Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Programa de Pós-Graduação em Ciência da Computação

UbiComSPL: DESENVOLVIMENTO BASEADO EM MDA, DE LINHA DE PRODUTO DE SOFTWARE NO DOMÍNIO DE APLICAÇÕES UBÍQUAS

RAPHAEL PEREIRA DE OLIVEIRA

São Carlos

Julho / 2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

O48ud

Oliveira, Raphael Pereira de.
UbiComSPL : desenvolvimento baseado em MDA, de linha de produto de software no domínio de aplicações ubíquas / Raphael Pereira de Oliveira. -- São Carlos : UFSCar, 2009.
69 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Engenharia de software. 2. Família de produtos de software. 3. Reuso. 4. MDA. 5. Computação ubíqua. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"UbiComSPL: Desenvolvimento baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubíquas"

RAPHAEL PEREIRA DE OLIVEIRA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)

Prof. Dr. Wanderley Lopes de Souza
(DC/UFSCar)

Prof. Dr. Júlio César Sampaio do Prado Leite
(PUC-Rio)

São Carlos
Julho/2009

Alzido de Oliveira (in memoriam)

A- Alzido de Oliveira, humilde, sábio
L- Leque de informação, carinho, ternura
Z- Zelou pela sua família
I- Idolatrado no seu caminho
D- Dedicou-se ao ensino
O- Ódio não conhecia

D- Deixou legados de sabedoria
E- Espalhados por todo o mundo

O- O dia do encontro chegou
L- Levado por anjos de Deus
I- Ilumina agora nossos caminhos
V- Vive ao lado Dele
E- Esperamos um dia te encontrar
I- Imenso é seu lugar em nossos corações
R- Recordações eternas de você, pai
A- Alzido de Oliveira

Raphael Pereira de Oliveira

Agradecimentos

Em primeiro lugar a Deus, por abrir esse caminho que foi trilhado com muito esforço e dedicação, aos meus pais, Alzido de Oliveira (in memoriam) e Francisca Isaldite Pereira de Oliveira, que sempre me deram apoio, incentivo, coragem e amor em todos os momentos da minha vida, ao meu orientador, Antonio Francisco do Prado, que através de sua sabedoria, experiência, incentivos e cobranças me guiou durante todo o percurso do mestrado, a CAPES pelo apoio financeiro e a todos que viveram comigo nesse período.

Sumário

Resumo 1
Abstract 2
Capítulo 1 3
1. Introdução 3
1.1. Motivação 4
1.2. Objetivos 5
1.3. Organização 6
Capítulo 2 7
2. Conceitos e Técnicas da Abordagem 7
2.1. Linhas de Produto de Software 7
2.1.1. Fundamentos de Linhas de Produto de Software 9
2.2. Model-Driven Architecture (MDA) 12
2.3. Computação Ubíqua 15
2.3.1. Conceitos 16
2.3.2. Desenvolvimento de Software para Computação Ubíqua 17
2.4. Ubiquitous Computing Framework (UCF) 19
2.5. Multiple-View Case (MVCASE) 20
Capítulo 3 29
3. Trabalhos Relacionados 29
Capítulo 4 32
4. Abordagem de Desenvolvimento baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubíquas 32
4.1. Engenharia de Domínio 34
4.1.1. Requisitos 35
4.1.2. Projeto 41
4.1.3. Implementação 46
4.2. Engenharia de Aplicação 50
Capítulo 5 52
5. Engenharia de Aplicação 52
5.1. Requisitos 53
5.2. Projeto 55
5.3. Implementação 57
Capítulo 6 61
6. Conclusões e Trabalhos Futuros 61
6.1. Contribuições 61
6.2. Trabalhos Futuros 63
Referências 65
Publicações 69

Lista de Figuras

Figura 1. Viabilidade na Utilização de LPS 8
Figura 2. ED e EA em LPS [Linden et al. 2007] 9
Figura 3. LPS e seus Produtos 10
Figura 4. Atividades Essenciais na Prática de LPS [SEI 2009] 11
Figura 5. Notação do Modelo de Features 12
Figura 6. Processo MDA 14
Figura 7. Grau de Transparência e Mobilidade na Computação Tradicional, Pervasiva, Móvel e Ubíqua [Jardim 2004] 17
Figura 8. Computação Ubíqua [Araújo 2003] 17
Figura 9. Metamodelo da UML 2.1 23
Figura 10. Arquivo de Integração gmfmap 24
Figura 11. Template para Geração de Código Java 25
Figura 12. Metamodelo de Classes da MVCASE 27
Figura 13. ED e EA na Abordagem Proposta 33
Figura 14. Identificação de Features no Domínio de Segurança Residencial 36
Figura 15. Casos de Uso e Features do Domínio de PRE 37
Figura 16. Modelo de Features no Domínio de Segurança Residencial 38
Figura 17. Modelo de Features do Domínio de PRE 39
Figura 18. PIM da Linha de Produto de Software PRE 41
Figura 19. Modelo de Classes do UCF 42
Figura 20. PSM da Linha de Produto de Software PRE 44
Figura 21. Modelo de Componentes – Feature Acessar Conteúdo PRE 45
Figura 22. Relacionamento Gerando um Objeto 47
Figura 23. Relacionamento Gerando uma Coleção de Objetos 48
Figura 24. Relacionamento de Dependência 49
Figura 25. Código Gerado a partir do PSM 50
Figura 26. Modelo de Casos de Uso com Casos de Uso do Produto 54
Figura 27. PIM do CellPhonePRE 55
Figura 28. PSM do CellPhonePRE 56
Figura 29. Modelo de Componentes para Acessar Conteúdo no PRE 57
Figura 30. Trecho do Código Gerado pela MVCASE para o CellphonePRE 58
Figura 31. Reuso do Core Asset UCF 59
Figura 32. Código final do produto CellPhonePRE 60
Figura 33. Reuso do Domínio de PRE em Outros Domínios 63

Lista de Tabelas e Listagens

Tabela 1. Mapeamento Features em Produtos no Domínio de Segurança Residencial 40
Tabela 2. Mapeamento de Features em Produtos no Domínio de PRE..... 40
Tabela 3. Features do Produto CellPhonePRE 54

Lista de Abreviações

CASE	*Computer Aided Software Engineering*
EMF	*Eclipse Modeling Framework*
GCU	*Grupo de Computação Ubíqua*
GEF	*Graphical Editing Framework*
GMF	*Graphical Modeling Framework*
GCU	*Grupo de Computação Ubíqua*
IDE	*Integrated Development Environment*
JET	*Java Emitter Template*
LPS	*Linha de Produto de Software*
MDA	*Model-Driven Architecture*
OMG	*Object Management Group*
PDS	*Processo de Desenvolvimento de Software*
PIM	*Platform Idenpendent Model*
PRE	*Portfólio Reflexivo Eletrônico*
PSM	*Platform Specific Model*
SGDB	*Sistema de Gerenciamento de Banco de Dados*
UFSCar	*Universidade Federal de São Carlos*
UML	*Unified Modeling Language*

Resumo

Essa dissertação apresenta uma Abordagem para Desenvolvimento baseado em *Model-Driven Architecture*, de Linha de Produto de Software orientada a domínios de Aplicações Ubíquas. A abordagem utiliza a *Model-Driven Architecture* para melhor atender as diversidades de arquiteturas da Computação Ubíqua e facilitar o reuso. Com foco no domínio do problema, desenvolve-se o *Core Asset*, núcleo da Linha de Produto de Software, que é reutilizado na construção dos produtos derivados da linha. Um dos artefatos do *Core Asset* é *Ubiquitous Computing Framework* construído para atender os requisitos não funcionais da Computação Ubíqua. Uma ferramenta CASE, denominada MVCASE, automatiza grande parte do processo da *Model-Driven Architecture*, desde a modelagem até a geração parcial de código.

Palavras-Chave: Linhas de Produto de Software, Reuso, MDA e Computação Ubíqua.

Citação: Oliveira, Raphael P. UbiComSPL - Desenvolvimento Baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubíquas. São Carlos, 2009. 78p. Dissertação de Mestrado - Departamento de Computação, Universidade Federal de São Carlos.

Abstract

This thesis presents an approach to Development based on Model-Driven Architecture, of Ubiquitous Applications Domain Software Product Lines. The approach uses Model-Driven Architecture to facilitate the modeling reuse of the same application in different architectures of the Ubiquitous Computing. With focus in the problem domain, the Software Product Lines core is developed and it is reused in the products construction. One of the assets that composed the Software Product Lines is the Ubiquitous Computing Framework, developed to attend the non functional requirements of Ubiquitous Computing. The CASE tool, called MVCASE, makes automatic part of the activities in the proposed approach.

Keywords: Software Product Line, reuse, MDA and Ubiquitous Computing.

Capítulo 1

1. Introdução

Recentes avanços em tecnologias e plataformas, os quais ocorrem em diversas áreas da computação, tem demandado um grande esforço da Engenharia de Software para oferecer novas abordagens para o desenvolvimento de software com qualidade e baixo custo. Uma dessas áreas é conhecida como Computação Ubíqua.

O termo Computação Ubíqua [Weiser 1991] refere-se a ambientes saturados de dispositivos computacionais (e.g., *desktops*, *laptops*, celulares, *smartphones*, *Personal Digital Assistants-PDAs*, *tablets*) e redes de comunicação sem fio (e.g., *Wi-Fi*, *Bluetooth*, *IrDA*, *WiMax*), que se integram naturalmente à atividade humana. Distribuída e envolvendo computadores com recursos significativos e também pequenos dispositivos móveis com capacidades limitadas (e.g., tela, memória, processamento, energia), a Computação Ubíqua apresenta requisitos adicionais ao das aplicações tradicionais, tais como: ciência de contexto e adaptação de serviços oferecidos. Para atender a esses requisitos, diferentes processos de desenvolvimento têm sido propostos [Carton et al. 2007, Fernandes et al. 2007]. Uma abordagem comum nesses processos é o uso da *Model-Driven Architecture (MDA)*. Na MDA as especificações e funcionalidades do software são representadas em

um Modelo Independente de Plataforma (*Platform Independent Model - PIM*), enquanto que as tecnologias e plataformas são especificadas através de um Modelo Específico de Plataforma (*Platform Specific Model - PSM*). Usando mecanismos de transformações, a partir de um PIM, vários PSM's podem ser obtidos, e a partir de cada PSM, um código pode ser gerado para uma plataforma específica. Dessa forma, pode-se ter para uma mesma aplicação ubíqua, diferentes PSM's gerados a partir de um PIM atendendo a grande diversidade de plataformas existentes na Computação Ubíqua.

Dentre os avanços da Engenharia de Software destaca-se também o desenvolvimento de *Linha de Produto de Software (LPS)*, que visa construir softwares em série, os quais possuem funcionalidades comuns, opcionais e especificas. A especificação dessas funcionalidades é realizada conforme o domínio no qual a LPS se enquadra. O conjunto de funcionalidades comuns e opcionais compõe o *core asset*, núcleo da LPS. A partir das extensões do *core asset*, através de pontos variáveis, são construídos produtos específicos da LPS. Dessa forma, pode-se obter um maior grau de reuso em relação aos processos que empregam apenas componentes e frameworks.

1.1. Motivação

As principais motivações para este projeto são:

- Investigar uma abordagem para o desenvolvimento de software na computação ubíqua, que seja orientada para o reuso de software;
- Explorar os conceitos da MDA, na computação ubíqua, onde existe uma grande variedade de dispositivos com diferentes arquiteturas;
- Combinar os conceitos e tecnologias de Linha de Produto de Software e MDA para apoiar o processo de desenvolvimento proposto;

- Validar a abordagem com estudos de casos no domínio da computação ubíqua; e
- Estender a ferramenta MVCASE para suportar e automatizar grande parte da abordagem proposta.

Assim, estimulados por essas motivações e procurando tirar proveito das vantagens de LPS e MDA, apresenta-se uma abordagem para o desenvolvimento, baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubíquas. Além da prática do reuso, a abordagem visa facilitar as implementações das aplicações ubíquas, considerando a diversidade de dispositivos e as diferenças dos seus contextos.

1.2. Objetivos

Um dos objetivos deste trabalho foi desenvolver uma abordagem para o Desenvolvimento, baseado em MDA, de LPS no Domínio de Aplicações Ubíquas. A abordagem denominada *Product Line Development based on MDA for the Ubiquitous Computing Domain (UbiComSPL)* é dividida em Engenharia de: Domínio e Aplicação. O processo é orientado pelas idéias da MDA, desde os requisitos até a implementação. A Engenharia de Domínio tem como objetivo a construção do *core asset* da LPS, onde artefatos das disciplinas de requisitos, projeto e implementação ficam disponíveis para reuso na construção dos produtos pela Engenharia de Aplicação.

Como outro objetivo desse trabalho tem-se a melhoria do processo de desenvolvimento de software, proporcionando uma diminuição do tempo de desenvolvimento, redução de custos, e aumento na qualidade.

Por fim, tem-se ainda outro objetivo importante que é o de incentivar e pesquisar o reuso de software combinando os conceitos de MDA e de LPS, num domínio onde o acesso a internet é realizado através de dispositivos móveis como celulares, *smartphones* e *PDAs*.

1.3. Organização

Essa Dissertação está organizada da seguinte forma: o Capítulo 2 introduz os principais conceitos e técnicas utilizados na abordagem proposta; o Capítulo 3 discute os trabalhos correlatos; o Capítulo 4 apresenta a abordagem proposta, incluindo o desenvolvimento do *core asset* pela Engenharia de Domínio; no Capítulo 5, apresenta-se a Engenharia de aplicação instanciando um produto a partir da LPS proposta, através de um estudo de caso; e finalmente o Capítulo 6 mostra as conclusões relativas a esse trabalho, apontando direções para trabalhos futuros.

Capítulo 2

2. Conceitos e Técnicas da Abordagem

Esse Capítulo apresenta os conceitos e técnicas de *Linhas de Produto de Software (LPS)*, *Model-Driven Architecture (MDA),* Computação Ubíqua, *Ubiquitous Computing Framework (UCF)* e *Multiple-View CASE (MVCASE)* usados no desenvolvimento da abordagem UbiComSPL.

2.1. Linhas de Produto de Software

O crescimento na utilização de softwares tornou-se uma característica chave para a construção de softwares modernos e competitivos. Não importando o quanto seja simples ou complexo, não importando o quanto seja grande ou pequeno, dificilmente encontramos algum produto moderno sem um software. Assim, a competitividade no desenvolvimento de software vem se tornado um interesse para empresas de todos os tamanhos e de todos os mercados. Como resultado, *Linhas de Produto de Software (LPS)*, ou família de produtos, tem ganhado grande atenção com o passar dos anos.

Várias razões levam empresas a utilizar LPS. A mudança em direção a uma LPS está geralmente baseada em considerações econômicas [Linden et al. 2007]. Os melhoramentos de custo e tempo de entrega estão fortemente relacionados com LPS: o processo de desenvolvimento de software utilizando LPS suporta larga uma escala de reuso. Com a utilização de LPS, 90% do

software como um todo pode ser construído através do reuso [Linden et al. 2007]. Assim, o custo no desenvolvimento do software e o tempo gasto em seu desenvolvimento podem ser consideravelmente reduzidos.

Infelizmente, essa melhora não vem de graça, e requer um investimento extra. Isso acontece principalmente por causa da construção do *core asset*, que é núcleo da LPS, e transformação da organização.

Por outro lado, o ponto de compensação, quando se adota uma LPS, é encontrado aproximadamente depois da construção do terceiro produto, sendo algumas vezes antes [Linden et al. 2007]. A Figura 1 ilustra esse estudo de viabilidade.

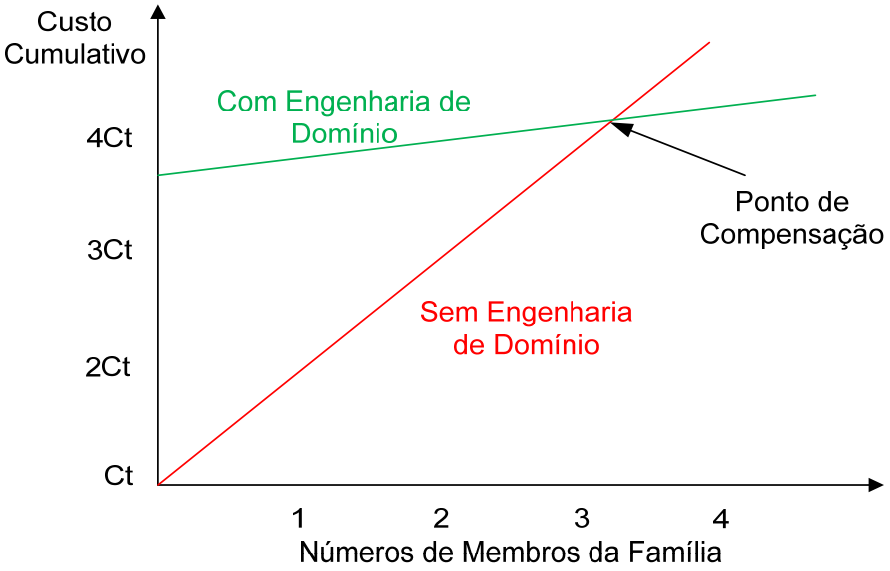

Figura 1. Viabilidade na Utilização de LPS [Linden et al. 2007]

Geralmente, através da redução nos custos de desenvolvimento, alcança-se também uma redução nos custos de manutenção [Linden et al. 2007]. Vários aspectos contribuem para essa diminuição; os mais notáveis são as quantidades totais de código e documentação, que são reduzidos.

Linhas de Produto de Software visam construir softwares em larga escala, utilizando artefatos reutilizáveis e previamente testados. Dessa forma, as falhas nos produtos obtidos da Linha tendem a ser menor do que em

produtos que são desenvolvidos pela Engenharia de Software Tradicional. Além disso, os produtos resultantes são mais fáceis de serem mantidos devido a uma diminuição na quantidade total de código e documentação utilizados na construção dos produtos pela LPS.

2.1.1. Fundamentos de Linhas de Produto de Software

O desenvolvimento de uma LPS compreende [Linden et al. 2007]: a *Engenharia de Domínio (ED)* para construção de um núcleo para reuso; e a *Engenharia de Aplicação (EA)* para construção dos produtos baseado no reuso, conforme mostra a Figura 2.

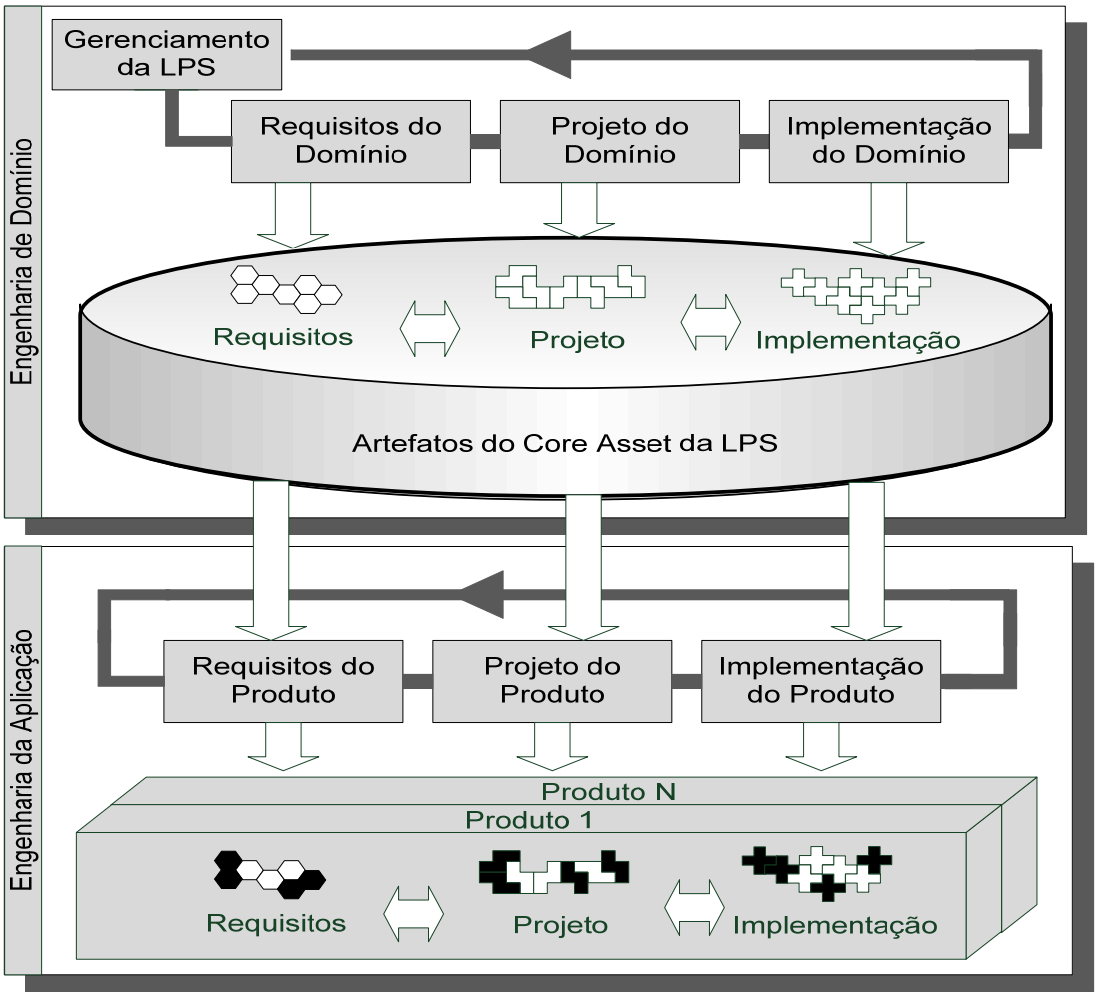

Figura 2. ED e EA em LPS [Linden et al. 2007]

Na ED, caracterizada pelo desenvolvimento para reuso, uma base é fornecida para o desenvolvimento de produtos. Em contradição a abordagens

tradicionais de reuso, que focam em artefatos de código, uma LPS abrange o reuso de artefatos que são relevantes em todo o ciclo de vida no desenvolvimento de um software, desde requisitos, passando pelo projeto e implementação.

Na EA, cujo enfoque é o desenvolvimento com o reuso, constroem-se os produtos derivados da LPS. A EA é orientada pelo *core asset* da LPS, o qual possui grande parte das funcionalidades comuns e opcionais, requeridas para um novo produto. Conforme a Figura 3, para a construção de um novo produto, especifica-se as funcionalidades comuns (*Kernel)* e opcionais *(Optional)*, encontradas no *core asset*, e posteriormente desenvolvem-se as funcionalidades específicas do produto (*Application*). O ideal é que no estágio de desenvolvimento do produto, aproximadamente 90% das suas funcionalidades estejam disponíveis para reuso e, os 10% restantes sejam desenvolvidos conforme as necessidades específicas do produto.

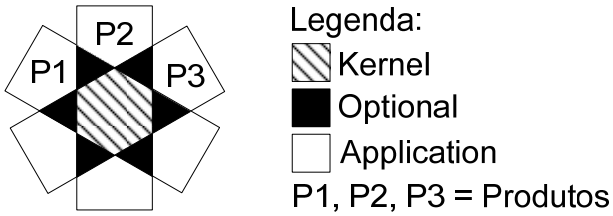

Figura 3. LPS e seus Produtos [Schmid 2008]

A Figura 4 resume as principais atividades definidas pelo *Software Engineering Institute (SEI)* [SEI 2009] simbolizando a prática de LPS. Os três círculos representam as três atividades essenciais do desenvolvimento do *core asset* (*Core Asset Development*), desenvolvimento do produto (*Product Development*) e do gerenciamento (*Management*).

Figura 4. Atividades Essenciais na Prática de LPS [SEI 2009]

Outro conceito relevante em LPS são as *features*. Uma *feature*, conforme [Chastek et al. 2001], é “um aspecto importante visível ao usuário, de boa qualidade, ou característica de um sistema de software ou sistemas de softwares”. As *features* identificadas no domínio são modeladas, em modelo de *features,* que representam os seus relacionamentos, com o objetivo de fornecer uma visualização das funcionalidades comuns e opcionais na LPS. A notação utilizada nesse modelo é resumida na Figura 5. O modelo de *features* é construído no início do desenvolvimento da LPS, logo após a identificação das *features* da LPS, facilitando a seleção de funcionalidades pertencentes a um produto.

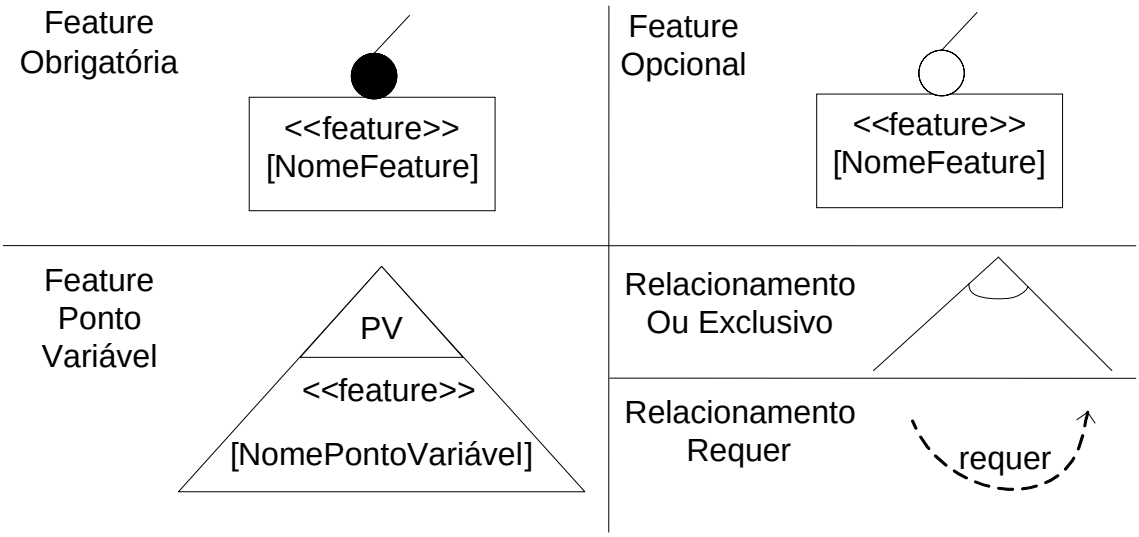

Figura 5. Notação do Modelo de Features [Woojin et al. 2007]

Como a abordagem proposta baseia-se na arquitetura da *Model-Driven Architecture (MDA)*, tem-se uma maior facilidade na implementação e manutenção dos produtos da LPS.

2.2. Model-Driven Architecture (MDA)

A abordagem proposta é baseada na *Model-Driven Architecture (MDA)*, onde as especificações em níveis mais abstratos são transformadas para níveis mais concretos até a implementação do software. Além de contribuir no reuso de modelos nas disciplinas de análise e projeto, a MDA facilita a implementação e manutenção da LPS através da geração parcial de códigos a partir de modelos mais abstratos. A MDA é importante na abordagem proposta pois um mesmo produto ubíquo, construído a partir de modelos da LPS, pode também ser executado em outras arquiteturas, como celulares, *pdas*, *smartphones*, e outros, bastando apenas gerar o código para a nova arquitetura a partir dos modelos.

Esse tipo de abstração de alto nível oferece três grandes vantagens no desenvolvimento de software [Pham et al. 2007]:

- Melhor entendimento do software: a especificação, compreensão e desenvolvimento se tornam mais fáceis com a utilização de modelos;
- "*Model once, generate anywhere*": basta modelar apenas uma vez e gerar quantos códigos forem necessários, tornando a manutenção mais fácil e econômica; e
- Aumento da qualidade e produtividade: como o conhecimento da implementação está ligado a um mecanismo transformador, este pode ser reusado e compartilhado entre diferentes projetos.

Com a MDA alcançam-se as vantagens do desenvolvimento de software dirigido por modelos [Kleppe et al. 2003, Frankel 2003]. A MDA é uma iniciativa da *OMG (Object Management Group)* [OMG 2009] que utiliza linguagens padronizadas para representar os artefatos de software nos diferentes níveis de abstração da arquitetura [MDA 2006].

Essa arquitetura possui dois conceitos relacionados. O primeiro é a independência de plataforma, onde os modelos mais abstratos são especificados, determinando "o que" o software é capaz de realizar. O segundo conceito é a dependência de plataforma, onde os interesses de implementação ou tecnológicos são especificados, determinado "como" o software irá prover tais interesses.

A MDA é composta por três níveis. No primeiro nível da MDA, para tornar a manutenção e evolução de software mais econômica, toda parte de análise do sistema deve ser capturada utilizando um modelo independente de plataforma (*Plataform Idependent Model - PIM*), enquanto que os interesses tecnológicos devem ser separadamente capturados no mecanismo de transformação, que transforma o PIM em modelos de baixo nível. Assim, se

houver uma mudança nas tecnologias de implementação ou nas plataformas de execução, a única coisa que precisará ser alterada será o mecanismo de transformação, sendo que, todo esforço investido na construção dos modelos independentes de plataforma permanecerá intacto [Pham et al. 2007].

No segundo nível da MDA, encontram-se os modelos específicos de plataforma (*Plataform Specific Model – PSM*). Esses modelos devem ser gerados a partir de um PIM através de um mecanismo de transformação, e representam as possíveis implementações do sistema (por exemplo, deve existir um PSM para cada tecnologia de implementação ou plataforma de execução).

No terceiro nível da MDA, têm-se os códigos fonte gerados a partir de um PSM (por exemplo, Java, C++), através de um gerador de códigos. A Figura 6 ilustra esse processo.

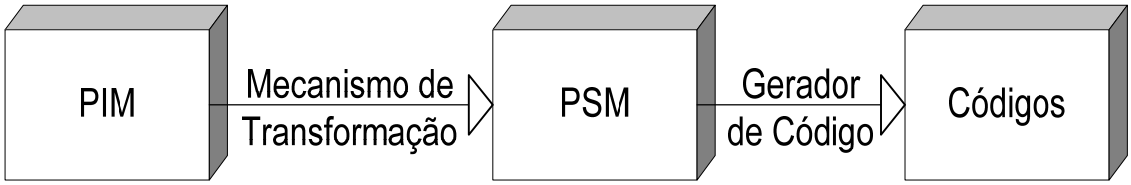

Figura 6. Processo MDA

Os projetos de software para a Computação Ubíqua sofrem constante atualização. O desenvolvimento, manutenção e evolução de tais softwares podem ser beneficiados com o uso da MDA. Pode-se destacar um conjunto desses benefícios [Pham et al. 2007]:

- Com a construção de sistemas utilizando modelos abstratos no lugar de código de baixo nível, características como manutenção, atualização e evolução, serão mais produtivas e econômicas, especialmente no contexto da Computação Ubíqua, onde as tecnologias tendem a mudar

muito rápido e, freqüentemente, atualizações incrementais são necessárias; e

- Trabalhar com o nível de abstração mais elevado permite que desenvolvedores de software enfrentem a heterogeneidade com mais facilidade.

Procurando explorar essas idéias, a abordagem UbiComSPL utiliza PIM's e PSM's para a construção da LPS e seus produtos, no domínio de aplicações ubíquas. Dessa forma, procura-se tirar proveito das vantagens oferecidas pela MDA que possibilita atender diferentes arquiteturas a partir de um mesmo modelo de LPS e seus produtos.

2.3. Computação Ubíqua

O avanço da tecnologia vem fazendo com que o tamanho e a usabilidade dos computadores se modifiquem de forma constante. Há três décadas aproximadamente, existiam grandes computadores centrais, conhecidos como mainframes, "*Era dos Mainframes*". Os usuários dos mainframes tinham que compartilhar esta única máquina para realizar diferentes tarefas. Já na década de 80, existiam os computadores menores e pessoais, em que cada pessoa possui seu próprio computador, a "*Era dos computadores pessoais*" [Jardim 2004]. Com a chegada da computação distribuída e da Internet [Weiser e Brown 2006], tem-se o modelo cliente-servidor, com a vantagem da utilização de recursos oferecidos por outros computadores interligados, a "*Era da Internet e Computação Distribuída*". Essa evolução possibilitou a presença dos computadores em diversos objetos, como carros, etiquetas de roupas, canetas, telefones, de forma invisível para o usuário, sendo denominada "*Era da Computação Ubíqua*". Essa nova era, também chamada de "*Calm Technology*",

tem como objetivo fazer com que os computadores, presentes em certo ambiente, auxilie humanos nas suas tarefas sem que sejam notados [Jardim 2004].

2.3.1. Conceitos

O termo "*Computação Ubíqua*" foi cunhado por *Mark Weiser* enquanto pesquisador do *Xerox Palo Alto Resarch Center (XPARC)* no final da década de 80. *Mark Weiser* definiu como característica da Computação Ubíqua uma intercomunicação entre os computadores presentes em vários objetos (*PDAs*, *PCs*, *laptops*, celulares, geladeiras).

Nesse contexto podem ainda ser definidos três tipos de computação [Jardim 2004]: Computação Tradicional, Computação Pervasiva e Computação Móvel. Na Computação Tradicional os meios de computação são distribuídos no ambiente de trabalho dos usuários de forma perceptível, com um baixo grau de tecnologia embarcada e com uma baixa mobilidade. A Computação Pervasiva os meios de computação são distribuídos no ambiente de trabalho dos usuários de forma imperceptível, com um alto grau de tecnologia embarcada, porém com pouca mobilidade. Na Computação Móvel, a capacidade de mobilidade é alta, porém os meios de computação são distribuídos no ambiente de trabalho dos usuários de forma perceptível, com baixo grau de tecnologia embarcada. A Computação Ubíqua integra a mobilidade da Computação Móvel com o alto grau de tecnologia embarcada da Computação Pervasiva. Assim, qualquer dispositivo computacional, enquanto em movimento, pode construir dinamicamente, modelos computacionais dos ambientes e configurar seus serviços dependendo da necessidade. A Figura 7

mostra os graus de tecnologia embarcada e mobilidade na Computação Tradicional, Pervasiva, Móvel e Ubíqua.

A Computação Ubíqua pode ser vista como uma junção das Computações Móvel e Pervasiva [Araújo 2003, Lyytinen e Yoo 2002] como mostra a Figura 8.

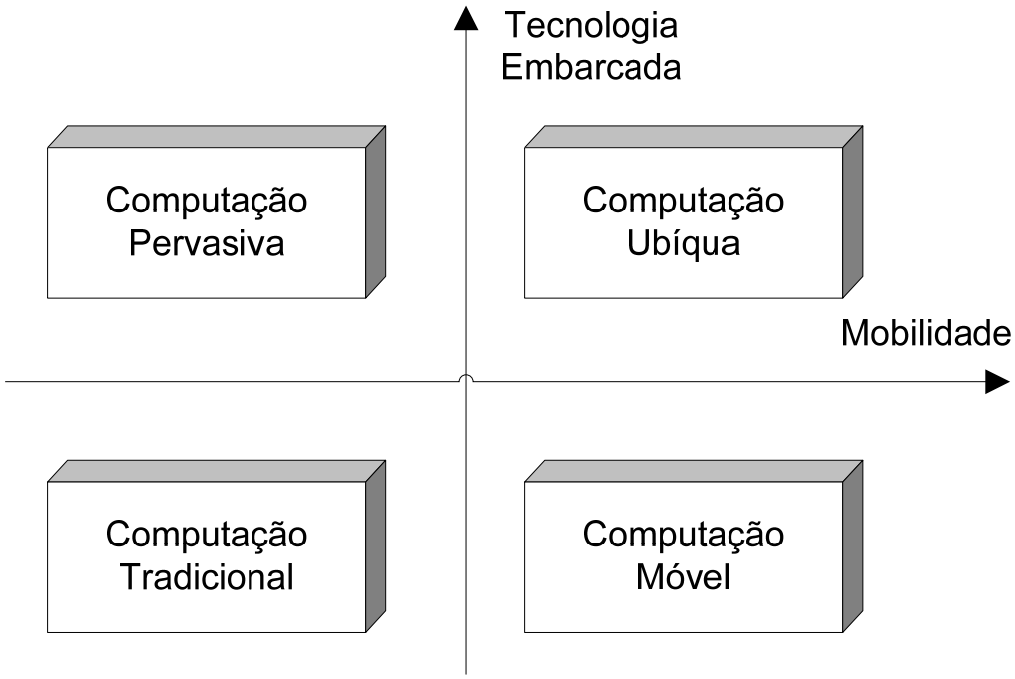

Figura 7. Grau de Tecnologia Embarcada e Mobilidade na Computação Tradicional, Pervasiva, Móvel e Ubíqua [Jardim 2004]

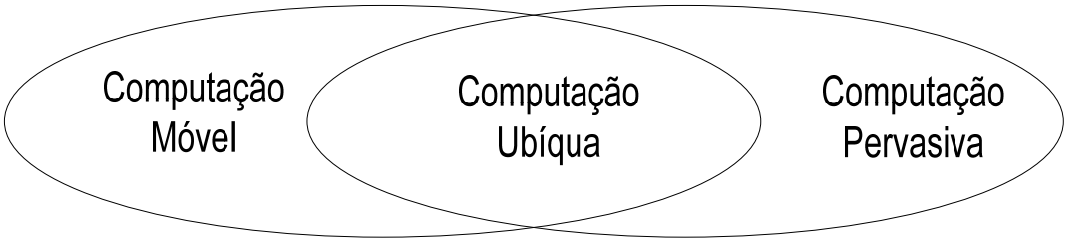

Figura 8. Computação Ubíqua [Araújo 2003]

2.3.2. Desenvolvimento de Software para Computação Ubíqua

A Computação Ubíqua é um vasto domínio que possui diversos dispositivos com características particulares (e.g., tamanho de tela, capacidade de entrada /

saída, memória, processamento, energia), plataformas e tecnologias que exigem diferentes arquiteturas de software.

Diversos requisitos não funcionais devem ser levados em consideração no desenvolvimento de software para Computação Ubíqua. Dentre esses requisitos têm-se [Spínola et al. 2007]: onipresença de serviços, pervasibilidade, ciência de contexto, heterogeneidade, interoperabilidade, privacidade, inteligência, invisibilidade, disponibilidade, adaptação dos serviços oferecidos, autonomia, confiabilidade, portabilidade, reusabilidade, descoberta e composição de serviços, e captura da experiência.

Para atender a esses requisitos diferentes processos de desenvolvimento têm sido propostos [Carton et al. 2007, Fernandes et al. 2007]. Dentre esses processos vem se destacando os que se baseiam nas idéias da MDA [Mostefaoui et al. 2008]. Conforme a MDA, as especificações e funcionalidades do software ubíquo são representadas em um PIM, enquanto que as tecnologias e plataformas são especificadas através de um PSM. Usando mecanismos de transformações, a partir de um PIM, vários PSM's podem ser obtidos. A partir de cada PSM, um código pode ser gerado para uma plataforma específica. Dessa forma, pode-se ter para uma mesma aplicação diferentes arquiteturas conforme se apresenta a Computação Ubíqua.

Considerando o rápido avanço tecnológico e o desenvolvimento da mesma aplicação para vários tipos de plataformas e tecnologias, a manutenção do software para os diferentes dispositivos acaba se tornando uma tarefa difícil. Baseada na MDA, o desenvolvimento de LPS no domínio de aplicações ubíquas, proposto nessa dissertação, oferece a vantagem de possibilitar que

grande parte do código seja gerada a partir da modelagem. Além disso a abordagem utiliza o *Ubiquitous Computing Framework (UCF)* [Santana et al. 2007, Santana et al. 2007a, Santana et al. 2007b, Forte et al. 2008] para apoiar a modelagem e implementação dos requisitos relacionados com a adaptação de conteúdo e o contexto das aplicações ubíquas.

2.4. Ubiquitous Computing Framework (UCF)

Dentre os *assets* que são comuns no *core* de uma LPS na Computação Ubíqua, destaca-se o framework, denominado *Ubiquitous Computing Framework (UCF)*, resultado das experiências de trabalhos desenvolvidos pelo *Grupo de Computação Ubíqua (GCU)* da *Universidade Federal de São Carlos (UFSCar)*. O UCF é uma evolução do *Extended Internet Content Adaptation Framework (EICAF)* [Forte et al. 2008], e vem sendo reutilizado na construção de diferentes aplicações na Computação Ubíqua.

O UCF estrutura as aplicações, realiza as adaptações de conteúdos em dispositivos móveis, considerando os contextos e as diferenças semânticas, e estabelece a comunicação entre cliente e servidor.

No UCF, o contexto é caracterizado por um conjunto de perfis que expressam as capacidades dos dispositivos de acesso, dados pessoais e preferências dos usuários, condições da rede de acesso e especificidades dos conteúdos requisitados. Utiliza ontologias na especificação do contexto das aplicações ubíquas. As adaptações de conteúdos são realizadas por serviços *Web* semânticos distribuídos, os quais podem ser compostos de forma seqüencial ou paralela para suportar serviços mais complexos.

Agentes de software são empregados no gerenciamento das aplicações, provendo a comunicação sob demanda, adaptação e configuração autônoma,

aquisição de contexto, mobilidade e o uso inteligente dos serviços *Web* semânticos. Na especificação dos agentes foi usada a *Multi-Agents System Modeling Language (MAS-ML)* [Torres e Lucena 2007], que oferece um conjunto de recursos que facilitam a Engenharia de Aplicação com multi agentes.

Considerando sua maturidade e seus recursos atualmente disponíveis, o UCF vem sendo utilizado no desenvolvimento de aplicações ubíquas em diferentes domínios como o de *Portfólios Reflexivos Eletrônicos (PRE)*, que hoje está operacional no curso de Medicina da UFSCar. Na abordagem proposta o UCF é responsável pelas adaptações de conteúdos nos diferentes contextos. A adaptação de conteúdo conforme o contexto é um aspecto transversal aos diferentes domínios das aplicações ubíquas, daí sua importância na abordagem proposta.

Para viabilizar a utilização do UCF, o mesmo foi integrado na ferramenta MVCASE. A MVCASE apóia o Engenheiro de Software nas diferentes etapas do processo de desenvolvimento de software proposta nessa dissertação. Com a integração do UCF na MVCASE, tem-se o apoio na modelagem e implementação dos *core assets* da LPS e dos seus produtos ubíquos.

2.5. Multiple-View Case (MVCASE)

Ao observar a história da humanidade, nota-se que a evolução humana está diretamente relacionada ao uso de ferramentas para aumentar as habilidades naturais dos seres humanos. Desde as ferramentas mais primitivas, destinadas à caça e à pesca, até os modernos satélites de comunicação, o homem sempre explorou o produto de sua inteligência na construção de artefatos para sobreviver e evoluir através das eras [Lucrédio 2004].

Na área da computação, mais especificamente na Engenharia de Software, não podia ser diferente. O desenvolvimento de software se apóia em uma grande diversidade de atividades que devem ser executadas para se produzir software com qualidade. Estas atividades exigem diferentes habilidades. Por exemplo, na disciplina de requisitos, o Engenheiro de Software deve agir como um interrogador, consultor, solucionador de problemas e um negociador [Pressman 2001]. Já a implementação exige um raciocínio lógico, voltado a construir linhas de código para instruir o computador a executar exatamente o que é esperado.

Devido a esta diversidade inerente à Engenharia de Software, ferramentas sempre foram utilizadas para auxiliar na execução das tarefas relacionadas ao desenvolvimento de software. Essas ferramentas, denominadas *Computer-Aided Software Engineering (CASE)*, existem desde os primórdios da computação, com os primeiros cartões perfurados, editores, compiladores e depuradores, até as modernas ferramentas para coleta de requisitos, projeto, construção de *Graphical User Interface (GUI)*, modelagem e gerenciamento de banco de dados, gerenciamento de configuração de software, entre outras, formando ambientes completos que cobrem todo ou a maior parte do ciclo de vida de desenvolvimento de software [Harrison et al. 2000].

Existe um grande número de ferramentas CASE no mercado, atuando nas diferentes atividades do *Processo de Desenvolvimento de Software (PDS)* e empresas as utilizam para reduzir tempo na construção de software e para aumentar a qualidade do seu processo. Na comunidade científica, ferramentas CASE são propostas para ilustrar, testar e validar novas idéias, buscando

aprimorar o conhecimento humano nessa área [Lucrédio 2004]. Uma dessas ferramentas é a *Multiple-View Case (MVCASE)* [Lucrédio et al. 2003] que apóia a modelagem baseada na *Unified Modeling Language (UML)* [UML 2009], com geração parcial de código.

Numa primeira versão, para *desktop's,* a ferramenta MVCASE suportava a modelagem de software segundo a UML 1.4. Com a evolução da UML, foi necessário atualizar a ferramenta e integrá-la com o *Integrated Development Environment (IDE)* Eclipse. Assim, a MVCASE foi reconstruída como um *plug-in* com suporte para a UML 2.1.

Um *plug-in* é uma unidade funcional da plataforma Eclipse que pode ser desenvolvida e integrada ao IDE Eclipse. Geralmente uma pequena ferramenta é escrita como um simples *plug-in*, entretanto uma ferramenta maior com mais funcionalidades, pode ser dividida em vários *plug-ins*. O mecanismo de *plug-ins* do Eclipse permite que novas funcionalidades sejam adicionadas através de extensões de *plug-ins* existentes.

Uma grande vantagem da MVCASE em se tornar um *plug-in* do Eclipse, além do suporte a UML 2.1, está na geração de códigos, os quais ficam disponíveis na própria plataforma Eclipse. Assim, os desenvolvedores terão como tarefa o refinamento dos códigos parciais gerados, como, por exemplo, a implementação do corpo de uma função.

O suporte de frameworks consolidados no IDE Eclipse permitiram o rápido desenvolvimento do *plug-in* MVCASE. Para o desenvolvimento do *plug-in* MVCASE foram utilizados os frameworks: *Eclipse Modeling Framework (EMF)* [EMF 2009]; *Graphical Editing Framework (GEF)* [GEF 2009]; e *Graphical Modeling Framework (GMF)* [GMF 2009].

O desenvolvimento de uma aplicação geralmente começa com a consideração de um modelo, depois avança para tarefas orientadas à interface. O *Eclipse Modeling Framework (EMF)* foi desenvolvido para facilitar o projeto e implementação de metamodelos, como o da UML 2.1. A Figura 9 apresenta uma parte do metamodelo da UML 2.1, utilizado pela MVCASE, projetado e persistido em um arquivo *ecore* (arquivo onde metamodelos são definidos) pelo EMF.

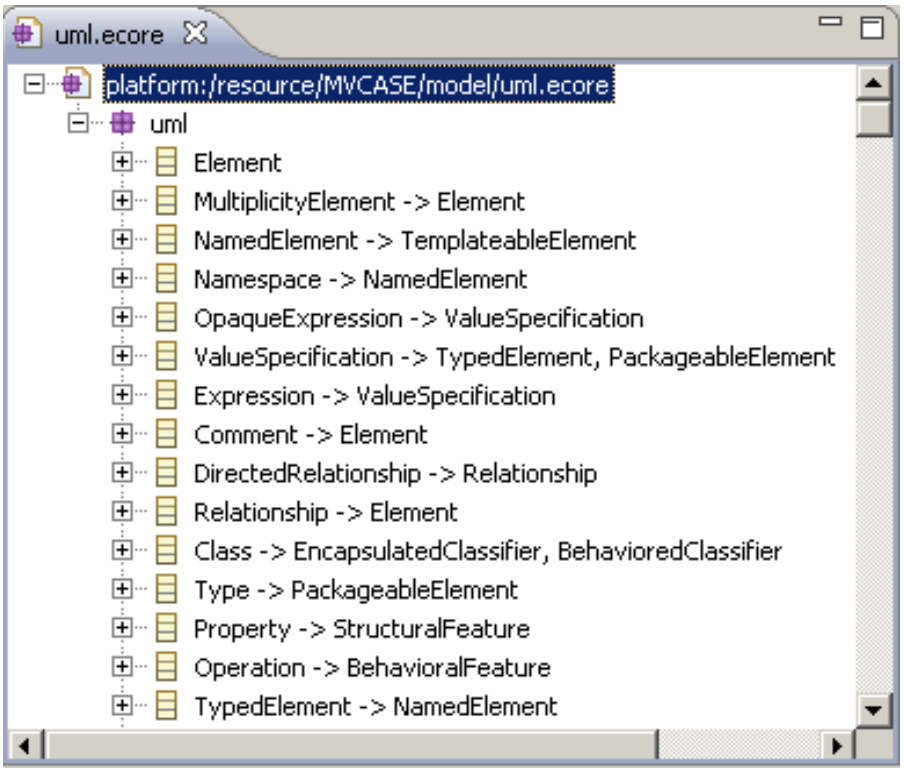

Figura 9. Metamodelo da UML 2.1

O *Graphical Editing Framework (GEF)* apoiou o desenvolvimento do editor para representações gráficas dos metamodelos da UML 2.1 e o *Graphical Modeling Framework (GMF)* integrou as funcionalidades do EMF e GEF na construção do *plug-in*. A Figura 10 mostra parte do arquivo de configuração, com a extensão *gmfmap*, que contem as especificações da integração do metamodelo da UML (EMF) com o editor para representações gráficas (GEF) para o Diagrama de Classes. Na propriedade *Element*,

especifica-se o elemento do metamodelo, na propriedade *Diagram Node*, especifica-se a representação gráfica para o elemento do metamodelo e na propriedade *Tool*, especifica-se o elemento da barra de ferramentas que cria a representação gráfica.

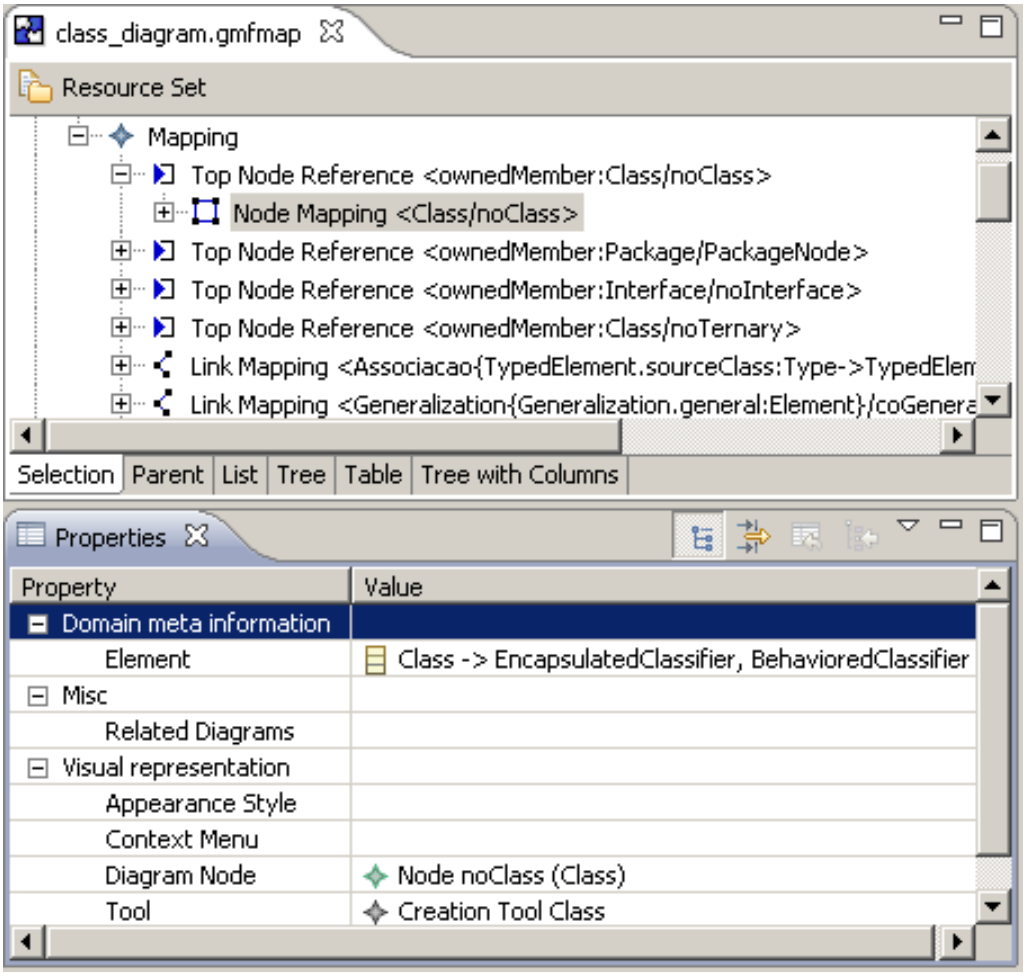

Figura 10. Arquivo de Integração gmfmap

Para a geração parcial de código, o *plug-in* MVCASE utiliza o *Java Emitter Templates (JET)* [JET 2009]. O JET é tipicamente utilizado na implementação de um "*gerador de código*". Um gerador de código é um importante componente quando se adota a arquitetura da MDA. Como o objetivo do MDA é descrever um sistema de software utilizando modelos abstratos e transformar esses modelos até a implementação, o JET é o componente do EMF adequado para automatizar este processo. O JET tem a

capacidade de gerar qualquer tipo de arquivo de saída baseado em um modelo, como por exemplo, Java [Java 2009], SQL, XML. Para isso, o JET utiliza *templates* que são responsáveis por interpretar um arquivo de entrada e gerar uma saída conforme o objetivo do *template*. A Figura 11 apresenta um pequeno trecho parte do *template* utilizado pela MVCASE para a geração de código Java. A tag <c:choose> é uma condicional onde os possíveis valores são declarados em <c:when test = [VALOR]> e <c:otherwise>. Nesse trecho a tag <c:choose> é responsável pela declaração do pacote, podendo ser um nome pré-definido ou *default.* A tag <c:if test = [VALOR]> também é uma condicional. No trecho apresentado, a primeira tag <c:if test = [VALOR]> verifica se é necessário declarar os *imports,* que tratam coleção em java, no arquivo de saída e a segunda tag <c:if test = [VALOR]> verifica qual é visibilidade da classe e a imprime no arquivo de saída.

Class.java.jet

```
<c:choose select="$containPackage">
        <c:when test = "0" >
package <c:get select="$PackageDefault/@name" />;
        </c:when>
        <c:otherwise>
package <c:get select="$currPackage/@name" />;
        </c:otherwise>
</c:choose>

<c:if test="($currClass/propertyAssociation/@* != '')
import java.util.Collection;
import java.util.Iterator;
</c:if>


<c:if test="$currClass/@visibility != 'package'" >
```

Figura 11. Template para Geração de Código Java

Como o JET utiliza *templates* para geração de código, o *plug-in* MVCASE pode ser estendido para geração de códigos em outras linguagens, além de Java.

Atualmente o *plug-in* MVCASE suporta os seguintes diagramas da UML 2.1: Casos de Uso, Classes, Componentes, Seqüência e de Estados. A geração de código é realizada com base em Diagrama de Classes.

Os diagramas da MVCASE são classificados como sendo estruturais ou comportamentais. Os diagramas de Classes e Componentes fornecem uma visão estruturada em uma coleção de pacotes, classes, relacionamentos e componentes. Os diagramas de Caso de Usos, Seqüência e Máquina de Estados fornecem uma visão comportamental. Por exemplo, a Figura 12 mostra parte do metamodelo de Classes do *plug-in* MVCASE.

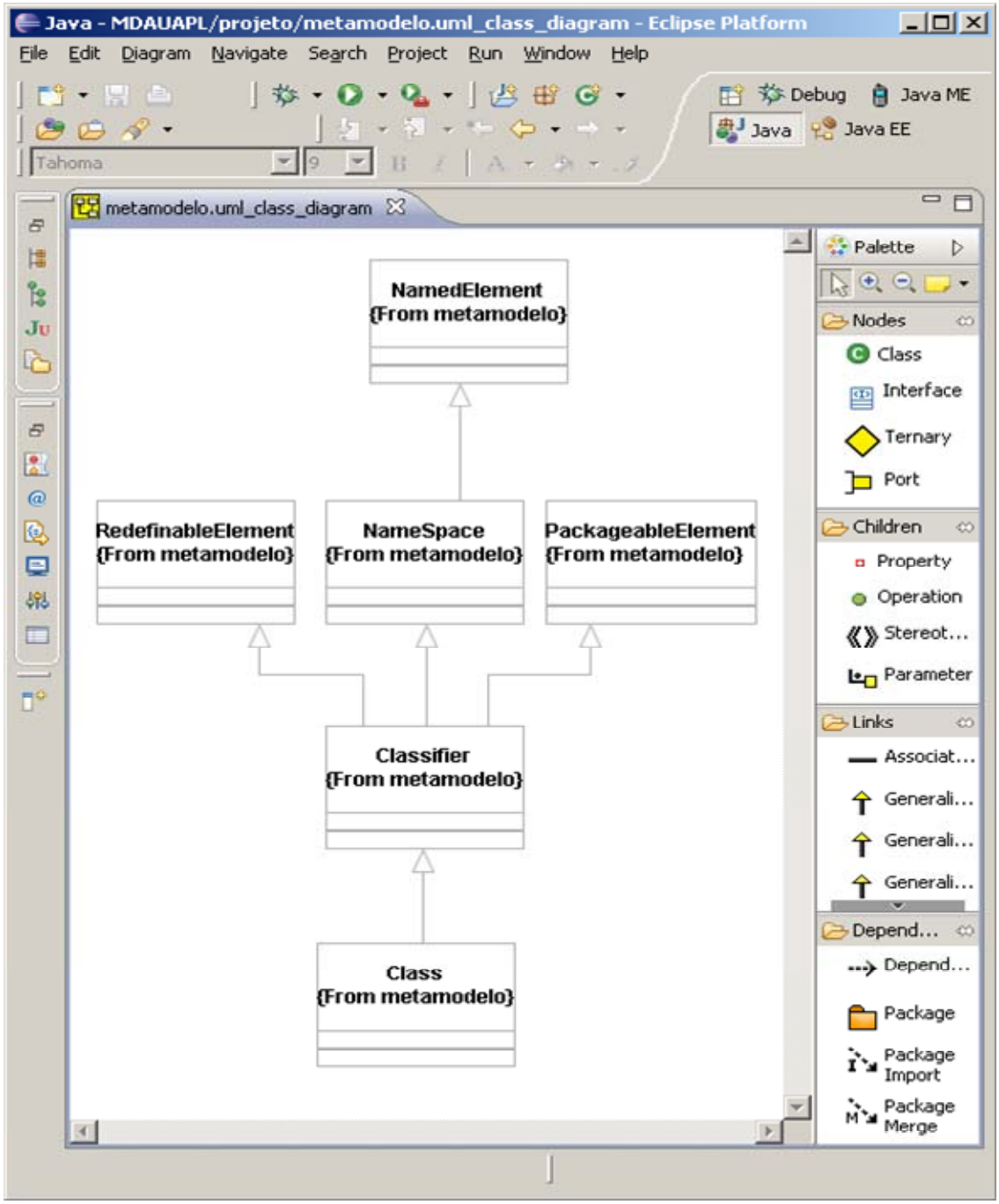

Figura 12. Metamodelo de Classes da MVCASE

Conforme Figura 12, a metaclasse *Class* herda de *Classifier* incorporando recursos das metaclasses *RedefinableElement*, *Namespace* e *PackageableElement*. Esses recursos possibilitam especificar o nome, atributos, métodos, relacionamentos (herança, composição, agregação,

associação, dependências, e outros), cardinalidades, pacotes, e outros elementos que definem uma Classe na UML 2.1.

Capítulo 3

3. Trabalhos Relacionados

Existem diferentes propostas para o desenvolvimento de aplicações ubíquas baseadas no reuso, que utilizam *frameworks*, *middlewares*, ferramentas, linguagens, metodologias e padrões de projeto.

No trabalho de Lee [Woojin et al. 2007] é proposta uma abordagem para o desenvolvimento *middlewares* em redes de sensores, utilizando LPS. Embora o método desse trabalho para o desenvolvimento do *core asset* e seus produtos sejam semelhantes ao da abordagem apresentada nesta dissertação, o mesmo não é baseado na MDA.

Os trabalhos de Carton [Carton et al. 2007] e Fernandes [Fernandes et al. 2007] empregam a MDA na Computação Ubíqua. A MDA é usada para facilitar o desenvolvimento de aplicações independentes de plataforma, uma vez que os requisitos da Computação Ubíqua podem ser altamente mutáveis. O trabalho de Carton combina *Apect-Oriented Software Development* e MDA, para separação de interesses, facilitando a elicitação das características adaptáveis (e.g., dispositivo, localização, usuário) para a obtenção de componentes pouco acoplados, que facilitam o reuso. A abordagem proposta, difere dessas abordagens por ser orientada para LPS, o que possibilita a

criação de uma família de produtos derivados de um mesmo núcleo, mas que sejam específicos para os diferentes dispositivos usados numa aplicação ubíqua.

No trabalho de Bragança e Machado [Bragança e Machado 2007] apresenta-se o uso da MDA na construção de uma LPS usando ferramentas baseadas no EMF e UML. A abordagem proposta difere deste trabalho por oferecer suporte para o desenvolvimento de LPS com o uso da ferramenta MVCASE em conjunto com o framework UCF, facilitando o reuso de software no domínio da Computação Ubíqua.

Dentre os trabalhos correlacionados com o uso de ferramentas CASE, como a MVCASE, tem-se a ArgoUML, a Omondo, a ferramenta IBM Rational Rose Data Modeler e o Netbeans.

A ArgoUML tem como vantagem ser um projeto Open Source que suporta a modelagem UML, porém apenas em sua versão 1.4. A MVCASE possui as mesmas características com o suporte a UML 2.1.

As ferramentas Omondo e IBM Rational Rose Data Modeler suportam a especificação com a UML 2.1 e a transformação de modelos conforme a MDA. Contudo, a maior parte das funcionalidades dessas ferramentas não é gratuita, dificultando sua utilização para muitas Instituições e pequenas Empresas.

A ferramenta de modelagem do Netbeans é semelhante atual versão da MVCASE. A escolha da MVCASE vem do conhecimento de suas especificações e códigos, o que facilita sua evolução e configuração para os diferentes projetos de pesquisa desenvolvidos em Instituições de Ensino além de ser uma ferramenta *Open-Source*.

Atualmente, a grande maioria das ferramentas utilizadas, principalmente para apoiar a modelagem de software, não é gratuita e não tem seu código fonte aberto para possíveis extensões nas diferentes pesquisas. A MVCASE é uma ferramenta gratuita, que pode auxiliar o Engenheiro de Software, contribuindo para produzir software com melhor qualidade.

Capítulo 4

4. Abordagem de Desenvolvimento baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubíquas

Motivados pelas idéias, conceitos e tecnologias apresentadas, e tendo como base o trabalho de Oliveira [Oliveira et al. 2009], elaborou-se a abordagem para o Desenvolvimento, baseado em MDA, de LPS no Domínio de Aplicações Ubíquas (*Product Line Development based on MDA for the Ubiquitous Computing Domain - UbiComSPL*), proposta nessa dissertação. Conforme mostra a Figura 13, a abordagem é constituída de duas fases: Engenharia de Domínio (ED) e Engenharia de Aplicação (EA). Em cada uma das fases têm-se as disciplinas de Requisitos, Projeto e Implementação, conhecidas dos processos de desenvolvimento de software, e modificadas para atender as necessidades da MDA, de LPS e da Computação Ubíqua.

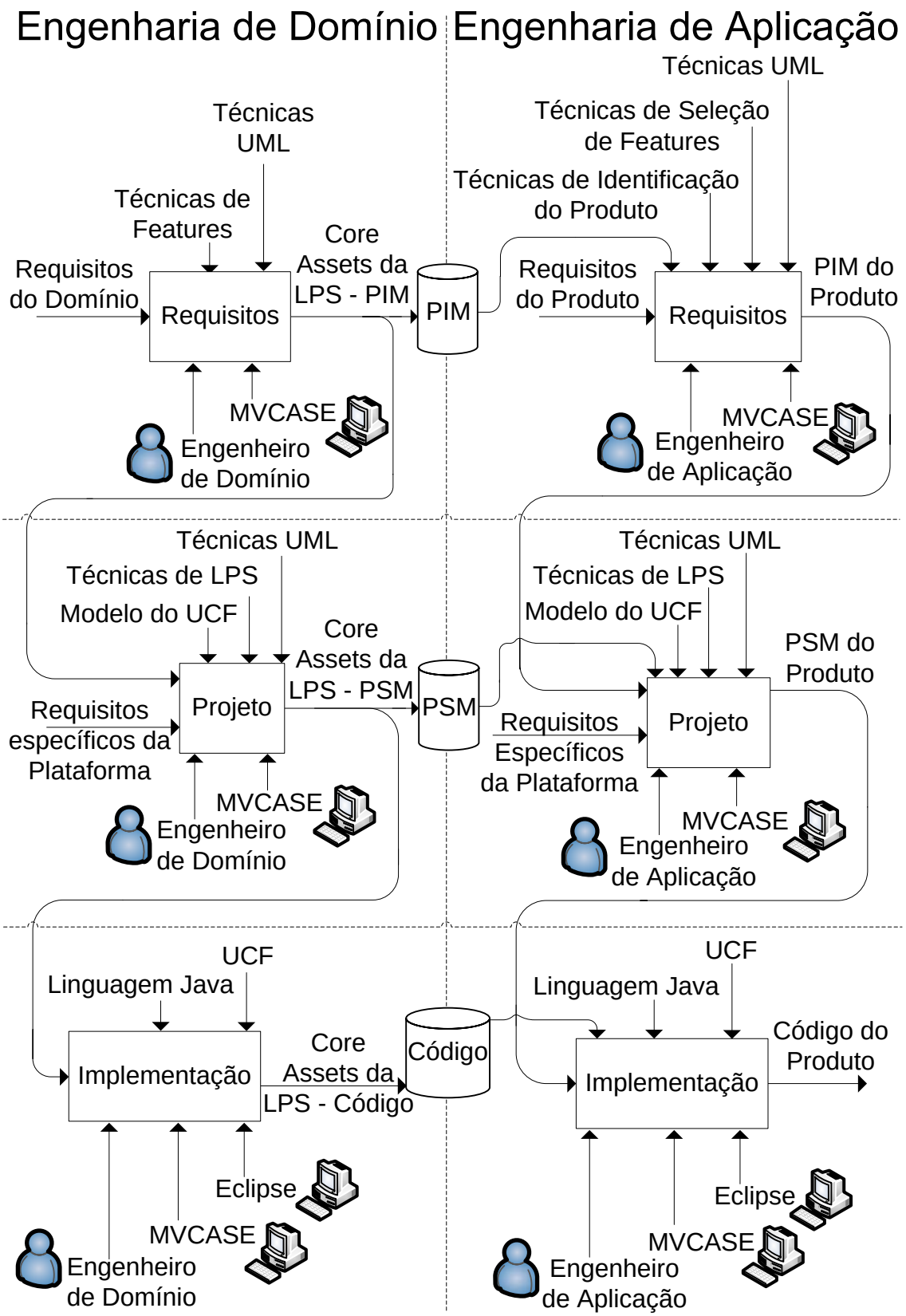

Figura 13. ED e EA na Abordagem Proposta

A ED parte dos requisitos do domínio do problema para obter os *core assets* da LPS, para serem reutilizados na EA, onde são construídos os produtos derivados da LPS. Tanto na ED para construção dos *core assets*, como na EA para construção dos produtos, seguindo os conceitos da MDA, são obtidos os PSMs a partir dos PIMs, e os códigos das implementações a partir dos PSMs.

Para facilitar o entendimento da ED, apresentada nesse capítulo, a abordagem proposta utiliza exemplos de alguns domínios de aplicações ubíquas, destacando o de *Portfólios Reflexivos Eletrônicos (PRE)*. Este domínio tem sido pesquisado pelo GCU do Departamento de Computação, em parceria com o Curso de Medicina, da UFSCar. Um Portfólio Reflexivo Eletrônico é um instrumento de registro de documentos on-line que permite a reflexão dos documentos por parte dos alunos e professores do curso de Medicina, realizado de forma sistemática. A seguir apresenta-se a Engenharia de Domínio.

4.1. Engenharia de Domínio

A Engenharia de Domínio (ED) é responsável pelo desenvolvimento dos *assets* que juntos formam o *core asset*, plataforma da LPS, contendo as funcionalidades necessárias para construção dos produtos. A primeira disciplina da ED, a disciplina de requisitos, trata da especificação dos requisitos

do domínio do problema, que são comuns a diferentes aplicações desse domínio.

4.1.1. Requisitos

Na primeira fase da ED, tem-se a engenharia de requisitos, que é responsável por identificar e gerenciar os requisitos, atendendo aos requisitos necessários para a construção dos produtos no domínio da LPS. Os requisitos de uma LPS podem ser comuns (encontrados em todos os produtos da LPS) ou opcionais (encontrados em alguns produtos da LPS).

Nessa etapa do desenvolvimento, os requisitos da LPS são elicitados, especificados, analisados e verificados. Inicialmente são identificados os *stakeholders*, que interagem no domínio do problema, tornando claras as suas funcionalidades na LPS. Em seguida, especificam-se os requisitos em modelos independentes de plataforma, de forma gráfica e textual. A análise desses modelos e uma verificação, do Engenheiro de Domínio, visam assegurar que esses documentos de requisitos representem as reais funcionalidades da LPS.

Uma das técnicas utilizadas para especificar os requisitos, nessa etapa, é o diagrama de caso de uso, que é a base para a identificação de uma *feature* da LPS. A Figura 14 mostra a identificação de *features* a partir de casos de uso no domínio de Segurança Residencial. No caso, a *feature* “Câmera” representa o caso de uso “Ativar Câmera de Vigilância” e a *feature* “Detector de Movimentos” representa o caso de uso “Ativar Detector de Movimentos”.

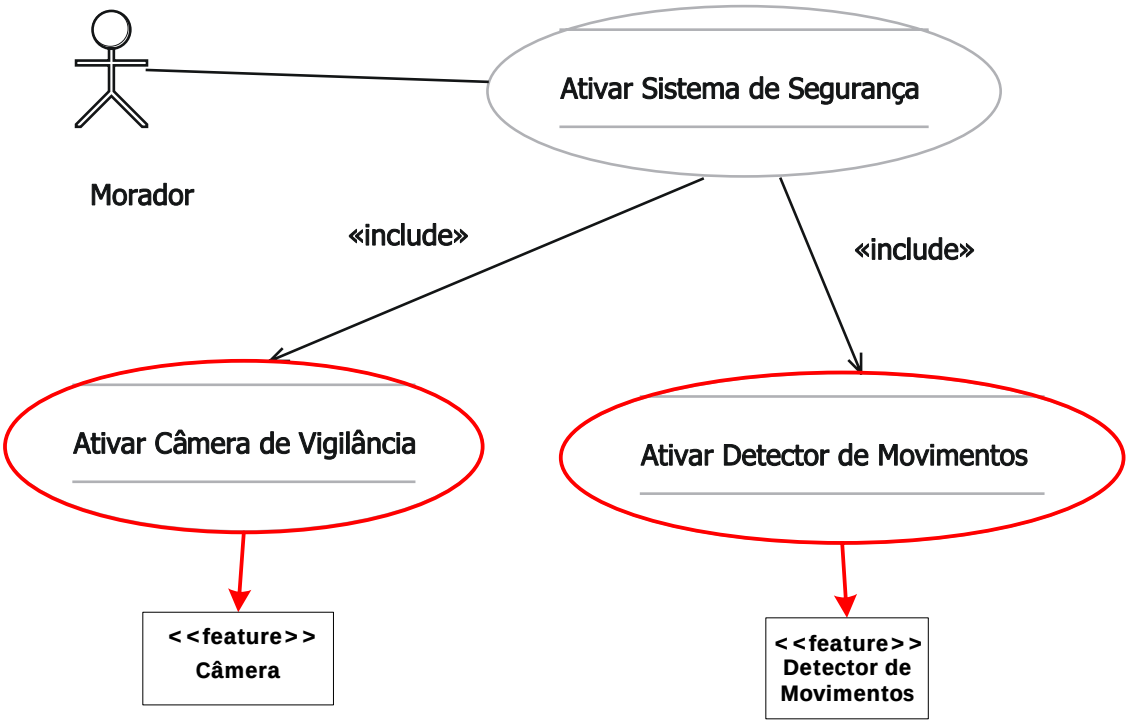

Figura 14. Exemplo de Identificação de Features no Domínio de Segurança Residencial

Uma *feature* pode representar um ou mais casos de uso. No domínio de PRE, por exemplo, tem-se a *feature* “Gerenciar Atividade”, que representa os casos de uso: “Gerenciar Reunião”, “Gerenciar Pequeno Grupo” e “Gerenciar Prática”, conforme mostra o modelo da Figura 15.

Para organizar e facilitar a compreensão, auxiliar nas transformações (PIM para PSM) e na geração de código a partir do PSM, esses modelos são anotados com estereótipos [Wirfs-Brock 1993]. No modelo da Figura 15, o estereótipo <<Kernel>> representa funcionalidades comuns a todos os produtos da LPS e o estereótipo <<Optional>> representa as funcionalidades opcionais conforme o produto a ser construído. Em seguida têm-se as técnicas para modelagem e mapeamento das *features* em produtos.

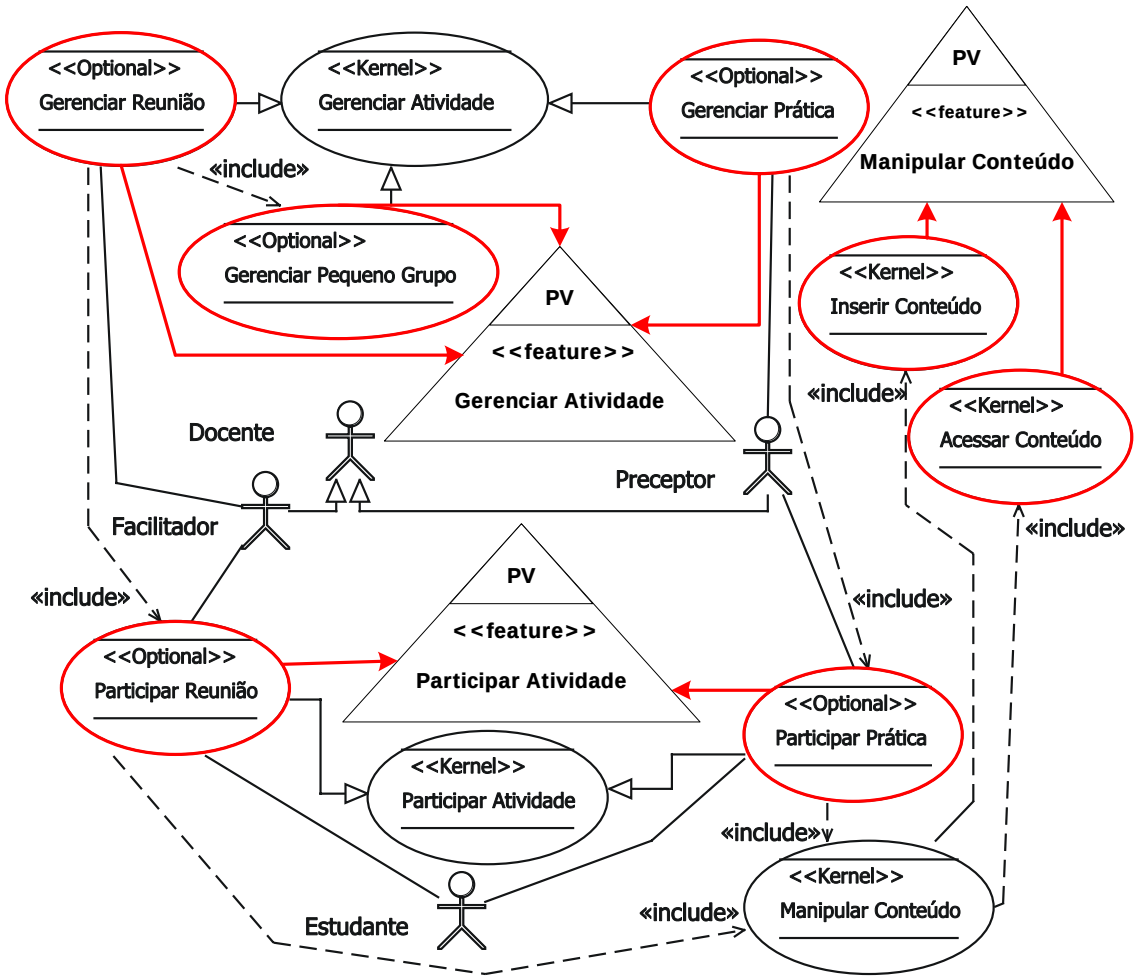

Figura 15. Casos de Uso e Features do Domínio de PRE

O modelo de *features* relaciona as *features*, fornecendo uma visualização das funcionalidades comuns e opcionais na LPS. O modelo de *features* no domínio de Segurança Residencial é apresentado na Figura 16. Nesse modelo, PV representam *features* que são pontos variáveis. O círculo preenchido (preto) indica um relacionamento que associa uma *feature* obrigatória ou variável, e o círculo vazio associa uma *feature* opcional ou variável. Por exemplo, têm-se no domínio de Segurança Residencial as *features* que representam pontos variáveis (“Fechadura da Porta” e “Segurança”), *features* obrigatórias (“Fechadura de Porta”, “Segurança”,

“Teclado Numérico”, “Impressão Digital” e “Câmera”) e *features* opcionais (“Detector de Movimentos”).

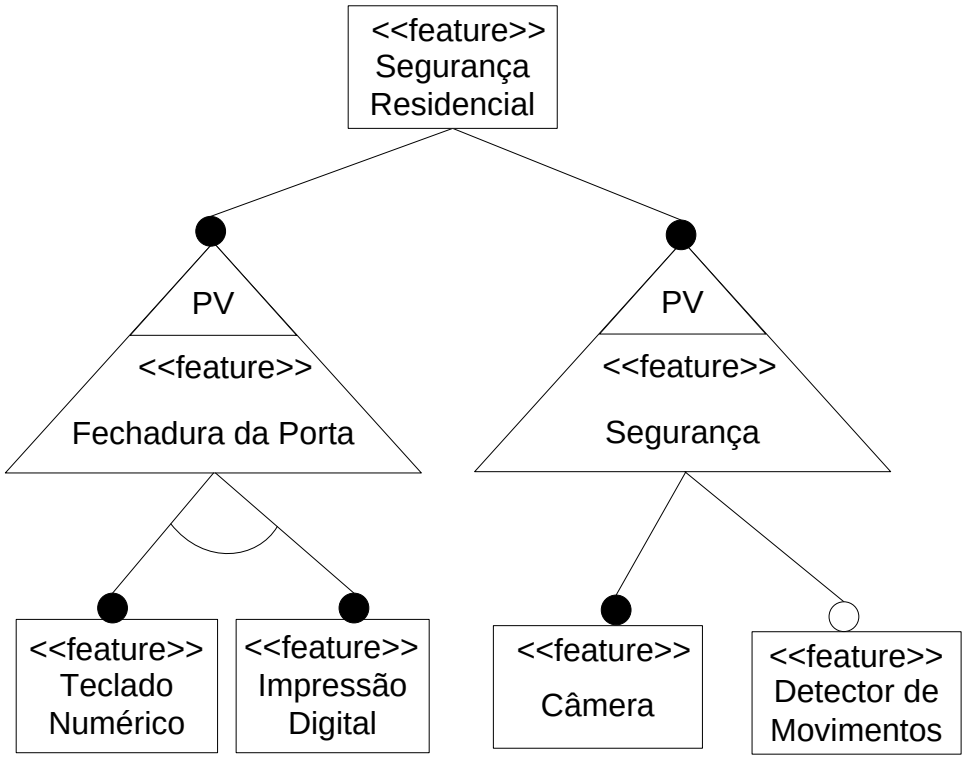

Figura 16. Modelo de Features no Domínio de Segurança Residencial

Um modelo de *features* pode representar *features* relacionadas aos requisitos funcionais do domínio do problema, e *features* relacionadas aos requisitos não funcionais, tais como o de adaptação de conteúdo e ciência de contexto. Organizando esses requisitos em domínios mais específicos pode-se refatorar aqueles que são comuns às diferentes aplicações, obtendo-se *features* que podem ser reutilizadas, evitando repetições. Por exemplo, no modelo da Figura 17, têm-se *features* que tratam das funcionalidades essenciais do PRE, relacionadas aos requisitos funcionais, e as *features* (sombreadas) relacionadas aos requisitos não funcionais de Adaptação de Conteúdo e Ciência de Contexto (UCF). Assim, as *features* de Adaptação de Conteúdo e Ciência de Contexto, podem ser refatoradas e organizadas em um domínio especifico, para serem reutilizadas em outros domínios que requisitam

essas funcionalidades. Nesse modelo, por exemplo, as *features* “Inserir Conteúdo”, “Acessar Conteúdo” e “Descobrir Contexto” são obrigatórias para todos os produtos da LPS, e a *feature* “Adaptar Conteúdo” é opcional, dependendo do produto.

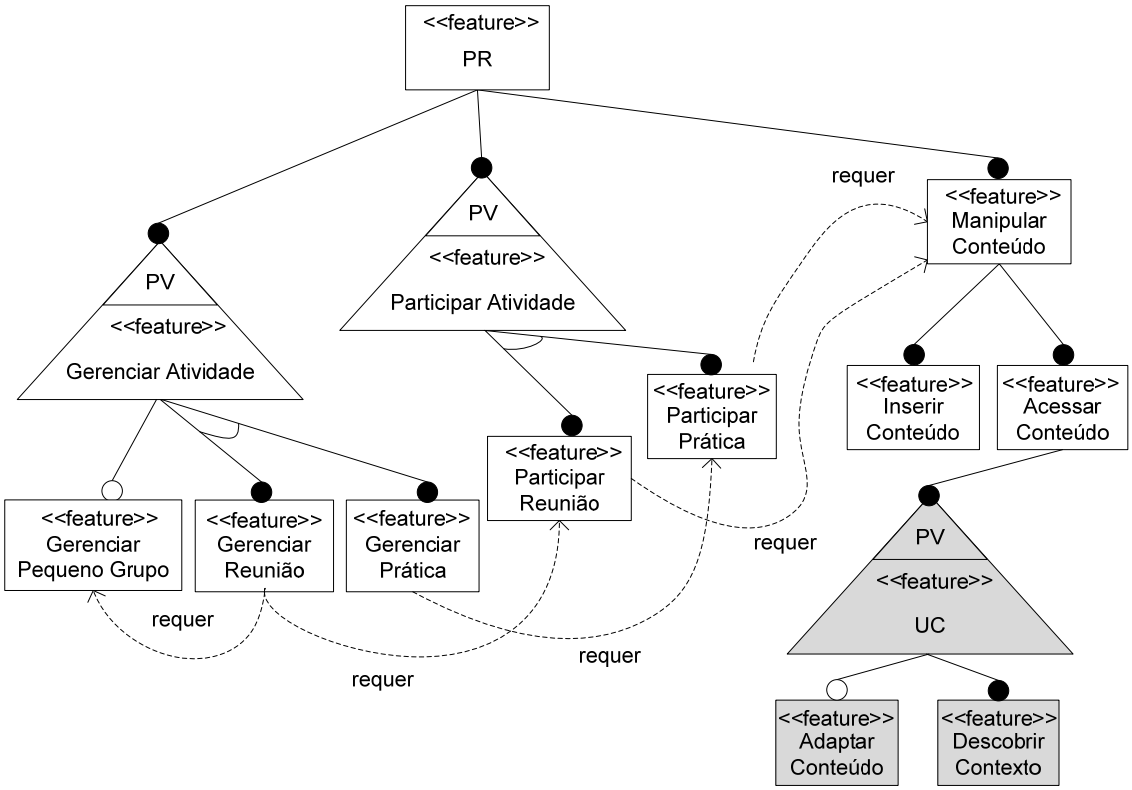

Figura 17. Modelo de Features do Domínio de PRE

Ainda neste passo da abordagem, as *features* são relacionadas com os possíveis produtos da LPS. Esse relacionamento permite a identificação de funcionalidades comuns e variáveis na LPS. No domínio de Segurança Residencial por exemplo, conforme Tabela 1, os possíveis produtos são: *Apto-Seguro*, responsável pela segurança em apartamentos, *Condominio-Seguro,* responsável pela segurança em condomínios e *Casa-Seguro*, responsável pela segurança em casas. A *feature* “Câmera” é uma *feature* comum a todos os produtos do domínio, enquanto que as outras *features* são opcionais conforme o produto a ser construído.

Tabela 1. Mapeamento Features em Produtos no Domínio de Segurança Residencial

Features / Produtos			Apto-Seguro	Condominio-Seguro	Casa-Seguro
Segurança Residencial	Fechadura Da Porta	Teclado Numérico		V	
		Impressão Digital	V		V
	Segurança	Câmera	V	V	V
		Detector de Movimentos		V	

No caso do domínio de PRE os principais produtos identificados foram: *LaptopPRE*, *CellphonePRE* e *PDAPRE*. O *LaptopPRE* visa apoiar as atividades do processo de aprendizagem realizadas dentro da Universidade. O *CellphonePRE* e *PDAPRE* apóiam as atividades realizadas em Unidades de Saúde da Família, Hospital-Escola e na comunidade. A Tabela 2 mostra o mapeamento das *features* em Produtos no Domínio de PRE.

Tabela 2. Mapeamento de Features em Produtos no Domínio de PRE

Features / Produtos			LaptopPRE	CellPhonePRE	PDAPRE
UCF	Adaptar Conteúdo			V	V
	Descobrir Contexto		V	V	V
PRE	Participar Atividade	Participar Reunião	V		
		Participar Prática		V	V
	Manipular Conteúdo	Inserir Conteúdo	V	V	V
		Acessar Conteúdo	V	V	V
	Gerenciar Atividade	Gerenciar Reunião	V		
		Gerenciar Prática		V	V
		Gerenciar Pequeno Grupo	V		

Baseado nos modelos de *features* o Engenheiro de Domínio especifica os modelos de classes, com seus atributos, comportamentos e relacionamentos, conforme mostra a Figura 18 no domínio de PRE. Esses modelos independentes de plataforma (PIM) são refinados com as informações que descrevem com mais detalhes as *features* em cada domínio.

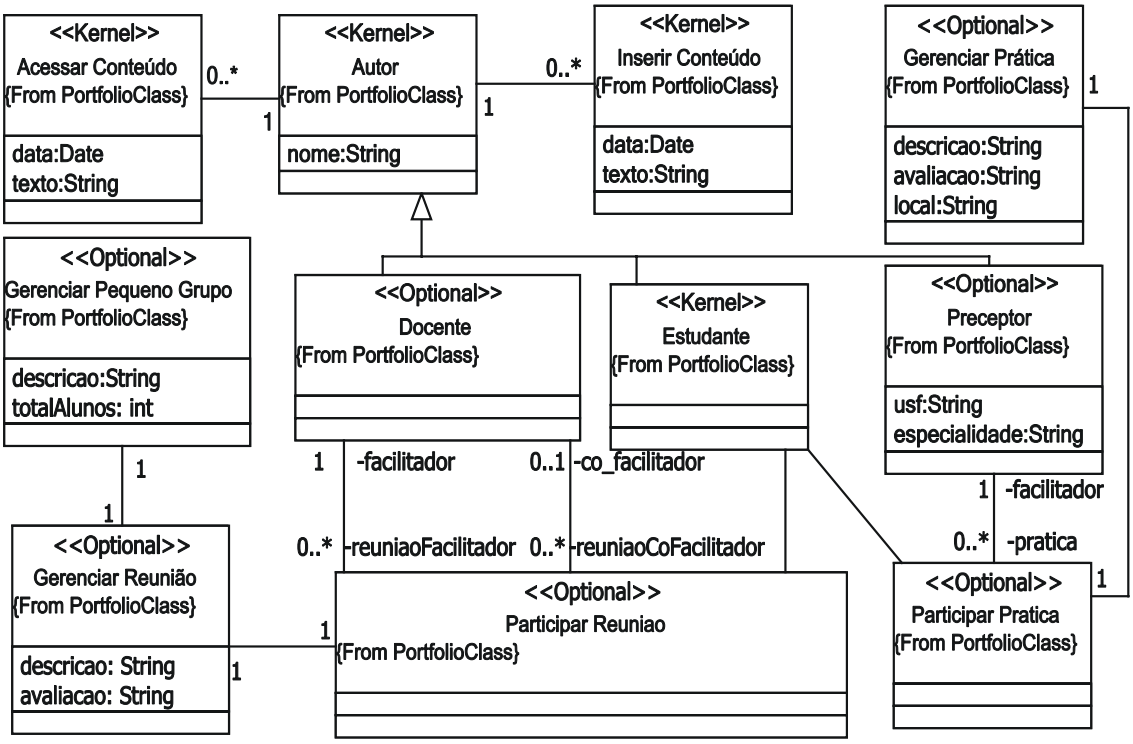

Figura 18. PIM da Linha de Produto de Software PRE

4.1.2. Projeto

A próxima fase da ED é caracterizada pela adoção da plataforma de hardware e software nos quais os produtos serão construídos e implantados. Tem-se nessa fase a definição da arquitetura da LPS, o refinamento dos modelos de classes da fase anterior, e a especificação dos componentes baseado nos modelos de classes.

Na abordagem, tem sido adotada uma arquitetura conforme o padrão *Model-View-Controller (MVC)* [Krasner e Pope 1988, Gamma et al. 1995]. Esta arquitetura é bem conhecida e facilita a reutilização dos *core assets* organizados nas camadas Visão, Controle e Modelo.

Uma das técnicas utilizadas nessa etapa é o reuso dos modelos de classes de frameworks e componentes previamente construídos. Um dos frameworks que vem sendo utilizado é o *Ubiquitous Computing Framework (UCF)*, que trata das *features* de Adaptação de Conteúdo e Ciência de

Contexto. Com a integração do UCF na MVCASE, conforme mostra a Figura 19, tem-se uma maior facilidade para a reutilização de suas classes durante a fase de projeto.

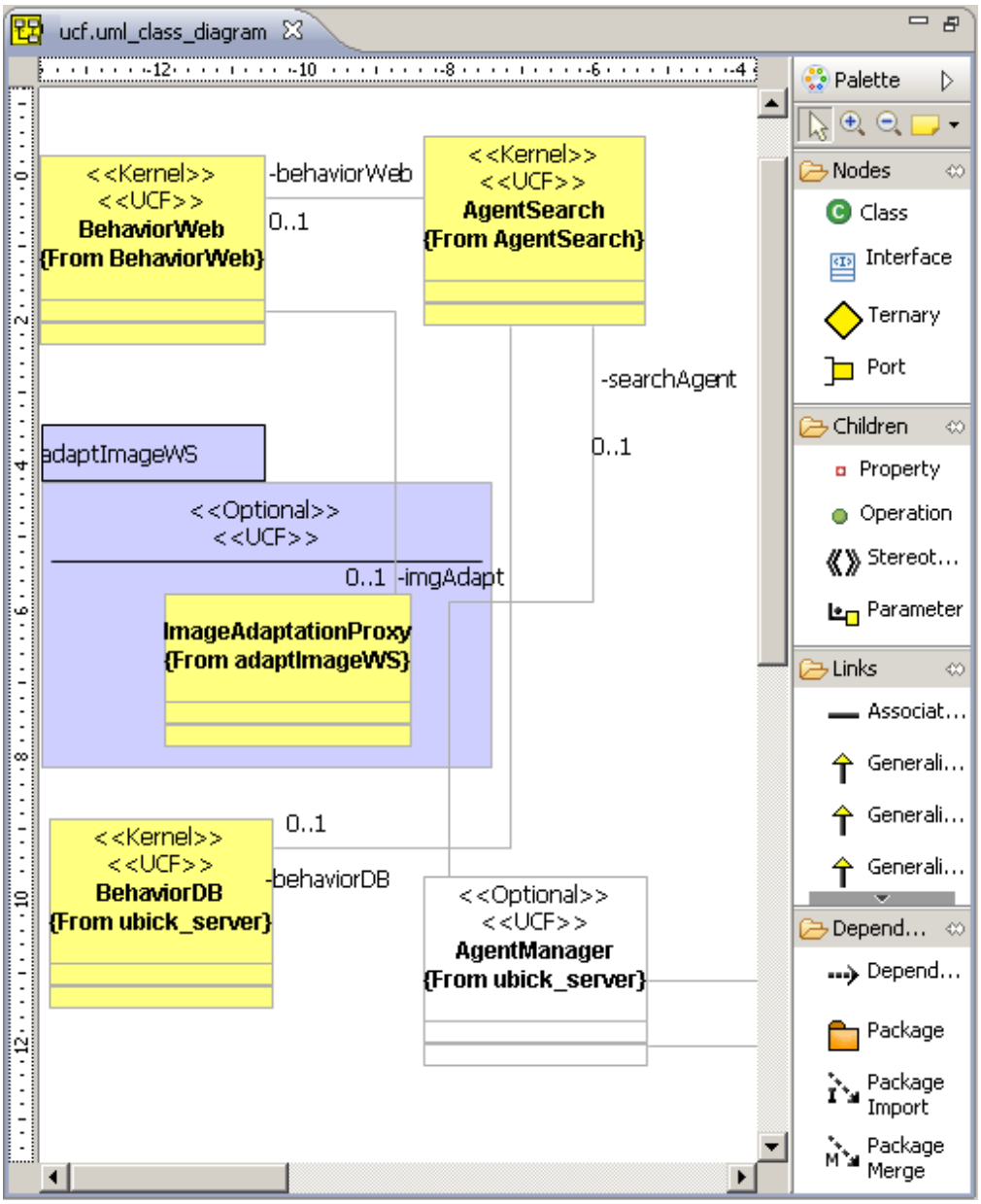

Figura 19. Modelo de Classes do UCF

Nessa etapa do desenvolvimento, o Engenheiro de Domínio refina os modelos de classes independentes de plataforma (PIM), obtidos na etapa anterior, considerando o reuso dos modelos do UCF. No refinamento é realizada a especificação detalhada de: protótipos dos métodos e atributos dos objetos.

No projeto da LPS para o domínio de PRE foi adotada a plataforma java para o produto *LapotopPRE* e *Java Micro-Edition (JME)* [Java 2009b] para os produtos móveis. Os *Sistemas de Gerenciamento de Bancos de Dados (SGDB)* adotados foram o Postgres e *Record Management System (RMS) para* o banco de dados nos dispositivos móveis. O servidor de aplicação utilizado foi *Glassfish* da *Sun* [Glassfish 2009].

Para a plataforma java e JME, foram refinados e organizados os modelos especificados na disciplina de Requisitos. A refatoração organizando por domínios específicos, como o de Adaptação de Conteúdo e Ciência de Contexto, facilita o entendimento e orienta implementação na próxima etapa da abordagem. Por exemplo, a Figura 20 mostra um modelo de classes de projeto no domínio de PRE, onde os requisitos não funcionais de Adaptação de Conteúdo e Ciência de Contexto foram separados para pertencer ao domínio do UCF. Assim, o estereótipo <<UCF>> indica que o elemento faz parte do *Ubiquitous Computing Framework* e o <<PRE>> indica um elemento do domínio de *Portfólio Reflexivo Eletrônico*. Ainda nesse modelo, técnicas de LPS para identificação de *assets* obrigatórios (*Kernel*) e opcionais (*optional*) são utilizadas através de esteriótipos. Outra decisão de projeto, já adotada na ED, é a que utiliza o padrão MVC para separar os assets da LPS nas camadas Visão, Modelo e Controle. Outros padrões e frameworks, como o *Java Server Faces (JSF)* [Sun 2009] e *Hibernate* [Hibernate 2009], também podem ser incorporados nesta etapa do projeto, ampliando o reuso dos *core assets*.

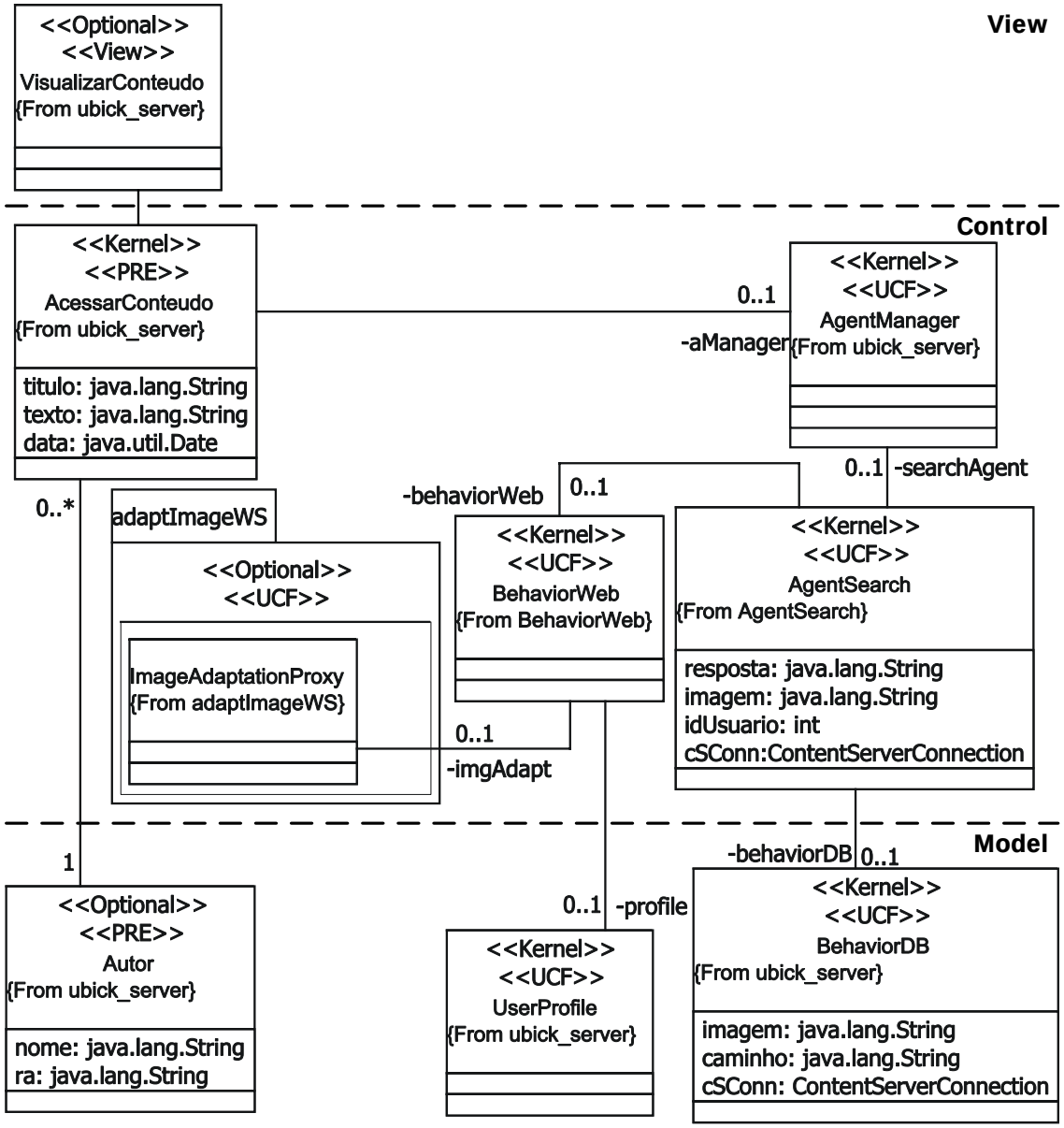

Figura 20. PSM da Linha de Produto de Software PRE

A estruturação das classes em componentes ou, se possível, em padrões e *frameworks*, facilita ainda mais a reutilização dos *core assets*. Assim, com uma análise dos modelos, e dependendo do conhecimento do Engenheiro de Domínio, pode-se refinar e estruturar as classes em componentes, padrões, e *frameworks*. Por exemplo, no domínio da Computação Ubíqua, os componentes para Adaptação de Conteúdo e Ciência de Contexto são reutilizados em diferentes domínios, como o do PRE. Portanto, esses

componentes foram projetados no UCF. Assim, os componentes do UCF são integrados aos componentes do domínio do problema, no caso o domínio PRE.

Para que se tenha uma idéia da especificação dos componentes a partir das especificações das classes, a Figura 21 apresenta um modelo de componentes no domínio de PRE. Assim, o componente *VisualizarConteudo* é responsável por fornecer uma interface para exibir os conteúdos nos dispositivos móveis usados pelo PRE (Celulares da Motorola e da Nokia). *AcessarConteudo* recupera os conteúdos. *Autor* representa o autor do conteúdo. *AgentManager* é responsável pelo gerenciamento dos agentes que buscam e adaptam conteúdos conforme as preferências do usuário. *AgentLog* registra os *logs* da aplicação usando agentes. *AgentUser* é responsável pela validação do usuário. *BehaviorDB* disponibiliza conteúdos solicitados pelo usuário. *AgentSearch* busca o conteúdo solicitado pelo usuário. *UserProfile* contem as preferências do usuário conforme o contexto em que o mesmo se encontra. *ContentAdaptation* faz a adaptação de conteúdo usando um serviço web descoberto pelo *BehaviorWeb*.

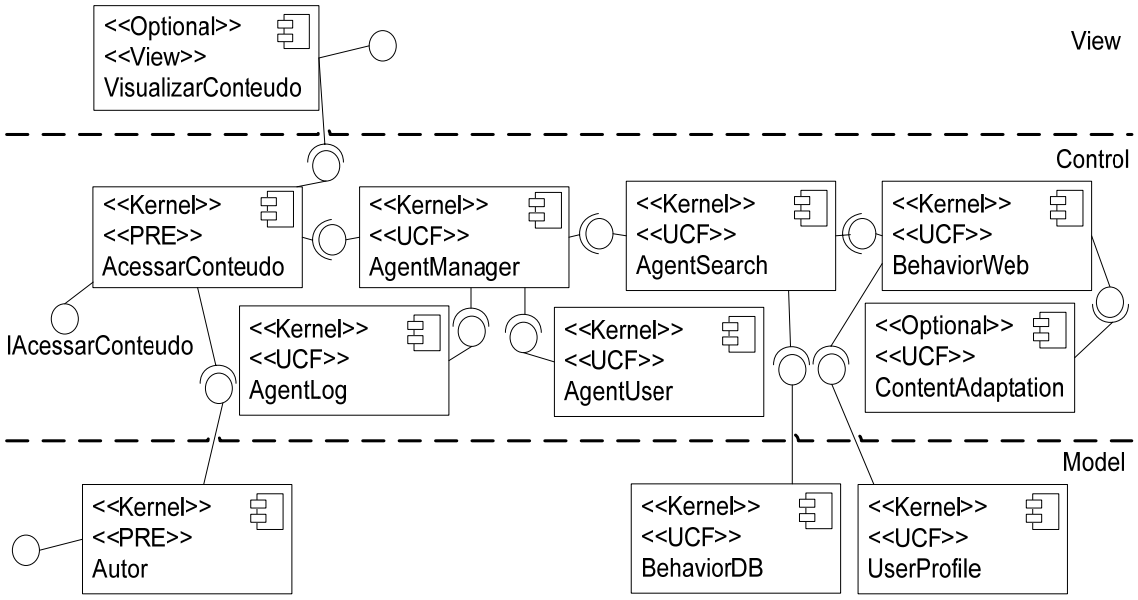

Figura 21. Modelo de Componentes – Feature Acessar Conteúdo PRE

4.1.3. Implementação

Nesta etapa, os componentes projetados são implementados e testados. Com o apoio da MVCASE, o Engenheiro de Domínio faz a geração parcial do código, baseado nas classes que compõem os componentes. Essas classes são testadas até atenderem suas reais funcionalidades.

As transformações dos PSM's em uma linguagem de implementação, utilizam os estereótipos para organizar e orientar a geração de código. Assim, por exemplo, os estereótipos informam quais componentes devem gerar código e quais não devem, devido ao reuso de componentes já disponíveis. Por exemplo, o estereótipo <<UCF>> não gera código por ser um componente reutilizado do framework *Ubiquitous Computing Framework.* No caso de um estereótipo do tipo <<View>> tem-se a informação de que o componente deve ser implementado como uma interface.

No caso dos relacionamentos entre classes o gerador de código da MVCASE, utiliza-se o nome do papel nos relacionamentos como atributo da classe. As Figuras 22, 23 e 24 mostram os diferentes tipos de relacionamentos e os respectivos códigos gerados. Ressaltem-se ainda as importações dos pacotes e classes reutilizadas na geração do código da aplicação.

Na Figura 22 tem-se um relacionamento com cardinalidade 1, que dá origem a um atributo aConteudo do tipo AcessarConteudo.

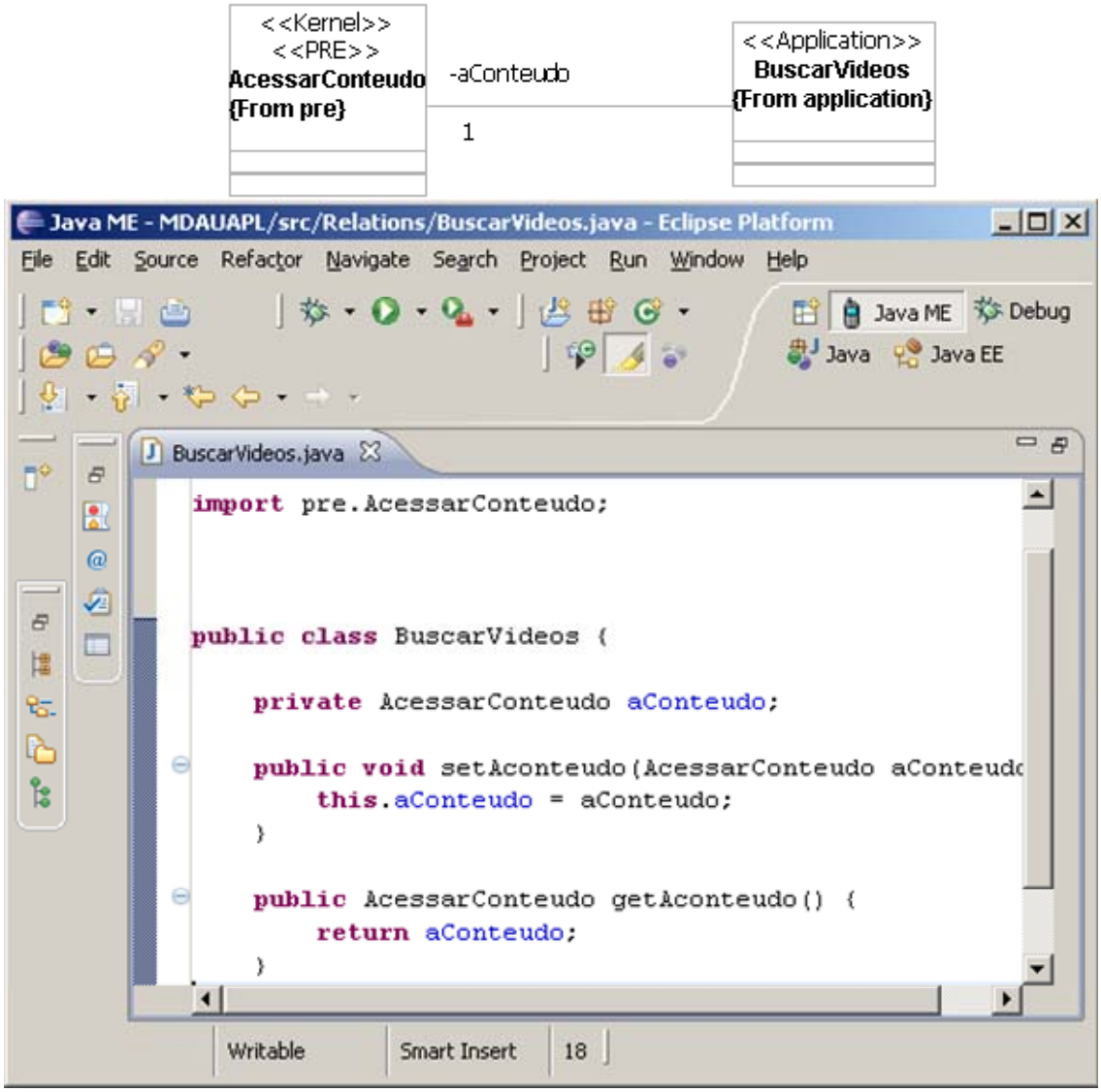

Figura 22. Relacionamento Gerando um Objeto

Na Figura 23 tem-se um relacionamento com cardinalidade *, que indica muitos. Nesse caso tem-se um atributo do tipo coleção. No caso o atributo aConteudo é do tipo *Collection*<AcessarConteudo>.

Relacionamento de dependência, como o de *UserProfile* no método *UserVerify*, apresentado na Figura 24, implica no uso dessa classe. O import não é necessário pois as classes encontram-se no mesmo pacote *ucf.*

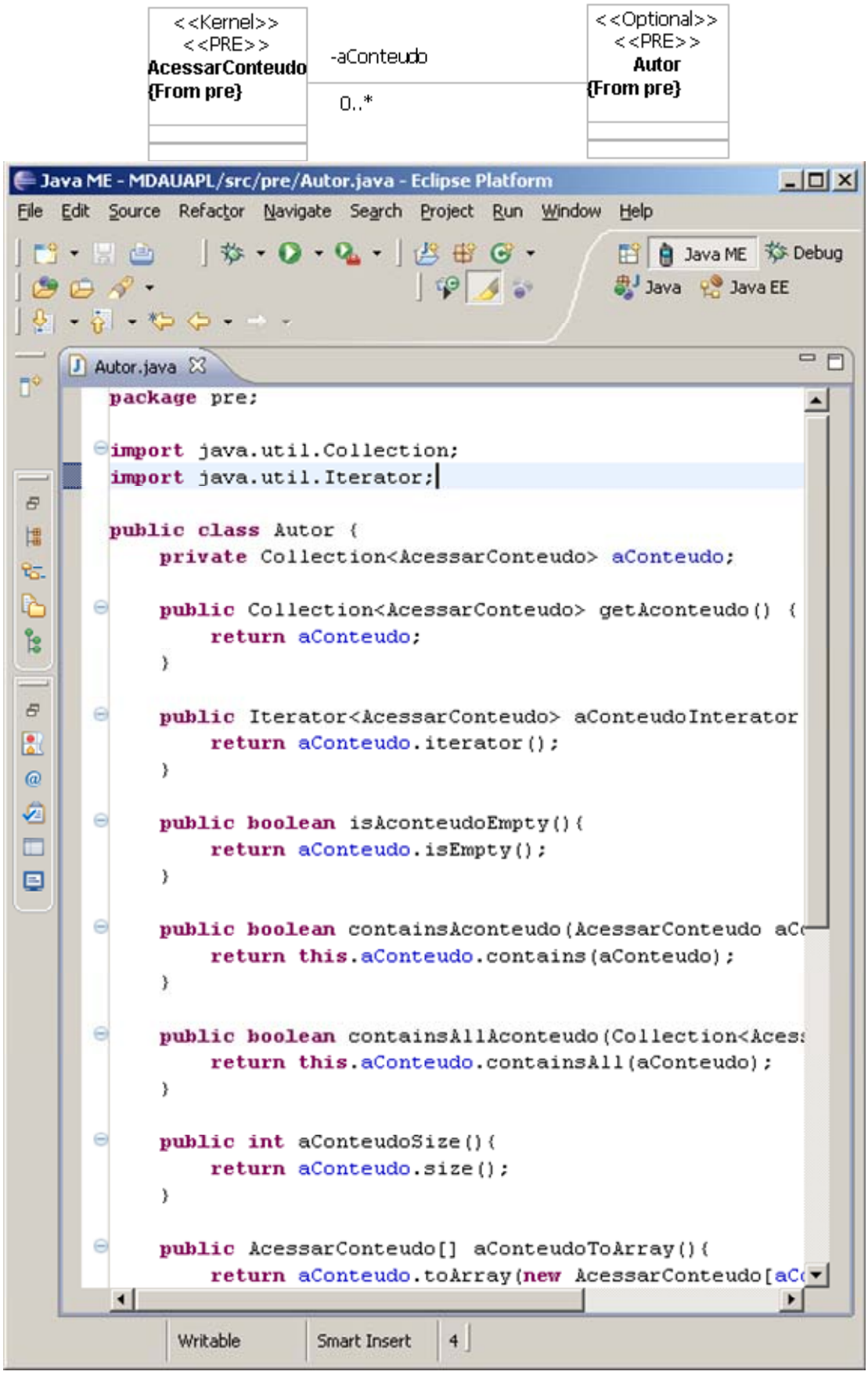

Figura 23. Relacionamento Gerando uma Coleção de Objetos

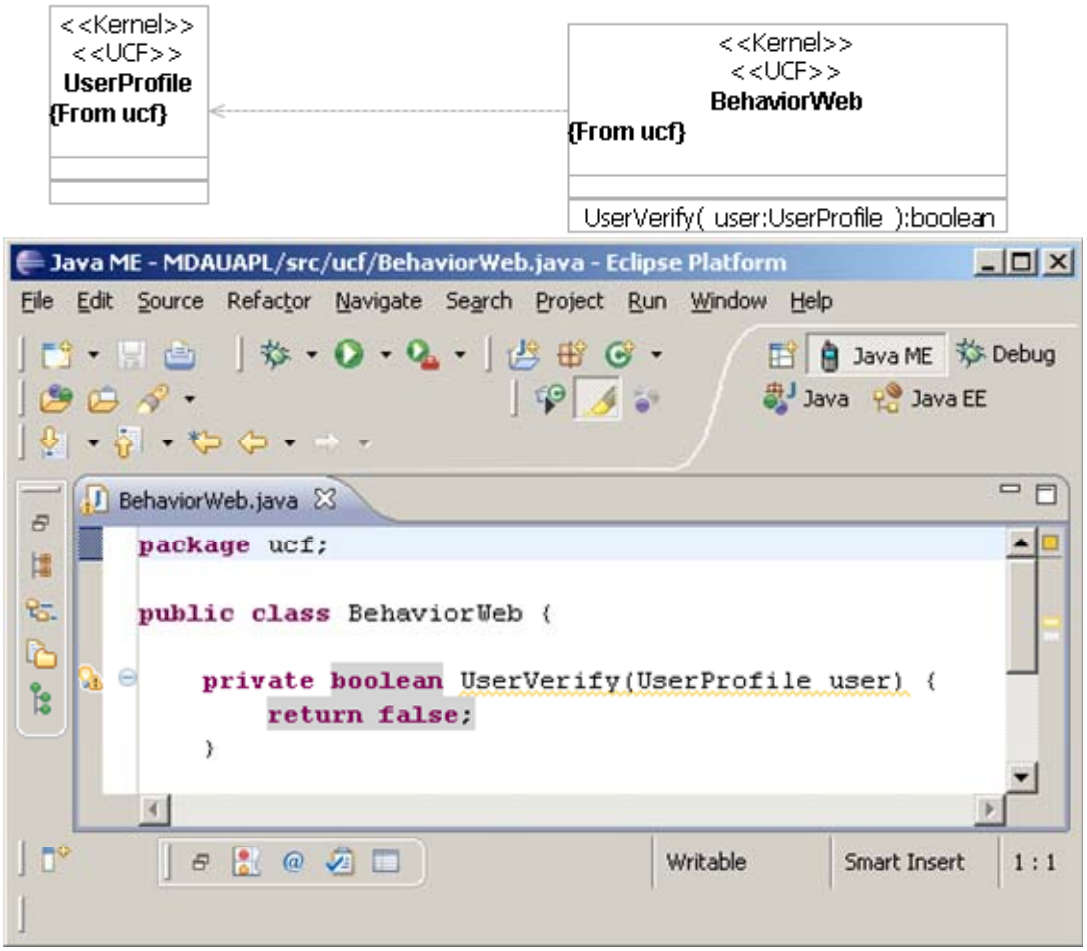

Figura 24. Relacionamento de Dependência

A Figura 25 apresenta um dos códigos java gerados no caso do componente *AcessarConteudo* do domínio de PRE. Uma vez que a MVCASE é um *plug-in* do IDE Eclipse, o código gerado pode ser complementado pelo Engenheiro de Domínio, nesse mesmo IDE, e em seguida executado e testado. Dependendo dos resultados dos testes, pode-se retornar às etapas anteriores para corrigir possíveis erros de implementação, projeto ou mesmo de requisitos.

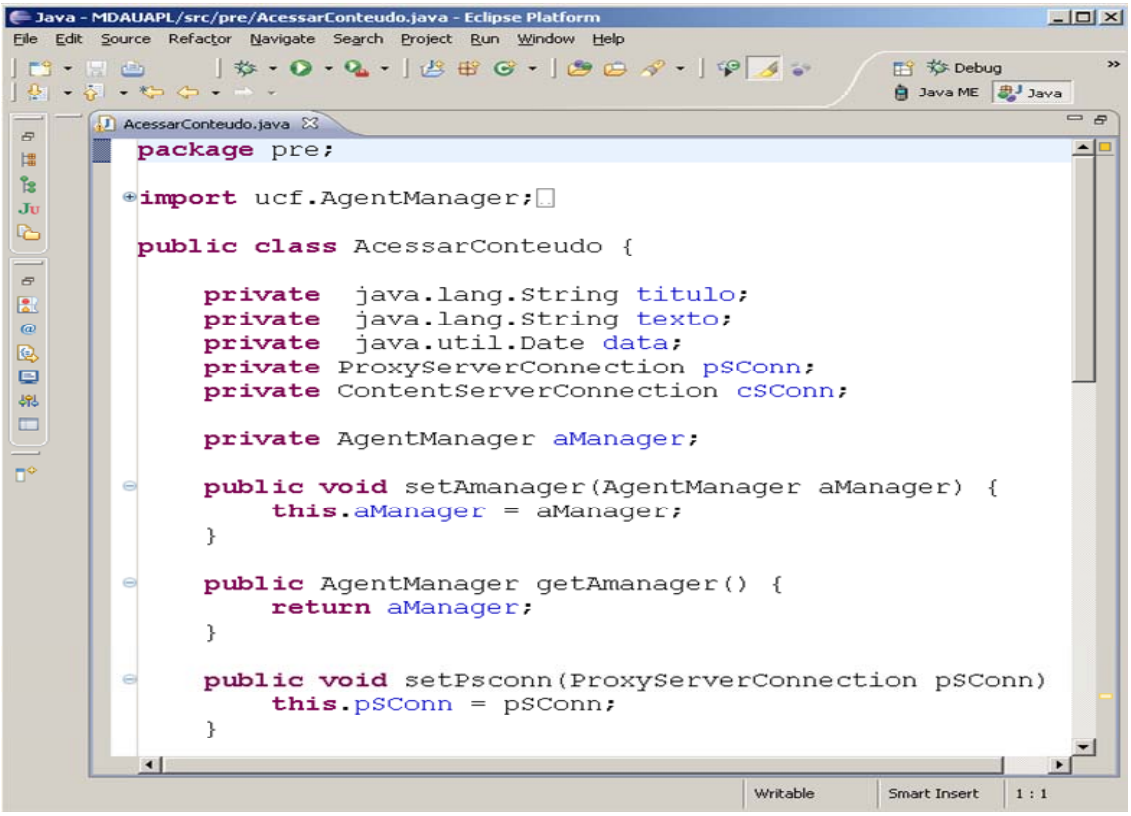

Figura 25. Código Gerado a partir do PSM

No final da ED têm-se os *core assets*, a nível de modelos e de códigos, organizados em pacotes e estruturados em componentes e *frameworks*, disponíveis para o reuso na construção dos produtos pela Engenharia de Aplicação.

4.2. Engenharia de Aplicação

Conforme a Figura 13, a próxima fase da abordagem trata da construção dos produtos da LPS, fazendo reuso dos *core assets* construídos na ED e disponíveis como frameworks, tais como o UCF, e biblioteca de componentes, tais como os do PRE.

Essa fase da abordagem é apresentada no próximo capítulo e é ilustrada com a instanciação de um produto derivado da LPS no domínio de PRE, que faz reuso do *core assets* apresentados na ED.

Capítulo 5

5. Engenharia de Aplicação

Na Engenharia de Aplicação (EA) os produtos derivados da LPS são construídos através do reuso dos *core assets* desenvolvidos pela ED. Como grande parte da complexidade e do esforço é investida na ED, devido ao desenvolvimento dos *core assets,* a construção dos produtos pela EA se torna uma tarefa que demanda menos tempo e esforço [Linden et al. 2007].

As disciplinas encontradas na EA são as mesmas da ED, conforme mostra a Figura 13, porem modificadas para reutilizarem os *core assets* disponíveis na LPS. O Engenheiro de Aplicação especifica o produto, ao longo do processo, usando as técnicas de cada disciplina, desde os Requisitos e Projeto até a Implementação. O reuso dos *assets* da LPS deve considerar as *features* obrigatórias e opcionais. Apenas as *features* específicas do produto é que precisam ser implementadas. Em principio, todas as *features* obrigatórias do core são reutilizadas e as *features* opcionais dependem de cada produto.

Nesse capítulo a Engenharia de Aplicação é apresentada através de um estudo de caso no domínio de PRE, o qual foi apresentado na ED. Nesse estudo de caso foi construído o produto *CellphonePRE*, que têm funcionalidades comuns e opcionais da LPS e funcionalidades específicas. Dentre as funcionalidades comuns têm-se o Acesso a Conteúdos no dispositivo

móvel celular. Como funcionalidades opcionais têm-se, por exemplo, o gerenciamento da prática profissional do domínio PRE. Outras funcionalidades, como visualização de imagens, textos e vídeos, conforme as preferências do usuário e características do dispositivo (perfil do dispositivo), são funcionalidades específicas do produto *CellphonePRE*. Dentre as preferências do usuário têm-se a exibição parcial de textos no celular e no perfil do dispositivo têm-se as dimensões da tela do dispositivo, quantidade total de memória e tipo do processador.

Nesse contexto, segue-se uma apresentação de cada etapa da Engenharia de Aplicação conforme a abordagem UbiComSLP.

5.1. Requisitos

Inicialmente, o Engenheiro de Aplicação modela os requisitos do *CellphonePRE* em diagramas de casos de uso que representam as funcionalidades desejadas pelos *stakeholders*, que interagem com o produto. Da mesma forma que na ED, faz-se o mapeamento dos casos de uso em *features*. Nesta etapa o Engenheiro da Aplicação pode reutilizar os modelos de Requisitos disponíveis pela ED. A Figura 26 mostra, por exemplo, um modelo de casos de uso do produto *CellPhonePRE*. Os estereótipos <<Kernel>> e <<Optional>> representam o reuso de casos de uso da LPS e o estereótipo <<Application>> representa casos de uso específicos do produto. Baseado nos modelos de casos de uso identifica-se as *features* do produto. Por exemplo, o caso de uso “Visualizar Vídeos” foi mapeado na *feature* “Visualizar Vídeos”.

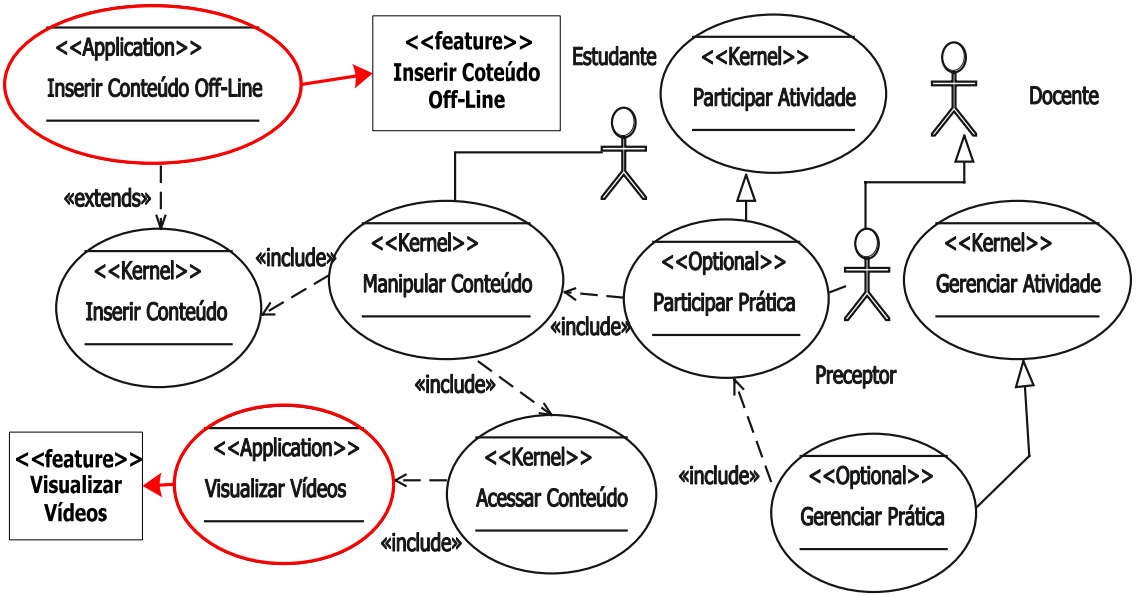

Figura 26. Modelo de Casos de Uso com Casos de Uso do Produto

Um dos artefatos produzidos nesta etapa é a lista com todas as *features* do produto. Por exemplo, a Tabela 3 relaciona algumas *features* do *CellPhonePRE* organizando-as conforme sejam reutilizadas do *core asset* da LPS (UCF e PRE) ou específicas do produto (Application).

Tabela 3. Features do Produto CellPhonePRE

Features		
UCF	Adaptar Conteúdo	
	Descobrir Contexto	
PRE	Participar Atividade	Participar Prática
	Manipular Conteúdo	Inserir Conteúdo
		Acessar Conteúdo
	Gerenciar Atividade	Gerenciar Prática
Application	Inserir Conteúdo Off-Line	
	Visualizar Vídeos	

Conforme o conceito da MDA, nesta etapa os modelos são independentes de plataforma (PIM). Dentre esses modelos, o de classes também deve ser especificado pelo Engenheiro da Aplicação, já considerando o reuso dos modelos PIMs dos *core assets* da LPS. A Figura 27 mostra um dos modelos de classes do produto *CellPhonePRE*, no qual os estereótipos

<<Kernel>> e <<Optional>> indicam classes reutilizadas e <<Application>> indica classes específicas do produto.

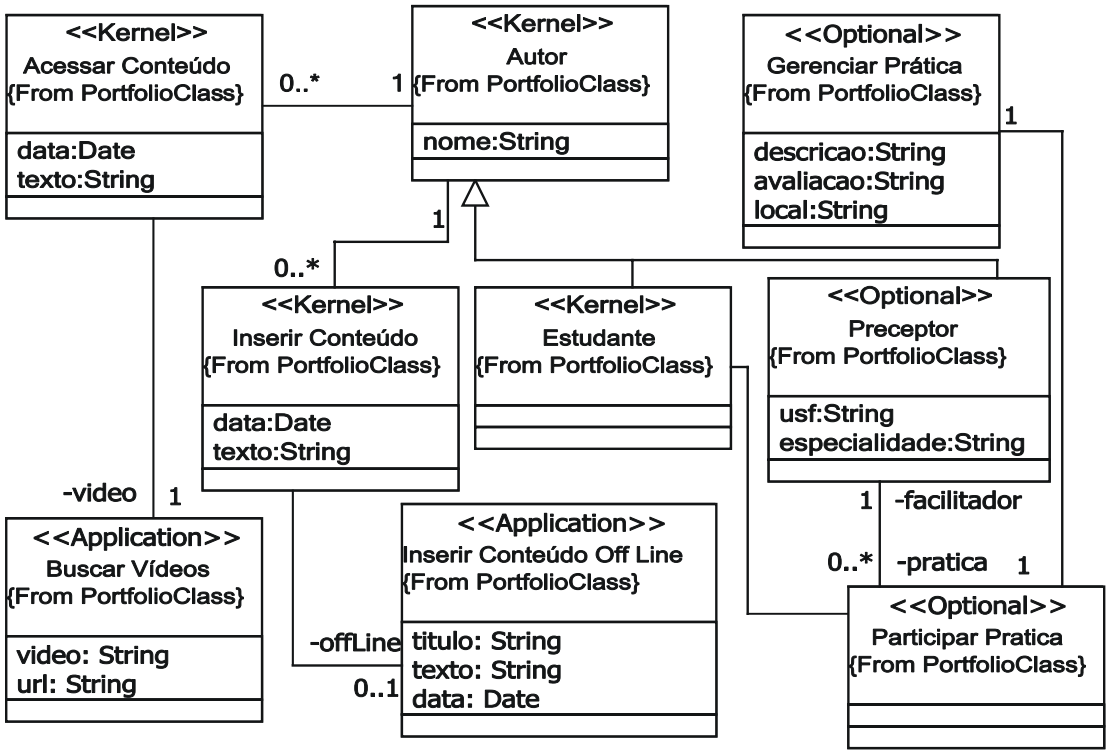

Figura 27. PIM do CellPhonePRE

5.2. Projeto

Nessa disciplina consideram-se as plataformas de hardware e software na qual o produto será construído e implantado. Adotando a mesma arquitetura e plataformas dos *core assets,* conforme os produtos a serem construídos, e adicionando os requisitos não funcionais da plataforma especifica do produto, faz-se o refinamento dos modelos da etapa de Requisitos, considerando o reuso dos *core assets* de projeto da LPS.

Para que se tenha uma idéia, a Figura 28 apresenta um modelo do produto *CellPhonePRE*, onde são representadas classes reutilizadas (<<Kernel>> e <<Optional>>) e classes específicas da aplicação (<<Application>>), segundo o padrão MVC.

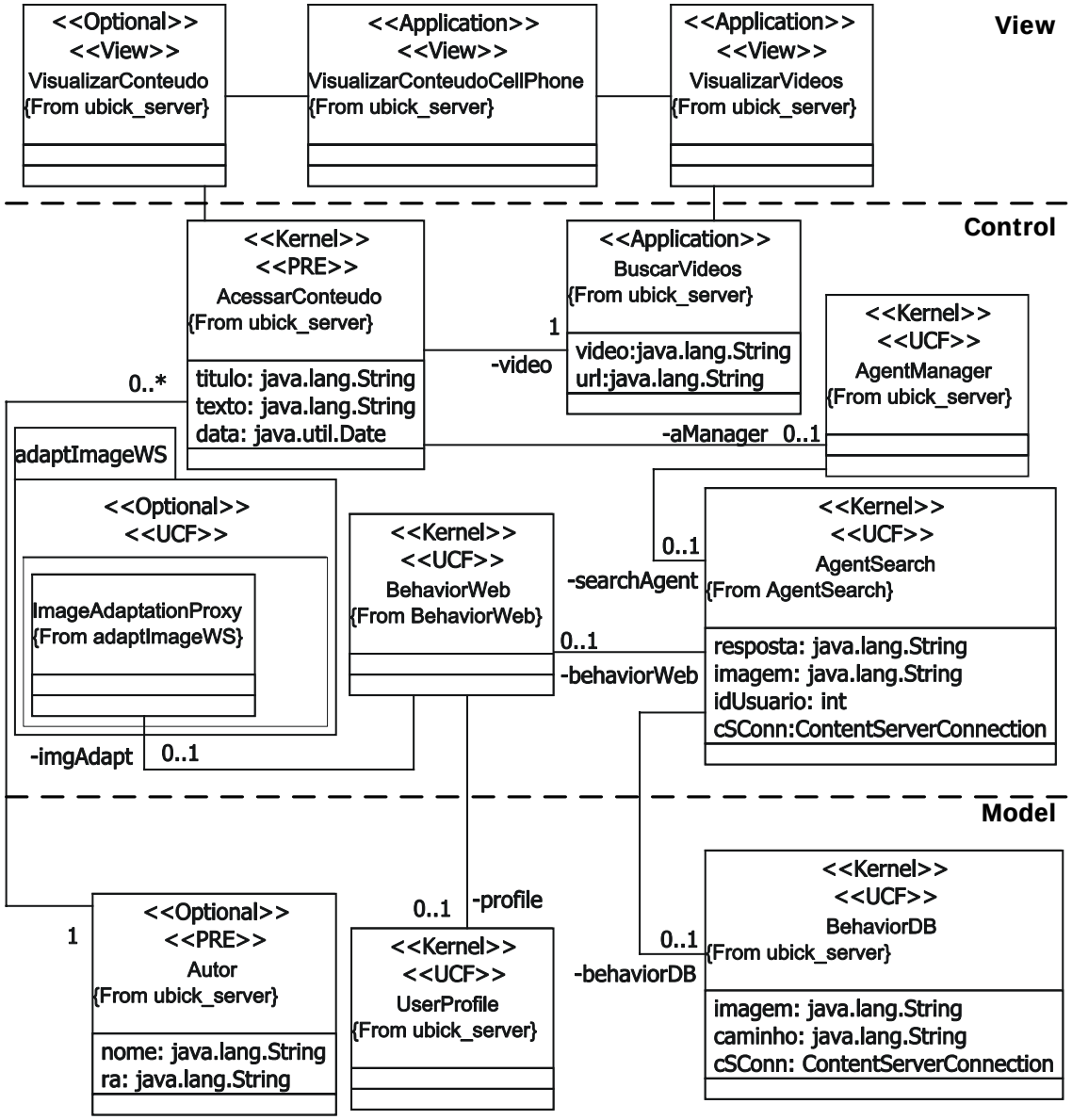

Figura 28. PSM do CellPhonePRE

Prosseguindo, elabora-se o modelo de componentes, considerando o reuso dos componentes disponíveis no *core asset* da LPS, conforme mostra a Figura 29. Nesse modelo, o componente *VisualizarConteudoCellPhone* é responsável pela visualização de conteúdos no *CellPhonePRE*. O componente *VisualizarVideo* é responsável pela visualização de vídeos no *CellPhonePRE*. O componente *BuscarVideo* recupera vídeos disponíveis em um servidor. Ressalte-se o reuso dos componentes do core da LPS, como no caso da

conexão com o componente *AcessarConteudo* (core da LPS) através da interface *IAcessarConteudo*.

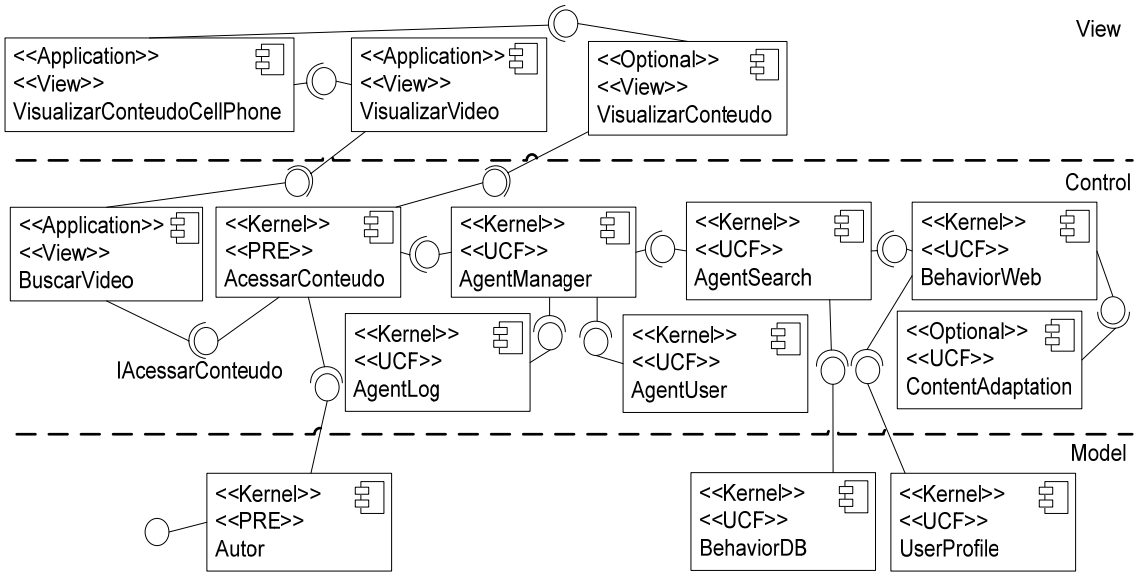

Figura 29. Modelo de Componentes para Acessar Conteúdo no PRE

5.3. Implementação

Finalmente os componentes do produto *CellPhonePRE* são implementados, também considerando o reuso dos *core assets* previamente implementados. Na MVCASE, com base nos PSM's, utilizando transformações dos modelos em códigos através de geradores de códigos, são gerados os códigos parciais do produto.

A geração parcial de código baseia-se no modelo de classes, desenvolvido na etapa de projeto. Esse modelo, estendido com os estereótipos, possibilita gerar o código das classes com seus atributos, protótipos de métodos e relacionamentos. Classes de componentes reutilizados do *core asset*, que já tenham suas implementações providas, tais como as do UCF, não geram código. Nesse caso, o código é provido pela importação da biblioteca, que contenha o framework ou componentes disponíveis para reuso na IDE (e.g., Eclipse).

A Figura 30 mostra um trecho do código gerado, o qual é complementado com os refinamentos realizados pelo Engenheiro de Aplicação no IDE Eclipse, conforme as especificidades do produto. O código dos *core assets* do PRE reutilizados, foi provido pela importação da biblioteca que os implementa, a qual deve estar disponível para o reuso, conforme mostra a Figura 31.

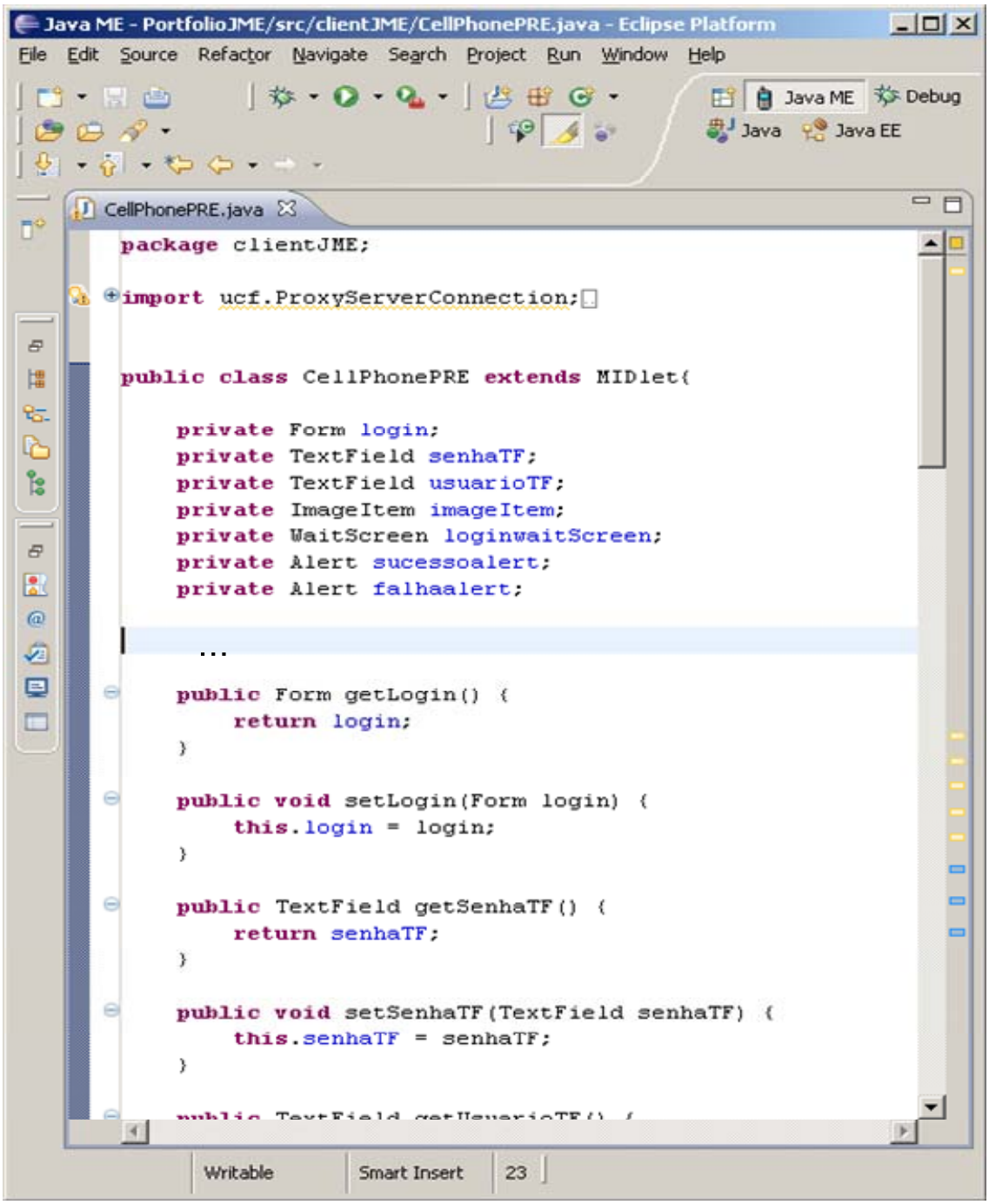

Figura 30. Trecho do Código Gerado pela MVCASE para o CellphonePRE

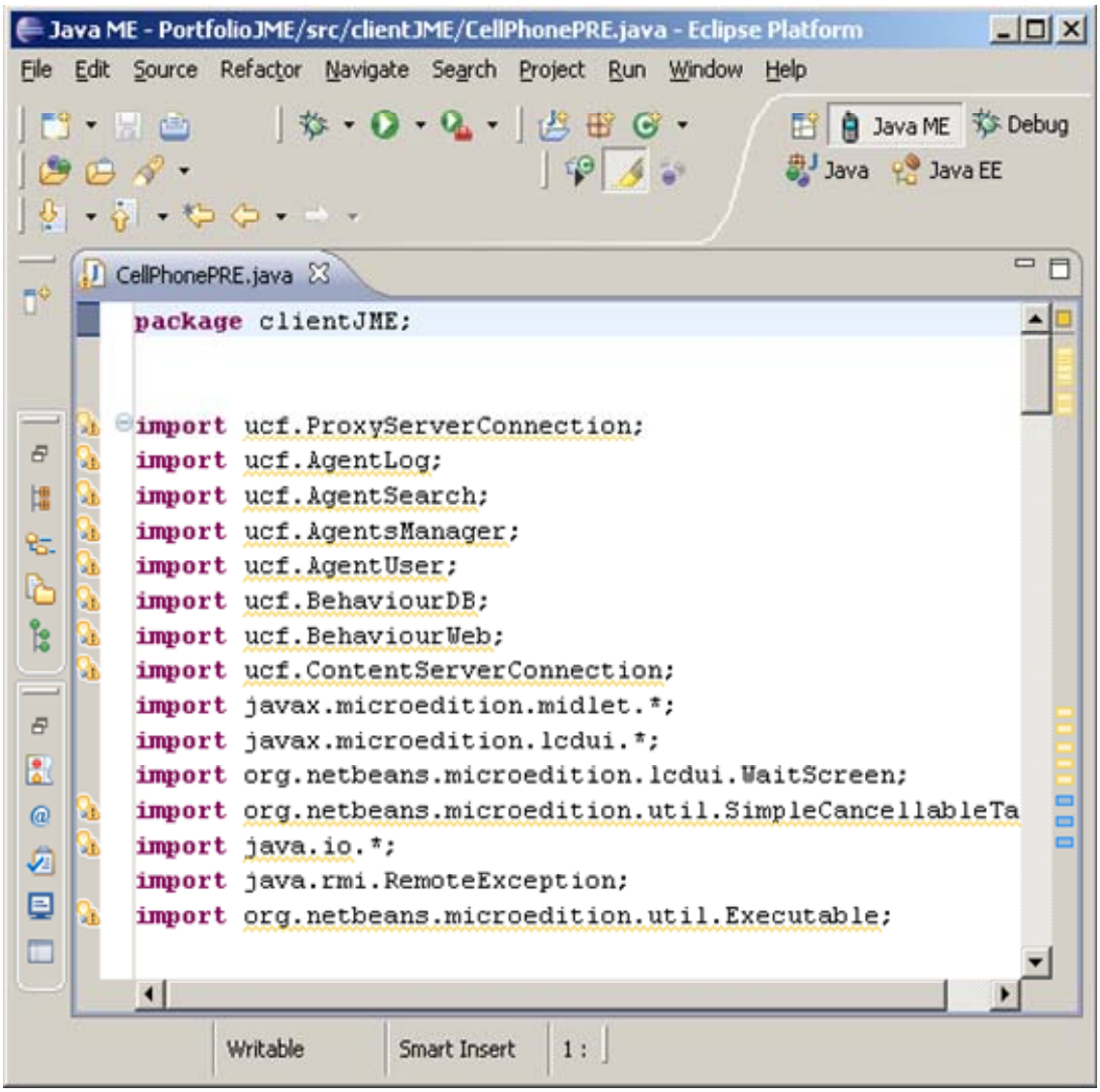

Figura 31. Reuso do Core Asset UCF

Na IDE Eclipse o Engenheiro da aplicação complementa o código dos métodos até obter o código final para sua execução. A Figura 32 mostra o mesmo trecho de código das Figuras 30 e 31, depois de realizadas as devidas complementações.

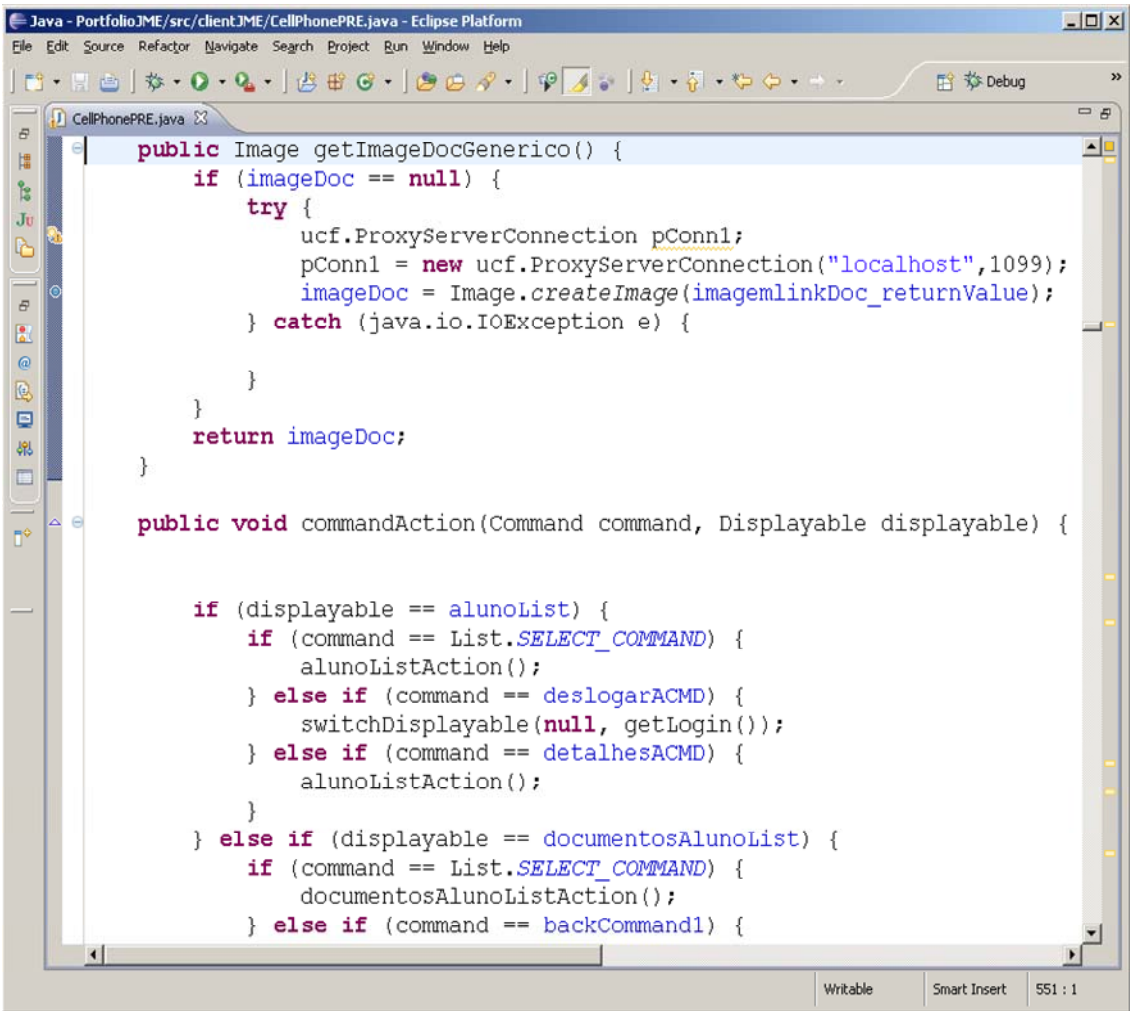

Figura 32. Código final do produto CellPhonePRE

Finalmente realizam-se os testes com o produto, para verificar se o mesmo atende aos requisitos desejados pelos *stakeholders*. Os resultados dos testes fornecem o *feed-back* para a sua validação. Caso ocorram erros, tanto na construção do produto quanto no *core asset* da LPS, estes devem ser corrigidos, gerando novas versões dos produtos ou até mesmo do *core asset*. Dependendo do erro encontrado, pode ser necessário retornar às etapas anteriores da abordagem para corrigi-los. Por exemplo, para um erro de requisito mal especificado, deve-se retornar na etapa de Requisitos para que este seja especificado novamente.

Capítulo 6

6. Conclusões e Trabalhos Futuros

Esta dissertação apresentou a Abordagem para o Desenvolvimento baseado em MDA de Linhas de Produto de Software no Domínio de Aplicações Ubíquas (UbiComSPL). A apresentação dessa abordagem foi organizada no desenvolvimento dos *core assets* da LPS pela Engenharia de Domínio (Capítulo 4) e reuso dos *core assets* da LPS pela Engenharia de Aplicação (Capítulo 5), onde foi apresentado um estudo de caso instanciando o produto *CellphonePRE*.

O projeto procurou combinar diferentes tecnologias atuais da Engenharia de Software, para propor uma abordagem que orienta e apóia o desenvolvedor na construção de LPS e seus produtos no domínio da Computação Ubíqua.

Com foco em LPS e MDA, o projeto enfatiza o reuso de software, procurando reduzir o tempo e o custo do desenvolvimento de novos produtos, além de facilitar a manutenção de software na Computação Ubíqua através da utilização de modelos.

6.1. Contribuições

Ao explorar as idéias, técnicas e apoios computacionais para a abordagem de LPS, baseada na MDA e no domínio de aplicações ubíquas, este trabalho contribui para a Engenharia de Software nas seguintes áreas:

- *Reuso de Software*, considerando que tem foco em LPS, baseia-se na MDA e torna disponíveis os *core assets* para reuso na construção dos produtos;
- *Computação Ubíqua*, considerando que a abordagem está orientada para o desenvolvimento de LPS cujos produtos são implementados para Computação Ubíqua, usando dispositivos móveis;
- *Linha de Produto de Software*. Embora já existam diversos processos, técnicas, arquiteturas e ferramentas, orientadas para o desenvolvimento de LPS em geral, a abordagem proposta diferencia das demais porque se propõe a construir produtos para aplicações ubíquas. As funcionalidades do núcleo da LPS foram definidas para atender grande parte dos requisitos da Computação Ubíqua, relacionados com a adaptação de conteúdo e ciência de contexto;
- *Model-Driven Architecture*, uma vez que integra as ferramentas MVCASE e Eclipse para suportar a modelagem, independente e dependente de plataformas, de uma LPS e seus produtos, incluindo a geração parcial do código e seu refinamento para execução e testes; e
- *Artefatos Reutilizáveis*. Os *assets* que implementam as *features* para adaptações de conteúdos, considerando os contextos, estão disponíveis num framework (UCF), para reuso tanto na modelagem como na implementação das aplicações ubíquas.

Conforme mostram as intersecções da Figura 33, uma parte do *core assets* da LPS, relacionada aos requisitos da adaptação de conteúdo e ciência de contexto (1) e presentes nos produtos ubíquos, e uma parte do *core assets*

no domínio do PRE (2 e 3), podem ser reutilizados em outros domínios como o de Comercio Eletrônico ou Telecomunicações.

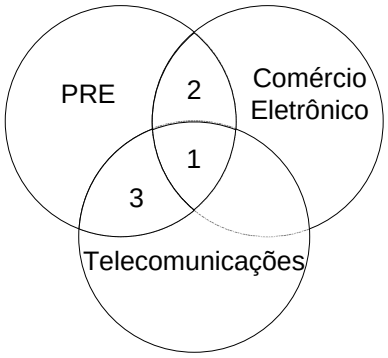

Figura 33. Reuso do Domínio de PRE em Outros Domínios

Concluindo, reforça-se que o reuso, através de LPS baseada na MDA, pode auxiliar o Engenheiro de Software, reduzindo o tempo de desenvolvimento das aplicações, diminuindo seus custos e também facilitando suas manutenções futuras através da reutilização de *core assets* da Engenharia de Domínio.

6.2. Trabalhos Futuros

Dentre os trabalhos futuros, que dão prosseguimento a este projeto de pesquisa, têm-se:

- Testes da abordagem em outros domínios, tais como o de Comércio Eletrônico e Telecomunicações;
- Melhoria do processo de transformação do PIM para o PSM. Atualmente essa transformação é realizada por refinamento dos modelos PIM para PSM, considerando as decisões de projeto e estendendo a UML com o uso de estereótipos. Outra idéia é usar as idéias da Modelagem Especifica de Domínio [Greenfield e Short 2004], para possibilitar uma transformação mais automática do PIM para o PSM e uma geração de

código que inclua outras decisões de projeto relacionadas com as camadas da visão e do modelo;

- Adição de novas técnicas para especificação e modelagem de LPS. Outras técnicas, além dos diagramas de *features*, caso de uso, classe e componentes, tais como as de ontologias, podem ajudar na compreensão, obtenção do conhecimento e elicitação dos requisitos do domínio do problema;
- Suporte para geração de código para outras plataformas além da JME. Por exemplo, pode-se gerar código para a plataforma Android [Google 2009]; e
- Possibilitar a reutilização de *core assets* da ED da forma mais automatizada possível. Por exemplo, logo na disciplina Requisitos, poderia-se selecionar as *features* do produto e a partir dessas obter-se automaticamente os modelos de projetos e respectivas implementações.

Referências

Araújo, R. B. (2003) “Computação Ubíqua: Princípios, Tecnologias e Desafios”. XXI Simpósio Brasileiro de Redes de Computadores.

Braganca, A. e Machado, R. J. (2007) “Model Driven Development of Software Product Lines”. Quality of Information and Communications Technology. QUATIC 2007. 6th International Conference, p. 199-203.

Carton, A., Clarke, S., Senart, A. e Cahill, V. (2007) “Aspect-Oriented Model-Driven Development for Mobile Context-Aware Computing”. Anais da International Conference on Software Engineering Workshops, p. 5-5.

Chastek, G., Donohoe, P., Kang, K. C. e Theil, S. (2001) “Product Line Analysis: A Practical Introduction” (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

EMF (2009) “Eclipse Modeling Framework”. Disponível em http://www.eclipse.org/modeling/emf/ Maio.

Fernandes, J., Machado, R. e Carvalho, J. (2007) “Model-Driven Software Development for Pervasive Information Systems Implementation”. Anais of International Conference on Quality of Information and Communications, p. 218-222.

Frankel, D.S. (2003) “Model Driven Architecture: Applying MDA to Enterprise Computing”. Willey Press. Hoboken, EUA.

Forte, M., Souza, W. L. e Prado, A. F (2008) “Using ontologies and Web services for content adaptation in Ubiquitous Computing”. Journal of Systems and Software (JSS), v. 81, No 3, p. 368-381, Elsevier.

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (1995) “Design Patterns: Elements of Reusable Design”. Reading, Massachusetts: AddisonWesley.

GEF (2009) “Graphical Editing Framework”. Disponível em http://www.eclipse.org/gef/ Maio.

Glassfish (2009) “Glassfish Community”. Disponível em https://glassfish.dev.java.net/ Maio.

GMF (2009) “Graphical Modeling Framework”. Disponível em http://www.eclipse.org/modeling/gmf/ Maio.

Google (2009). “Android Platform”. Disponível em http://code.google.com/intl/pt-BR/android/ Maio.

Greenfield, J., Short, K. (2004) “Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools”. In: Third International Conference, SPLC, Boston, USA.

Harrison, W., Ossher, H., e Tarr, P. (2000) “Software Engineering Tools and Environments: A Roadmap. In The Future of Software Engineering”. ACM, New York, 261-277.

Hibernate (2009) “Hibernate”. Disponível em http://www.hibernate.org/, último acesso em Maio.

Jardim, F. M. (2004) “Framework para implementação de serviços transmissores de vídeos voltados a PCs e dispositivos móveis, perceptivo à mudança contextual de localização”. Dissertação de Mestrado – UFSCAR – Departamento de Computação.

Java (2009) “The Source for Java Developers”. Sun Developer Network. Disponível em http://java.sun.com/ , último acesso em Maio.

Java (2009b) “Java Micro Edition”. Sun Developer Network. Disponível em http://java.sun.com/ , último acesso em Maio.

JET (2009) “EMF and Java Emitter Template (JET) – Tutorial”. Disponível em http://www.vogella.de/articles/EclipseEMF/article.html#emfjet Maio.

Schmid, K. (2008) “Introduction to Software Product Lines”. Presentation on RISS Rise Summer School, Recife, Pernambuco, Brazil, November, 27 to 29.

Kleppe, A. G., Warmer, J. B., Bast, W. e Watson, A. (2003) “MDA Explained: The Model Driven Architecture: Practice and Promise”. Addison-Wesley Professional.

Krasner, G. E. e Pope, S. T. (1988) “A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80”. Journal of Object-Orientated Programming, 1(3), Aug/Sep 1988, p. 26-49.

Linden, F. J. D., Rommes, E. e Schmid, K. (2007) “Software Product Lines in Action”. Editora Springer. Primeira Edição.

Lucrédio, D., Alvaro, A., Almeida, E. S., Prado, A. F. (2003) “MVCASE Tool - Working with Design Patterns”. The Third Lating American Conference on

Pattern Languages of Programming (SugarLoafPLoP 2003), Porto de Galinhas, PE, Brasil.

Lucrédio, D. (2004) “Extensão da Ferramenta MVCASE com Serviços Remotos de Armazenamento e Busca de Artefatos de Software”. Exame de Qualificação. UFSCAR – Departamento de Computação.

Lyytinen, K. e Yoo, Y. (2002) “Issues and Challenges in Ubiquitous Computing”. Case Western Reserve University in Cleveland. Ohio. ACM.

MDA (2006) “The Model-Driven Architecture - Guide Version 1.0.1”, OMG Document: omg/2003-06-01. Disponível em http://www.omg.org/cgi-bin/doc?omg/03-06-01

Mostefaoui, S. K., Maamar, Z., Giagles, G. M. (2008) “Advances in Ubiquitous Computing: Future Paradigms and Directions”. IGI Publishing, Hershey, New York.

Oliveira, R.P., Prado, A.F, Souza, W. L. e Biajiz, M. (2009) “Development based on MDA, of Ubiquitous Applications Domain Product Lines”. Publicado no 8th IEEE/ACIS International Conference on Computer and Information Science, p. 1005-1010.

OMG (2009) “The Object Management Group”. Disponível em http://www.omg.org/, Maio.

Pham, H. N., Mahmoud, Q. H., Ferworn, A. e Sadeghian, A. (2007) “Applying Model-Driven Development to Pervasive System Engineering,” in Proceedings of the 29th International Conference on Software Engineering Workshops.

Pressman, R., S. (2001) “Software Engineering: A Practitioner's Approach”. McGraw-Hill.

Santana, L.H.Z., Prado, A.F., Souza, W. L. e Biajiz, M. (2007) “Usando Ontologias, Serviços Web Semânticos e Agentes Móveis no Desenvolvimento Baseado em Componentes”. Publicado no Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, Campinas, v. 1. p. 163-176.

Santana, L., Martins, D., Forte, M., Souza, W., Prado, A., Biajiz, M. e Knoff, L. (2007a) “Serviço De Tradução De Linguagens De Marcação Para A Internet”. Anais Do Simpósio Brasileiro De Redes De Computadores, p. 541-554.

Santana, L. H. Z., Martins, D.S., Perlin, C. B., Prado, A. F., Souza, W.L. e Biajiz, M. (2007b) “Adaptação de Páginas Web para Dispositivos Móveis”. In: Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), Gramado. v. 1. p. 1-8.

SEI (2009) “Software Engineering Institute – Software Product Lines”. Disponível em http://www.sei.cmu.edu/productlines/index.html Maio.

Spínola, R., Massollar, J. e Travassos, G. (2007) “Checklist to Characterize Ubiquitous Software Projects”. Anais of Software Engineering Brazilian Simposion, p. 39-55.

Sun (2009) “JavaServer Faces Technology”. Sun Developer Network. Disponível em http://java.sun.com/javaee/javaserverfaces/ , último acesso em Maio.

Torres, S. e Lucena, C. (2007) “Modeling multi-agent systems”Comunications of the ACM, vol. 50, no. 5, p. 103 – 108.

UML (2009) “Unified Modeling Language Specification”. Disponível em http://www.uml.org/, Maio.

Weiser, M. (1991) “The Computer for the Twenty-First Century”. Scientific American, vol. 265, no. 3, p. 94-104.

Weiser, M. e Brown, J. S. (2006) “The Coming Age of Calm Technology”. Xerox PARC. Outubro.

Wirfs-Brock, R. (1993) "Stereotyping: A Technique for Characterizing Objects and Their Interactions". Object Magazine, vol. 3, p. 50-3.

Woojin, L., Sungwon, K. e Hyung, L. D. (2007) “Product Line Approach to Role-Based Middleware Development for Ubiquitous Sensor Network”. Computer and Information Technology. 7th IEEE International Conference on 16-19 Oct., p. 1032 – 1037.

Publicações

A partir da pesquisa apresentada nessa dissertação, foi publicado, em colaboração com diferentes pesquisadores, o seguinte artigo no evento internacional:

Oliveira, R.P., Prado, A.F, Souza, W. L. e Biajiz, M. (2009) “Development based on MDA, of Ubiquitous Applications Domain Product Lines”. Publicado no 8th IEEE/ACIS International Conference on Computer and Information Science, p. 1005-1010.