Uma ferramenta para rastreabilidade de core assets em linha de produtos de software

Anderson Fonseca e Silva[1], Vinicius Cardoso Garcia[2]

[1]Centro de Estudos e Sistemas Avançados do Recife – C.E.S.A.R. EDU

[2]Centro de Informática – Universidade Federal de Pernambuco

anderson.fonseka@gmail.com, vcg@cin.ufpe.br

Abstract. *This work presents a CASE tool for Software Product Line support, aiming to keep traceability between features and core assets such as requirements, components and others. Besides, the tool provide resources for domain analysis models with multiple viewpoints and to permit analyze impacts, in an overall SPL developed by software engineer, through the traceability matrix and mapping diagrams.*

Resumo. *Este trabalho apresenta uma ferramenta CASE para auxiliar as atividades de manutenção da rastreabilidade entre as características projetadas por modelos de Linha de Produtos de Software e artefatos modelados pela UML (i.e. casos de uso, componentes, dentre outros). Fornecendo além de recursos de múltiplas visões para os modelos de domínio, projetados pelo engenheiro de software, a análise de impactos no gerenciamento de uma LPS, através de uma matriz de rastreabilidade e de diagramas de mapamento.*

1. Introdução

Para Pohl et al. (2005) e Eisenecker (2000), a Engenharia de Linha de Produtos de Software é tipicamente organizada em termos de dois principais processos: engenharia de domínio e a engenharia de aplicação.

Na engenharia de domínio (ED) é estabelecida uma plataforma reutilizável, definindo as partes comuns e variáveis de um produto de software através do conceito de *features*[1] definido por Kang et al. (1990), que estabelece um método onde o usuário identifica o que normalmente se espera em uma aplicação de um determinado domínio.

A engenharia de aplicação (EA) permite a derivação de produtos a partir das definições estabelecidas pela plataforma criada no processo de engenharia de domínio, explorando as variações da LPS e garantindo o vínculo correto entre as variações e as necessidades especificas da aplicação.

Para a garantia do vínculo entre a engenharia de domínio e a engenharia de aplicação, é importante entender o conceito de rastreabilidade definido pelo IEEE[2] (1990), como o grau entre o qual um relacionamento pode ser estabelecido entre dois ou mais produtos de um processo de desenvolvimento.

1 Será utilizado o termo *feature* ao invés de característica no decorrer deste artigo.

2 IEEE Standard Glossary of Software Engineering Terminology

Com base na pesquisa apresentada por Lisboa (2010), as ferramentas de análise de domínio estudadas, demonstraram uma ausência com relação ao mapeamento entre *features* e os artefatos que constituem a aplicação gerada. Ainda conforme Lisboa (2010), é importante a utilização de ferramentas que forneçam o suporte ao analista de domínio para o gerenciamento e a representação das *features,* e que a ausência de produtos integrados, força o uso de ferramentas independentes para a execução de tais atividades, causando um aumento no risco de problemas, tais como: *delays,* degradação de produtividade e interoperabilidade durante a execução do processo.

Este artigo apresenta a Fit2Mapping, uma ferramenta para auxiliar analistas de domínio e engenheiros de software, no projeto e modelagem de artefatos inerentes a uma LPS e elementos da UML[3] de forma integrada. Permitindo a associação manual entre os elementos dos modelos de *features* (LPS) e *core assets* (UML), a ferramenta torna possível a identificação de impactos na manutenção de uma LPS, através do uso de múltiplas visões, diagramas de mapeamento e uma matriz de rastreabilidade.

Nas seções seguintes desse artigo, serão apresentadas na Seção 2, o Estado da Arte; na seção 3, a ferramenta Fit2Mapping; na seção 4, os Trabalhos relacionados, as Considerações finais e Trabalhos futuros.

2. Estado da Arte

2.1. Linha de Produtos de Software

Segundo Nortrop (2008), uma linha de produtos de software é um conjunto de sistemas, que compartilham características em comum, satisfazendo necessidades específicas de um segmento de mercado em particular ou missão, e que é desenvolvida a partir de um grupo de *core assets*, de forma prescrita. As motivações principais para a adoção de linha de produtos são: alcançar ganhos de produtividade em larga escala, melhorar o *time-to-market,* manter presença de mercado, melhorar a qualidade do produto, lidar com evolução e complexidade, e obter maior agilidade de mercado.

Para Eisenecker (2000), uma Engenharia de LPS é tipicamente organizada em termos de dois principais processos: engenharia de domínio e engenharia de aplicação.

2.1.1. Engenharia de domínio

Para Pohl et al. (2005), a engenharia de domínio é o processo no qual as partes comuns e variáveis de uma linha de produtos são definidas. A plataforma reutilizável definida pela engenharia de domínio, consiste de todos os tipos de artefatos de software (especificação de casos de uso, diagramas de análise e projeto, código-fonte, roteiro de testes, dentre outros) onde a rastreabilidade deve ligá-los fornecendo um reuso sistemático e consistente.

Os principais focos na engenharia de domínio segundo Pohl et al. (2005) são:

- Escopo, especificação e modelagem das *features* comuns e variáveis de uma LPS;
- Definição de uma arquitetura flexível que compreende as *features* comuns e variáveis;
- A produção de um conjunto de *core assets* (*frameworks*, componentes, bibliotecas e aspectos) que são aderentes a arquitetura de uma LPS.

3 http://www.uml.org

2.1.1.1. Feature-Oriented Domain Analysis (FODA)

Kang et al. (1990) definem a *Feature-Oriented Domain Analysis* (FODA), como uma técnica utilizada para a análise de domínio. O conceito de orientação a *features* estabelece um método, onde o usuário identifica, o que normalmente se espera em uma aplicação de um determinado domínio.

Sochos et al. (2004) resumem as *features* como um meio de:

- Modelar grandes domínios;
- Gerenciar variabilidades de produtos em uma LP;
- Encapsular as necessidades descritas nos requisitos;
- Facilitar a comunicação entre os envolvidos no sistema.

A FODA prevê a modelagem de *features* para a identificação de partes comuns e variáveis de um produto, com o objetivo de capturar o entendimento dos usuários finais e clientes a respeito das capacidades gerais de aplicações em um domínio. Esse modelo prevê o agrupamento lógico de funcionalidades de mesmo interesse, podendo gerar relacionamentos obrigatórios, alternativos ou opcionais.

A Figura 1, apresenta um modelo de *features* para o domínio de carros, tratando a seleção de *features* alternativas para o sistema de transmissão de um veículo, podendo este ser automático ou manual. Esse mesmo modelo ainda apresenta como opção a seleção de ar condicionado para o veículo, sendo esta *feature* algo opcional.

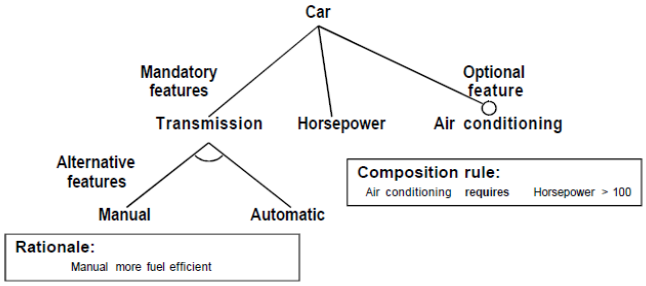

Figura 1. Modelo de features proposto pela FODA [Kang et al. 1990].

2.1.2. Engenharia de Aplicação

A engenharia de aplicação permite a derivação de produtos a partir das definições estabelecidas pela plataforma criada na processo de engenharia de domínio, explorando as variações da LPS e garantindo o vínculo correto entre as variações e as necessidades especificas da aplicação.

Os principais focos na engenharia de aplicação segundo Pohl et al. (2005) são:

- Obter o mais alto grau possível de reutilização de componentes e explorar as partes comuns e variáveis da LPS durante a criação da linha de aplicações;
- Documentar os artefatos da aplicação (casos de uso e código-fonte) vinculando-os aos artefatos de domínio (*features*, requisitos de LP, dentre outros);
- Vincular as variações de acordo com as necessidades da aplicação, dos requisitos até a arquitetura, para componentes e casos de testes.

2.2. Variabilidade

A variabilidade significa inconstância, e uma possível forma de gerenciamento dessa variabilidade é definido por Pohl et al. (2005), como uma das propriedades chave para distinguir uma engenharia de linha de produtos de software, do desenvolvimento de sistemas únicos e reuso de software. As subseções seguintes, apresentam os modelos de variabilidades existentes na literatura e que compõem uma parte essencial na composição de uma LPS.

2.2.1. Modelo de variabilidade utilizando UML

O modelo de variabilidade utilizando UML é proposto por Claub (2001), onde são utilizados mecanismos de extensões e objetiva tornar-se parte da UML através de *profiles*, propondo a introdução de três extensões: uma para o modelo de *features* e duas menores para a modelagem de variabilidade. A extensão voltada para a modelagem de *features* é uma tentativa de se criar diagrama de *features* dentro da UML, fornecendo uma maneira sistemática na organização de variabilidades de um grupo de sistemas de software.

2.2.2. Modelo de variabilidade conceitual

O modelo de variabilidade conceitual proposto por Berg (2005), objetiva a rastreabilidade entre dois artefatos desenvolvidos durante as várias fases da engenharia de software. São utilizados nesse modelo os conceitos de espaço-problema e espaço-solução introduzidos por Czarnecki e Eisenecker (2000), onde o espaço-problema refere-se às especificações do sistema estabelecidas durante a análise de domínio e a fase de engenharia de requisitos, enquanto que o espaço-solução refere-se aos sistemas concretos criados durante as fases de arquitetura, projeto e implementação.

2.2.3. Modelo de variabilidade ortogonal (MVO)

Pohl et al. (2005) conceituam o modelo de variabilidade ortogonal como um modelo que define a variabilidade de um produto de software, relacionando os modelos definidos em outros modelos de desenvolvimento de software, como por exemplo, modelos de *features*, modelos de casos de uso, modelos de classes, modelos de componentes e modelos de testes.

A justificativa para a criação desse modelo segundo Bühne et al. (2004), é que o modelo de features é insuficiente para representar variabilidade nos requisitos de uma LP. Os ganhos com essa abordagem é a redução do tamanho do modelo e da complexidade devido a somente um aspecto de variação ser documentado no modelo ortogonal.

Para a construção da ferramenta apresentada neste artigo, foram seguidas as definições propostas por Pohl et al. (2005), em função da utilização do modelo de variabilidade ortogonal (MVO) relacionado à modelagem das *features*.

2.3. Rastreabilidade

Gotel e Finkelstein (1994) definem rastreabilidade de requisitos como a habilidade de descrever e seguir o ciclo de vida de um requisito em ambas as direções, como por exemplo, sua origem através de seu desenvolvimento e sua especificação até a subsequente implantação e uso, através de sucessivos refinamentos e iterações em qualquer dessas fases.

Os *stakeholders* do projeto possuem diferentes objetivos de rastreabilidade e, se-

gundo Stehle (1990), a perspectiva do gerente de projetos é que a rastreabilidade dê suporte a forma como cada requisito é satisfeito pelos componentes do sistema. Do ponto de vista do gerente de requisitos, a rastreabilidade facilita a ligação entre os requisitos e suas origens e razões, capturando as informações necessárias para o entendimento e evolução dos requisitos, verificando a adequação destes. Edwards e Howell (1991) indicam que durante o projeto, a rastreabilidade permite que projetistas consigam manter o mapeamento entre o que acontece quando uma solicitação de mudança é implementada antes que o sistema seja reprojetado.

Dessa forma, Aizenbud-Reshef et al. (2006) afirmam que com uma rastreabilidade completa é possível obter uma estimativa de custos e uma previsão de mudanças mais precisa, sem depender do conhecimento do programador em todas as áreas em que serão afetadas por essas modificações. As ferramentas automatizadas de suporte para rastreabilidade, começaram como processadores de texto, planilhas e sistema de banco de dados e que se tornaram mais fáceis com o advento de tecnologias de hipertexto. Porém a maior desvantagem desse método permanecia: as informações de rastreabilidade criadas e mantidas de forma manual, com o objetivo de gerenciar e validar as mudanças, se tornavam desatualizadas à medida que o software evoluía.

Com o passar do tempo surgiram ferramentas de propósitos específicos como IBM RequisitePro [IBM 2011] e o Telelogic DOORS [IBM 2005], que introduziram soluções de rastreabilidade mais avançadas, com suporte ao gerenciamento de informações, validando e monitorando as mudanças dos elementos mapeados e indicando ligações suspeitas. Essas ferramentas também possuem integração com outras ferramentas para facilitar a rastreabilidade dos requisitos a outros produtos do ciclo de vida de um software. Porém, mesmo com esses avanços, o trabalho de rastreabilidade ainda permanece manual devido às informações sobre os requisitos permanecerem expressos em forma de texto informal, necessitando do entendimento humano para validar as ligações. Com isso, a maioria das ferramentas não fornecem ricos esquemas de rastreabilidade, permitindo somente rastrear artefatos de forma simples.

3. Fit2Mapping

A partir desta seção, este artigo apresenta a especificação da Fit2Mapping, que foi desenvolvida para suportar a modelagem das engenharias de domínio e aplicação inerentes a uma LPS com foco na rastreabilidade entre os elementos projetados. As seções seguintes apresentam o Metamodelo e o Padrão Arquitetural adotado pela Fit2Mapping.

A ferramenta é de uso livre (gratuito), disponibilizada através de um site[4], utilizando o licenciamento MIT[5], onde é necessário a inclusão do aviso de *copyright* e de permissão em todas as cópias ou parte substanciais do software.

Atualmente a ferramenta se encontra em versão alfa, porém, apresentando funcionalidades que subsidiam e validam a proposta inicial apresentada por este artigo. Na Tabela 1, são apresentadas as principais funcionalidades, e como as mesmas se encaixam no contexto da engenharia de LPS, UML e contribuições propostas pela ferramenta (F2M):

4 http://code.google.com/p/fit2mapping/
5 http://www.opensource.org/licenses/mit-license.php

Tabela 1. Principais funcionalidades

Funcionalidades	Contexto		
	LPS	UML	F2M
Modelagem de *features* com segmentação e múltiplas visões do modelo.	X		X
Modelagem de casos de uso.		X	X
Diagrama de especificação de requisitos (caso de uso, cenários e passos);			X
Reuso de cenários e passos entre casos de uso.			X
Mapeamento entre *features* e *core assets.*			X
Matriz de rastreabilidade (mapa das associações entre features, casos de uso e componentes).			X
Árvore de configuração do produto. (Somente leitura)	X		X

3.1. Metamodelo

A Figura 2, apresenta o metamodelo do Fit2Mapping para a realização da rastreabilidade entre as *features* e *core assets*.

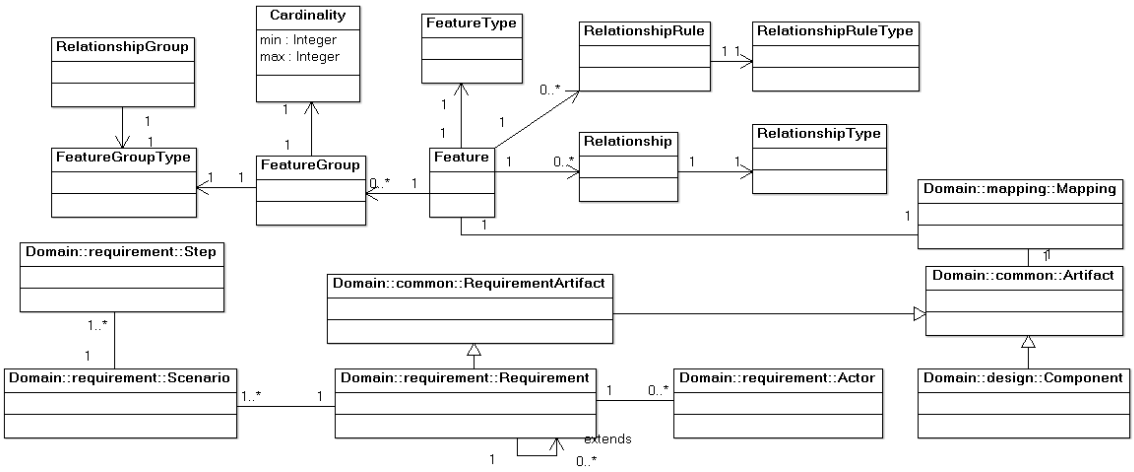

Figura 2. Metamodelo da Fit2Mapping

Na Figura 2, um requisito de domínio (*Requirement)* é composto por cenários (*Scenario*) e passos (*Step*), podendo possuir atores (*Actor*), e, incluir (*includes*) ou estender (*extends*) outros requisitos. O metamodelo possui um classe central (*Mapping*) para o mapeamento entre as *features* e os artefatos (*Artifact*), que além dos requisitos de domínio, permite a relação com componentes (*Component*).

3.2. Padrão Arquitetural

A ferramenta possui uma abordagem tradicional em três camadas (*Layers*), desenvolvida na plataforma Java[6], composta por uma interface gráfica do usuário (GUI), utilizando Swing e JavaD para a criação dos elementos visuais, uma camada de negócios e uma camada de persistência em XML, tendo a sua execução em modo *desktop*. Além da utilização dos componentes fornecidos pela plataforma Java, também foram utilizados: **MigLayout**[7] – Gerenciamento de layouts em Swing; **IText**[8] – Geração de arquivos em

6 http://www.oracle.com/technetwork/java/javase/downloads/index.html
7 http://www.miglayout.com
8 http://itextpdf.com

formato PDF; e **FaMa-FW**[9] **–** para a análise automatizada de modelos de *features.*

A motivação para a utilização dessas tecnologias foi a disponibilização de forma gratuita, e por atenderem às necessidades de implementação da ferramenta.

A Fit2Mapping utiliza os conceitos para a modelagem de *features* baseada na Feature-Oriented Domain Analysis (FODA) [7] e na notação estendida baseada em cardinalidades proposta por Czarnecki (2000), além da utilização do modelo de variabilidade ortogonal proposta por Pohl et al. (2005), permitindo o reuso de uma *feature* através do relacionamento com mais de uma *feature-parent*.

3.3. Múltiplas visões do modelo de *features*

A Figura 3, apresenta um modelo[10], onde o diagrama-base de *features* (*Web Portal Diagram*), é segmentado em outros dois diagramas (*Web Server* e *Additional Services Diagram*), na configuração do produto, as informações aparecem todas unidas a partir dos diagramas elaborados em uma única árvore.

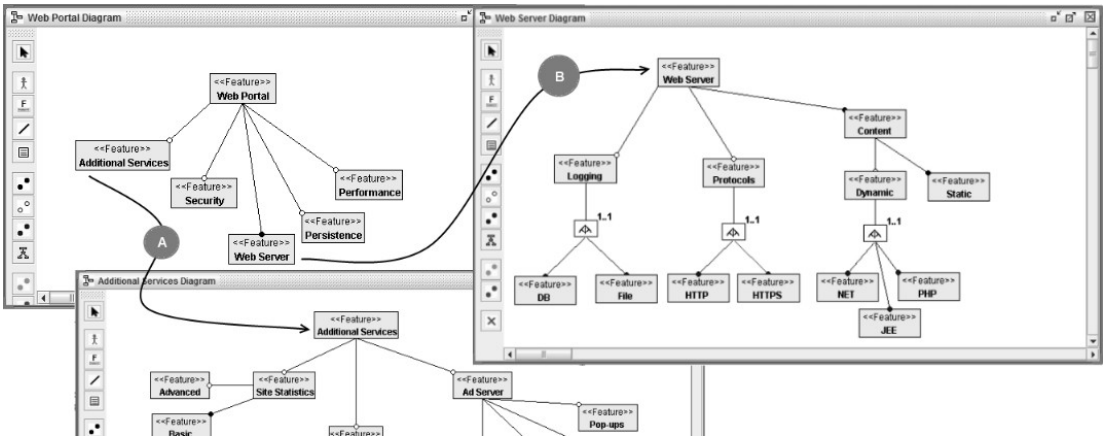

Figura 3. Segmentação no modelo de *features* – as setas (A e B) passando através dos diagramas, apresentam uma forma de extensão dos modelos de *features,* facilitando o detalhamento e a visualização.

3.4. Diagrama de Mapamento e Matriz de Rastreabilidade

Os diagramas (Fig 4.1 e 4.2), representam as principais funcionalidades propostas por este artigo, onde o primeiro, representa o mapeamento entre casos de uso e *features*, e o segundo, uma matriz de rastreabilidade entre os elementos no modelo.

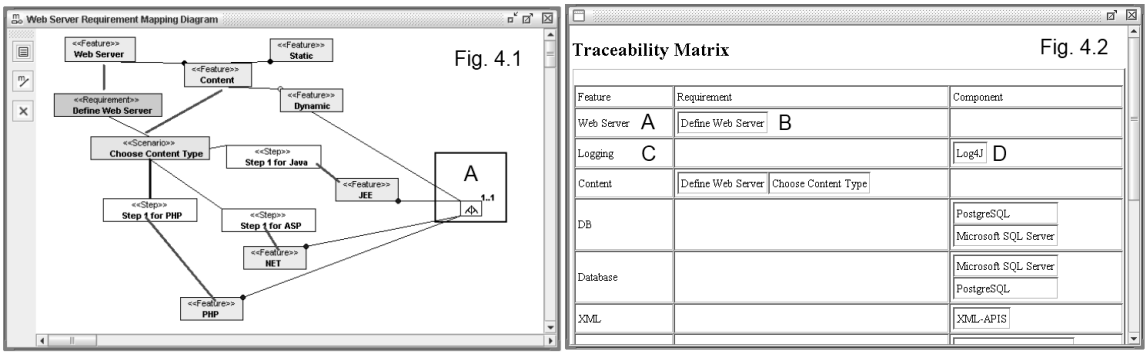

Figura 4. Diagrama para mapeamento de *features* e artefatos (Fig 4.1), o símbolo abaixo da letra A, representa uma agrupamento do tipo XOR-OR Czarnecki (2000). Matriz de rastreabilidade (Fig 4.2).

9 http://www.isa.us.es/fama/

10 Projeto disponível em: http://code.google.com/p/fit2mapping/downloads/list

4. Trabalhos Relacionados

Nessa seção serão apresentadas quatro ferramentas, selecionadas de acordo com os seguintes critérios:

- O contexto do processo de engenharia de domínio e a modelagem de *features*;
- A proposta de um metamodelo que prevê a rastreabilidade entre os modelos resultantes da análise de domínio e artefatos como requisitos, código-fonte, dentre outros;
- A utilização da notação baseada em cardinalidades;.

O estudo apresentado por Lisboa (2010), no item *Relacionamento entre features e requisitos*, identificou quatro ferramentas, sendo elas: *Doors Extension* [Buhne et al. 2005], *DREAM* [Park et al. 2004], *Odyssey* [Braga et al. 1999], *PLUSS Toolkit* [Eriksson et al. 2006] e *RequiLine* [Rwth 2005]. Porém, para a composição dessa seção, foram selecionadas apenas duas ferramentas: *PLUSS Toolkit* e *RequiLine*, por disponibilizarem a documentação e/ou o produto para a avaliação, acrescida pela *pure::variants* [GmbH 2009] considerada líder de mercado nesse segmento e a *FeaturePlugin* [Fmp 2011] desenvolvida pelo Czarnercki.

4.1. PLUSS Toolkit [4]

É um conjunto de extensões para o Telelogic DOORS[11] e o IBM-Rational Rose[12] desenvolvidos em 2005 pela Umes University e Land SystemsHSgglunds, ambas da Suécia. A abordagem adotada pela PLUSS Toolkit foca no mapeamento entre *features* e casos de uso, e neste contexto é utilizada uma notação de fluxo de eventos para a descrição de cenários e realizações de casos de uso. O objetivo dessa abordagem foi a utilização de uma linguagem natural, permitindo o interesse de uma maior diversidade de *stakeholders* nos modelos, e que não descarta a utilização de diagramas UML como suplemento para um melhor entendimento quando necessário.

4.2. RequiLine [13]

A RequiLine, desenvolvida pelo Research Group Software Construction (RWTH), Alemanha, em 2005, tem como principais características: suporte para a modelagem de *features* e requisitos, gerenciamento de linhas de produto, configuração do produto e checagem de consistência, interface de consultas, gerenciamento de usuários, suporte a visões e importação e exportação de dados, porém, a ferramenta não disponibiliza o seu metamodelo para análise. O objetivo principal da ferramenta é suprir as deficiências que outras ferramentas de requisitos possuem referentes ao gerenciamento de variações e dependências.

4.3. pure::variants [11]

A ferramenta desenvolvida pela GmbH, é uma versão comercial para a modelagem de *features* e configuração, que se baseia em dois conceitos: modelos de *features* e *family models.* As partes comuns e variáveis são elaboradas através dos modelos de *features*, enquanto que a modelagem dos ativos são elaborados pelo *family models*, onde são descritos o software em termos de elementos arquiteturais. A pure::variants utiliza uma árvore de visualização mas sem cardinalidades na composição das *features*, permite a modelagem de restrições globais entre *features* oferecendo de forma interativa, configura-

11 http://www-01.ibm.com/software/rational
12 http://www-01.ibm.com/software/ awdtools/developer/ rose /#

ções baseadas na linguagem Prolog. A ferramenta permite que somente um grupo *alternative* ou *or*, sejam vinculados a um mesmo elemento-pai, as cardinalidades são descritas através de expressões, a partir da *Family Model Element Properties*, na propriedade *range.*

4.4. FeaturePlugin [12]

A FeaturePlugin é uma ferramenta para a modelagem de features utilizando a notação baseada em cardinalidades, especializações do diagrama de features e configuração baseada no diagrama de *features*. O metamodelo do FeaturePlugin pode ser estendido para associar atributos adicionais às *features*, como prioridades, *binding times* ou estado da implementação.

4.5. Discussão

Nas seções 4.1 a 4.4, foram apresentadas ferramentas que apoiam a engenharia de domínio e a engenharia de aplicação. A Tabela 2, apresenta um resumo das ferramentas estudadas, onde é possível concluir a ausência dos cinco itens (agrupamento de *xor-or features* baseado em cardinalidades (i), segmentação do modelo de *features* (ii), rastreabilidade de *core assets* (iii), modelagem de diagramas UML (iv) e configuração do produto (v)) implementados, não oferecendo dessa forma, um ambiente integrado para as atividades de projeto e suporte à rastreabilidade de artefatos entre modelos de LPS e UML, comparadas com a ferramenta Fit2Mapping e sua aderência às características apresentadas.

Tabela 2. Resumo comparativo das ferramentas estudadas

Ferramentas	Distribuição	(i)	(ii)	(iii)	(iv)	(v)
PLUSS Toolkit	Extensões IBM Rational Rose, Telelogic DOORS e MS-Word	-	-	Requisitos, utilizando os ambientes estendidos	Sim (IBM Rational Rose)	-
RequiLine	Standalone .NET	-	-	Requisitos, de forma integrada	-	Sim
pure::variants	Eclipse plugin	Somente um *xor* ou *or-group* por *parent-feature*	-	Caliber RM e o Telelogic DOORS	-	Sim
FeaturePlugin	Eclipse plugin	Sim	-	-	-	Sim
Fit2Mapping	Java Standalone	Sim	Sim	Sim	Sim	Sim

5. Conclusões e Trabalhos futuros

Este artigo apresentou uma ferramenta para auxiliar o engenheiro de software nas atividades de projeto de uma LPS, bem como recursos para a modelagem de diagramas baseados na UML, permitindo a associação entre *features* e *core assets*, o que possibilita a verificação de quais os artefatos da engenharia de software compõem o produto gerado pela engenharia de aplicação em uma LPS.

As ferramentas com características em rastreabilidade que subsidiaram a construção da Fit2Mapping, permitiram uma melhor análise para a integração de modelos de LPS com modelos UML. O estudo desta integração forneceu a base para a criação do metamodelo da Fit2Mapping visando o mapeamento entre *features* e *core assets.* Este comparativo também subsidiou a implementação dos modelos UML

(requisitos e componentes) inicialmente disponíveis na ferramenta, bem como a criação do modelo de especificação de casos de uso (casos de uso, cenários e passos).

Em versões futuras, a ferramenta disponibilizará diagramas para a elaboração de modelos UML de *analise & design* (diagrama de classes e sequencia), possibilitando a vinculação destes com os modelos de *features* projetados. Também será implementada e geração de relatórios a partir da engenharia de aplicação, permitindo que sejam identificados os artefatos relacionados com o produto gerado a partir da seleção através da árvore de configuração do produto.

6. Agradecimentos

Este trabalho foi parcialmente financiado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software(INES[13]) e financiado pelo CNPq e FACEPE, subsídios nº 573964/2008-4, APQ-1037-1.03/08 e APQ-1044-1.03/10 e Agência Brasileira (CNPq processos nº 475743/2007-5 e 140060/2008-1).

Referências

[1] (IEEE, 1990) Institute of Electrical and Electronic Engineers. IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). New York, NY: Institute of Electrical and Electronics Engineers, 1990

[2] CZARNECKI, K. Eisenecker, U: Generative Programming: Methods, Tools and Applications. Addison Wesley Longman (2000).

[3] CZARNECKI, Krysztof. Antkeiwicz, Michael. (2005) Mapping Features to Models: A Template Approach Based on Superimposed Variants. Generative Programming and Component Enginering. 4th International Conference.

[4] ERIKSSON, M., Morast, H., Borstler, J., Borg, K. (2006) An Empirical Evaluation of the PLUSS Toolkit, Technical Report, Depto of Computing Science, Umea University, UMINF-06.31.

[5] J. PARK, M. Moon, K. Yeom. (2004) Dream: domain requirement asset manager in product lines, in: International Symposium on Future Software Technology (ISFST), Xian, China.

[6] KANG, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Featureoriented domain analysis (foda) feasibility study. Technical Report Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University.

[7] LISBOA, L., Garcia, V., Lucrédio, D., Almeida, E., Meira, S., Fortes, R. (2010) A systematic review of domain analysis tools. Elsevier.

[8] POHL, Bockle and Linden. (2005) Software Product Line Engineering - Foundations, Principles and Techniques, pages 9-10, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[9] R.M.M. Braga, C. Werner, M. Mattoso. (1999) Odyssey: a reuse environment based on domain models, in: IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET), pp.49–57, Richardson, Texas.

[10] S. BUHNE, K. Lauenroth, P. Klaus. (2005) Modelling requirements variability across product lines, in: IEEE International Conference on Requirements Engineering, pp. 41–52.

[11] GMBH. Pure:variants. User Manual Version 3.0. (2009). pure-systems GmbH. http://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.p df, Aces sado em: abril, 2011.

[12] FMP: Feature Modeling Plug-in. http://gsd.uwaterloo.ca/book/export/html/16, Acessado em: abril, 2011.

[13] RWTH. RequiLine. User Manual Version 1.0, (2005) Research Group Software Construction. RWTH Aachen. http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/_requiline/Support/documentation.php#user_manual. Acessado em: fevereiro, 2011.

13 INES - http://www.ines.org.br