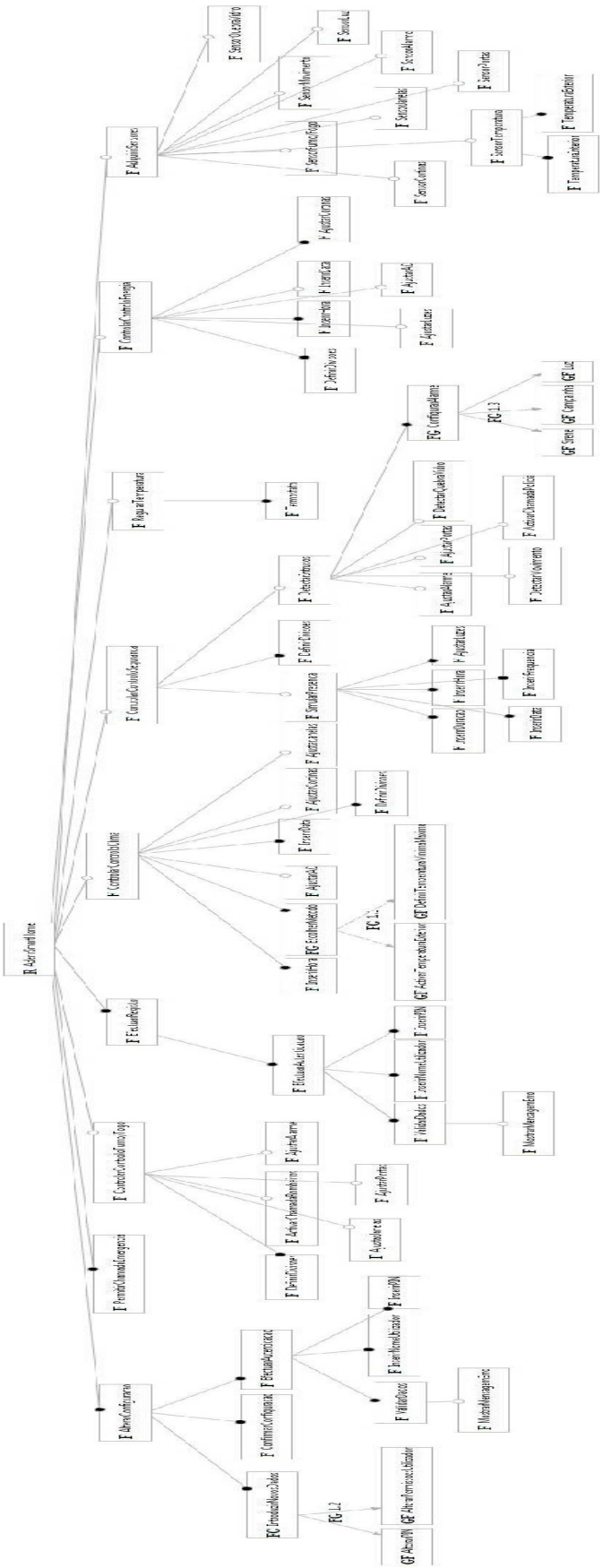

Figura 9.15 Modelo de *features* resultante da transformação

Foi modelada a Visão Transversal obtida na secção 6.1 (Figura 6.5), na LDE criada, sendo o resultado apresentado na Figura 9.16.

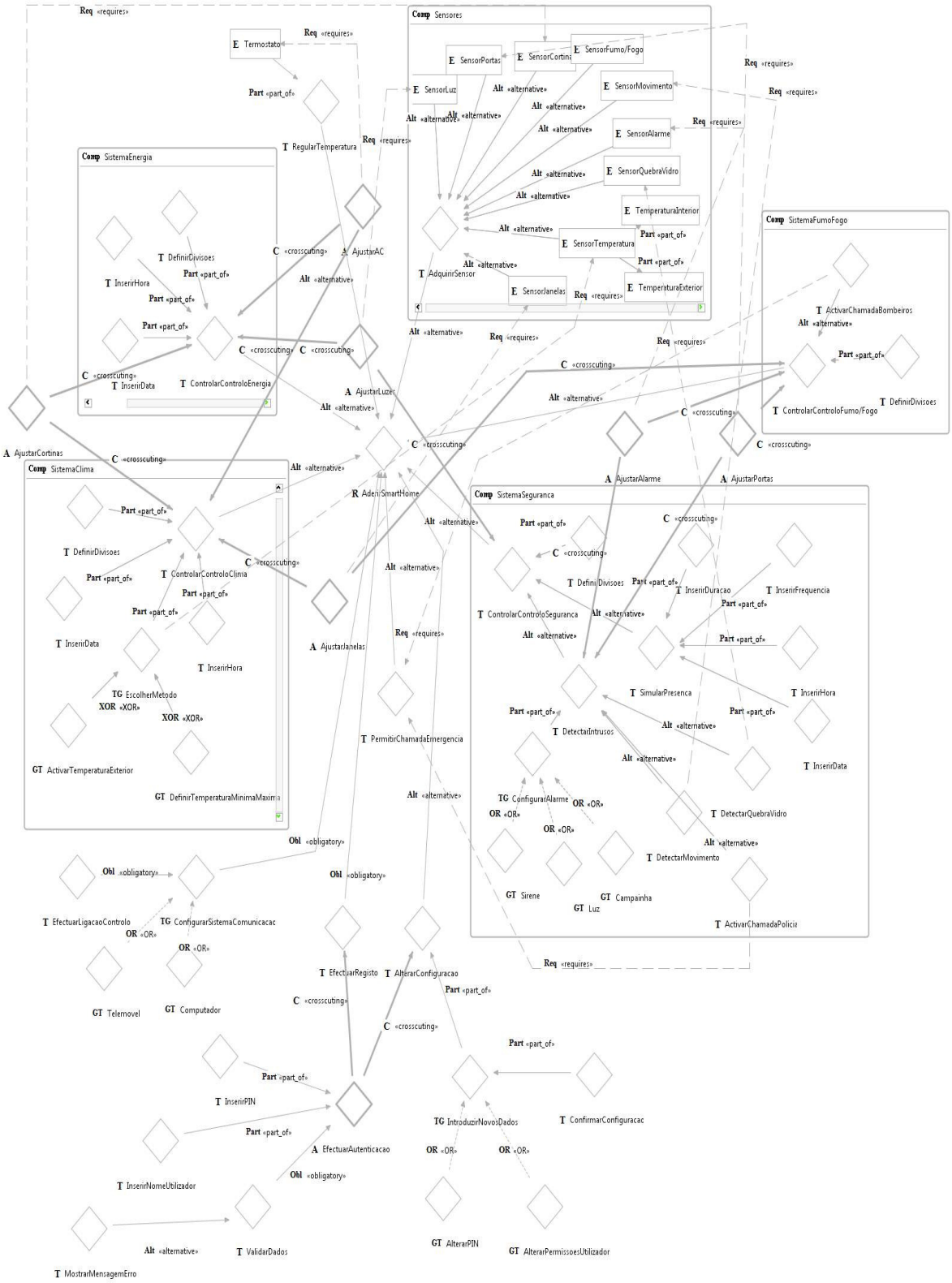

Figura 9.16 Visão Transversal para o caso de estudo *Smart Home* usando a LDE Theme-SPL

A Figura 9.17 mostra o modelo de *features* resultante da transformação aplicada, que como se verifica é semelhante ao obtido na Figura 6.6, secção 6.1.

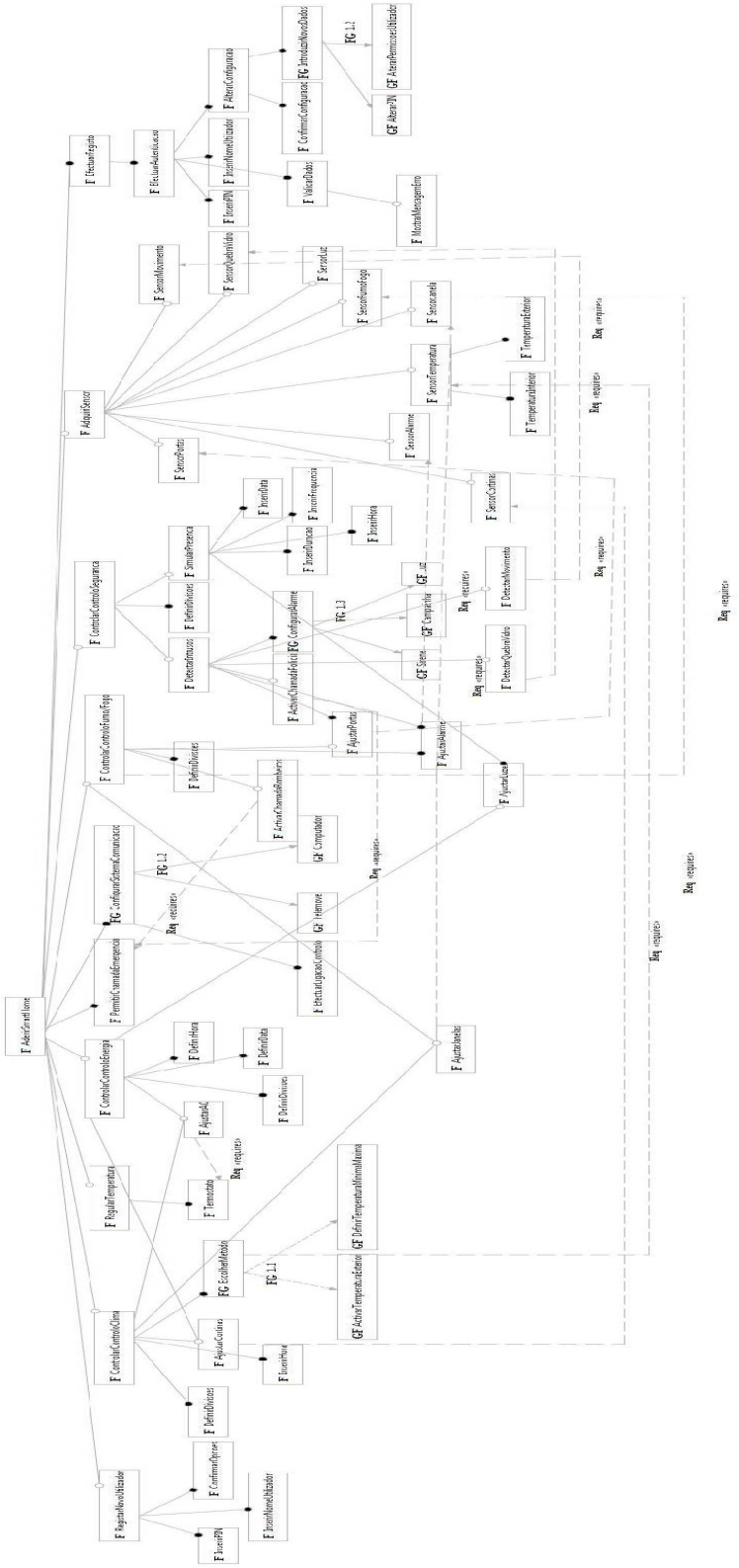

Figura 9.17 Modelo de *features* resultante da transformação

Com a análise das Figuras 9.14 a 9.17, podemos concluir que a sintaxe abstracta é mantida nos modelos criados com a ferramenta, uma vez que os modelos elaborados na LDE são bastante semelhantes aos modelos apresentados na secção 6.1.

Em suma, não alterando os modelos de Theme tornar-se-ia mais difícil transformar os mesmos em modelos de *features*, a não ser que se definisse uma abordagem baseada em processamento de linguagem natural para obter os modelos de *features* através de modelos Theme, que não é o objectivo desta dissertação.

Estendendo a abordagem Theme com conceitos LPS torna-se possível, através de regras de transformação, obter modelos de *features* através de modelos Theme-SPL de forma sistemática.

9.4 Sumário

Neste capítulo foram descritos os passos de como foi realizada a validação da ferramenta. A validação foi efectuada de dois modos. Utilizou-se um caso de estudo de complexidade média para avaliar se as novas construções na linguagem inseriram limitações na expressividade. E, realizou-se um questionário com um grupo de utilizadores.

No próximo capítulo são apresentadas as conclusões finais a retirar do trabalho, as limitações do mesmo e o trabalho futuro a desenvolver.

10. Conclusão

10.1 Contribuições

Esta dissertação apresentou uma abordagem de como se podem desenvolver Linhas de Produtos de Software, através da modelação de aspectos e das várias maneiras de os satisfazer, recorrendo à abordagem Theme. Pelo que, as principais contribuições desta dissertação são: a beneficiação da Engenharia de Requisitos para Engenharia de Linhas de Produtos de Software com a Engenharia de Requisitos Orientada a Aspectos, e a extensão da utilização da abordagem Theme para as linhas de produtos.

A abordagem Theme-SPL é composta por dois processos principais, Engenharia de Domínio e Engenharia de Aplicação, tal como a ELPS. Os processos foram aplicados de dois modos: num primeiro modo, analisando a abordagem Theme, foi definido um conjunto de heurísticas para elaborar o modelo de *features* através de modelos de Theme; no segundo modo foi definido um conjunto de heurísticas para estender a abordagem Theme com conceitos de LPS para usar Desenvolvimento Orientado a Modelos. Estas heurísticas são elaboradas tendo em conta as heurísticas definidas no primeiro modo e verificando o que se precisa dos requisitos para se poder gerar o modelo de *features*.

O segundo modo foi aplicado desenvolvendo-se uma ferramenta, Linguagem de Domínio Específico, de apoio à abordagem. Esta ferramenta permite que através da análise de requisitos e de modelos Theme-SPL, sejam aplicadas um conjunto de regras de transformação para identificar as *features* e elaborar o seu modelo. Ou seja, tendo em conta os modelos de Theme-SPL, são aplicadas um conjunto de regras de transformação, definidas em *Atlas Transformation Language* (ATL), que permitem fazer o mapeamento automático de Theme-SPL para o modelo de *features*.

A abordagem foi aplicada em dois casos de estudo, Software de Telemóveis e *Smart Home*. Para a validação da ferramenta foram utilizados os casos de estudo *Smart Home* e Observador Sanitário.

10.2 Limitações do Trabalho

Embora a validação tenha ajudado muito com os alunos de mestrado que têm conhecimentos em LPS, nomeadamente no modelo de *features*, e em Engenharia de Requisitos Orientado a Aspectos, tendo os mesmos obtido aprovação nas cadeiras de Engenharia de Requisitos e Desenvolvimento de Software e de Linguagens para Domínios Específicos, a validação seria ainda melhor se aplicada a pessoas da indústria com conhecimento em desenvolvimento orientado a aspectos e LPS, e portanto idealmente seria melhor ter uma validação industrial para ter uma avaliação mais rigorosa.

10.3 Trabalho Futuro

Como trabalho futuro pode-se criar uma ferramenta que transforme, através de linguagem natural, os modelos de Theme (originais) em modelos de *features*. Também se podem realizar outros casos de estudo baseados em sistemas reais aplicando a abordagem Theme-SPL de maneira a melhorá-la.

Estes são apenas alguns dos possíveis trabalhos que se poderão realizar futuramente e que poderão melhorar a abordagem Theme-SPL e consequentemente a engenharia de requisitos das Linhas de Produto de Software.

Bibliografia

(António, 2009) António S. (2009). *Adaptação da Framework i* para Linhas de Produtos*. Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia. Lisboa.

(António et al., 2009) António S., Araújo J., Silva C., *Adapting the Framework i* for Software Product Lines*, Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGIM 2009), Workshop da ER'09, Lecture Notes in Computer Science, Vol. 5833, Springer-Verlag, Novembro de 2009.

(Alencar et al., 2006) Alencar F., Castro J., Moreira A., Araújo J., Silva C., Ramos R., Mylopoulos J., *Integration of Aspects with i* Models*, 8th International Bi-Conference Workshop on Agent-Oriented Information Systems, Maio 2006, Hakotade, Japão.

(Alferez el at., 2008) Alferez M., Kulesza U., Moreira A., Araújo J., and Amaral V., *Traceability between Features and UML-Based Requirements Models: A Model-Driven Approach for Product Lines Engineering*, in (submited to the 20th Conference on Advanced Information Systems Engineering - CAISE'08), Montpellier, France, Springer, Junho 2008.

(AMPLE, 2009) AMPLE, "Ample Project", http://www.ample-project.net/. Último acesso: Junho 2009

(Araújo et al., 2002) Araújo J., Moreira A., Brito I., Rashid A., *Aspect-Oriented Requirements with UML*. Workshop on Aspect-Oriented Modeling with UML. 2002. Abril 22-26, Enschede, Holanda.

(Araújo et al., 2004) Araújo J., Whittle J., Kim, D., *Modeling and Composing Scenario-Based Requirements with Aspects*, presented at 12th IEEE International Requirements Engineering Conference (RE2004), Kyoto, Japão, Setembro 2004.

(Araújo et al., 2005) Araújo J., Baniassad E., Clements P., Moreira A., Rashid A., Tekinerdogan B. (2005). *Early Aspects: The Current Landscape*. TR: CMU/SEI-2005-TN-xxx.http://trese.cs.utwente.nl/earlyaspects-AOSD2005/Papers/EarlyAspects-LandscapePaper-FirstDraft-2005.pdf.

(AspectJ Project, 2007) AspectJ Project (2007). http://www.eclipse.org/aspectj/. Último acesso: Junho 2009

(ATL, 2009) *Atlas Transformation Language* (ATL), Página Web: http://wiki.eclipse.org/ATL/Concepts. Último acesso: Novembro 2009

(ATL User Guide, 2009) *Atlas Transformation Language* (ATL), User Guide. Página Web: http://wiki.eclipse.org/ATL/User_Guide. Último acesso: Novembro 2009

(Baptista, 2009) Baptista D. (2009). *Adaptar a Abordagem KAOS para Especificar Linhas de Produtos de Software*. Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia. Lisboa.

(Baptista et al., 2009) Baptista D., Araújo J. e Silva C. (2009). *Adaptar a abordagem KAOS para especificar as Linhas de Produtos de Software.* CAPSI2009. Viseu, Portugal.

(Chitchyan et al. 2006) Chitchyan R., Khan S., Rashid A., *Modelling and Traceability of Composition Semantics in Requirements*, em Workshop Early Aspects AOSD'06, 2006

(Dijkstra, 1976) Dijkstra E., *A Discipline of Programming*, Prentice-Hall, 1976.

(Baniassad e Clarke, 2004) Baniassad E., Clarke S.*, Theme: An approach for aspect-oriented analysis and design*, 26th International Conference on Software Engineering (ICSE), IEEE Press, Edinburgh, Escócia, Maio 2004.

(Baniassad et al., 2006) Baniassad E., Clements C., Araújo J., Moreira A., Rashid A., Tekinerdogan B., *Discovering Early Aspects*, IEEE software, 2006.

(Bezivin, 2005) Bezivin J., *On the Unification Power of Models, Software and System Modeling* (SoSym) 4(2):171–188, 2005

(Bézivin et al., 2005) Bézivin J., Hillairet G., Jouault F., Kurtev I., Piers W., *Bridging the MS/DSL Tools and the Eclipse Modeling Framework*. In Proceedings of the International Workshop on Software Factories at OOPSLA 2005, San Diego, Estados Unidos da América, 2005.

(Blair et al., 2004) Blair G., Blair L., Rashid A., Moreira A., Araújo J.and Chitchyan R. (2004), *Engineering Aspect-Oriented Systems*. Capítulo no livro Aspect-Oriented Software Development. Editor(s): M. Aksit, S. Clarke, T. Elrad, R. Filman.

(Bonifácio e Borba, 2009) Bonifácio R., Borga P., *Modeling Scenario Variability as Crosscutting Mechanisms* 8th ACM International Conference on Aspect-Oriented Software Development (AOSD'09), Charlottesville, Virginia (USA), Março 2-6, 2009

(Brito e Moreira, 2004) Brito I., Moreira A., *Integrating the NFR framework in a RE model*, Workshop Proceedings of Early Aspects 2004, Março 2004, Lancaster, Reino Unido, pp. 28-34.

(Brito et al., 2007) Brito I., Vieira F., Moreira A., Ribeiro R., *Handling Conflicts in Aspectual Requirements Compositions*, Transactions on Aspect-Oriented Software Development: Special Issue on Early Aspects, 2007.

(Clarke, 2002) Clarke S., *Extending standard UML with model composition semantics*, Science of Computing Programming, 2002.

(Clarke e Baniassad, 2005) Clarke S., Baniassad E., L. A. *Aspect Oriented Analysis and Design*, Addison-Wesley Professional, ISBN 0321246748, Abril, 2005.

(Clarke e Walker, 2001) Clarke S., Walker R., *Composition patterns: An approach to designing reusable aspects*. International Conference on Software Engineering, 2001.

(Classen et al., 2001) Classen A., Heymans P., Laney R., Nuseibeh B., Tun T. (2001). *First International Workshop on Variability Modelling of Software-intensive Systems*. Janeiro. Irlanda. http://www.sse.uni-due.de/vamos/2007/index.html. Último acesso: Maio 2009.

(Clements e Northrop, 2002) Clements P., Northrop L., 2002, *A Framework for Software Product Line Practice*, 1st edition Boston, MA, USA: Addison-Wesley.

(Chitchyan et al., 2005) Chitchyan R., Rashid A., Sawyer P., Garcia A., Pinto M., Bakker J., Tekinerdogan B., Clarke S., Jackson A., *Survey of Aspect-Oriented Analysis and Design Approaches,* AOSD-Europe, Maio 2005.

(Czarnecki e Helsen, 2003) Czarnecki K., Helsen S., *Classification of model transformation approaches*, In OOPSLA 2003 Workshop on Generative Techniques in the context of Model Driven Architecture, Anaheim, Califórnia, USA, Outubro 2003.

(Czarnecki et al., 2004) Czarnecki K., Helsen S., Eisenecker U., *Staged Configuration Using Feature Models*, Software Product Lines: Third International Conference SPLC 2004: Boston MA USA, 2004.

(Czarnecki et al., 2005) Czarnecki K., Helsen S., Eisenecker U., *Staged configuration through specialization and multi-level configuration of feature models*. In: Practice SPIa (ed), 2005; 143-169.

(Declarative QVT, 2009) Declarative QVT, Página Web: http://wiki.eclipse.org/M2M/Relational_QVT_Language_(QVTR). Último acesso: Novembro 2009

(Deursen et al., 2000) Deursen A., Klint P., and Visser J., *Domain-Specific Languages: An Annotated Bibliography*. SIGPLAN Notices. 35(6): 26-36, June 2000.

(DSL Tools, 2009) DSL Tools: http://msdn.microsoft.com/en-us/library/bb126235.aspx. Último acesso: Novembro 2009.

(Kelly and Tolvanen, 2008) Kelly S., Tolvanen J., *Domain Specific Modeling Enabling Full Code Generation*, Wiley-IEEE Computer Society Press, 2008.

(Filman e Friedman, 2000) Filman R., Friedman D., *Aspect-oriented programming is quantification and Obliviousness*. Proceedings of the Workshop on Advanced Separation of Concerns, in conjunction with OOPSLA'00 (2000)

(Eclipse/GMF wiki, 2009) Eclipse/GMF wiki: http://wiki.eclipse.org/GMF_Documentation. Último acesso: Junho2009.

(Eclipse model-to-model, 2009) Eclipse Model-to-Model, Página Web: http://www.eclipse.org/m2m/. Último acesso: Novembro 2009

(Gomma, 2004) Gomaa H,. *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. Addison-Wesley, 2004.

(GME, 2009) GME: The Generic Modeling Environment. Página Web. http://www.isis.vanderbilt.edu/projects/gme/. Último acesso: Outubro 2009.

(Graphviz, 2009) *Graph Visualization Software* http://www.graphviz.org/. Último acesso: Dezembro 2009.

(Jacobson, 2003) Jacobson I., *Use Cases and Aspects – Working Seamlessly Together*, Journal of Object Technology, Julho-Agosto 2003, Volume 2(4), pp. 7-28.

(Jayaraman et al., 2007) Jayaraman P., Whittle J., Elkhodary A., Gomaa H., *Model Composition and Feature Interaction Detection in Product Lines Using Critical Pair Analysis*, Lecture notes in computer science. Springer, 2007; 151.

(Kang et al., 1990) Kang K., Cohen S., Hess J., Nowak W., Peterson S., 1990. *Feature-Oriented Domain Analysis (FODA)* Feasibility Study. Tech. Report, CMU/SEI-90-TR-021, Pittsburgh, PA.

(Kelly and Tolvanen, 2008) Kelly S., Tolvanen J., (2008), *Domain Specific Modeling Enabling Full Code Generation*, Wiley-IEEE Computer Society Press.

(Kiczales et al., 1997) Kiczales G., Lamping J., Mendhekar A., *RG: A Case-Study for Aspect-Oriented Programming.* In: SPL97. Palo Alto Research Center, Technical Report.

(Kiczales et al., 1997) Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J., *Aspect-Oriented Programming*. Em ECOOP'97 – Object-Oriented Programming, 11th European Conference, LNCS 1241, pp. 220-242. 1997

(Kotonya e Sommerville, 1998) Kotonya G., Sommerville I., *Requirements Engineering: Processes and Techniques.* John Wiley, 1998.

(Kovačević et al., 2007) 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.

(Lamsweerde, 2001) Lamsweerde A., *Goal-Oriented Requirements Engineering: A Guided Tour*; Université Catholique de Louvain; Louvain-la-Neuve; Bélgica; 2001

(Lamsweerde, 2003) Lamsweerde A., *KAOS Tutorial*, Cediti, Setembro 5, 2003, http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf. Último acesso: Abril 2009

(LaTeX Project, 2009) http://www.latex-project.org/. Último acesso: Dezembro 2009.

(LDE wiki, 2009) LDE wiki: http://en.wikipedia.org/wiki/Domain_Specific_Language. Último acesso: Junho 2009

(Liu e Yu, 2001) Liu L., Yu E., *From Requirements to Architectural Design - Using Goals and Scenarios*. ICSE-2001 Workshop: From Software Requirements to Architectures, Maio 2001, Toronto, Canada.

(MDA, 2009) Model-Driven Architecture, Página Web: http://www.omg.org/mda. Último acesso: Novembro 2009.

(Moodle, 2008) Página Web: http://moodle.fct.unl.pt/course/view.php?id=1728. Último acesso: Janeiro 2010.

(Mussbacher et al., 2007) Mussbacher G., Amoyt D., Araújo J., Moreira A., Weiss M., *Visualizing Aspect-Oriented Goal Models with AoGRL*, Second International Requirements Engineering Visualization, 2007.

(Norman et al., 1998) Norman S. Murray, Norman W. Paton, Carole A. Goble, Joanne B. (1998), *Kaleidoquery- A flow-based Visual Language and its Evaluation*, In: Proceedings of the working conference on Advanced visual interfaces. New York, USA.

(OMG, 2009) The Object Management Group (OMG), Página Web: http://www.omg.org/. Último acesso: Novembro 2009.

(OMG, 2005) Unified Modeling Language Version 2.0 (2005), Object Management Group (OMG). http://www.omg.uml. Último acesso: Setembro 2009

(Operational QVT, 2009) Operational QVT, Página Web: http://www.eclipse.org/m2m/qvto/doc/. Último acesso: Novembro 2009

(Pohl et al., 2005) Pohl K., Böckle G., Van Der Linder F., *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer, 2005.

(Rashid et al., 2003) Rashid A., Moreira A., Araújo J., *Modularisation and Composition of Aspectual Requirements*, Proceedings of 2nd International Conference on Aspect-Oriented Software Development, ACM, 2003.

(Rumbaugh et al., 2005) Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Second Edition. Addison-Wesley, 2005.

(SEI/CMU, 2006) SEI/CMU. *A Framework for Software Product Line Practice*. Version 4.2. Software Engineering Institute, Carnegie Mellon University, 2006

(Silva et al., 2008) 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.

(Soares et al., 2006) 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

(Sommerville e Sawyer, 1997) Sommerville I., Sawyer P., *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.

(SQL, 2009) SQL.org http://www.sql.org/. Último acesso: Dezembro 2009.

(Van Deursen e Klint, 2002) Van Deursen A., Klint P., *Domain-Specific Language Design Requires Feature Descriptions*. In Journal of Computing and Information Technology, 10, 1, pp. 1-17. 2002.

(Via Verde, 2005) Via Verde (2005). http://www.viaverde.pt/ViaVerde/vPT. Último acesso: Abril 2009

(Völter e Stahl, 2006) Völter M., Stahl T., *Model-Driven Software Development - Technology, Engineering, Management*, Wiley, 2006.

(Sousa, 2009) Sousa V., *Model Driven Development Implementation of a Control Systems User Interfaces Specification Tool*, Dissertação de mestrado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Portugal, Fevereiro 2009.

(Whittle e Araújo, 2004) Whittle J., Araújo J., *Scenario Modeling with Aspects, IEE Proceedings -Software*, vol. 151, pp. 157-172, 2004.

(W3C HTML, 2009) Página Web: http://www.w3.org/html/. Último acesso: Dezembro 2009.

(Yu et al., 2004) Yu Y., Leite J., Mylopoulos J., *From goals to aspects: discovering aspects from requirements goal models*, Proceedings of the 12th IEEE International Requirements Engineering Conference (RE), pp.38 – 47. Kyoto, Japão, IEEE CS Press, 2004.

(Yu et al., 2008) Yu Y, Leite J, Lapouchnian A, Mylopoulos J., *Configuring features with stakeholder goalsProc*., of the 2008 ACM Symposium on Applied computing,. ACM: Brazil, 2008.

Anexo A. Questionário sobre a LDE Theme-SPL

A.1 Introdução

O objectivo deste questionário é avaliar a usabilidade de uma ferramenta para criação de modelos Theme-SPL e transformação dos mesmos em modelos de *features*.

Desde já obrigada pela colaboração.

A.2 Resolução de Problemas

O sistema real do Observador Sanitário é adaptado de (Soares et al., 2006). 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 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;
- Queixas alimentares: Casos onde se suspeita de ingestão de comida contaminada;
- Queixas diversas: Casos relacionados com várias razões, que não estão mencionados 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.

Tabela A.1 Requisitos para o Observador Sanitário

R No.	Descrição dos requisitos
R1	O **observador sanitário** é composto por: **efectuar queixas**, pode-se **obter informação** sobre os serviços de saúde, **registar queixas** pelo assistente, **enviar relatório** com estatísticas e **efectuar consulta** sobre a **queixa**.
R2	O queixoso tem que **efectuar autenticação** no sistema para poder **efectuar** uma **queixa**.
R3	Para **efectuar autenticação** é necessário inserir o login e a *password*.
R4	Os **dados** são **validados**.
R5	Se os dados são válidos é possível inserir a queixa.
R6	Se os dados são inválidos é **mostrada** uma **mensagem** de **erro** ao utilizador, utilizando o visor do computador, "Login ou *password* errado (s) ".
R7	Para **efectuar** uma **queixa**, o queixoso necessita de inserir os seus dados, a data da queixa, o tipo de queixa, observações sobre a queixa, qual a situação da queixa, e as análises que foram efectuadas através de um assistente.
R8	Os tipos de queixas podem ser dois ou três de entre os seguintes: animais, alimentares ou diversas.
R9	A situação da queixa pode ser uma entre as seguintes alternativas: aberta, suspensa ou fechada.
R10	Pode-se **obter informação** sobre um ou mais dos seguintes tipos: doenças, campanhas de vacinação, unidades de saúde que utilizam uma dada especialidade, e quais as especialidades utilizadas numa unidade de saúde.
R11	O assistente tem que **efectuar** a **autenticação** no sistema para **registar** uma **queixa**.
R12	A **queixa** é **registada** no sistema pelo assistente.
R13	É **enviado** um **relatório** com as estatísticas das várias queixas, utilizando as queixas registadas para efectuar o mesmo.
R14	A **consulta** sobre a **queixa** é efectuada utilizando os dados das queixas existentes.

Percorrendo a lista de requisitos, são identificados os seguintes potenciais temas, ilustrados na Figura A.1.

Efectuar Autenticação	**Enviar Relatório**	**Obter Informação**
Efectuar Consulta Queixa	**Mostrar Mensagem Erro**	**Registar Queixa**
Efectuar Queixa	**Observador Sanitário**	**Validar Dados**

Figura A.1 Lista de Temas

São identificadas as entidades, ilustradas na Figura A.2, nos requisitos.

Assistente	Queixoso
Computador	Visor

Figura A.2 Lista de Entidades

A Visão de Relação de Temas, Figura A.3, é elaborada tendo em conta as heurísticas definidas na secção 5.3.

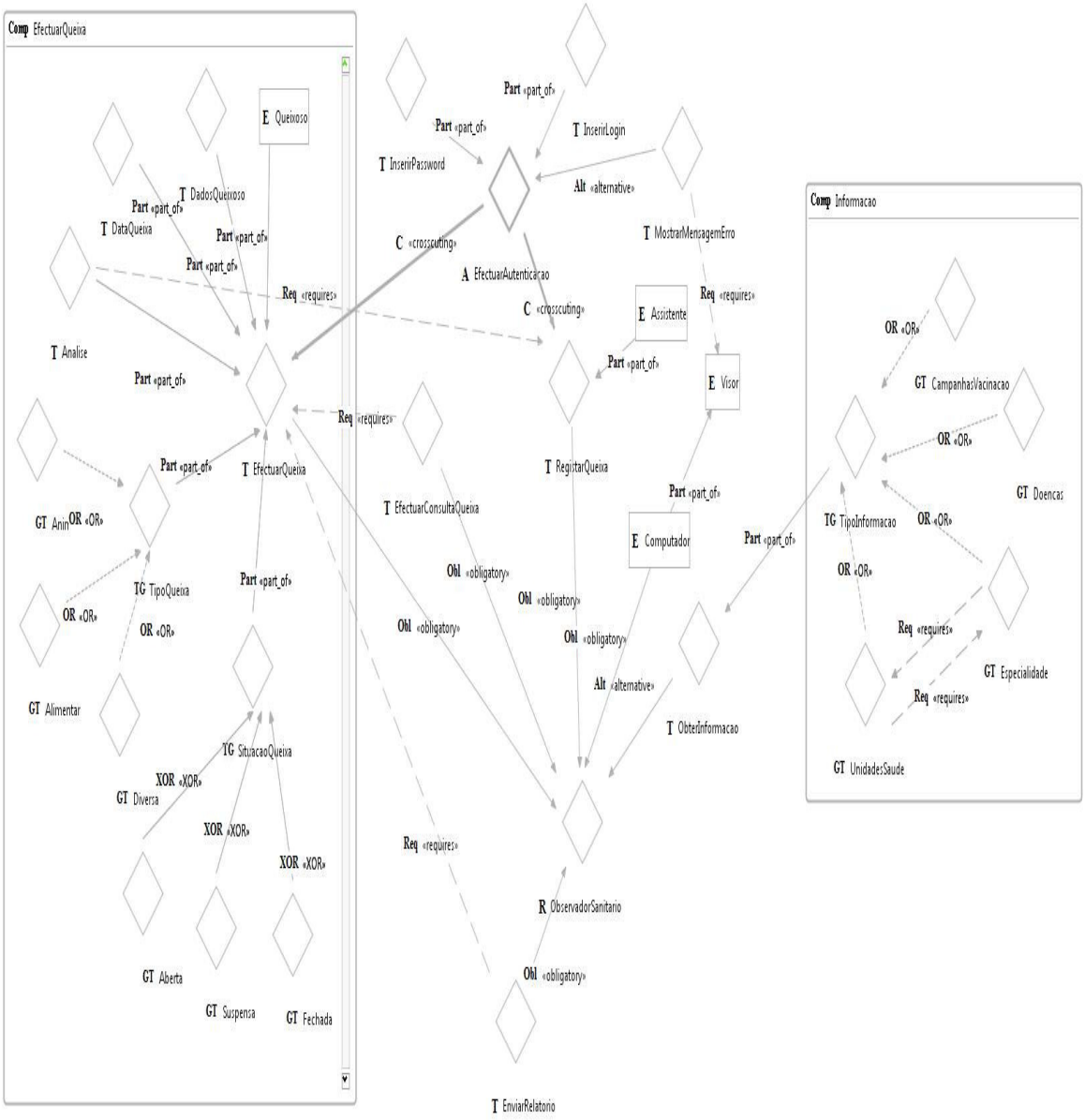

Figura A.3 Modelo Theme-SPL elaborado na LDE Theme-SPL

Crie o modelo de *features* para o caso de estudo considerado, tendo em conta o modelo Theme-SPL ilustrado.

Tempo gasto a identificar as *features* e elaborar o modelo de *features.* ☐ minutos

A.3 Nível de Satisfação

As seguintes questões são relativas à abordagem apresentada, ao modelo apresentado anteriormente e à usabilidade da ferramenta.

O questionário é composto por 18 questões, e o utilizador deve classifica-las numa escala de 1 (muito mau) a 5 (muito bom).

1. Compreendeu o modelo Theme-SPL?

Sim ☐ Não ☐ Talvez ☐

2. Se não compreendeu o que faltou?

☐

3. Qual o grau de dificuldade para identificar as *features*?

☐ ☐ ☐ ☐ ☐

1 2 3 4 5

4. O modelo Theme-SPL contribuiu para o ajudar a identificar as *features*?

☐ ☐ ☐ ☐ ☐

1 2 3 4 5

5. E contribui-o para o ajudar a construir o modelo de *features*?

☐ ☐ ☐ ☐ ☐

1 2 3 4 5

6. Com que facilidade criou o modelo de *features* com base em Theme-SPL?

☐	☐	☐	☐	☐
1	2	3	4	5

7. Comparando o modelo feito a mão e o gerado qual o resultado esperado?

☐	☐	☐	☐	☐
1	2	3	4	5

8. O que acha que o modelo transformado fez melhor em relação ao que fez a mão?

9. E o que acha que fez pior?

10. Considera a ferramenta útil?

☐	☐	☐	☐	☐
1	2	3	4	5

11. Teve dificuldades a utilizar a ferramenta?

☐	☐	☐	☐	☐
1	2	3	4	5

12. Qual a maior dificuldade que encontrou no uso da ferramenta?

13. Como acha que a ferramenta lida com o problema de escalabilidade dos modelos?

1 2 3 4 5

14. Acha que os mecanismos do desenvolvimento orientado a aspectos são vantajosos para a construção do modelo de *features*?

Sim Não

15. Porquê?

16. Quais são, na sua opinião, os pontos fracos desta ferramenta?

17. E quais pontos fortes da ferramenta?

18. O que sugere para que a ferramenta possa ser melhorada?

Anexo B. Manual de utilizador da LDE para Theme-SPL

1. Introdução

Neste manual de utilizador vão ser mostrados todos os passos necessários à correcta utilização da LDE Theme-SPL.

Serão apresentados primeiramente quais os requisitos necessários para instalar a ferramenta, sendo apresentada a sua correcta instalação.

Nas secções seguintes serão apresentadas todas as acções que é possível fazer com a LDE, assim como os vários elementos e ligações constituintes de cada acção, sendo sempre apresentada uma breve descrição e exemplo de cada elemento e ligação, para melhor esclarecimento por parte do utilizador.

1.1 Requisitos

Para utilizar a ferramenta é necessário ter previamente instalado a plataforma Eclipse e os *plugins* EMF e GMF. Pode-se fazer o download da ferramenta eclipse na seguinte página: http://www.eclipse.org/downloads/.

No Eclipse ir a Help → Software Updates → Find and Install, depois:

- Search for new features to install
- Calisto Discovery Site

Finalmente escolher e instalar os seguintes componentes:

- Eclipse Modeling Framework (EMF)
- Graphical Modeling Framework (GMF)

1.2 Instalar e executar a LDE Theme-SPL

Antes de fazer a instalação da LDE, é necessário certificar que a versão do compilador Java é a 6.0. Para isso vai-se a Window -> Preferences -> Java -> Compiler -> Compiler compliance level. Após este passo, e a instalação do EMF/GMF, faz-se a instalação da LDE. Para isso vai-se a File -> Import. Em Import escolher General -> Existing Projects into Workspace. Em Select archive file -> Browse indicar o caminho para o ficheiro Theme-SPL.zip, seleccionam-se as pastas correspondentes e carrega-se em Finish.

1.3 Criação do Projecto

Para se criar um projecto vai-se a File -> New -> Project. Na janela que aparece ir a General -> Project. Carregar em Next, escrever o nome do projecto e carregar em Finish.

Após a criação do projecto, deve-se criar o espaço de edição dos modelos. Para isso é necessário, no Eclipse, ir a Run → Open Run Dialog. Por fim, no Run Dialog faz-se duplo clique em Eclipse Application e faz-se Run. Na nova extensão do Eclipse vai ser necessário criar um novo projecto. Sendo assim escolhe-se New → Project → General → Project. Agora neste novo projecto escolhe-se New → Example → Theme Diagram e dá-se o nome desejado, como é mostrado na Figura B.1.

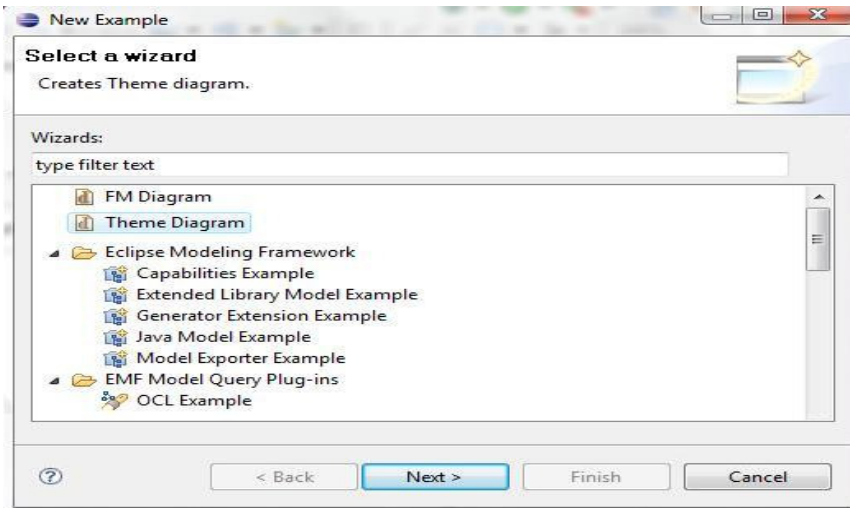

Figura B.1 Criação do diagrama Theme Diagram

1.4 Criação de Modelos

Seguidamente é apresentado o modo de utilização do editor da ferramenta.

- Menu de selecção

A ferramenta apresenta no lado direito um menu de selecção, onde são mostrados os elementos necessários para construir os diagramas (Figura B.2). O menu de selecção apresenta três tipos de elementos: *Links* (as ligações entre os diferentes elementos), *Themes* (elementos que representam os conceitos do Theme) e *Compartment* (onde é representado o compartimento onde são armazenados os elementos dos diagramas).

Existem duas maneiras de criar nós (Figura B.2): (i) ir à secção *Themes* e arrastar o elemento desejado para o editor (elipse vermelha); (ii) dentro do editor, ao aparecer a lista dos elementos possíveis de criar dentro do mesmo, escolher o elemento desejado (elipse cinzenta).

Para criar as ligações entre os elementos vai-se à secção *Links* (elipse verde) e escolhe-se a ligação desejada. Neste caso foi criado um *link alternative*, um *obligatory* e um *part_of* (Figura B.2).

Para criar compartimentos existem dois meios de criação: (i) ir ao menu de selecção e escolher o elemento *ThemesCompartment* (assinalado com a elipse castanha); (ii) na janela de edição, quando aparecem quais os elementos disponíveis (elipse cinzenta) escolher o elemento (neste caso, será Comp).

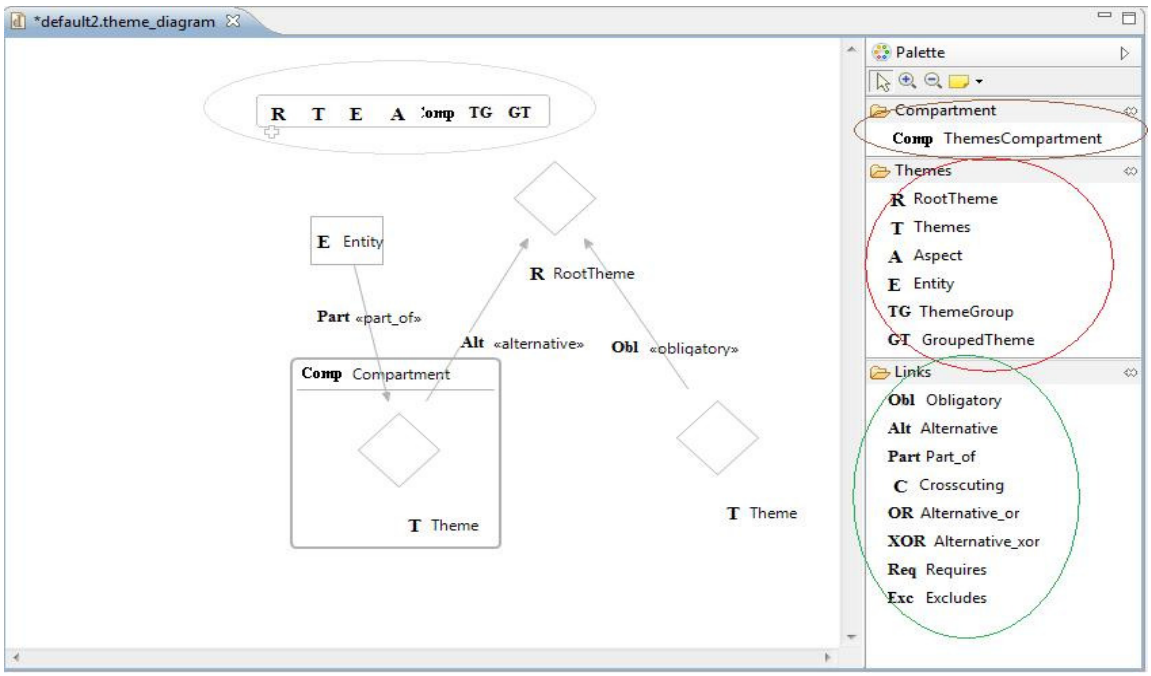

Figura B.2 Criação de temas, entidades, compartimentos e *links*

Tal como o editor dos modelos Theme-SPL, o editor para o modelo de *features* apresenta no lado direito um menu de selecção, onde são mostrados os elementos necessários para construir os diagramas (Figura B.3). O menu de selecção apresenta dois tipos de elementos: *Links* (as ligações entre os diferentes elementos) e *Features* (elementos que representam os conceitos do modelo de *features*).

Para criar *Features* existem duas maneiras de criar estes elementos (Figura B.3): (i) ir à secção *Features* e arrastar o elemento desejado para o editor (elipse vermelha); (ii) dentro do editor, ao aparecer a lista dos elementos possíveis de criar dentro do mesmo, escolher o elemento desejado (elipse cinzenta).

Para criar as ligações entre os elementos vai-se à secção *Links* (elipse verde) e escolhe-se a ligação desejada. Neste caso foi criado um *link optional* e dois *mandatory* (Figura B.3).

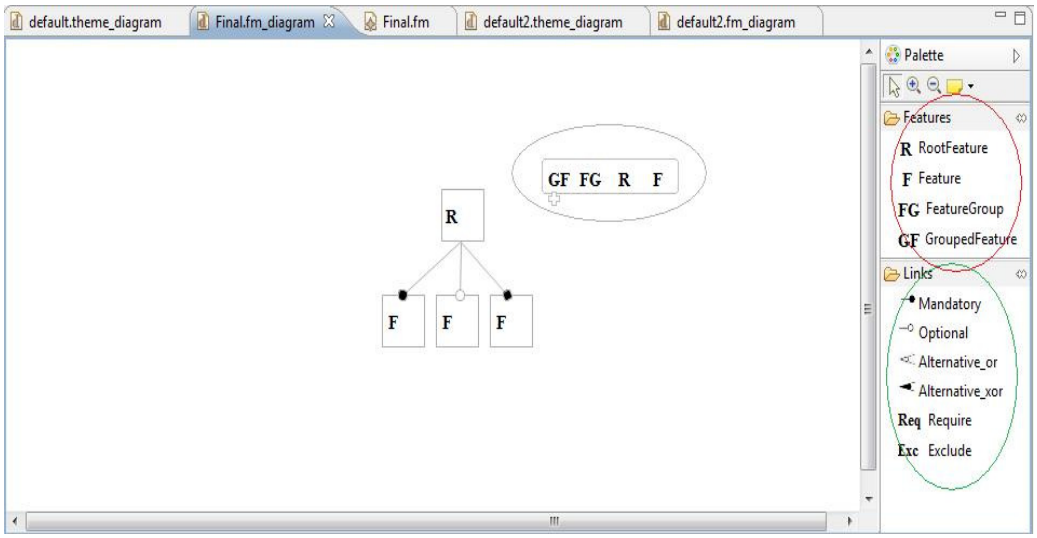

Figura B.3 Criação de *features* e links

1.5 Criar um projecto ATL

Para se criar um projecto vai-se a File -> New -> Project. Na janela que aparece ir a ATL -> ATL Project. Carregar em Next, escrever o nome do projecto e carregar em Finish.

Clicando com o botão direito do rato no projecto ATL ir a New -> Other. Na janela que aparece ir a ATL -> ATL File. Carregar em Next irá abrir o ATL File Wizard, que deverá ser preenchido conforme ilustra a Figura B.4, e clicar em Finish após o Wizard

preenchido. Copiar o ficheiro ATL existente no ficheiro Theme-SPL.zip para o ficheiro ATL criado.

Para aplicar as regras é necessário criar uma nova configuração. Para criar uma new lauch configuration seleccionar Run Configurations, seleccionar ATL Transformations e clicar em New. As configurações devem ser preenchidas como ilustrado na Figura B.5, no IN Source Models é dado o caminho para o modelo que se pretende transformar (Theme-SPL), e em OUT é dado o nome do modelo resultante da transformação (FM). Em seguida clicar em Run e na 2ª instância do eclipse irá aparecer o modelo de *features* resultante da transformação.

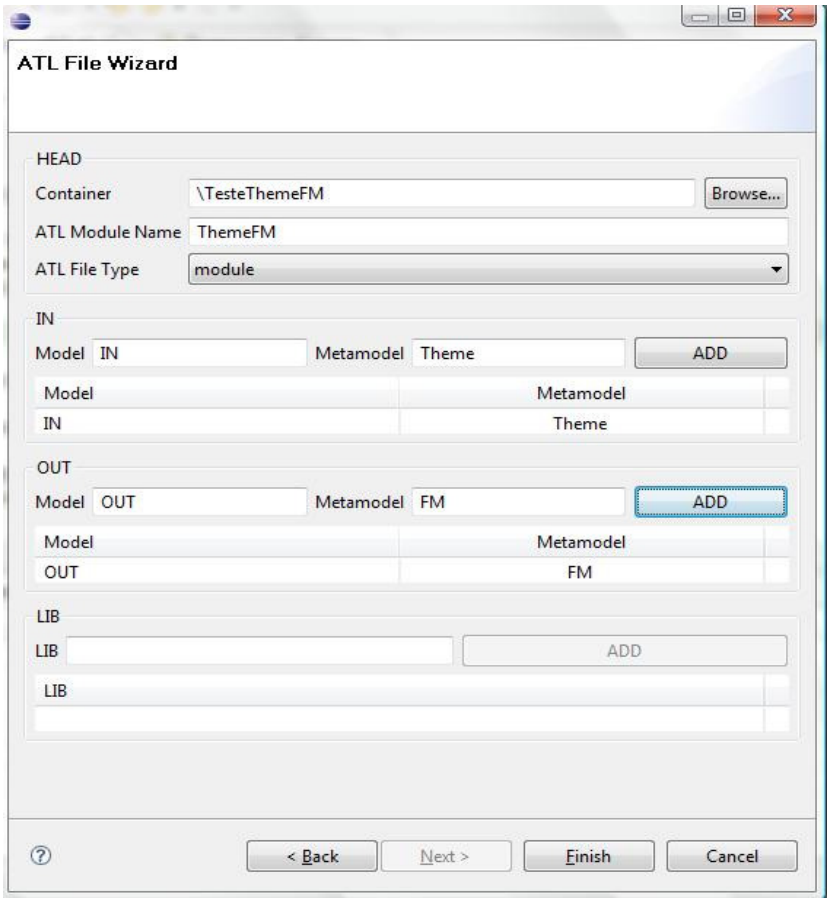

Figura B.4 ATL File Wizard

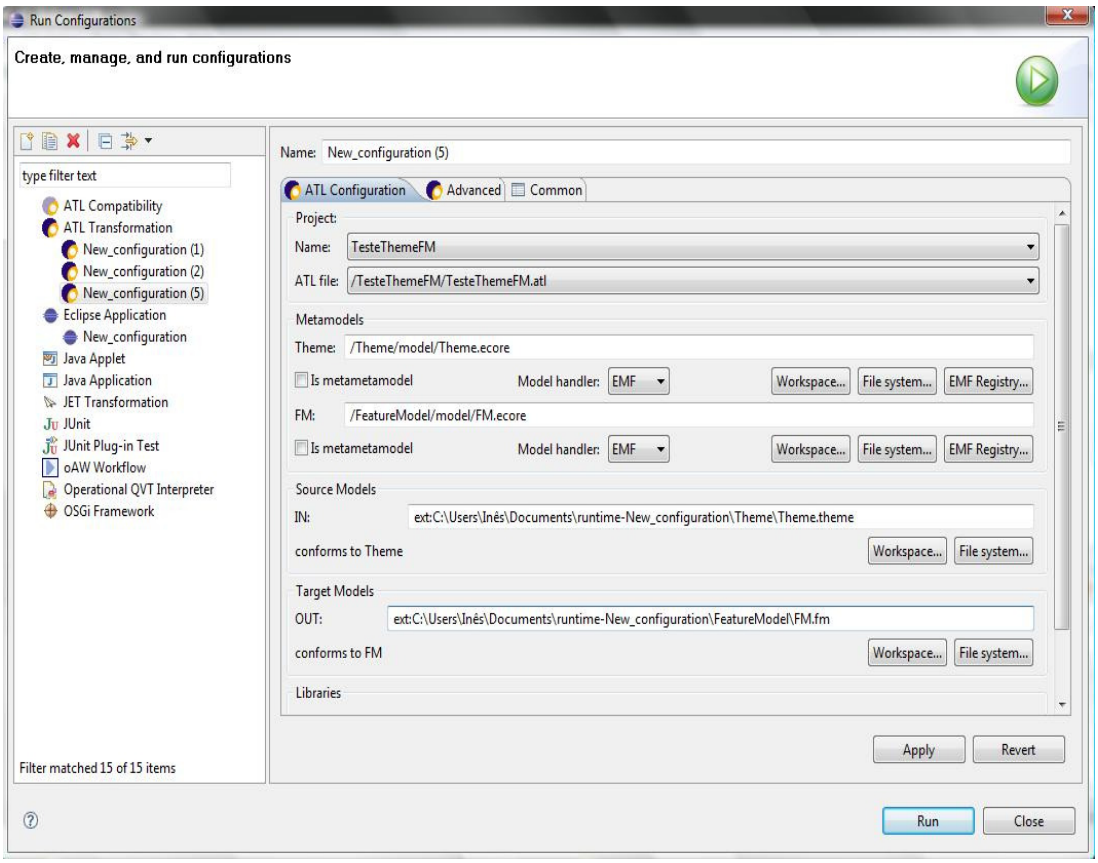

Figura B.5 Run Configurations