2
Linhas de Produtos de Software

"Reunir-se é um começo, permanecer juntos é um progresso e trabalhar juntos é um sucesso." (Henry Ford)

Este capítulo apresenta os principais conceitos relacionados ao desenvolvimento e gerenciamento de Linhas de Produtos de Software (LPS). Na Seção 2.1 é dada uma visão geral sobre LPS. Posteriormente, é apresentada a abordagem de Linhas de Produtos de Sistemas Multi-agentes (LP-SMA) na Seção 2.2. Por fim, um breve resumo sobre o capítulo é apresentado na Seção 2.3.

2.1
Conceitos

Uma das atuais tendências que motivam o aumento da produtividade através da reutilização em Engenharia de Software é a produção de técnicas e ferramentas que viabilizem o desenvolvimento de famílias de sistemas, ao invés do desenvolvimento de aplicações individuais. Essa família de sistemas (Parnas 1976) pode ser vista como uma Linha de Produto de Software (LPS). Ao longo dos últimos anos, várias abordagens para o desenvolvimento de linhas de produtos foram propostas (Clements and Northrop 2002, Pohl et al. 2005, Weiss and Lai 1999, Czarnecki and Eisenecker 2000, Greenfield et al. 2004). Uma LPS (Clements and Northrop 2002, Pohl et al. 2005) determina uma família de sistemas que compartilha um conjunto de características comuns e gerenciadas e que satisfazem as necessidades de um segmento particular de mercado. Tais sistemas são desenvolvidos a partir de um conjunto de artefatos compartilhados, usando um método sistemático para construir as aplicações. As abordagens de LPS motivam o desenvolvimento de uma arquitetura flexível e reusável com o intuito de permitir a rápida e efetiva customização de diferentes aplicações pertencentes ao domínio determinado pela família de sistemas.

A abordagem de LPS é promissora no sentido de melhorar a produtividade do processo de desenvolvimento de software e proporcionar uma série de benefícios (Clements and Northrop 2002, Pohl et al. 2005, Kolb et al. 2005), tais como: (i) redução do custo e tempo de desenvolvimento - pois os pro-

dutos podem ser rapidamente derivados a partir da LPS e com isso podem ser entregues mais rapidamente, aumentando assim a produtividade; e (ii) garantia da qualidade - essa qualidade pode ser obtida através das técnicas de implementação de variabilidades utilizadas no desenvolvimento das arquiteturas LPS. Tais técnicas podem permitir uma melhor modularização das *features* diante dos cenários de evolução e reduzir o impacto das mudanças feitas na arquitetura da LPS.

Inicialmente, no desenvolvimento de uma família de produtos de software é necessário identificar as similaridades e as variabilidades (diferenças) entre os produtos em termos de requisitos funcionais ou não funcionais. Essas funcionalidades comuns e variáveis identificadas são chamadas de *features* e representam requisitos reusáveis e configuráveis. Uma *feature* (Czarnecki and Helsen 2006) é uma propriedade ou uma funcionalidade da família de sistemas que é relevante para algum dos interessados no seu desenvolvimento. Dentre as várias abordagens propostas para o desenvolvimento e a representação das variabilidades em uma LPS (Weiss and Lai 1999, Bayer et al. 1999, Gomaa 2004, Kang et al. 1990), o método FODA (*Feature-Oriented Domain Analysis*) proposto por (Kang et al. 1990, Kang et al. 2002), é um dos mais conhecidos e um dos precursores da abordagem de LPS. Este método foi desenvolvido pelo Instituto de Engenharia de Software (SEI). Esta abordagem destaca-se pelo modelo de *features*, que é amplamente utilizado nas abordagens de LPS. Um modelo de *features* (Kang et al. 1990, Kang et al. 2002) é utilizado para modelar as partes comuns e variáveis entre os produtos que compõem uma família de produtos. Segundo (Kang et al. 1990, Kang et al. 2002), as *features* podem ser classificadas em:

- **obrigatórias**: funcionalidades que devem estar em todos os produtos;
- **opcionais**: uma funcionalidade que pode ou não estar em um produto;
- **alternativas**: exatamente uma das funcionalidades deve estar no produto, ou seja, representam funcionalidades mutuamente exclusivas.

A Figura 2.1 apresenta a notação do modelo de *features* de acordo com o método FODA.

De acordo com (Krueger 2002a), três estratégias diferentes de adoção da abordagem de LPS podem ser usadas no seu desenvolvimento, que são: pró-ativa, extrativa e reativa. A primeira é a pró-ativa, cuja estratégia é o desenvolvimento de todos os artefatos da LPS, partindo da análise, projeto e implementação de domínio e indo posteriormente para a engenharia da aplicação. Além disso, esta abordagem considera que a LPS atenda o maior número de produtos existentes de um certo domínio. Na abordagem extrativa,

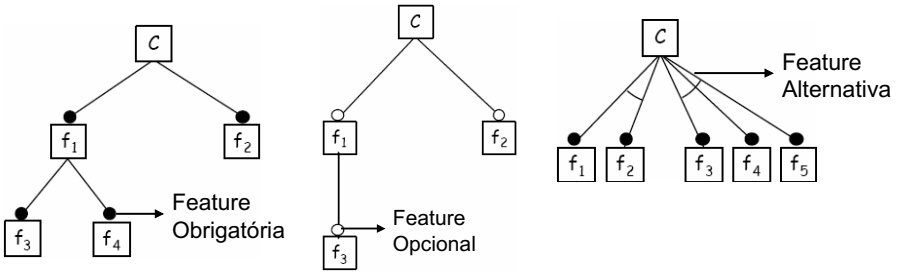

Figura 2.1: Notação FODA do modelo de features.

a LPS é desenvolvida a partir de um conjunto de softwares existentes, isto é, extraindo as características comuns e variáveis desse softwares. Por fim, na abordagem reativa, a LPS é desenvolvida incrementalmente. Assim, a LPS vai sendo ampliada quando existem demandas de novos requisitos ou produtos. De uma forma geral, um conjunto de produtos de uma LPS é construído visando um segmento de mercado ou domínio de aplicação, compartilhando uma arquitetura comum e utilizando componentes de uma base comum, conforme ilustrado na Figura 2.2.

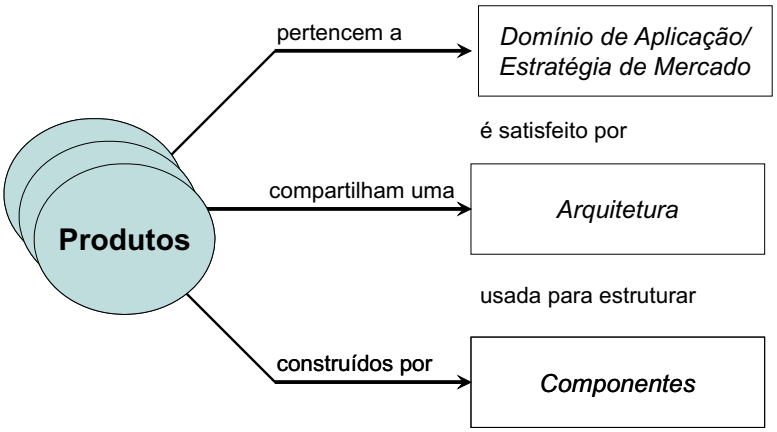

Figura 2.2: Linhas de Produtos de Software.

Segundo (Clements and Northrop 2002), as três atividades essenciais para se construir uma linha de produto (LP) são: desenvolvimento do núcleo de artefatos, também conhecida como engenharia de domínio, desenvolvimento do produto, também conhecida como engenharia da aplicação, e o gerenciamento da linha de produto, como apresentado na Figura 2.3. Essas atividades são descritas a seguir: (i) *Desenvolvimento do núcleo de artefatos* - nesta fase os artefatos e a arquitetura comum são desenvolvidos. Esse núcleo de artefatos consiste de componentes, *frameworks*, estratégia de produção, dentre outros artefatos. Basicamente, durante a atividade de desenvolvimento do núcleo de artefatos (Engenharia de Domínio) três atividades fazem parte

dessa fase: análise de domínio, projeto de domínio e implementação do domínio (Prieto-Diaz and Arango 1991). Ao final desta atividade um conjunto de artefatos podem ser reutilizados pelos produtos derivados a partir dessa LPS; (ii) *Desenvolvimento do produto* - nesta fase os produtos são construídos seguindo o plano de produção da LPS. O desenvolvimento de um produto de software é também chamado de derivação (Deelstra et al. 2005). Esta derivação ocorre a partir da seleção e configuração de um conjunto de artefatos. Durante esta atividade é possível existir a iteração com a atividade anterior, pois alguns requisitos do produto podem não ter sido especificados a priori, tendo assim que serem desenvolvidos; e (iii) *Gerenciamento da linha de produto* - esta atividade tem como objetivo garantir que todas as atividades sejam realizadas de forma coordenada e planejada. Nesta atividade também deve-se estabelecer como ocorrerá a evolução da linha de produto, ou seja, como as atividades de desenvolvimento do núcleo de artefatos e desenvolvimento do produto devem interagir.

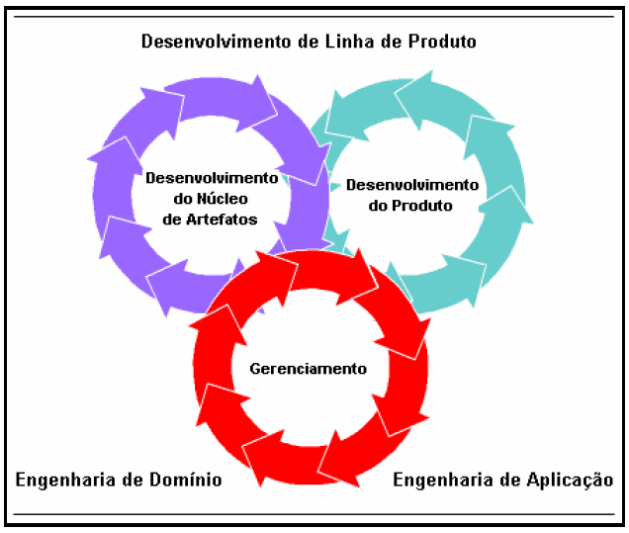

Figura 2.3: Atividades da Linha de Produto.

As arquiteturas de LPS são desenvolvidas com o foco principal em reutilização. Dessa forma, várias abordagens que promovem a reutilização no desenvolvimento de software são candidatas naturais para implementação das variabilidades encontradas em componentes de arquiteturas de LPS, tais como: o desenvolvimento baseado em componentes (Szyperski 2002), *frameworks* orientados a objetos (Fayad et al. 1999) e bibliotecas de software, padrões de projeto (Buschmann et al. 1996, Gamma et al. 1995) e arquivos de configuração. Além destas, a programação orientada a aspectos também vem sendo explorada como uma técnica utilizada na implementação das variabilidades de LPS (Capítulo 3.4).

2.2 Linhas de Produto de Sistemas Multi-agentes

Recentemente, o paradigma de Engenharia de Software Orientada a Agentes (ESOA) surgiu como uma abordagem promissora da Engenharia de Software para permitir a análise, projeto e desenvolvimento de sistemas complexos (Jennings 2001, Wooldridge and Ciancarini 2000). Um sistema complexo é freqüentemente caracterizado pela grande quantidade de pequenos softwares que interagem. Com os avanços das aplicações de software, a introdução de componentes autônomos nos sistemas de software tornou-se um requisito comum em diversas aplicações (Jennings 2001). Apesar de existirem várias definições para o termo agente na literatura, uma delas é a seguinte: "agente de software é uma entidade que está situada em um ambiente e capaz de ação autônoma neste ambiente para atingir seus objetivos projetados" (Wooldridge and Jennings 1995). Agentes de software são entidades autônomas que podem representar e tomar iniciativas pelos usuários de software (Briot 1998).

Dentre as propriedade de agentes de software, Wooldridge e Ciancarini em (Wooldridge and Ciancarini 2000) definiram um agente inteligente como uma entidade que possui as seguintes propriedades: (i) *autonomia*: os agentes atuam em um determinando ambiente sem a intervenção de qualquer entidade; (ii) *habilidade social*: os agentes interagem/comunicam com outros agentes; (iii) *reatividade*: os agentes reagem as mudanças ocorridas em um ambiente; e (iv) *pró-atividade*: os agentes podem atuar tanto em resposta ao ambiente quanto agir de forma autônoma, ou seja, de forma a tomar a iniciativa. Diante disso, a ESOA é considerada adequada para tratar certos tipos de sistemas (Jennings 2001).

Sendo assim, diante dos benefícios trazidos pela ESOA, algumas pesquisas recentes começaram a explorar a integração de Sistemas Multi-agentes (SMA) com LPS, caracterizando assim uma Linha de Produto de Sistemas Multi-agentes (LP-SMA) (Pena et al. 2006a, Pena et al. 2006c). O objetivo é unir os benefícios da ESOA com a LPS (Pena et al. 2006b). Essa área ainda é muito recente na literatura, não havendo tantas publicações nem mesmo uma definição formal do que venha a ser uma LP-SMA. Por isso, segundo a nossa visão e experiência em LP-SMA, nós a caracterizamos como uma arquitetura LPS, a qual faz uso de agentes de software para modelar, projetar e implementar *features* obrigatórias, opcionais e alternativas.

2.3 Resumo

Este capítulo foi dividido em duas partes. Na primeira parte, os principais conceitos relacionados a LPS foram descritos. Como apresentado, uma LPS define como aspecto central a implementação de uma arquitetura base, onde todos os produtos derivados desta LPS irão compartilhá-la. Além disso, essa arquitetura contempla as características similares e variáveis dos produtos de um determinado domínio da LPS. Para poder conseguir contemplar as características variáveis existentes nos produtos, um conjunto de componentes são disponibilizados no núcleo de artefatos com o intuito de permitir a construção/customização de cada um dos produtos da LPS. Na segunda parte do capítulo, foi apresentada a abordagem de LP-SMA. Essa abordagem utiliza o conceito de sistemas multi-agentes, principal abstração utilizada no contexto deste trabalho. Um agente de software é visto como uma entidade autônoma bastante adequada para o desenvolvimento de sistemas complexos, devido as suas propriedades (autonomia, pró-atividade, dentre outras).