Plano de Tese de Doutorado

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Linha de Produtos de Software para Aplicações Ubíquas usando Programação Orientada a Aspectos

- **Orientando:** Vander Alves
- **Orientador:** Paulo Borba
- **Área de concentração:** Engenharia de Software
- **Previsão de início:** Março de 2003
- **Previsão de término:** Dezembro de 2006

Recife, 5 de Março de 2004

Conteúdo

Resumo

A busca por produtividade, qualidade, e competitividade tem estimulado a adoção de processos de desenvolvimento de software nas organizações. Mais recentemente, processos baseados em linha de produtos de software têm focalizado o desenvolvimento não apenas de um produto, mas sim de uma família de produtos. Com isso, espera-se ganhos mais significativos em produtividade e qualidade.

No cerne de uma linha de produtos de software, está a atividade de instanciar uma arquitetura genérica para a arquitetura de um produto específico. Normalmente, utiliza-se o paradigma de orientação a objetos (OO) para tal fim; no entanto, este paradigma tem algumas limitações [20, 21], como o entrelaçamento de código com diferentes propósitos, aumentando a dificuldade de entendimento e, pois, diminuindo a manutenibilidade dos sistemas. Por outro lado, com adaptações do paradigma OO como a orientação a aspectos [15], novas técnicas de programação vem surgindo com o objetivo de corrigir as limitações do paradigma original.

O objetivo da tese aqui proposta é propor e avaliar um método de instanciação, customização, e configuração da arquitetura de uma linha de produtos de software para aplicações ubíquas, utilizando-se do paradigma de orientação a aspectos. Em particular, será investigado como esse paradigma possibilita a composição de componentes de forma mais flexível do que alcançado pela orientação a objetos.

1 Introdução

Os sistemas computacionais estão se tornando ubíquos. Através de um celular com poder computacional, atualmente podemos acessar e manipular informação praticamente em qualquer lugar e a qualquer instante. Assim como o celular, vários outros dispositivos eletrônicos passarão a ter poder computacional. De fato, teremos um aumento significativo no impacto das tecnologias da informação na sociedade. Consequentemente, neste cenário de computação ubíqua, os sistemas têm que atender a altos padrões de qualidade, principalmente com relação aos fatores de disponibilidade e amigabilidade.

Para obter disponibilidade e amigabilidade, os sistemas têm que ser adaptáveis, provendo tanto a atualização dinâmica de partes de sua funcionalidade quanto a instalação dinâmica de novas funcionalidades, normalmente sem intervenção do usuário e sem interrupção dos serviços oferecidos. Sem adaptabilidade, várias aplicações não poderiam ser executadas transparentemente em dispositivos computacionais que dispõem de poucos recursos e são inerentes à computação ubíqua. De fato, nestes dispositivos, as instalações e atualizações de software têm que ser pequenas e frequentes para que serviços adicionais sejam carregados e integrados sob demanda da aplicação, de acordo com as necessidades do usuário em momentos e contextos específicos. A adaptabilidade também é importante para ajudar a garantir a alta disponibilidade dos sistemas, pois atualizações dinâmicas, sem interrupção, são necessárias sempre que novas versões do código são disponibilizadas.

Disponibilidade, amigabilidade, e adaptabilidade contribuem para aumentar a já complexa tarefa de desenvolvimento dos sistemas atuais. De fato, esses sistemas devem satisfazer a uma série de requisitos funcionais e não–funcionais como persistência, concorrência, distribuição, e adaptabilidade; além disso, o processo de desenvolvimento deve ser produtivo, e espera-se que o software seja exetensível e reusável [4]. Para aplicações ubíquas, em particular, observa-se a demanda pelo desenvolvimento de famílias de aplicações destinadas a um determinado segmento de mercado. Adicionalmente, tais aplicações apresentam limitações de recursos como memória, o que dificulta ainda mais se alcançarem requisitos de qualidade sem que se sacrifique a performance dos sistemas.

A fim de superar o desafio do desenvolvimento das aplicações atuais, utilizam-se paradigmas como a orientação a objetos e processos de software. O paradigma de orientação a objetos é implementado por linguagens e se apóia em padrões de projeto e padrões arquiteturais [5, 8] . Como resultado, oferece meios efetivos para se alcançar ganhos com reuso, tornando futuros projetos mais produtivos, e com manutenção de software. No entanto, a orientação a objetos apresenta limitações [20, 21], como a dificuldade em modularizar requisitos sistêmicos, como os requistos não–funcionais. A fim de diminuí-las, novas extensões da orientação a objetos surgiram, dentre as quais se destaca a programação orientada a aspectos.

Processos de Software também surgiram para guiar o desenvolvimento dos sistemas. Estes processos definem as atividades, os artefatos resultantes, e os encarregados por sua execução. Com isso, ajudam a reduzir a complexidade do desenvolvimento, tornando-o mais previsível e reproduzível, e a aumentar a qualidade do produto resultante. Algumas limitações dos processos existentes são que não enfatizam a implementação [14]. Deste modo, preocupações como reuso e extensibilidade, alcançadas durante o projeto, podem ser perdidas na implementação, resultando na queda da qualidade final do software. Algumas extensões de processos focalizando a implementação ja estão sendo de-

finidas [2, 19, 24]. Mais recentemente, nota-se que os processos de software estão sendo generalizados em meta-processos denominados Linhas de Produtos de Software, que focalizam o desenvolvimento de uma família de produtos direcionados a um determinado segmento de mercado e fundamentados numa mesma base de artefatos.

Pretende-se nesta tese explorar a adequação da abordagem de Linha de Produtos de Software no contexto de aplicações ubíquas, e refinar esta aborgadem para esta classe relevante de aplicações, utilizando a técnica de programação orientada a aspectos. Em particular, aplicaremos esta técnica para instanciar a arquitetura de uma linha de produtos de software a fim de derivar produtos específicos, e para alcançar maior flexibilidade na composição de componentes do que na orientação a objetos.

2 Programação Orientada a Aspectos

Determinados requisitos de um sistema, como os requisitos não–funcionais (concorrência, persistência, distribuição, adaptabilidade, dentre outros), não são facilmente modularizados apenas com a orientação a objetos. De fato, a implementação desses requisitos encontra-se geralmente dispersa num conjunto de classes, envolvendo colaborações e protocolos elaborados entre as mesmas [15]. Consequentemente, há uma distância expressiva entre decisões de projeto e a sua implementação; além disso, o código resultante está propenso a ser mais complexo, menos modular e, pois, de qualidade inferior. Técnicas como padrões de projeto e padrões arquiteturais tentam minimizar este problema [5, 8], mas observa-se que se trata de uma limitação do paradigma.

A fim de se minimizar ainda mais essa limitação, surgiu recentemente a programação orientada a aspectos (AOP) [15]. Um aspecto é um módulo representando um requisito sistêmico, que potencialmente permeia toda a aplicação. O código deste requisito, escrito em AOP, é localizado e tem uma interface clara com o resto do sistema; torna-se, então, o requisito de projeto mais próximo de sua implementação, e o código resultante mais modular e, pois, de maior qualidade. Resultados recentes indicam a adequação da AOP na modularização de requisitos não–funcionais [25] e representação e implementação mais concisas de padrões de projeto [11].

Em [18] é apresentada e disponibilizada uma extensão orientada a aspectos da linguagem Java [9], a qual é fácil de aprender e usar e é compatível com a plataforma Java. Abordagens equivalentes à AOP também foram definidas [21, 22].

3 Linha de Produtos de Software

Observa-se, atualmente, uma crescente competitividade no mercado de produção de software. A fim de se manterem competitivas, várias organizações têm adotado algum processo de desenvolvimento de software. Tal processo visa direcionar os recursos das organizações eficientemente para a produção de software de qualidade.

Para alcançar tal objetivo, o processo de desenvolvimento de software estabelece uma seqüência de atividades a serem desempenhadas, determina quem deve realizá-las e quais os artefatos produzidos. Um processo recente que tem alcançado larga aceitação internacional é o Rational Unified Process (RUP), que é iterativo, centrado na arquitetura, e orientado a casos de uso [14]. O RUP e vários outros processos enfatizam

o desenvolvimento de um produto de software.

No entanto, nota-se que as organizações em geral produzem uma família de produtos de software, ao invés de apenas um produto. De fato, é comum empresas se estabelecerem visando um certo segmento de mercado e, a fim de conquistá-lo, desenvolverem uma família de produtos. Apesar de suas particularidades, tais produtos possuem similaridades, pois são potencialmente desenvolvidos a partir de um conjunto comum de artefatos. Dessa forma, é essencial que o processo de desenvolvimento contemple também a família de produtos.

Com o intuito de atender a esse contexto mais amplo, surgiu a área de Linha de Produtos de Software (LPS) [17]. Uma linha de produtos de software é uma família de produtos de software direcionados a um determinado segmento de mercado e fundamentados numa mesma base de artefatos. Três atividades centrais definem uma LPS: desenvolvimento de artefatos, desenvolvimento de produtos, e gerenciamento. O desenvolvimento de artefatos produz componentes a serem reusados pelos diversos produtos da LPS; um artefato essencial é a arquitetura da LPS, que também é usada como modelo para as arquiteturas específicas de cada produto. O desenvolvimento de produtos ocorre de forma similar aos processos de desenvolvimentos mencionados anteriormente. A diferença é que, numa LPS, há ênfase explícita no reuso dos artefatos desenvolvidos, e, em particular, da arquitetura da LPS. O gerenciamento garante a interação entre as duas atividades anteriores e a sua conformidade com os objetivos da organização.

4 Linha de Produtos de Software usando AOP

Como mencionado anteriormente, um artefato essencial em uma LPS é a arquitetura. A sua relevância é ainda maior do que em um processo de desenvolvimento convencional, pois, em uma LPS, a arquitetura é modelo para construção de vários produtos e não de apenas um. Por esse motivo, seu projeto merece cuidados especiais. Em particular, devem-se estudar os pontos em que este artefato deve sofrer variações e com isto acomodar os diversos produtos da linha. Esta tarefa é usualmente auxiliada por técnicas como parametrização, delegação, e herança.

Uma atividade essencial associada à variação da arquitetura básica de uma LPS é a integração ou composição de componentes. Componentes são incorporados à arquitetura para que a mesma seja instanciada para um produto específico. Este processo ocorre frequentemente, mas seu custo é amortizado entre os vários produtos, e seus benefícios incluem a validação da arquitetura e previsibilidade de atributos de qualidade dos produtos resultantes.

O uso da técnica de AOP na variação da arquitetura de uma LPS e na integração de componentes a esta arquitetura é tópico ainda não explorado [17] e potencialmente oferece benefícios como reversibilidade, localidade, modularidade, e automação, contribuindo para a eficiência da LPS e para a qualidade de seus produtos. Numa LPS de aplicações ubíquas, em particular, em que há restrições significativas de recursos como memória, acreditamos que a AOP venha garantir requisitos de qualidade como modularidade sem que se afete a performance, o que se observa ser mais difícil de se alcançar apenas com a orientação a objetos e padrões de projeto.

5 Objetivos

Propõem-se, nesta tese, alcançar os seguintes objetivos:

- estudar a técnica de AOP, comparando-a à programação orientada a objetos;
- definir método, padrões e frameworks que utilizem AOP para implementar variações da arquitetura de uma LPS no contexto de aplicações ubíquas;
- implementar ferramentas de apoio ao desenvolvimento;
- realizar estudos de caso e experimentos com sistemas reais, tentando identificar, quantificar e qualificar os benefícios e desvantagens do método aqui proposto em relação a um processo de desenvolvimento convencional.

6 Atividades

Nesta seção são indicadas as atividades a serem realizadas durante o doutorado para alcançar os objetivos descritos na seção anterior.

6.1 Estudo de Programação Orientada a Aspectos

Será realizado um estudo sobre Programação Orientada a Aspectos [15] e abordagens equivalentes [21, 27]. Conduziremos também experimentos com linguagens desse paradigma [22, 18]. Adicionalmente, com base em critérios a serem definidos, será realizada uma comparação da AOP com o paradigma OO.

6.2 Integração de AOP à LPS

A atividade de desenvolvimento de produtos de uma LPS será revisada e modificada para que a instanciação da arquitetura da LPS para um produto específico seja feita com auxílio de AOP. Isto incluirá o acréscimo tanto de requisitos não–funcionais (persistência, concorrência, distribuição, e adaptabilidade) como de requisitos funcionais à arquitetura básica. Será investigado também como a AOP pode auxiliar na integração de componentes a esta arquitetura básica.

6.3 Implementação de ferramentas de desenvolvimento

O uso de ferramentas que auxiliem o desenvolvimento de software é essencial, pois tem um impacto imediato em confiabilidade, produtividade, e competitividade. A disponibilidade de ferramentas se torna ainda mais importante no caso de uma LPS, já que esta se baseia de forma crítica no reuso de artefatos e na integração de componentes. Pretendemos implementar uma ferramenta que adiciona e remove os aspectos de distribuição, concorrência, persistência, e adaptabilidade à arquitetura básica da LPS. Proporemos também outra ferramenta que auxilie no acréscimo de requisitos funcionais a esta arquitetura. A infra-estrutura para o desenvolvimento dessas ferramentas já se encontra disponível [7].

6.4 Avaliação do método

Iremos realizar estudos de caso e experimentos aplicando o método definido nesta tese ao desenvolvimento de partes de aplicações ubíquas reais [23]. O estudo será feito junto à indústria local e utilizará a ferramenta de apoio. Dessa forma, poderemos identificar e comprovar as vantagens e desvantagens do nosso processo em relação à produtividade de equipes de desenvolvimento e aos seguintes fatores de qualidade de software: adaptabilidade, extensibilidade, e reusabilidade.

7 Cronograma

No cronograma a seguir são apresentadas as atividades a serem realizadas durante o doutorado no decorrer do tempo estimado para a conclusão do mesmo. Observe que os números **1** e **2**, colocados abaixo dos anos de 2003 até 2006, referem-se respectivamente, ao primeiro e segundo semestres de cada ano. O símbolo '•' indica que a tarefa referente a linha da célula marcada com o mesmo está em andamento.

Cronograma de março/2003 a dezembro/2006.

	2003		2004		2005		2006	
Tarefas	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
Cursar disciplinas	•	•						
Exame de Qualificação	•	•	•					
Proposta de Tese				•	•			
Estudo de Programação Orientada a Aspectos	•	•	•					
Integração de AOP à LPS		•	•	•				
Implementação de ferramentas de desenvolvimento					•	•	•	
Avaliação do método					•	•	•	
Finalização da escrita da tese							•	•

Assinaturas

Vander Ramos Alves
(Orientando)

Paulo Borba
(Orientador)

Referências

[1] Mendhekar A., Kiczales G., and et al. RG: A Case–Study for Aspect–Oriented Programming. Technical Report SPL97–009 P9710044, Xerox PARC, Palo Alto, CA, February 1997.

[2] Vander Alves and Paulo Borba. An Implementation Method for Distributed Object-Oriented Applications. In *XV Brazilian Symposium on Software Engineering*, pages 161–176, Rio de Janeiro, Brazil, October 2001.

[3] AOP. Aspect–Oriented Programming homepage. At http://www.parc.xerox.com/-aop.

[4] Grady Booch. The limits of software, 2002. Software Development Forum, PARC,Palo Alto,CA.

[5] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.

[6] Consel C. Program Adaptation based on Program Transformation. *ACM Workshop on Strategic Directions in Computing Research*, 1996.

[7] Fernando Castor and Paulo Borba. A language for specifying Java transformations. In *V Brazilian Symposium on Programming Languages*, pages 236–251, Curitiba, Brazil, 23th–25th May 2001.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[9] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, second edition, 2000.

[10] Demeter Research Group. Online material on adaptive programming, Demeter/Java, and APPCs. At http://www.ccs.neu.edu/research/demeter/, 1998.

[11] Jan Hannemann and Gregor Kiczales. Design pattern implementation in Java and AspectJ. In *Proceedings of OOPSLA'02, Object Oriented Programming Systems*. ACM Press, November 2002. To appear.

[12] William Harrison and Harold Ossher. Subject–Oriented Programming (a critique of pure objects). In *OOPSLA'93*, pages 411–428. ACM Press, September 1993.

[13] Lieberherr K. J., Silva-Lepe I., and et al. Adaptive Object–Oriented Programming Using Graph–Based Customization. *Communications of the ACM*, 37(5):94–101, 1994.

[14] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[15] Gregor Kiczales. Aspect–oriented programming. *ACM Computing*, 28(154), December 1996.

[16] Karl Lieberherr and Doug Orleans. Preventive program maintenance in Demeter/Java. In *International Conference on Software Engineering*, pages 604–605, Boston, MA, 1997.

[17] Linda Northrop. A Framework for Software Product Lines, 2001. Disponível em http://www.sei.cmu.edu/plp/framework.html.

[18] Cristina Lopes and Gregor Kiczales. Recent developments in AspectJ. *Workshop on Aspect–Oriented Programming at ECOOP'98*, July 1998.

[19] Tiago Massoni. Um Processo de Software com Suporte para Implementação Progressiva. Master's thesis, Centro de Informática – Universidade Federal de Pernambuco, Fevereiro 2001.

[20] H. Ossher, M. Kaplan, A. Katz, W. Harrison, and V. Kruskal. Specifying suject–oriented composition. *TAPOS*, 2(3):179–202, 1996. Special Issue on Subjectivity in OO Systems.

[21] Harold Ossher and Peri Tarr. Using subject–oriented programming to overcome common problems in object–oriented software development/evolution. In *International Conference on Software Engineering, ICSE'99*, pages 698–688. ACM, 1999.

[22] Harold Ossher and Peri Tarr. Hyper/J: multi–dimensional separation of concerns for Java. In *22nd International Conference on Software Engineering*, pages 734–737. ACM, 2000.

[23] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. Empirical studies of software engineering: a roadmap. In *ICSE - Future of SE Track*, pages 345–355, 2000.

[24] Sérgio Soares. Desenvolvimento Progressivo de Programas Concorrentes Orientados a Objetos. Master's thesis, Centro de Informática – Universidade Federal de Pernambuco, Fevereiro 2001. Disponível em http://www.cin.ufpe.br/∼scbs/syscoop.

[25] Sérgio Soares, Eduardo Laureano, and Paulo Borba. Implementing Distribution and Persistence Aspects with AspectJ. In *Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications, OOPSLA'02*, pages 174–190. ACM Press, November 2002. ACM SIGPLAN Notices 37(11).

[26] SOP. Subject–oriented programming homepage. At http://www.research.ibm.-com/sop.

[27] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton. N degrees of separation:multi–dimensional separation of concerns. In *1999 International Conference on Software Engineering*, pages 107–119. ACM, 1999.