ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

COMPARAÇÃO ENTRE FERRAMENTAS PARA LINHA DE PRODUTOS DE SOFTWARE

Trabalho de Conclusão de Curso

Engenharia da Computação

Rogério Aguiar de Lima Júnior
Orientador: Prof. Sérgio Castelo Branco Soares

Recife, junho de 2008

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

COMPARAÇÃO ENTRE FERRAMENTAS PARA LINHA DE PRODUTOS DE SOFTWARE

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Rogério Aguiar de Lima Júnior
Orientador: Prof. Sérgio Castelo Branco Soares

Recife, junho de 2008

UNIVERSIDADE
DE PERNAMBUCO

Rogério Aguiar de Lima Júnior

COMPARAÇÃO ENTRE FERRAMENTAS PARA LINHA DE PRODUTOS DE SOFTWARE

Resumo

A evolução dos sistemas computacionais está possibilitando cada vez mais o acesso e manipulação de informações em qualquer lugar. Essa ubiqüidade de sistemas de informação baseados em *software* está tornando necessário o desenvolvimento cada vez mais rápido de sistemas de *software*. Linha de Produtos de *Software* é uma abordagem de desenvolvimento de famílias de produtos de *software*, que atendem um determinado segmento de mercado, baseada na composição de artefatos e na modelagem do domínio, bastante promissora e que tem apresentado uma grande aceitação no ambiente corporativo. Esse trabalho se propõe a fazer uma análise comparativa das principais soluções de *software* gratuitas e comerciais que visam fornecer suporte às diversas etapas do desenvolvimento de uma linha de produtos.

Abstract

The evolution of computational systems is enabling more access to and manipulation of information anywhere. This ubiquity of software based information systems is making a necessity the faster development of software systems. Software Product Line is a very promising approach for developing families of software products, which pertain to a specific market segment, based on artifacts composition and domain modeling, which has been presenting a great acceptance in the corporate environment. This work proposes to make a comparative analysis of the main free and commercial software solutions which aim to provide support for the diverse steps of developing a *software* product line.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)

OO – Orientação a Objetos
RUP – *Rational Unified Process*
XP – *eXtreme Programming*
SPL – *Software Product Line*
AOP – *Aspect Oriented Programming*
DSL – *Domain Specific Language*
FODA – *Feature Oriented Domain Analysis*
IDE – *Integrated Development Environment*
EMF – *Eclipse Modeling Framework*
XML – *eXtensible Markup Language*
XSLT – *eXtensible Stylesheet Language Transformations*
XSL – *eXtensible Stylesheet Language*
GEF – *Graphical Editing Framework*

Agradecimentos

Gostaria de agradecer, em primeiro lugar, a Deus por ter me dado saúde e paz para realizar esse trabalho de conclusão de curso e, em segundo lugar, ao meu Professor Orientador Sérgio Soares pela dedicação e empenho dispensados não só a este trabalho, mas também a minha pessoa, como figura fundamental nesta orientação, à minha família e à minha noiva Karlise, por tantas ausências justificadas, por tanto stress, pela paciência, pelo incentivo nas horas mais difíceis e pela crença na minha capacidade. Fica aqui o meu muito obrigado a todos que de alguma forma contribuíram para que esse dia tão esperado chegasse, a conclusão do meu curso de graduação.

Capítulo 1

Introdução

A evolução dos sistemas computacionais tem possibilitado cada vez mais o acesso e a manipulação de informações em qualquer lugar. Essa ubiqüidade de sistemas de informação baseados em *software* nos vários aspectos da vida comum está tornando necessário o desenvolvimento cada vez mais rápido de sistemas de *software*.

Entretanto, com o aumento da demanda, houve também o aumento de fatores de qualidade exigidos pelos usuários finais dos sistemas, tais como disponibilidade, tolerância a erros, usabilidade etc. Estes requisitos não-funcionais estão sendo cada vez mais cobrados dos desenvolvedores dos *softwares*.

Visando atender as demandas das aplicações atuais, os desenvolvedores se concentram no uso de paradigmas de programação tais como Orientação a Objeto (OO), que tem sido visto como uma solução razoável para atingir reuso de software.

Conseguindo atingir o objetivo de reuso de *software*, a diminuição do esforço utilizado no desenvolvimento e a utilização de *software* previamente testado, os desenvolvedores têm mais tempo para atingir os requisitos não funcionais.

Do lado dos processos de desenvolvimento de software, têm-se como exemplo o *Rational Unified Process* (RUP) e o *eXtreme Programming* (XP). Porém, estas metodologias, mesmo aliadas à OO, não conseguem atingir um nível de reuso adequado para o desenvolvimento de produtos de *software* que são constituídos em famílias.

Com o objetivo de tratar da questão de desenvolvimento de famílias de *software*, surgiu o conceito de Linha de Produtos de *Software* (do inglês *Software Product Lines* – SPL)[1].

As empresas estão descobrindo que essa prática de construir conjuntos de sistemas relacionados a partir de artefatos comuns pode trazer melhorias quantitativas consideráveis em

produtividade, *time to market* (o tempo que leva desde a concepção de um produto até ele atingir o mercado), qualidade de produto e satisfação do consumidor.

Uma Linha de Produtos de *Software* é um conjunto de sistemas de *software* que compartilham de um conjunto de *features* (características) comum e gerenciado, que satisfazem as necessidades de um segmento de mercado particular ou missão e que são desenvolvidos a partir de um conjunto comum de artefatos de uma maneira pré-determinada [1].

As principais motivações para o uso de Linhas de Produtos de *Software* são:

- Ganhos em produtividade em larga escala;
- Diminuição do *time to market*
- Manter a presença de mercado
- Manter um crescimento sem precedentes
- Melhorar a qualidade do produto
- Aumentar a satisfação do cliente
- Atingir objetivos de reuso
- Permitir customização em massa

Em face de todos os benefícios que a utilização do conceito de Linha de Produtos de *Software* pode trazer, é necessário que se leve em consideração fatores que podem acarretar num alto custo de entrada para a implementação deste conceito em uma empresa.

As empresas desejam diminuir seu custo e aumentar seu lucro, para isso um bom suporte ferramental se torna necessário. Para apoiar a empresa na implantação de Linha de Produtos de *Software* existem várias ferramentas que auxiliam desde a modelagem do domínio até a implementação real dos produtos. O suporte ferramental para Linha de Produtos de *Software* é amplo, e este trabalho se propõe a analisar fatores de custo e qualidade das ferramentas *pure::variants*[3], *fmp*[2], *XFeature*[4] e *Gears*[5].

1.1 Objetivos

Neste trabalho será feito um comparativo entre as ferramentas para Linha de Produtos de *Software* mais utilizadas do mercado com o objetivo de tornar mais fácil o processo de escolha, por parte de uma empresa, de qual ferramenta utilizar para suportar sua linha de produtos.

Os objetivos e metas deste trabalho são:

- Identificação dos recursos existentes em cada ferramenta de SPL;
- Levantamento dos pontos fracos e fortes das ferramentas analisadas;
- Identificação dos pontos de possíveis melhorias das ferramentas;
- Identificação dos cenários onde cada ferramenta melhor se aplica.

1.2 Metodologia

Uma vez que este trabalho se propõe a fazer uma avaliação de um conjunto de ferramentas, será analisado um estudo de caso baseado na literatura, o *EShop*[2].

O EShop consiste numa família de produtos de lojas eletrônicas, que faz venda de produtos *on-line*. Essas lojas possuem as seguintes características:

- Forma de pagamento: pode ser cartão de crédito, cartão de débito e ordem de compra.
- Detecção de fraudes.
- Envio: pode ser algum método customizado (padrão, gratuito), utilizar algum *gateway* de envio (UPS, FedEx, CanadaPost, USPC).
- Política de senha:
    - Expiração: número de dias ou nunca expira.
    - Caracteres necessários: letras maiúsculas, números, caracteres especiais, letras minúsculas.

Essa família de produtos permite uma customização das características das lojas, apresentadas acima. Apesar de não representar uma modelagem completa de uma loja eletrônica essa família de produtos possui características suficientes para ilustrar os recursos das ferramentas de *feature model* que serão analisadas.

A família permite ilustrar diversas situações de cardinalidades (no mínimo um e no máximo infinitos tipos de caracteres necessários para a senha, por exemplo), opcionalidade de

features (detecção de fraudes, por exemplo), obrigatoriedade de *features* (forma de pagamento, por exemplo), *features* com atributos (número de dias de expiração, por exemplo).

Visando atingir os objetivos deste trabalho, foram executados os seguintes passos:

1. **Instalação**: baixar e instalar as versões das ferramentas a serem analisadas. As ferramentas a serem avaliadas são: *pure::variants*, *fmp*, *gears* e *XFeature*.
2. **Estudo**: nesta tarefa as documentações das ferramentas instaladas serão estudadas, visando facilitar a implementação do estudo de caso em cada ferramenta;
3. **Análise crítica**: Nesta tarefa, os recursos que foram utilizados em cada ferramenta serão analisados de acordo com os seguintes critérios: *background* (comercial ou acadêmico, se ainda está em desenvolvimento), técnica de modelagem e notação utilizada (tipos de visualização e manipulação de dados disponíveis), implementação (*plugin* ou ferramenta *stand-alone*), facilidade de uso e custo.
4. **Estudo de caso**: o estudo de caso selecionado será implementado em todas as ferramentas que serão analisadas. Nessa tarefa serão identificadas as vantagens que uma ferramenta pode ter sobre as outras.
5. **Matriz de recursos**: serão montadas tabelas comparativas que permitam que as ferramentas sejam analisadas pelos seus recursos de uma maneira visualmente simples.

1.3 Escopo

Este trabalho se limitará a avaliar as ferramentas *pure::variants*, *fmp*, *Gears* e *XFeature*. Os atributos destas ferramentas que serão analisados serão apenas os especificados na metodologia na Seção 1.2.

1.4 Organização do trabalho

O Capítulo 2 introduz os principais conceitos relacionados a Linha de Produtos de *Software*. Já o Capítulo 3 apresenta a análise de cada ferramenta. O Capítulo 4 apresenta as tabelas comparativas. Por fim, o Capítulo 5 apresenta a conclusão do trabalho.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os principais conceitos utilizados neste trabalho. Os mesmos serão apresentados de forma clara e concisa com o intuito de que diferentes pessoas de diferentes áreas de estudo possam compreender.

2.1 Linha de Produtos de Software

Linha de Produtos de Software (do inglês *Software Product Lines*, SPL) [1] é um *framework* de processos que focam no desenvolvimento de uma família de produtos com o objetivo de atingir um mercado específico e baseado numa base comum de artefatos. Família de produtos de *software* é um conjunto de produtos de *software* com características suficientemente similares para permitir a definição de uma infra-estrutura comum de estruturação dos itens que compõem os produtos (membros) e a parametrização das diferenças entre produtos. Um exemplo de uma família de produtos é a família de produtos para controle de satélites (Ccontrol Channel Toolkit, Clements and Northrop, 2002):

- Arquitetura de software comum
- Componentes: escalonadores, Órbita, Manobrista, Planejamento de acesso aos dados e Avaliação.
- Exemplo de variação: pontos de variação permitem que ferramentas específicas dos produtos gerem dados de órbita ou que fontes externas forneçam dados de órbita.

Em uma linha de produtos, há uma arquitetura genérica que é comum a todos os produtos da linha; essa arquitetura é adaptada permitindo a criação de um produto particular. Por exemplo, o sistema operacional *Windows Vista* pode ser considerado uma linha de produtos, uma vez que

ele possui várias versões com a mesma base arquitetural, que é alterada, adicionando-se ou removendo componentes, para permitir a geração das diversas versões.

Cada produto da linha é definido a partir de uma seleção de *features*, que são atributos que caracterizam as funcionalidades do produto. Essas seleções dos diferentes pontos de variabilidade (diferenças tangíveis entre produtos que podem ser reveladas e distribuídas entre os artefatos da Linha de Produtos) de um produto são o conjunto de pontos onde ele difere dos outros produtos da linha. Existem vários níveis de variabilidade: de código, de recursos, de arquitetura etc.

Os benefícios de uma Linha de Produtos de *Software* são derivados do reuso de artefatos de uma maneira estratégica e pré-definida. Quando o repositório de artefatos da linha de produtos estiver estabelecido, os ganhos serão identificados nas seguintes áreas:

- **Requisitos**: existem requisitos comuns da linha de produtos. Os requisitos dos produtos individuais são apenas deltas (diferenças) desta base estabelecida de requisitos, o que poupa uma análise de requisitos mais extensiva.
- **Arquitetura**: A arquitetura de *software* de um programa ou sistema computacional é a estrutura do sistema, que é composta de componentes de *software*, as propriedades visíveis externamente dos mesmos e a relação entre eles. A arquitetura é uma parte fundamental para atingir os objetivos dos produtos, uma vez que a arquitetura tenha sido desenhada corretamente numa linha de produtos, ela será usada para cada produto através de instanciação, poupando tempo e risco. O processo de instanciação consiste na adição ou remoção de partes da arquitetura que irão entrar ou não em um determinado produto da linha, permitindo que a arquitetura seja customizada para atender exatamente as necessidades do produto. Enquanto a academia tem focado em deixar a arquitetura definida explicitamente, com componentes e conectores, usando linguagens específicas de definição de arquitetura para descrever e gerar aplicações, a indústria tem normalmente apenas uma compreensão conceitual da arquitetura, com pouca descrição explícita e, menos ainda, formal e, também, com soluções ad-hoc para conexão entre elementos da arquitetura. A instanciação da arquitetura de um produto em uma linha de produtos de *software* é feita, idealmente, como um processo automático, que tem como entrada a seleção de características que estarão no produto, os componentes disponíveis que implementam a arquitetura e o mapeamento das relações entre estes.
- **Componentes**: Existem três níveis de reuso de componentes:
    - Reuso de componentes em versões subseqüentes dos produtos: o tipo mais comum que é aplicado com facilidade pelas empresas. A idéia é que o código que foi

desenvolvido em uma versão do produto não seja descartado, e sim reaproveitado para versões subseqüentes do produto, exceto em casos que requeiram uma completa reescrita da aplicação.

- Reuso de componentes em versões do produto e em vários produtos: é nesse ponto que foca o uso de linha de produtos de *software*. Consiste no desenvolvimento de componentes que funcionam não apenas em um produto, mas nos vários produtos da linha, e não apenas em uma versão dos produtos da linha, mas também nas versões subseqüentes.
- Reuso de componentes em versões do produto, em vários produtos e em diferentes organizações: muito longe de estar maduro, é pouco aplicado na indústria. Engloba o reuso explicado no ponto anterior, e vai além, propondo que os componentes sejam desenvolvidos de forma que possam ser utilizados por outras empresas, que podem, ou não, estar desenvolvendo produtos para o mesmo domínio que a empresa que desenvolve os componentes.

Em alguns casos, até todos os componentes disponíveis na linha de produtos podem ser usados em um determinado produto. Esses casos podem ser considerados utópicos, visto que na maior parte das vezes os componentes precisaram de alguma (mesmo que pouca) customização para atender as necessidades de um produto. Esses componentes podem ser alterados através de herança, parâmetros ou, até mesmo, programação orientada a aspectos (do inglês *Aspect Oriented Programming* – AOP[7]).

- **Modelagem e análise**: modelos de performance e suas respectivas análises são artefatos da linha de produtos. Problemas comuns de distribuição, por exemplo, tais como: sincronização, ausência de *deadlocks*(situação em que ocorre um impasse e dois ou mais processos ficam impedidos de continuar suas execuções) etc, são dados como eliminados em novos produtos, uma vez que os mesmos foram resolvidos nos artefatos comuns da linha de produtos.
- **Teste**: similarmente a requisitos, os artefatos de teste (plano de testes, processos de testes, casos de testes, dados de testes etc) já estão na linha de produtos, e só precisam ser adaptados baseados nas variações do produto relacionado.
- **Planejamento**: o plano de produção, que é responsável pelo planejamento geral de todos os produtos, já está desenvolvido. As estimativas para novos produtos são mais confiáveis, uma vez que os projetos existentes fornecem uma excelente base para comparação.

- **Pessoal**: Menos pessoas são necessárias para construir os produtos.

2.2 Feature Model

Feature é uma propriedade do sistema que é relevante para algum *stakeholder* (pessoa influenciada pelo desenvolvimento de um produto de *software*) e é usada para capturar o que há de comum ou discriminar entre produtos de uma mesma linha. Um *feature model* consiste de um ou mais diagramas de *features*, que organizam os *features* em hierarquias.

Modelagem de *features* permite que se modelem as propriedades comuns e variáveis dos membros da linha de produtos, durante todos os estágios da engenharia de linhas de produtos. No caso do EShop, por exemplo, a modelagem de *features* identificaria que as lojas possuem *features* em comum, como o envio de produtos, e *features* opcionais, como detecção de fraude. São essas *features* opcionais, as alternativas e as com cardinalidade que permitem a diferenciação entre os diversos produtos da linha, uma vez que as *features* obrigatórias sempre estão presentes em todos os produtos da linha. A criação de um novo produto resume-se então, no nível de *feature model* à seleção de quais *features* não-obrigatórias estarão disponíveis no produto.

Num estágio inicial, a modelagem de *features* permite a definição do escopo da linha de produtos, definindo quais *features* serão suportados pela linha, e quais não serão. Além disso, *feature models* permitem a derivação de linguagens específicas de domínio (do inglês *Domain Specific Languages* – DSL), que são usadas para especificar membros da linha de produtos usando Programação Generativa[8][9].

Modelagem de *features* foi proposta como parte do método *Feature Oriented Domain Analysis* (FODA) [10], e desde então tem sido aplicada em diversos domínios. Modelagem de *features* baseada em cardinalidade[11] estende a modelagem de *features* original do FODA com cardinalidade de *features* e de grupos, atributos de *features*, referências no diagrama de *features*, e anotação definidas pelo usuário. Cardinalidade de *features* e de grupos permite que se definam, respectivamente, quantas cópias da mesma *feature* podem existir no modelo, e quantos filhos do grupo podem ser selecionados. As cardinalidades normalmente se encontram especificadas como: 0 a 1, 1 ou mais, 2 a 4, etc. Atributos de *features* permitem que as *features* possuam um valor em cada instância do *feature model*, valor este que pode ser uma *string*, um inteiro, um booleano, etc. Por fim, anotações permitem que o usuário adicione propriedades às *features*, alterando diretamente o meta-modelo, de modo que todas as *features* possuam a propriedade customizada.

2.3 Configuration Knowledge

O *feature model*, por si só, apenas representa a modelagem do domínio, mas não representa o modo como os produtos deste domínio serão gerados a partir dos artefatos da linha de produtos. O mapeamento entre o *feature model* e os artefatos de implementação é o que se chama de *Configuration Knowledge*[9].

Os artefatos de implementação podem estar modelados direto no modelo que representa o *configuration knowledge* ou em um modelo a parte, neste caso o *configuration knowledge* associará os artefatos de um modelo com as *features* do *feature model*.

Quando não estão dentro do *configuration knowledge*, os artefatos de implementação encontram-se em um modelo que assume diferentes nomes, dependendo da ferramenta utilizada, tais como *family model*, *component model*, *architecture model*, etc.

O mapeamento existente no *configuration knowledge* é essencialmente um conjunto de regras que definem que artefatos de implementação (classes, arquivos de recursos, etc) entram em cada produto da linha. Tomando o EShop como exemplo, uma classe chamada FraudDetection que trata da detecção de fraudes estaria representada no *configuration knowledge* como uma regra que diz se a *feature* detecção de fraudes estiver selecionada em um produto, a classe entraria nesse produto também, essencialmente uma relação de implica (*feature* implica em artefato de implementação).

Por trás do processo de geração de produtos de uma linha de produtos de *software* se encontra uma *engine* de resolução de expressões lógica, que checa se todas as restrições presentes no *feature model* foram respeitadas em cada instância, e, para cada seleção de *features*, verifica que artefatos de implementação estarão habilitados na instância. Uma vez que todas as verificações tenham sido realizadas, a saída do processo de geração de produtos vai ter como resultado, para cada produto, uma lista dos artefatos de implementação habilitados no produto. Dependendo da ferramenta utilizada, o usuário pode especificar diretamente na ferramenta o que será feito com essa lista, como gerar um projeto individual na IDE utilizada com uma cópia de cada artefato de implementação, ou invocar um sistema de empacotamento dos produtos que geraria os executáveis finais, permitindo assim uma maior customização do processo de geração.

Capítulo 3

Análise das ferramentas

Neste capítulo será apresentada a análise de cada ferramenta. Os critérios analisados de cada ferramenta são: *background*, custo, implementação, técnica de modelagem e notação utilizada e facilidade de uso.

Será analisado um estudo de caso em cada ferramenta, que permitirá uma comparação das mesmas tendo uma linha base comum.

3.1 fmp

O *fmp* (*Feature Model Plugin*) é um *plugin* para o Eclipse desenvolvido pelo *Generative Software Development Lab*, da *University of Waterloo* [17]. Ele utiliza como base o *Eclipse Modeling Framework*, o que, de acordo com os desenvolvedores, reduziu significativamente o esforço de desenvolvimento.

O *fmp* apenas apresenta suporte à modelagem do domínio, através de *feature models*. Ele não possui nenhuma funcionalidade de *configuration knowledge*, nem nada que permita, dentro do próprio *fmp*, o mapeamento necessário entre *features* e artefatos da linha de produtos para permitir a geração dos produtos.

3.1.1 Background

O *fmp* apresenta um *background* puramente acadêmico, não sendo conhecido se ele é utilizado comercialmente. O site do projeto indica que o mesmo está completo na versão 0.6.6 e que não está sendo mais mantido, em favor de um projeto chamado Ecore.fmp, que permitirá a edição de modelos EMF Ecore [12], que são modelos não especificamente de *features*, e sua visualização e

desenvolvimento como *feature models*. O ECore.fmp está definido como em progresso, mas não há atualizações no *site* desde 13 de Julho de 2007, e também não há nenhuma versão disponível para *download*.

Apesar de o *fmp* estar definido como completo, uma nova versão, 0.7.0, considerada em desenvolvimento, está disponível no *site*. Essa nova versão muda o sistema de restrições no *feature model*, substituindo o formato de expressões antigo, baseado em XPath 2.0[18], por uma linguagem de expressões mais simplificada.

3.1.2 Custo

Por ser uma ferramenta *Open Source*, o *fmp* é completamente gratuito e, além disso, o código fonte do mesmo está disponível para qualquer um que deseje contribuir com correções e novos *features*.

3.1.3 Implementação

O *fmp* foi implementado baseado na versão 3.2 do Eclipse (a versão mais recente é a 3.3), utilizando o *Eclipse Modeling Framework* para definição do meta-modelo de *feature model* e utilizando os recursos de geração de código do EMF.

Os modelos gerados pelo *feature model* são gravados no sistema como um arquivo XML (do inglês *Extensible Markup Language*), o que pode permitir que geradores, utilizando XSLT[13] (do inglês *Extensible Stylesheet Language Transformation*), por exemplo, possam processar o modelo. Por ser um *plugin* para o Eclipse e possuir uma API bem definida, possibilita que outras ferramentas se integrem com o mesmo; é sabido que pelo menos uma ferramenta[14] se integra com o *fmp*.

3.1.4 Técnica de modelagem e notação utilizada

O *fmp* implementa modelagem de *features* baseada em cardinalidades[11], ou seja, permite a definição de cardinalidades de *feature* e de grupo, atributos de *feature*, referências e anotações definidas pelo usuário.

Cardinalidades de *feature* e grupo permitem que sejam especificados o número mínimo e máximo de filhos de uma *feature* que podem ser selecionados, 0 ou mais, 1 a 4, por exemplo. Atributos de *feature* permitem que cada feature possua um valor, que pode ser uma *string*, um booleano, um inteiro, etc, o que aumenta a expressividade do modelo. Anotações definidas pelo

usuário permitem que o usuário adicione novas propriedades as *features*, além das básicas (nome, cardinalidade, valor), permitindo assim uma maior customização do modelo.

Para suportar anotações definidas pelo usuário, o *fmp* permite que o usuário altere o meta-modelo do *feature model*, permitindo, por exemplo, que toda *feature* tenha um atributo chamado “quantidade”; ou qualquer coisa que o usuário deseje colocar.

A Figura 1 demonstra a interface do *fmp*.

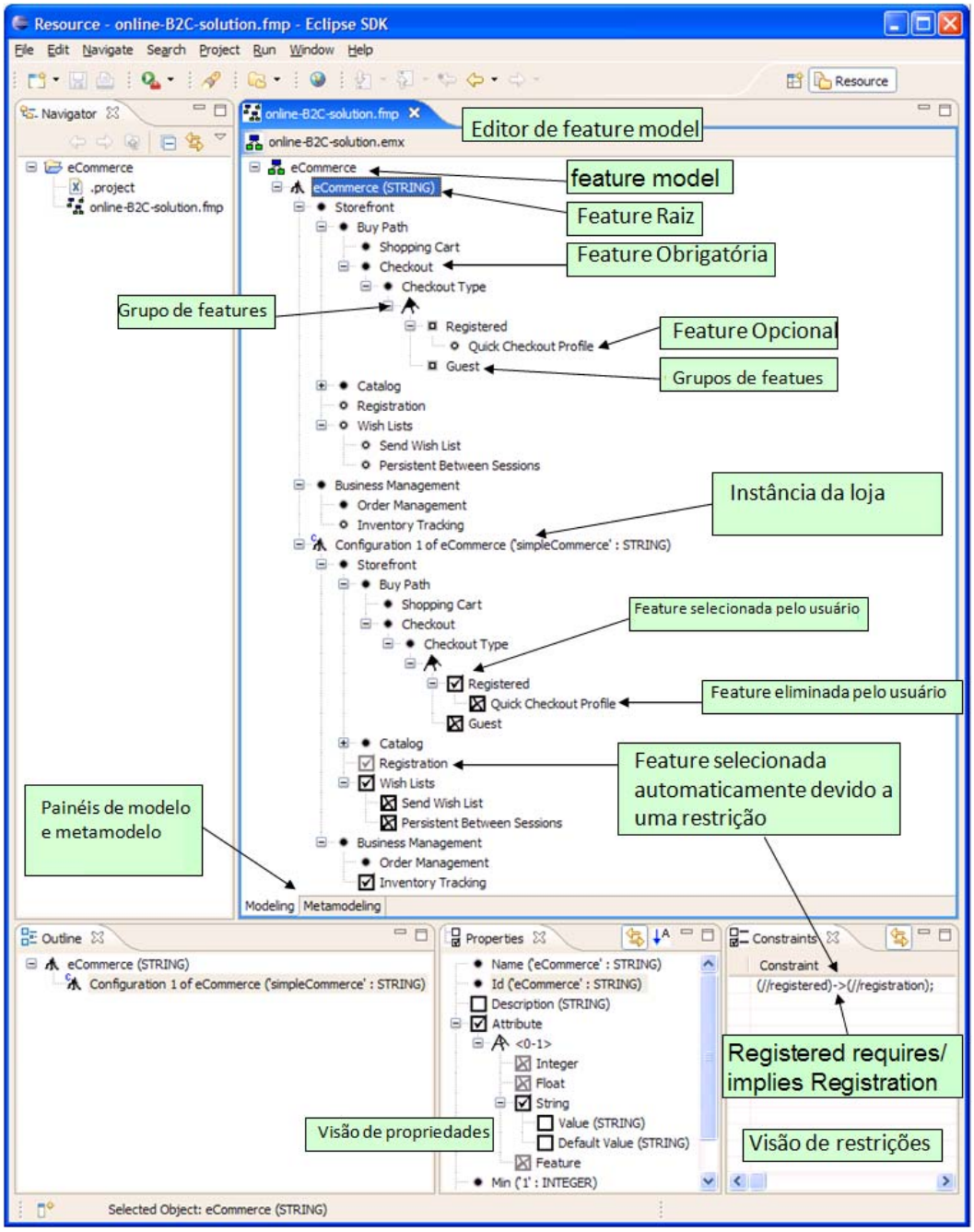

Figura 1. Interface do *fmp*

O *fmp* só possui uma maneira de visualizar e editar o *feature model*, que é em forma de árvore, como demonstrado na Figura 1, onde encontramos todos os recursos disponíveis no *fmp*, tais como *features* obrigatórias, opcionais, em grupo, instâncias, seleção de *feature* pelo usuário e automática devido a uma restrição, a edição de restrições, de propriedades da *feature*. Como foi mencionado anteriormente, o *fmp* permite a definição de restrições sobre o *feature model*, de modo que o próprio *fmp* não permita que o usuário selecione a *feature* 'x' quando houver uma restrição dizendo "se a *feature* 'y' estiver selecionada, não selecione a *feature* 'x'". Todas as restrições encontram-se na *feature* raiz do *feature model*.

Na aba de propriedades o usuário pode editar os atributos presentes no meta-modelo (que podem ter sido adicionados pelo próprio usuário), tais como cardinalidade, descrição, nome. Para criação de instâncias, o que é chamado, na terminologia do *fmp*, de configuração de instâncias, o usuário deve clicar com o botão direito na *feature* raiz do *feature model* e selecionar a opção "*new configuration of feature*", o que gerará uma árvore não preenchida para que as *features* sejam selecionadas, respeitando as restrições existentes no modelo.

3.1.5 Facilidade de uso

Por ser uma ferramenta integrada ao Eclipse, os desenvolvedores que já estão familiarizados com o desenvolvimento no Eclipse provavelmente não se sentirão desconfortáveis ao utilizar o *fmp*. A linguagem utilizada para definições de restrições, XPath, não apresenta uma boa legibilidade, mas a versão em desenvolvimento do *fmp*, disponível no *site* da ferramenta, apresenta uma linguagem de definição de restrições bastante simples.

Como a definição de instâncias do *feature model* encontra-se no mesmo modelo, junto com a definição do *feature model*, isso pode vir a ser um inconveniente com modelos grandes que possuam uma grande quantidade de instâncias. O *fmp* permite que um modelo importe outro, permitindo que uma instância fique definida em um arquivo, e o *feature model* em outro, porém, nos testes realizados com a ferramenta, essa funcionalidade apresentou problemas e não funcionou corretamente.

A notação utilizada no editor de *feature model* é simples, porém não é intuitiva. Entretanto, após a leitura da documentação da ferramenta e pouco tempo de uso, o usuário poderá se acostumar facilmente com os ícones e notações utilizados.

3.1.6 Estudo de caso

Uma vez que o estudo de caso é originalmente do *fmp*, não houve problemas para a implementação do mesmo. Diversos recursos do fmp foram utilizados, tais como: grupo de *features*, cardinalidades, *features* opcionais, *features* com valor, referências e configurações. A criação do modelo do estudo de caso foi feita para que o modelo criado fosse exatamente o igual ao apresentado no artigo que descreve o estudo de caso. As *features* foram criadas na ordem apresentada no modelo, foram criados inicialmente um sub-modelo para cada uma das *features:* pagamento, envio e política de senha, uma vez que estes estivesse completos, o modelo da loja foi criado utilizando referências para essas *features*, seguido da criação de uma instância do modelo.

O estudo de caso mostrou que é fácil criar um *feature model* e instâncias do mesmo utilizando o *fmp*. Como não há *configuration knowledge*, não houve nenhuma associação de artefatos reais com as *features* do domínio, nem a geração de produtos individuais.

A figura 2 apresenta o *EShop* desenvolvido no *fmp*. Pode-se identificar que todas as *features* do EShop foram modeladas claramente no *fmp*, podemos ver a *feature* opcional de deteção de fraude (opcionalidade representada por uma bola branca ao lado do nome), podemos ver a *feature* com atributo inteiro *InDays* que define o número de dias para a expiração da senha e grupos de *features* com cardinalidade, como tipos de pagamento, onde se pode escolher um ou mais tipos.

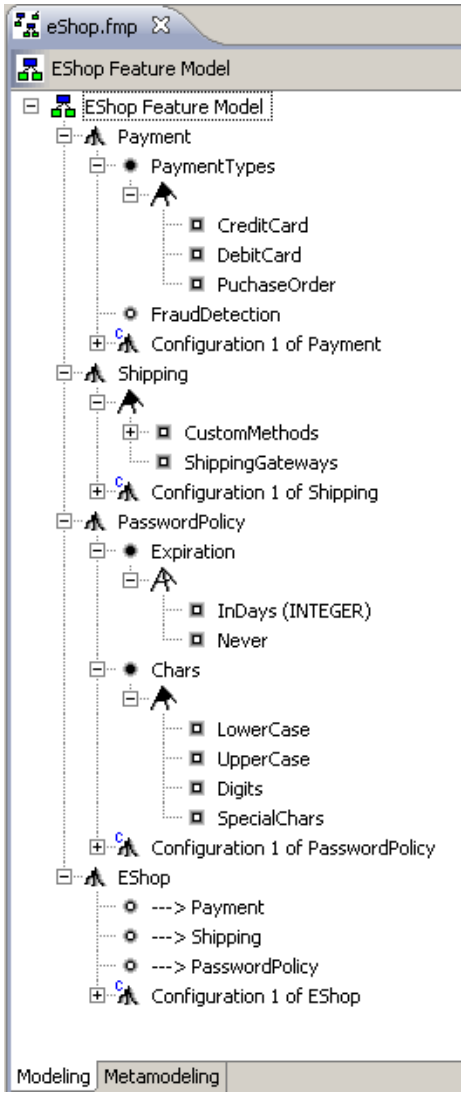

Figura 2. *EShop* no *fmp*

3.2 XFeature

O *XFeature* é um *plugin* para o Eclipse desenvolvido pela *P&P Software*, que é uma empresa que se originou no *Institute of Automatic Control of ETH* (*Swiss Federal Institute of Technology*).

O *XFeature* foi criado para demonstrar um conceito de uma ferramenta para automatizar o processo de modelagem e configuração de artefatos reusáveis de *software*. A ferramenta se apresenta como inovadora pela possibilidade de customização do meta-modelo da família de produtos[15].

O *XFeature* apresenta suporte à modelagem do domínio, através de *feature models* e não possui a funcionalidade de *configuration knowledge*, que permite o mapeamento necessário entre *features* e artefatos da linha de produtos para permitir a geração dos produtos.

3.2.1 Background

O *XFeature* apresenta um *background* puramente acadêmico e corporativo. Não é conhecido se ele é utilizado comercialmente, porém, o site da *P&P* lista como clientes passados e atuais da empresa as seguintes instituições: *European Space Agency*, *Alenia-Spazio*, *Astrium GmbH*, *Alcatel Space of France*, *Roche Diagnostics AG*, logo, é possível que alguns destes clientes esteja utilizando o *XFeature*. O site do projeto indica que o mesmo está atualmente na versão 2.1.2, que funciona na versão mais recente do Eclipse, a 3.3. O projeto encontra-se em constante desenvolvimento.

Originalmente o *XFeature* foi desenvolvido visando o seu uso em aplicações espaciais (sistemas de controle embutidos para naves espaciais, por exemplo), porém, não há nenhuma restrição na maneira que foi desenvolvido que não permita o seu uso em outros contextos.

3.2.2 Custo

Por ser uma ferramenta *Open Source* licenciado sob *a GNU General Public Licence*, o *XFeature* é completamente gratuito, e, além disso, o código fonte do mesmo está disponível para qualquer um que deseje contribuir com correções e novos *features*, que podem ser aceitas no projeto oficial a critério da *P&P*.

3.2.3 Implementação

O *XFeature* foi implementado baseado na versão 3.3 do Eclipse, ao contrário do *fmp*, ele não utilizou o EMF para a definição do meta-modelo de *feature model*. O seu editor é baseado no GEF (*Graphical Editing Framework*) do Eclipse.

O XFeature depende fortemente de *XML* e de transformações *XSL*. *Scripts Ant* são utilizados para a verificação e geração de arquivos de configuração da aplicação, tais como os que customizam o editor dos modelos.

3.2.4 Técnica de modelagem e notação utilizada

Ao contrário do *fmp* e do *pure::variants*, o *XFeature* é extremamente flexível, permitindo que a técnica de modelagem de *features* seja escolhida pelo usuário, sendo fornecidas quatro

configurações diferentes: *FD*, *FMP*[11], *ICSR*[15] e *SimplifiedICSR*. Uma das configurações é justamente a utilizada pelo *fmp*. As configurações *FD* e *SimplifiedICSR* são ambas baseadas na configuração *ICSR*. Essas configurações definem quais são os elementos presentes no modelo de *features* que será utilizado pelo usuário. *ICSR* é bastante similar a FMP, tendo a mais a idéia de macros de *features*, que permitem que o *feature model* seja dividido em várias partes e composto através dessas macros, similarmente à funcionalidade de macro presente em algumas linguagens de programação. *FD* e *SimplifiedICSR* removem algumas características de *ICSR*, *SimplifiedICSR* remove a idéia de macros e *FD* que impõe algumas restrições sobre quais os possíveis tipos de filhos para *features* macro.

O processo de criação de modelos no *XFeature* é consideravelmente mais complicado do que nas outras ferramentas. Primeiramente se define qual será a configuração utilizada (*FD*, *FMP*, *SimplifiedICSR* ou *ICSR*). Em seguida, é criado o que o XFeature chama de *family model*, que é o meta-modelo do domínio. Após a validação do *family model* sob a configuração utilizada, o usuário deve clicar em um botão que gera dois meta-modelos: o de aplicação e o de *display*. O meta-modelo de aplicação é o que será utilizado como configuração dos modelos que vão conter as instâncias da linha de produto, e o meta-modelo de *display* define quais os elementos visuais que aparecerão no editor. Apesar de ser mais complicado, o *XFeature* permite que o usuário tenha mais controle sobre como são criados os *feature models*, uma vez que o usuário não fica preso à única configuração presente nas outras aplicações.

Uma vez criados os modelos das instâncias da linha de produtos, eles podem ser validados contra seu meta-modelo, que foi definido pelo usuário anteriormente. Quando os modelos estiverem validados, eles podem ser utilizados como entrada pra outras ferramentas que possuam a parte de *configuration knowledge* para gerar os produtos finais.

A Figura 3 demonstra a interface do *XFeature*.

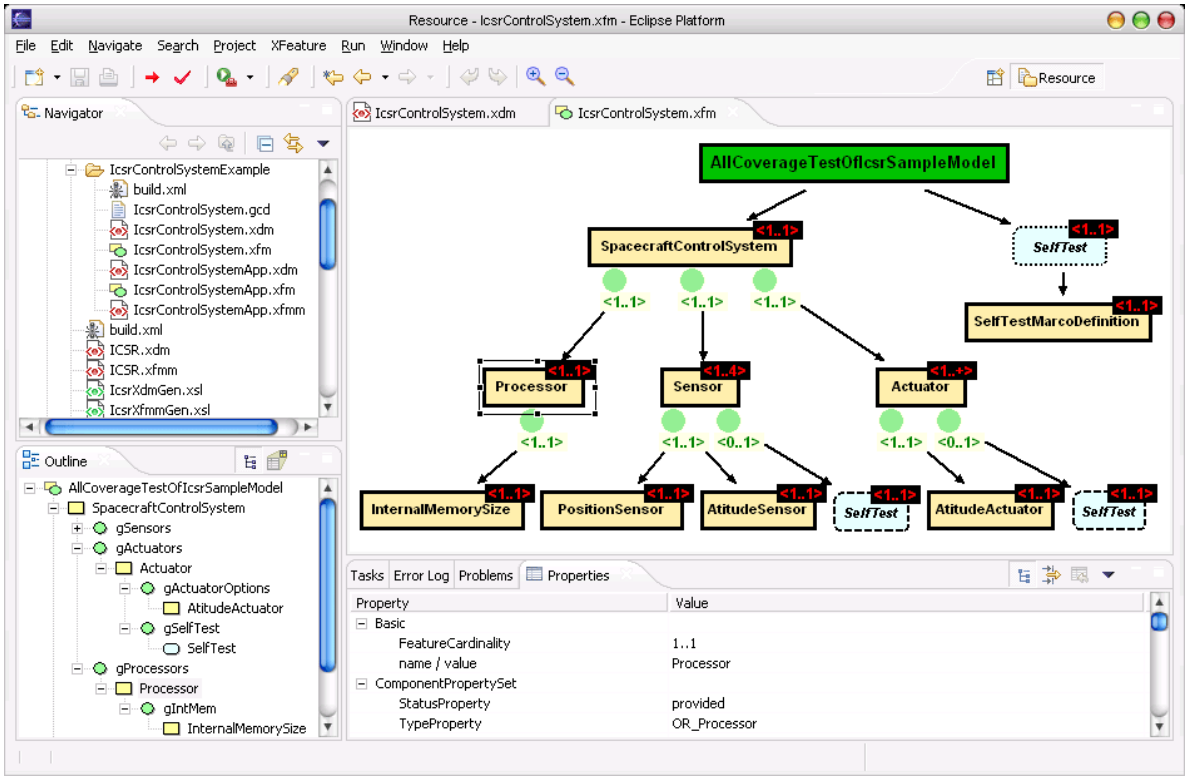

Figura 3. Interface do *XFeature*

O *XFeature* só possui uma maneira de visualizar e editar os seus modelos, que é em forma de árvore, como demonstrado na Figura 3, onde encontramos *feature* raiz, no topo da árvore, *features* opcionais, pontilhadas, e *features* obrigatórias, em caixas amarelas, assim como o editor de propriedades de *features*. O *XFeature* também permite a definição de restrições globais sobre o *feature model*. O usuário cria um modelo de restrições, que passa pelo *global constraints compiler*, que vai gerar um conjunto de arquivos *XSL* que permitem a verificação das restrições.

3.2.5 Facilidade de uso

Igualmente ao *fmp*, o *XFeature* é integrado com o Eclipse. Entretanto, a facilidade de uso normalmente associada com o fato de ser uma ferramenta integrada com um ambiente de desenvolvimento conhecido é praticamente inexistente com o *XFeature*. A quantidade de arquivos de validação, geração, *display*, meta-modelos etc, é enorme. A barreira de entrada para que um usuário aprenda para que serve cada arquivo é de difícil transposição.

É consideravelmente repetitiva e trabalhosa a criação de instâncias do *feature model*, uma vez que o usuário não é apresentado a um modelo completo para seleção, ele deve criar a árvore inteira, nó por nó, podendo criar apenas os nós definidos no *feature model*.

De todas as ferramentas analisadas, o *XFeature* foi a menos intuitiva e com a pior usabilidade.

3.2.6 Estudo de caso

O estudo de caso foi desenvolvido utilizando como meta-modelo para definição do *feature model* a configuração *FMP* disponível no *XFeature*. Como a configuração *FMP* provê os recursos disponíveis no *fmp*, não houve dificuldade para criar o *feature model*. As figuras 4 e 5 apresentam os modelos criados durante o estudo de caso no *XFeature*.

A notação difere um pouco do *fmp*, mas podemos ver todas as *features* encontradas no estudo de caso do *fmp* nas figuras 4 e 5. A cardinalidade é representada visualmente como um *range* entre os sinais de menor '<' e maior '>', onde <1..1> significa que apenas uma feature deve ser selecionada, <0..1> significa que 0 ou 1 *feature* devem ser selecionadas e <1..*> significa que uma ou mais *features devem ser selecionadas.*

A criação de instâncias no *XFeature* é consideravelmente mais trabalhosa do que no *fmp*, uma vez que elas têm que ser criadas nó a nó, dado que não somos apresentados a um *feature model* completo para seleção de apenas as *features* desejadas. Assim como o *fmp*, não há *configuration knowledge* e não houve nenhuma associação de artefatos reais com as *features* do domínio, nem a geração de produtos individuais.

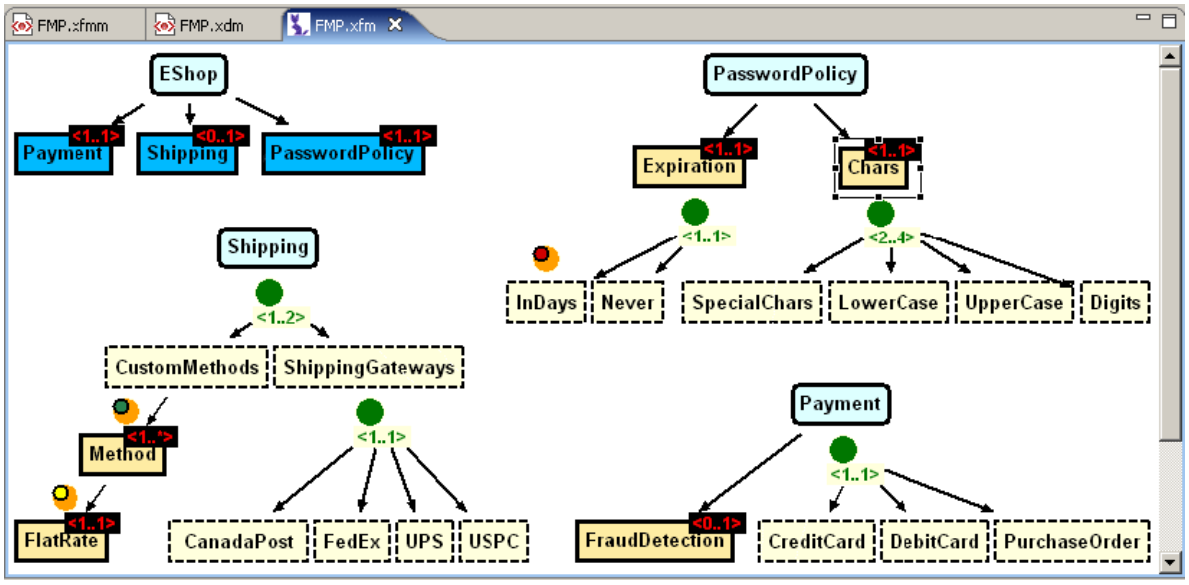

Figura 4. Feature model do *EShop* no *XFeature*

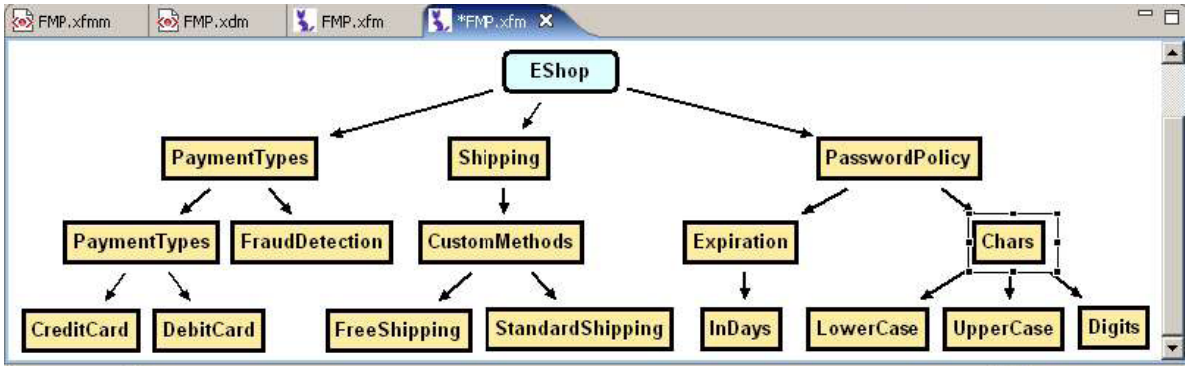

Figura 5. Instância do *EShop* no *XFeature*

3.3 pure::variants

O pure::variants é um *plugin* para o Eclipse desenvolvido pela *pure-systems GmbH*, uma empresa que se originou no *Institute Otto-von-Guericke-Universität Magdeburg* e no *Fraunhofer Instituts Rechnerarchitektur und Softwaretechnik*. A empresa tem como objetivo o desenvolvimento de softwares para sistemas embarcados, que se baseia no desenvolvimento de componentes de software e de ferramentas de desenvolvimento de *software*.

O *pure::variants* foi desenvolvido para suportar o desenvolvimento e a implantação de linhas de produtos e famílias de *software*. O *pure::variants* provê suporte no desenvolvimento durante as atividades de análise, modelagem, implementação e implantação.

O pure::variants apresenta suporte à modelagem do domínio, através de *feature models* e possui a funcionalidade de *configuration knowledge*, que permite o mapeamento necessário entre *features* e artefatos da linha de produtos para permitir a geração dos produtos.

3.3.1 Background

O *pure::variants* provavelmente teve um início acadêmico, dado que a *pure-systems* surgiu em institutos acadêmicos, mas cresceu a ponto de se tornar uma ferramenta comercial extremamente rica.

O projeto encontra-se em constante desenvolvimento e, similarmente ao *XFeature*, foi inicialmente criado visando o mercado de sistemas embarcados, mas também é usado em outros ambientes, especialmente na indústria automotiva.

3.3.2 Custo

O *pure::variants* é uma ferramenta comercial, logo, requer licença para uso. Ela possui uma versão gratuita para testes, que não pode ser usada comercialmente e possui restrições no tamanho dos modelos. É sabido que o custo do *pure::variants* varia muito dependendo da quantidade de licenças adquiridas. Mas sabe-se que normalmente o preço de uma licença para uma máquina na versão *developer* custa em torno de 200 euros por mês.

O custo também varia dependendo de qual versão da ferramenta será utilizada, a *Professional* ou a *Enterprise*. A versão *Enterprise* possui controle de versões integrado (utilizando CVS, por exemplo), permite que o usuário desfaça modificações no modelo mesmo após ele ter sido fechado (não apenas quando ele está aberto, como ocorre na *Professional*), colaboração *on-line* e gerenciamento de modelos centralizado.

3.3.3 Implementação

O *pure::variants* funciona com a versão 3.3 do Eclipse, mas não se utiliza dos *frameworks* de modelagem e editores gráficos utilizados pelo *fmp* e pelo *XFeature*.

O *pure::variants* utiliza uma arquitetura cliente-servidor, onde o cliente é o *plugin* do Eclipse que é utilizado pelo usuário, enquanto o servidor, que faz o trabalho de modificação real dos modelos, é um aplicativo C++ que fica rodando em *background*.

3.3.4 Técnica de modelagem e notação utilizada

O *pure::variants* utiliza uma notação baseada em FODA, descrita em um artigo[16]. Ele possui 3(três) tipos de modelos diferentes: o *feature model* – onde são definidas as características comuns e variabilidades da linha de produtos, o *family model* – a parte do *configuration knowledge* onde se encontram os componentes que compõem os artefatos da linha de produtos, e o *variant model* – que é o modelo que define uma instância da linha de produtos.

A divisão dos 3 modelos é clara e consideravelmente mais intuitiva do que os inúmeros meta-modelos e modelos que o *XFeature* define, e a definição das instâncias em arquivos separados facilita a organização da linha. Além disso, as mudanças no *feature model* são sincronizadas automaticamente com os *variant models*.

Uma vez criados os *variant models*, eles podem ser validados para determinar se cumprem as restrições do *feature model*, que foi definido pelo usuário anteriormente. Uma vez que estes modelos estejam validados, eles podem ser usados juntamente com o *family model* e o *feature model* para a geração dos produtos da linha, utilizando os recursos de geração do *pure::variants*.

As Figuras de 6 a 9 demonstram a interface do *pure::variants*.

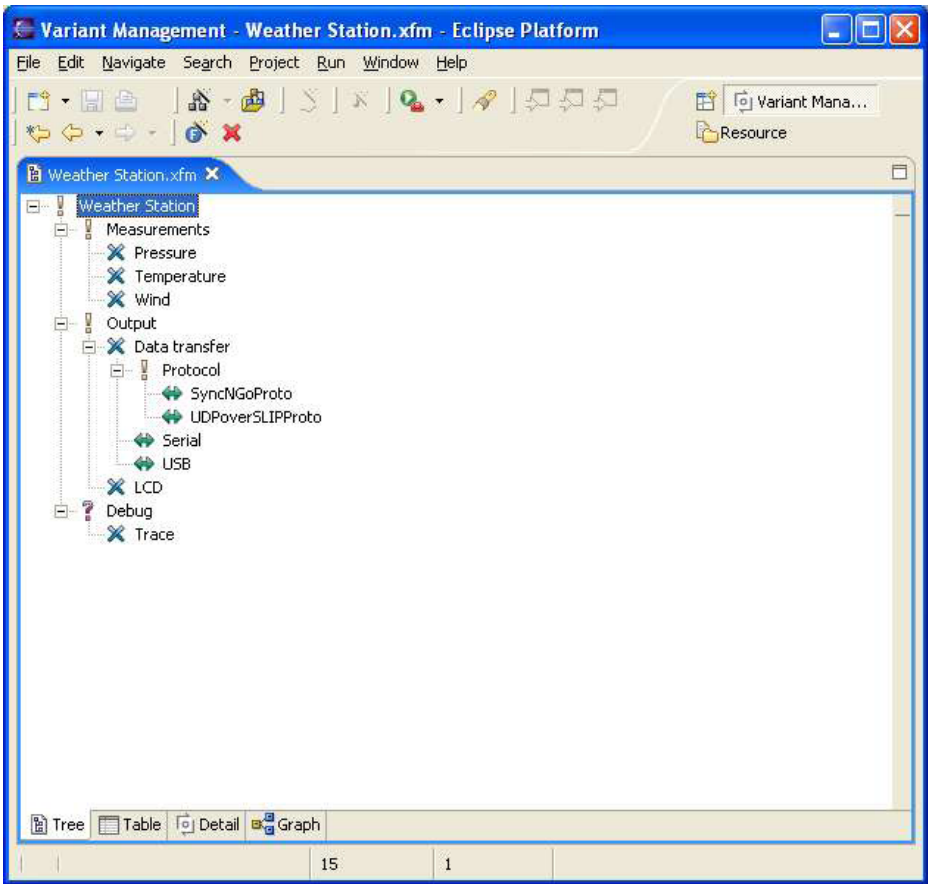

Figura 6. Interface de Edição do *Feature Model* em modo árvore do *pure::variants*

Na Figura 6 encontramos *features* obrigatórias (com exclamação), *features 'or'* (com um X), *features* alternativas (com ←→) e *features* opcionais (com interrogação).

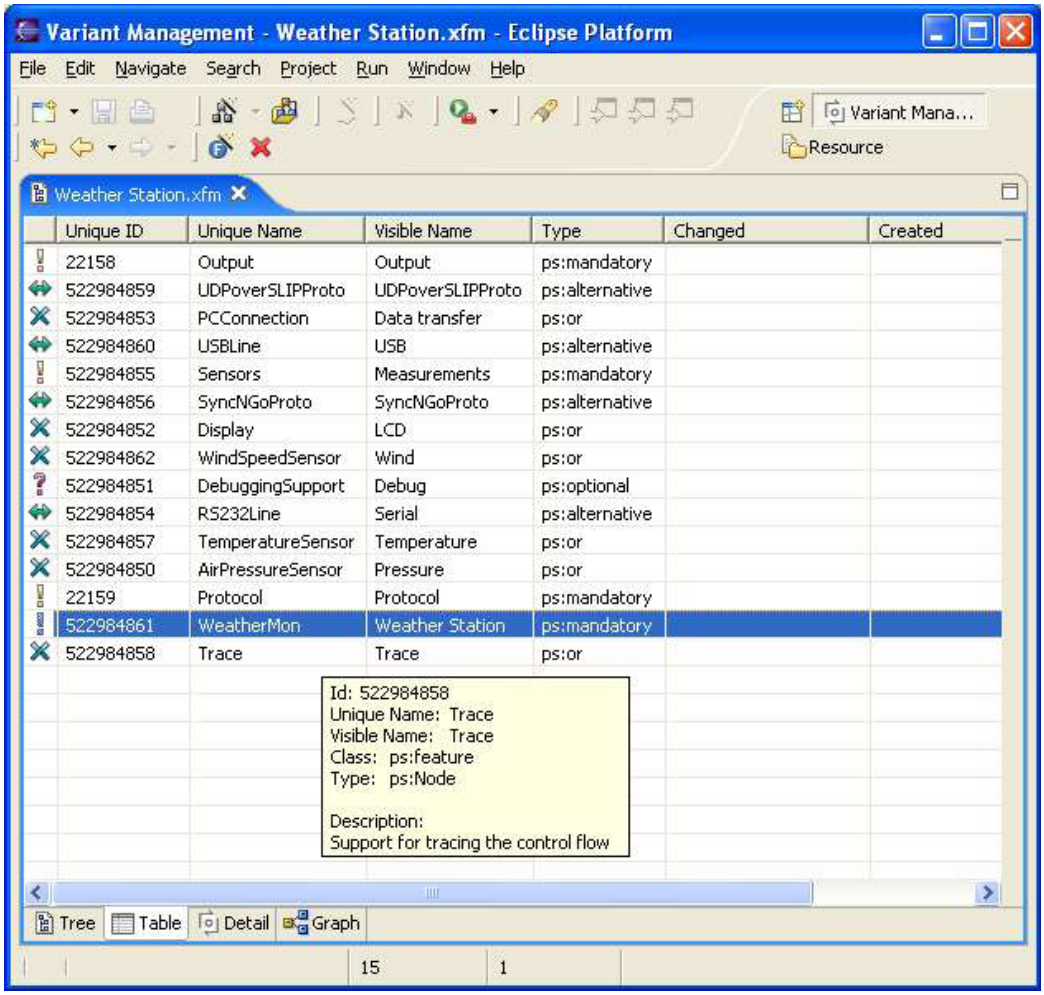

Figura 7. Interface de Edição *do Feature Model* em modo tabela do *pure::variants*

A Figura 7 simplesmente oferece uma forma diferente de visualização, em forma de tabela, onde se pode ver também o ID, o nome único, o nome visível e o tipo da *feature*.

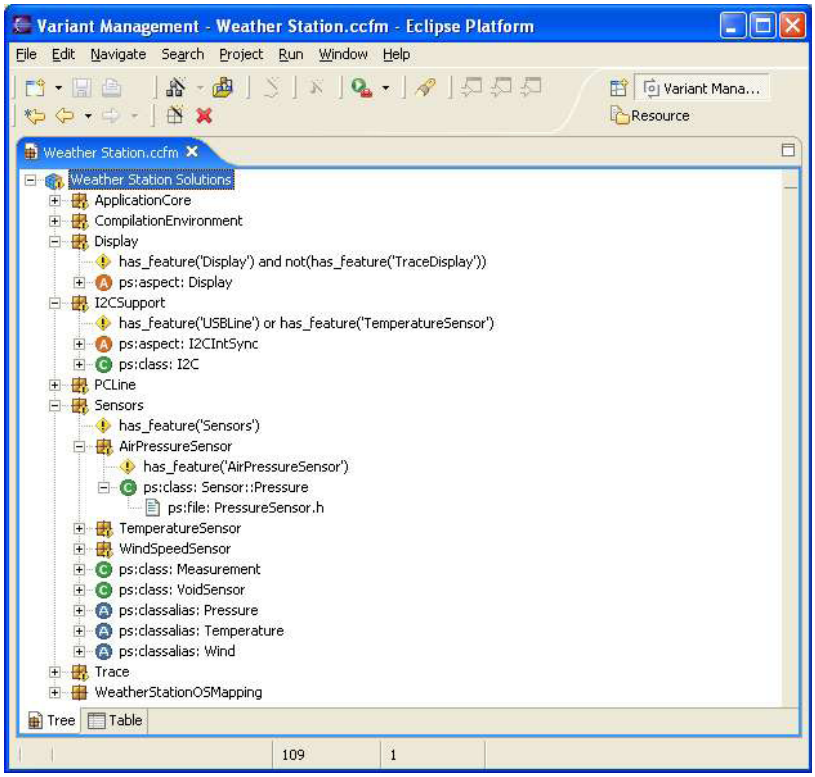

Figura 8. Interface de Edição do *Family Model* do *pure::variants*

A Figura 8 apresenta o *Family Model*, onde encontramos componentes (as caixas marrons), restrições (as expressões após a placa com exclamação), classes (as bolas verdes com um ‘C’), aspectos (as bolas laranjas com um ‘A’), arquivos (o papel dobrado na borda).

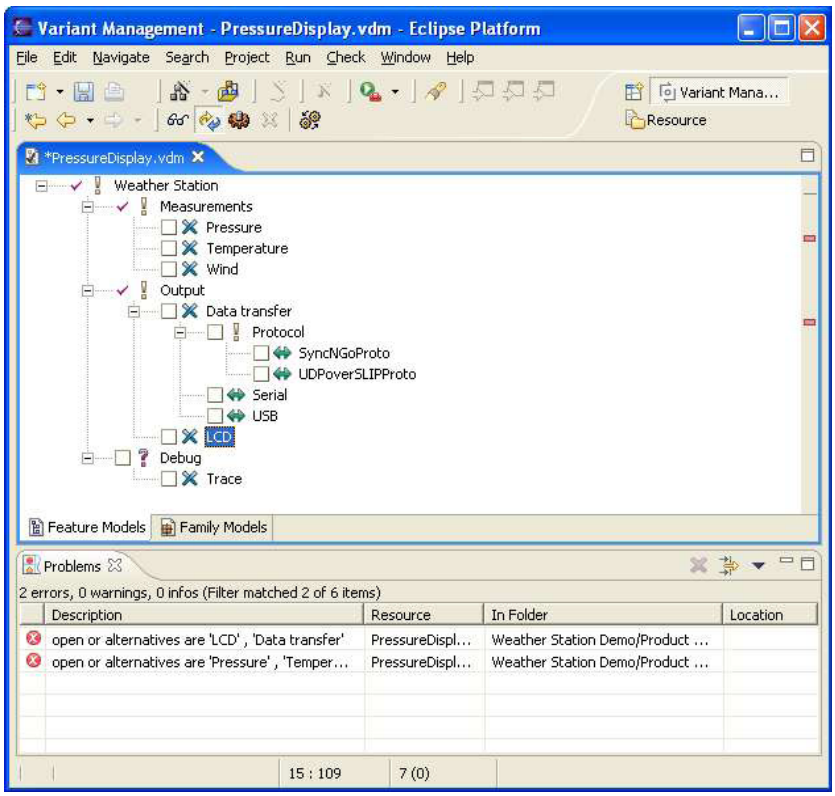

Figura 9. Interface de Edição de um *Variant Model* do *pure::variants*

A Figura 9 mostra a interface de seleção de *features* em uma instância do *feature model* no *pure::variants*, as *features* são selecionadas marcando-se a *checkbox* adjacente às mesmas.

O produto resultante de uma instância pode ser visualizado de forma similar ao *family model* utilizado como base. Todos os elementos que não entraram na instância são omitidos do *family model* durante esta visualização.

Além disso, o *pure::variants* provê um engenho de transformação baseado em XLST, que pode ser estendido pelo usuário, permitindo a customização dos arquivos que serão gerados quando uma instância for transformada.

3.3.5 Facilidade de uso

Similarmente ao *fmp*, o *pure::variants* é integrado ao Eclipse, o que facilita o seu uso por desenvolvedores que já utilizam o ambiente de desenvolvimento. Entretanto, o *pure::variants* possui uma quantidade maior de editores e *wizards* que o *fmp*, além de introduzir mais conceitos, o que pode dificultar o uso inicial da ferramenta.

Como a definição de instâncias do *feature model* encontra-se em arquivos separados, isso facilita a organização da estrutura de arquivos do projeto. O uso de prolog para a definição de

restrições poderia ser considerado uma péssima escolha. Entretanto, pvscl, a linguagem alternativa própria do *pure::variants* para definição de restrições é consideravelmente mais simples e intuitiva do que prolog. Usuários mais avançados terão todo o poder que prolog oferece, enquanto usuários comuns poderão usar pvscl sem dificuldades.

3.3.6 Estudo de caso

A divisão entre os modelos no estudo de caso desenvolvido no *pure::variants* foi bastante clara. O *feature model* especificou o domínio, porém sem a possibilidade de definir cardinalidades como utilizado no *fmp* e no *XFeature*, entretanto, o uso de *features* '*or*' e '*alternative*' puderam substituir os casos existentes no estudo de caso.

O *variant model* era atualizado automaticamente refletindo as mudanças do *feature model*, e, similarmente ao *fmp*, era apresentado em forma de árvore completa para a seleção de *features*. Uma vez feita a seleção de *features*, o usuário deve clicar na opção '*check model*' para verificar se a seleção de *features* respeita todas as restrições no modelo.

O *family model* não foi utilizado, uma vez que isso iria requerer o desenvolvimento de uma aplicação de lojas *on-line*, o que estava fora do escopo do projeto. Entretanto, ele poderia ter sido usado para a definição de componentes *Java* que só entrariam num produto quando devidas *features* fossem selecionadas. Após ajustar o sistema de transformação para gerar o que o usuário deseja, o processo de derivação é um simples clique no botão '*transform model*' de um *variant model*.

As Figuras 10 e 11 apresentam os modelos gerados durante o estudo de caso no *pure::variants*.

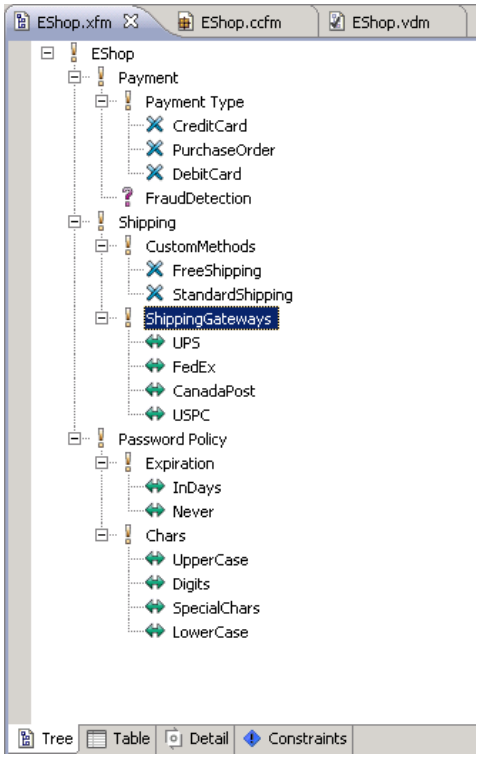

Figura 10. Feature model do *EShop* no *pure::variants*

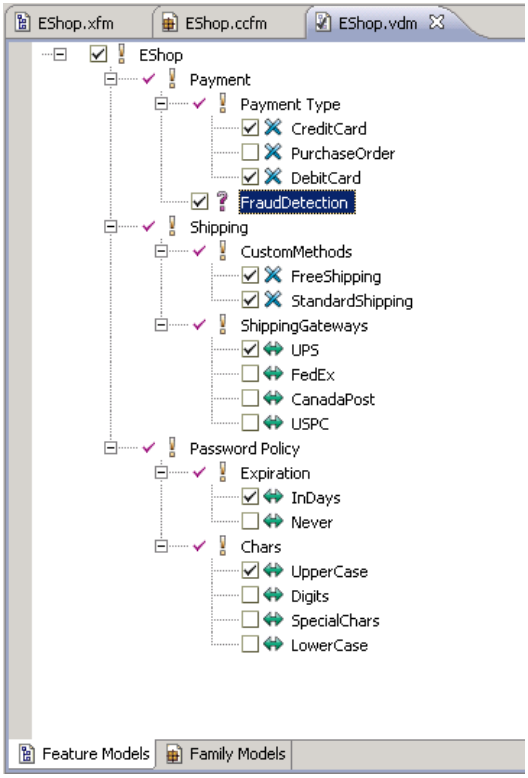

Figura 11. Instância do *EShop* no *pure::variants*

3.4 Gears

Ao contrário de todas as ferramentas analisadas até agora, a *Gears*, desenvolvida pela *BigLever Software Inc.*, não é um plugin para o Eclipse, e sim uma ferramenta completamente *stand-alone*. A empresa tem como objetivo o desenvolvimento de *softwares* para sistemas embarcados, que se baseia no desenvolvimento de componentes de *software* e de ferramentas de desenvolvimento de *software*.

A *Gears* é uma das ferramentas de *feature model* mais populares no mundo comercial, o que motivou a idéia de avaliá-la, porém não foi possível utilizá-la, pois os desenvolvedores não forneceram uma versão demo. Entretanto, tentou-se descobrir o máximo sobre a mesma para apresentar neste trabalho.

3.4.1 Background

Ao contrário das outras ferramentas analisadas, a *Gears* é essencialmente uma ferramenta com fundo comercial, em contraste às avaliadas anteriormente, que possuíam fundo acadêmico. De acordo com a *BigLever* a *Gears* é utilizada com sucesso pela *LSI Logic*, pela *Salion* e pela *HomeAway*, citando resultados impressionantes obtidos após o uso da ferramenta. O projeto continua sendo desenvolvido e suportado pela empresa atualmente.

3.4.2 Custo

O custo exato da Gears é desconhecido, porém, na seção de avaliação disponível no site, o pacote de avaliação "*getting started*" custa 5 mil dólares e inclui 3 dias de um projeto de desenvolvimento e consultoria piloto *onsite* (alguém da *BigLever* iria na empresa pessoalmente acompanhar desenvolvimento), meio dia de preparação e pós-processamento interativo *off-site* (na própria *BigLever*) para resumir os resultados e garantia de satisfação, se o cliente não estiver 100% satisfeito, o custo do pacote é reembolsado.

3.4.3 Técnica de modelagem e notação utilizada

As informações sobre a técnica de modelagem e notação utilizadas pela *Gears* são escassas. Sabe-se que ela utiliza uma metodologia de linhas de produtos chamada *3-Tiered Software Product Line Methodology*, que aparenta ser similar ao que o *pure::variants* oferece, que contém os seguintes níveis: *base tier* – gerenciamento de variações e geração de produtos, *middle tier* – desenvolvimento focado nos artefatos *core*, *top tier* – evolução do *portfolio* baseada em *features*.

A *base tier* pode ser considerada a parte de *variant models* e geração de produtos do *pure::variants*, enquanto *middle tier* seria o *family model* e o *top tier* seria o *feature model.*

A Figura 7 demonstra a interface da *Gears*.

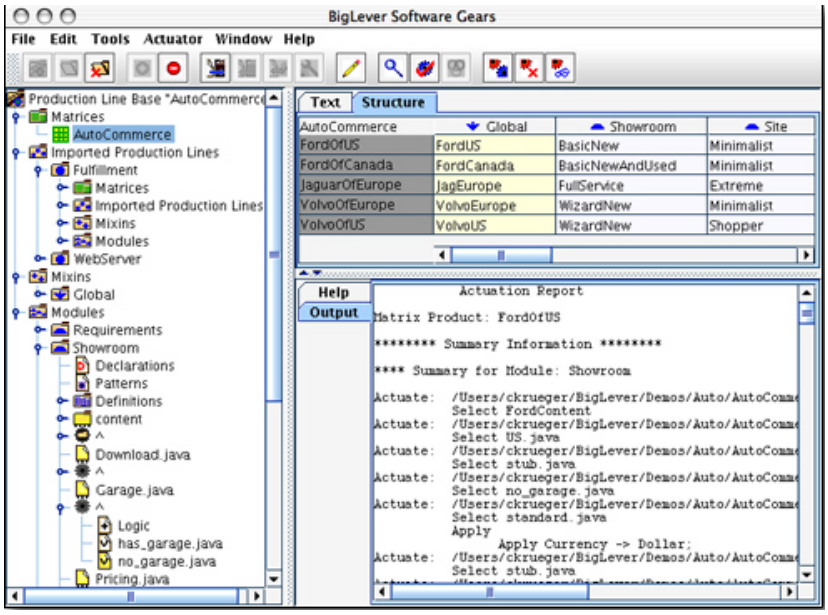

Figura 12. Interface da *Gears*

Capítulo 4

Comparação das Ferramentas

Visando facilitar a comparação entre as ferramentas, utilizando os critérios que foram analisados no capítulo anterior, assim como alguns recursos que as ferramentas possuam, ou não, este capítulo apresentará as tabelas comparativas entre as ferramentas.

4.1 Ferramentas de Linhas de Produtos de *Software*

A Tabela 1 apresenta uma comparação entre as ferramentas analisadas com os critérios que foram analisados a fundo no Capítulo 3. Ela resume todas as informações apresentadas nas seções do Capítulo 3 sobre cada ferramenta.

Tabela 1. Principais características das ferramentas

Critério	Ferramentas			
	Fmp	*XFeature*	*pure::variants*	*Gears*
Background	Acadêmico	Acadêmico	Industrial	Industrial
Suporte a processo	Não	Não	Sim	Sim
Notação utilizada	Árvore	Árvore e Diagrama	Árvore, Diagrama e Tabela	Desconhecido
Implementação	*Plugin*	*Plugin*	*Plugin*	*Stand-alone*
Tipo de licença	*Open Source*	*Open Source*	Gratuito e Comercial	Comercial
Código fonte	Disponível	Disponível	Não disponível	Não disponível
Técnicas de modelagem	Baseada em cardinalidade	Variável	Derivada de *FODA*	Desconhecido

A Tabela 1 apresenta critérios de alto nível, incluindo alguns que são mais do interesse da gerência do que dos desenvolvedores, tais como tipo de licença, *background* e suporte a processo (se suportam mais do que apenas a modelagem do domínio, se suportam a geração do produtos, documentação dos modelos, edição do código, etc), que afeta vários níveis do desenvolvimento de um projeto, não apenas a programação.

Já a Tabela 2 apresenta recursos disponíveis nas ferramentas que são mais pertinentes aos desenvolvedores que irão utilizá-las no dia a dia. Ela apresenta uma comparação entre alguns dos recursos disponíveis nas ferramentas analisadas. Alguns destes recursos foram mencionados no Capítulo 3, porém alguns que foram considerados importantes pelo autor também são apresentados, como controle de versão, se as *features* podem ser documentadas, se a ferramenta possui checagem de consistência dos modelos, etc.

Tabela 2. Recursos das ferramentas

Critério	Ferramentas		
	fmp	*XFeature*	*pure::variants*
Representação de dados em forma tabular	Não	Não	Sim
Checagem de consistência	Sim	Sim	Sim
Documentação das *features*	Sim	Sim	Sim
Controle de Versão	Não	Não	Sim
Possui *configuration knowledge*	Não	Não	Sim
Derivação de produtos	Não	Não	Sim
Restrições	XPath e linguagem própria	XSL	Prolog / pvSCL
Cardinalidade em relacionamentos	Sim	Sim	Não

Com esses dados em mão, espera-se que a tomada de decisão por parte dos desenvolvedores e da gerência sobre qual ferramenta utilizar para facilitar o desenvolvimento de uma Linha de Produtos de S*oftware* seja simplificada, podendo ser fundamenta em critérios que sejam importantes para a empresa, critérios estes que podem variar de uma empresa para outra.

Capítulo 5

Conclusão

A utilização do conceito de Linhas de Produto de *Software* tem gerado muitos casos de sucesso na indústria, e, definitivamente, é algo que deve ser considerado por empresas que desejam manter a competitividade no mercado em que trabalham.

Este trabalho apresentou uma análise comparativa entre 4(quatro) das principais ferramentas de apoio ao desenvolvimento baseado em Linha de Produtos de *Software*. Uma das ferramentas, a *Gears*, não pôde ser analisada a fundo, uma vez que não havia uma versão para testes disponível *on-line*. Este estudo demonstrou as principais características das ferramentas analisadas e apresentou os dados de forma resumida e concisa, permitindo que gerentes e desenvolvedores possam fazer uma escolha consciente sobre qual ferramenta devem utilizar na sua linha de produtos. Pôde-se perceber que se a empresa deseja mais do que apenas modelar o domínio, a escolha certa seria uma das ferramentas pagas, uma vez que as gratuitas não possuem o recurso de *configuration knowledge*.

Como trabalhos futuros, outras ferramentas existentes podem ser analisadas, assim como uma análise mais a fundo das que foram apresentadas neste trabalho pode ser realizada, como a utilização do recurso de *configuration knowledge* do *pure::variants*, além de também a realização de estudos empíricos do uso das ferramentas em um ambiente real de desenvolvimento de linhas de produtos.

Bibliografia

[1] CLEMENTS, Paul e Northrop, LINDA. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002

[2] ANTIEWICZ, M. e CZARNECKI, K.. *FeaturePlugin: Feature Modeling Plug-in for Eclipse*. In Eclipse '04: Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange, OOPSLA, Vancouver, British Columbia, Canada, Pages 67 - 72, ACM Press, 2004.

[3] PURE::VARIANTS, página web http://www.pure-systems.com/Variant_Management.49.0.html, acessado em 20 de abril de 2008.

[4] XFEATURE, página weeb http://www.pnp-software.com/XFeature/, acessado em 20 de abril de 2008.

[5] GEARS, página web http://www.biglever.com/, acessado em 20 de abril de 2008.

[6] KRUEGER, Charles. *Easing the transition to software mass customization*. In Proceedings of the 4th International Workshop on Software Product-Family Engineering. Spain, October 2001.

[7] KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., LOPES, C., LOINTGIER, J. e IRWIN, J.. Aspect-*oriented programming*. ECOOP, 1997.

[8] CZARNECKI, K.. *Overview of Generative Software Development*. In Proceedings of the European Commission and US National Science Foundation Strategic Research Workshop on Unconventional Programming Paradigms, September, 15–17, 2004, Mont Saint-Michel, France, 2004.

[9] CZARNECKI, K. e EISENECKER, U.. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[10] KANG, K., COHEN, S., HESS, J., NOWAK, W. e PETERSON, S.. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report CMU/SEI-90TR -21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.

[11] CZARNECKI, K., HELSEN, S. e EISENECKER, U.. *Formalizing cardinality-based feature models and their specialization*. Software Process Improvement and Practice, 10(1):7–29, jan/mar 2005.

[12] EMF – ECLIPSE MODELING FRAMEWORK, http://www.eclipse.org/emf/, acessado em 20 de abril de 2008.

[13] XSLT – EXTENSIBLE STYLESHEET TRANSFORMATIONS, página web http://www.w3.org/TR/xslt, acessado em 20 de abril de 2008.

[14] CIRILO, E., KULESZA, U. e LUCENA, C.. *GenArch: Uma Ferramenta baseada em Modelos para Derivação de Produtos de Software*. Anais da Sessão de Ferramentas do SBCARS 2007.

[15] CECHTICKY, V., PASETTI, A., ROHLIK, O. e SCHAUFELBERGER, W.. *XML-Based Feature Modelling*. In J. Bosch and C. Krueger, editors, Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004, Madrid, Spain, July 5--9, 2009. Proceedings, volume 3107 of Lecture Notes in Computer Science, pages 101--114. Springer-Verlag, 2004.

[16] BEUCHE, D., PAPJEWSKI, H. e SCHRODER-PREIKSCHAT W.. *Variability Management with Feature Models*. Proceedings of the Workshop on Software Variability Management, 2003: p. 72-83.

[17] GENERATIVE SOFTWARE DEVELOPMENT GROUP, página web http://gsd.waterloo.ca/, acessado em 20 de abril de 2008.

[18] XML PATH LANGUAGE (XPATH) 2.0, página web http://www.w3.org/TR/xpath20, acessado em 20 de abril de 2008.