1
Introdução

"Comece pelo começo e prossiga até chegar ao fim: então pare." (Lewis Carroll)

Linhas de Produtos de Software (LPS) são uma abordagem promissora para melhorar a produtividade do processo de desenvolvimento de software. Isto porque consegue reduzir o custo, o tempo de desenvolvimento e a manutenção de sistemas complexos (Kolb et al. 2005). Uma LPS pode ser vista como uma família de sistemas (Parnas 1976) que endereça um segmento de mercado específico. Ela objetiva explorar as partes comuns e variáveis de uma família de sistemas, de modo a permitir a criação de produtos similares a partir de um conjunto de artefatos compartilhados, usando um método sistemático para construir aplicações (Clements and Northrop 2002, Pohl et al. 2005). Ao longo dos últimos anos, várias abordagens para o desenvolvimento de LPS foram propostas (Clements and Northrop 2002, Pohl et al. 2005, Weiss and Lai 1999, Czarnecki and Eisenecker 2000, Greenfield et al. 2004). Essas abordagens visam definir uma arquitetura comum para todas as aplicações de um certo domínio/escopo. Tais arquiteturas compreendem um conjunto de *features* comuns e variáveis de uma família de produtos. Uma *feature* (Czarnecki and Eisenecker 2000) é uma propriedade ou requisito da LPS que é relevante para algum interessado no seu desenvolvimento, e que é usado para capturar elementos comuns ou diferenças entre produtos de uma LPS.

As arquiteturas de LPS são desenvolvidas com o foco principal em reutilização. Por isso, várias abordagens que primam por um certo nível de reutilização são utilizadas para a implementação de tais arquiteturas, tais como o desenvolvimento baseado em componentes (DBC) (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), arquivos de configuração, compilação condicional e programação orientada a aspectos (Kiczales et al. 1997). No processo de instanciação dos produtos de uma LPS, um conceito chave é o gerenciamento de variabilidade (Krueger 2002b), ou seja, as diferenças específicas entre os produtos. Neste contexto, é necessário modelar e implementar tais variabilidades de maneira a tornar fácil o seu gerenciamento.

Por outro lado, recentemente, a Engenharia de Software Orientada a Agentes (ESOA) surgiu como uma nova tendência da Engenharia de Software para permitir o desenvolvimento de aplicações distribuídas complexas (Jennings 2001, Wooldridge and Ciancarini 2000). Como parte dessa complexidade dos sistemas, a introdução de comportamento autônomo (agentes de software) tornou-se um requisito comum em muitas aplicações (Wooldridge and Ciancarini 2000, Garcia et al. 2003b). Agentes de software trazem vantagens únicas para o desenvolvimento de software, como suporte à (semi-)automação de tarefas, tais como: recomendação de produtos e informações aos usuários (Uchyigit and Ma 2008). Pesquisas recentes têm investigado a integração de Sistemas Multi-agentes (SMA) e LPS, caracterizando assim uma Linha de Produto de Sistemas Multi-agentes (LP-SMA) (Pena et al. 2006a, Pena et al. 2006c). Porém, ainda existem alguns desafios a serem enfrentados no contexto de LP-SMA (Pena et al. 2006b). Por exemplo, LPS para sistemas distribuídos, gerenciamento de sistemas evolutivos baseado em agentes e propriedades *auto-** de agentes. Uma LP-SMA agrega um conjunto de *features* de agentes (agentes de software) comuns e variáveis em uma família de produtos de SMA. Os pontos de variação associados às *features* de agentes estendem-se a suas capacidades funcionais: autonomia, reatividade, pró-atividade (Jennings 2001, Wooldridge and Ciancarini 2000).

1.1 Problemática

Similar ao desenvolvimento de sistemas individuais, as abordagens de LP-SMA também precisam lidar com a existência de cenários de evolução. Portanto, durante a evolução de uma LP-SMA é preciso considerar mecanismos de implementação apropriados para implementar uma determinada variabilidade. A ineficácia de mecanismos de implementação de variabilidades pode trazer algumas conseqüências indesejadas como: aumento da complexidade, dificuldades nas atividades de manutenção da arquitetura de LP-SMA e redução da estabilidade da arquitetura. De fato, a escolha e combinação de técnicas de implementação de variabilidades para implementação dos artefatos de uma arquitetura de LPS ainda é um desafio (Anastasopoulos and Muthig 2004). É desejável que a evolução de uma LPS seja feita com o mínimo impacto possível, de forma que os artefatos existentes sejam pouco alterados.

A evolução de uma LP-SMA ocorre através da inclusão, modificação ou remoção de *features* de agentes, as quais podem ser opcionais, alternativas ou obrigatórias. Essas *features* podem ser também transversais, causando um impacto em diversos pontos da LP-SMA, e tornando a sua inclusão muito cus-

tosa. Idealmente, as arquiteturas de LP-SMA precisam ser estáveis e flexíveis para suportar tais mudanças. Segundo (Kelly 2006), a definição de estabilidade refere-se a avaliação de características de projetos. Uma característica de projeto é estável quando, as diferenças observadas de uma característica são pequenas sobre duas ou mais versões do software. O projeto modular e estável das variabilidades em uma LP-SMA é particularmente desafiador, pois elas tendem a exercer um efeito transversal global sobre as outras e sobre as decomposições da arquitetura de agentes (Garcia and Lucena 2008). Assim, é importante utilizar técnicas de implementação que nos habilitem a configurar em módulos as variabilidades e interdependências associadas com as *features* de agentes. Caso contrário, a estabilidade de uma arquitetura de LP-SMA naturalmente irá degenerar com o tempo, sendo perpetuada através de todas as arquiteturas dos produtos de SMA gerados. A especificação modular de configurações possíveis da LP-SMA é essencial para evitar a degeneração da arquitetura a mudanças típicas.

Estudos recentes têm observado que a estabilidade do projeto é diretamente dependente de mecanismos de implementação (Figueiredo et al. 2008, Alves et al. 2006, Greenwood et al. 2007, Alves et al. 2005, Apel and Batory 2006, Kastner et al. 2007). Portanto, é importante prover mais evidências empíricas para demonstrar qual mecanismo de implementação, considerando contextos e domínios específicos, é mais apropriado para promover a estabilidade de arquiteturas de LP-SMA diante de cenários de evolução. Exemplos de tais mecanismos são: compilação condicional, injeção de dependência, arquivos de configuração, orientação a aspectos, agentes de software, geradores de código, frameworks OO.

Devido aos problemas dos mecanismos existentes de orientação a objetos (OO) para lidar com a modularização e composição de *features*, trabalhos recentes têm explorado o uso de técnicas de desenvolvimento de software orientadas a aspectos para facilitar a implementação de arquiteturas flexíveis e customizáveis. A orientação a aspectos (OA) é um paradigma que fornece mecanismos e abstrações para modularizar interesses transversais (*crosscutting concerns*). Esses interesses transversais são propriedades ou funcionalidades que se espalham por diversos módulos do sistema.

Algumas pesquisas recentes mostram os benefícios de usar técnicas de programação orientada a aspectos (POA) para melhorar a gerência e modularização de *features* em LPS (Figueiredo et al. 2008, Alves et al. 2006, Alves et al. 2005, Griss 2000), baseadas em frameworks (Kulesza et al. 2006a) e SMA (Garcia et al. 2003a). O aumento da complexidade de aplicações baseadas em agentes motiva o uso de POA, porque objetiva modularizar *features*

transversais. *Features* transversais produzem código replicado, espalhado e entrelaçado, e tendem a ocorrer freqüentemente no contexto de SMA e LPS. Esses tipos de problemas podem causar dificuldades para gerenciamento, manutenção e reuso de *features* em LPS. Assim, AOP tem sido proposta para permitir uma melhor modularização das *features*, a fim de melhorar a reusabilidade e a manutenção (Kiczales et al. 1997). Porém, como a abordagem de LP-SMA é recente, não existem evidências empíricas da avaliação de atributos de modularidade e estabilidade considerando tal abordagem. Além disso, não existem trabalhos que avaliem a estabilidade de projeto em LP-SMA, considerando a introdução de *features* de agentes em cenários de evolução.

1.2 Limitações dos Trabalhos Relacionados

Alguns trabalhos têm conduzido alguns estudos sobre a gerência de variabilidades em LPS usando aspectos e compilação condicional (Figueiredo et al. 2008, Alves et al. 2006, Alves et al. 2005, Griss 2000). Porém, poucos estudos empíricos relacionam a abordagem de LPS usando vários atributos de modularidade e estabilidade, com exceção do trabalho de (Figueiredo et al. 2008). Em (Apel and Batory 2006) é apresentado um estudo comparativo entre a programação orientada a *features* (FOP - *feature oriented programming*) e POA para implementação de *features* em uma LPS. (Kastner et al. 2007) apresenta um estudo de caso que refatora um sistema legado em uma LPS utilizando OA para implementar as *features*. No contexto de agentes, existem dois trabalhos (Garcia et al. 2003a, Lobato et al. 2008) que avaliam qualitativamente e quantitativamente algumas características específicas de agentes de software, tais como: mobilidade, autonomia, adaptação, colaboração, dentre outros.

Apesar de existirem alguns estudos que exploram o uso de POA e compilação condicional como técnicas de modularização de *features* no contexto de LPS e SMA, não há trabalhos que apresentem estudos empíricos com foco em LP-SMA, analisando o impacto da inclusão de *features* de agentes. Além disso, não existem estudos empíricos consistentes que avaliam a estabilidade do projeto de uma LP-SMA em cenários de evolução com o intuito de garantir a longevidade dessas arquiteturas. A maioria dos estudos existentes na literatura focam em interesses transversais como: persistência (Kulesza et al. 2006b, Soares et al. 2006), *exception handling* (Filho et al. 2006) e padrões de projeto (Hannemann and Kiczales 2002, Garcia et al. 2005). Também não existem trabalhos que avaliem diferentes implementações de agentes de software usando diferentes plataformas de SMA, tais como, JADE (JADE 2001) e Ja-

dex (Braubach and Pokahr 2002), e comparando diferentes técnicas de implementação, tais como, arquivos de configuração, compilação condicional e POA. Diante disso, é importante conduzir mais estudos empíricos para analisar as circunstâncias que uma determinada técnica de variabilidade é mais apropriada. Este conhecimento é essencial para suportar o desenvolvimento de LP-SMA estáveis, as quais são alvos de diferentes tipos de mudanças.

1.3 Solução Proposta

A gerência de variabilidades e a separação de interesses em LP-SMA são características importantes para permitir a redução da complexidade e garantir uma fácil evolução diante de cenários de mudanças. Tais características também permitem um melhor gerenciamento no processo de derivação/customização de produtos. Modularidade é um dos atributos de software que contribuem para analisar e melhorar a qualidade de sistemas.

Nesse contexto, este trabalho tem por objetivo realizar uma avaliação empírica para avaliar a modularidade e estabilidade de diferentes técnicas de implementação de *features* de agentes em LP-SMA. O objetivo é comparar diferentes implementações de *features* de agentes usando duas plataformas de SMA, JADE e Jadex, em combinação com outras técnicas vigentes de modularização de variações (compilação condicional, arquivos de configuração e orientação a aspectos) no desenvolvimento e evolução de LP-SMA. O objetivo é analisar como plataformas de SMA específicas e mecanismos de programação são benéficos ou prejudicam a modularidade e estabilidade de uma LP-SMA.

Neste contexto, duas LP-SMAs foram desenvolvidas com o intuito de avaliar qualitativamente e quantitativamente os pontos positivos e negativos das diferentes técnicas de implementações. As duas LP-SMAs foram desenvolvidas seguindo uma abordagem reativa (Krueger 2002a). Um conjunto de cenários de mudanças foram avaliados, e estes estão relacionados principalmente com a inclusão de diferentes tipos de *features* de agentes. Na primeira LP-SMA foi comparado o uso de duas técnicas de implementação utilizando a plataforma JADE: (i) técnicas de orientação a objetos (OO) combinada com compilação condicional; e (ii) técnicas de orientação a aspectos (OA) usando a linguagem AspectJ. Na segunda LP-SMA, foram analisadas três versões da mesma LP-SMA: (i) plataforma JADE com arquivos de configuração; (ii) plataforma Jadex com arquivos de configuração; e (iii) plataforma JADE enriquecido com técnicas de POA. Uma das diferenças chaves das implementações baseadas em Jadex é que ele é baseado no modelo BDI (*belief-desire-intentions*). Tais diferenças são importantes para

permitir a comparação de POA com mais de uma forma de implementar e estruturar *features* de agentes. O estudo é baseado em um conjunto de métricas já definidas e utilizadas anteriormente em outros estudos empíricos (Sant'Anna et al. 2007, Sant'Anna et al. 2003, Figueiredo et al. 2008) e métricas de impacto de mudanças (Yau and Collofello 1985). Os procedimentos dos estudos foram estruturados de acordo com os parâmetros propostos por (Wohlin et al. 2000).

1.4 Objetivos

Este trabalho tem como objetivo principal avaliar e discutir, através de métricas de software, o impacto do uso de três técnicas de implementação de variabilidade em LP-SMA: compilação condicional, arquivos de configuração e orientação a aspectos. Sendo assim, com um determinado conjunto de métricas poder-se-á avaliar a modularidade e estabilidade da LP-SMA. Como objetivos específicos, pode-se citar:

- Projetar e implementar duas LP-SMAs para o domínio *Web* utilizando as plataformas JADE e Jadex de desenvolvimento de SMA. Na implementação das LP-SMA, diferentes técnicas de implementação são utilizadas a fim de permitir uma melhor gerência de variabilidades dos produtos;
- Comparação sistemática qualitativa e quantitativa das diferentes versões das LP-SMAs desenvolvidas com base nas técnicas de implementação e nas plataformas utilizadas. Tal comparação é feita através de atributos de modularidade e estabilidade de suas respectivas técnicas de modularização, considerando cenários reais de evolução nessas LP-SMAs;
- Organizar estes estudos seguindo os princípios da Engenharia de Software Empírica e de modo a torná-los quasi-experimentos, seguindo as diretrizes estabelecidas por Wohlin et al. (Wohlin et al. 2000);
- Selecionar, aplicar e analisar um conjunto de métricas de modularidade e estabilidade para comparação das implementações das diferentes versões de uma mesma LP-SMA, no contexto de cenários de evolução;
- Relatar algumas lições aprendidas de como implementar *features* de agentes em LP-SMAs utilizando diferentes técnicas de implementação aliado as plataformas de desenvolvimento de SMA.

1.5 Estrutura da Dissertação

O restante deste trabalho está organizado da seguinte forma:

- **No Capítulo 2** é dada uma visão acerca dos conceitos de linhas de produtos de software e de linhas de produtos software de sistemas multi-agentes.
- **No Capítulo 3** são apresentados os conceitos relativos às técnicas de implementação de variabilidades utilizadas neste trabalho.
- **No Capítulo 4** são apresentados os conceitos sobre engenharia de software empírica, modelos de qualidade e as métricas utilizadas neste trabalho.
- **No Capítulo 5** são descritas a avaliação da modularidade e estabilidade das duas LP-SMAs desenvolvidas neste trabalho.
- **No Capítulo 6** são apresentadas as discussões e lições aprendidas a partir dos estudos empíricos realizados.
- **No Capítulo 7** são apresentados trabalhos relacionados ao desta dissertação, no que concerne sobretudo a estudos empíricos já realizados na área.
- **No Capítulo 8** são apresentadas as considerações finais, bem como possíveis trabalhos futuros.