GenArch: Uma Ferramenta baseada em Modelos para Derivação de Produtos de Software

Elder Cirilo[1], Uirá Kulesza[1], Carlos José Pereira de Lucena[1]

[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO) - Rio de Janeiro – RJ – Brazil

{ecirilo,uira,lucena}@inf.puc-rio.br

Resumo. *Este artigo apresenta uma ferramenta baseada em modelos para derivação de produto, denominada GenArch. A ferramenta é centrada na definição de três modelos – característica, arquitetura e configuração, os quais permitem a instanciação automática de linhas de produto de software ou frameworks. A ferramenta foi implementada na forma de um plug-in da plataforma Eclipse e utiliza a tecnologia EMF para manipulação de seus modelos. Anotações Java foram também definidas para permitir a geração de versões iniciais dos modelos a partir do código de implementação da arquitetura da linha de produto.*

1. Introdução

Pesquisas atuais e experiências práticas [3, 6, 12] sugerem que para se obter um progresso significativo com respeito ao reuso de software é necessário focalizar a modelagem e desenvolvimento de linhas de produtos ao invés de se desenvolver sistemas individualmente. A engenharia de linha de produtos procura explorar os pontos comuns e variáveis existentes entre sistemas de uma maneira sistemática. Através de uma linha de produtos, diferentes customizações de aplicações de um mesmo domínio podem ser rapidamente criadas a partir de um conjunto de artefatos reusáveis (arquitetura comum, componentes, framework, modelos, etc).

De acordo com Krueger [7], a adoção de uma abordagem de desenvolvimento de linha de produtos é mais difícil que simplesmente construir um sistema. Em parte porque durante várias décadas foram desenvolvidas métodos, técnicas e ferramentas centrados principalmente na construção de sistemas de software únicos, havendo poucas ferramentas para o gerenciamento e manipulação de variações de linha de produtos de software. Recentemente várias pesquisas e experiências industriais tem motivado o desenvolvimento de abordagens que lidam explicitamente com o gerenciamento de variabilidades de linhas de produtos. Algumas ferramentas para gerência de variabilidades e derivação automática de produtos de software têm sido desenvolvidas, tais como, Gears e pure::variants. Essas ferramentas apresentam funcionalidades úteis para a derivação de produtos de software, mas são, em geral, complexas e pesadas para serem usadas pela comunidade de desenvolvedores com pouca familiaridade com vários dos novos conceitos propostas pela comunidade de linhas de produto de software.

Nesse contexto, esse artigo apresenta a ferramenta GenArch que busca auxiliar engenheiros de software nas atividades de adoção e derivação de linhas de produto de software. A derivação automática de produtos na ferramenta é realizada através da definição de três modelos: característica, configuração e arquitetura. Versões iniciais desses modelos são geradas automaticamente pela ferramenta a partir da anotação do código de implementação da arquitetura de linha de produto.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta uma visão geral da ferramenta, apresentando seu funcionamento geral e sua arquitetura baseada em tecnologias Eclipse. A Seção 3 descreve brevemente a preparação do framework JUnit para ser instanciado automaticamente pela GenArch. Finalmente, a seção 4 apresenta as conclusões do trabalho e direções para pesquisas futuras.

2. GenArch – Generative Architecture Plugin

GenArch é uma ferramenta baseada em modelos que visa simplificar a derivação automática [13] de novos membros de linha de produtos de software (LPS) ou instâncias de frameworks OO. A ferramenta é baseada nos conceitos de Programação Generativa (PG) [4]. PG contempla métodos e técnicas que permitem a geração automática de membros de uma linha de produtos a partir de especificações de alto nível (ex: modelos). Ela motiva a separação do espaço de problema do espaço de solução. O espaço de problema representa conceitos, abstrações e características (features) existentes no domínio de uma linha de produto. O espaço de solução agrega a arquitetura de software e respectivos componentes de implementação que são usados na geração de um membro da família de sistemas. O mapeamento entre os espaços de problema e solução é definido em programação generativa, através do conhecimento de configuração. O conhecimento de configuração define como combinações específicas de características no espaço de problema são mapeadas para uma configuração específica de componentes no espaço de solução.

A ferramenta GenArch propõe a definição de três modelos os quais são usados para representar as variabilidades e elementos de implementação de uma LPS, são eles: (i) modelo de arquitetura; (ii) modelo de característica; e (iii) modelo de configuração. Cada modelo representa: o espaço de solução, espaço de problema e conhecimento de configuração da organização proposta por Programação Generativa. A seguir cada modelo é brevemente descrito.

O modelo de arquitetura (espaço de solução) define uma representação visual dos elementos de implementação da LPS, tais como, classes, aspectos, templates e arquivos de configuração ou figuras. Tais elementos são organizados em diferentes componentes que definem a arquitetura do sistema. O GenArch permite a geração automática da versão inicial do modelo de arquitetura através do *parsing* dos diretórios contendo os elementos de implementação da arquitetura de LPS. O modelo de arquitetura é criado de forma a permitir relacionar os elementos de implementação com as variabilidades representadas no modelo de característica.

O modelo de característica (espaço de problema) é usado no GenArch para representar as variabilidades da LPS. A ferramenta adota o modelo de característica proposto por Czarnecki et al [4], o qual permite modelar características obrigatórias, opcionais e alternativas, com suas respectivas cardinalidades, restrições (*constraints*) e atributos. O plugin FMP (*Feature Modeling Plugin*) [1] é usado para especificar o modelo de característica.

Finalmente, o modelo de configuração (conhecimento de configuração) é responsável por definir o mapeamento entre características e elementos de implementação. Um conjunto de relações de dependência entre as características (espaço de problema) e os elementos de implementação (espaço de solução) são definidas, de forma a decidir quais elementos de implementação devem ser instanciados

a partir do fornecimento de uma instância do modelo de característica (um produto), durante o processo de derivação de produto suportado automaticamente pelo GenArch.

2.1. Visão Geral de Funcionamento

A ferramenta GenArch é baseada em modelos, sendo fundamental sua especificação para permitir a derivação automática da linha de produto. Versões iniciais dos modelos são geradas automaticamente a partir do *parsing* do diretório contendo elementos de implementação de um projeto Java. Alguns desses elementos podem também conter anotações GenArch específicas que são consideradas na geração dos modelos.

A Figura 1 apresenta uma visão geral do funcionamento da ferramenta GenArch, assim como ilustra as tecnologias usadas no seu desenvolvimento. Inicialmente ocorre a preparação do código dos elementos de implementação para permitir a geração inicial dos modelos (passo 1). Nessa etapa o engenheiro de domínio, usa um conjunto explícito de anotações para indicar no código dos elementos de implementação da linha de produto: (i) quais elementos corresponde à implementação específica das características variáveis (variabilidades) da linha de produto (anotação do tipo `@Feature`); e (ii) quais elementos correspondem a pontos de extensão (*hotspots*) da linha de produto (anotação do tipo `@Variability`).

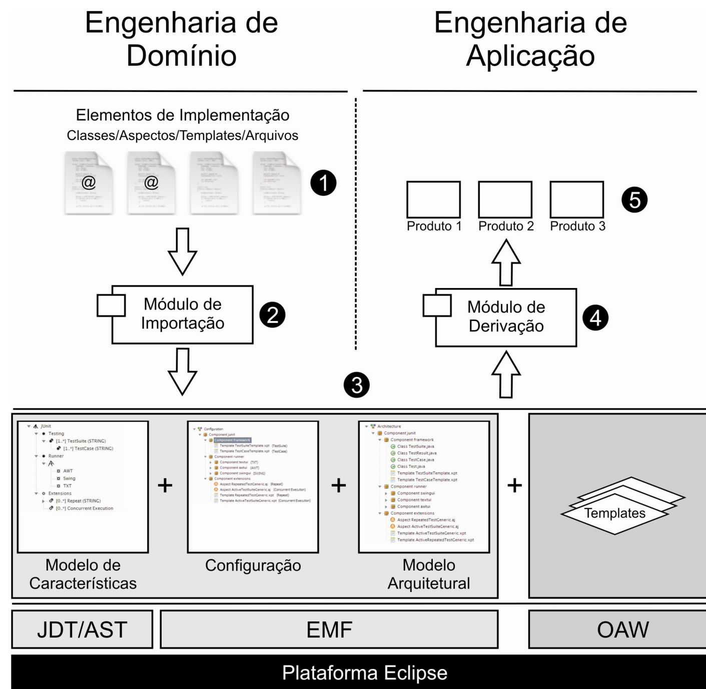


Figura 1. Arquitetura da ferramenta GenArch

Em seguida, o módulo de importação da ferramenta GenArch pode realizar um *parsing* no diretório contendo os elementos de implementação com suas respectivas anotações e gera uma versão inicial do modelo (passo 2) de arquitetura, de característica, de configuração e dos templates. A geração dos modelos de característica e configuração é baseada nas anotações (criadas no passo 1) que indicam as implementações específicas de características. Os templates são criados a partir das anotações que indicam quais são os pontos de extensão da linha de produto. O módulo

de importação pode ser acionado a qualquer momento. No início da iteração quando se deseja importar os modelos a partir de código existente ou em um momento qualquer, por exemplo, quando novos requisitos foram implementados na linha de produtos.

Após essa geração inicial, o engenheiro de domínio deve refinar e incrementar as versões iniciais dos modelos gerados (passo 3). As seguintes modificações são tipicamente realizadas: (i) adição de novas características ou reorganização das existentes; (ii) adição de novas relações de dependência entre características e elementos de implementação no modelo de configuração, sobretudo aquelas que não foram possíveis de serem especificadas através de anotações; e (iii) refinamento da implementação dos templates para contemplar a customização de alguma variabilidade a partir de informação coletada pelo modelo de característica.

Após o refinamento de cada um dos modelos, o módulo de derivação da ferramenta GenArch permite a instanciação automática de membros da linha de produto (passo 4). Essa derivação demanda inicialmente a especificação de uma instância do modelo de característica que representa aquele produto. Ou seja, o engenheiro de aplicação realiza a resolução de características, escolhendo um conjunto específico de elementos de implementação que satisfaça o produto que ele deseja. A ferramenta GenArch usa essa instância do modelo de característica, o modelo de configuração e o modelo de arquitetura para então gerar o produto desejado pelo usuário.

O módulo de derivação varre todo o modelo de arquitetura e para cada elemento ele verifica se existe uma relação de dependência no modelo de configuração entre tal elemento e alguma característica. Caso exista tal relação de dependência (ou seja, o elemento só deve ser instanciado caso uma dada característica tenha sido selecionada), a instância do modelo de características é consultada e dependendo da configuração da característica o elemento é adicionado ao produto. Se não existir uma relação de dependência o elemento de implementação será adicionado ao produto, representando nesse caso uma característica obrigatória. Esse processo de geração demanda a criação de um projeto Java/Eclipse (passo 5) contendo todos os elementos de implementação que implementam as características selecionadas, assim como o processamento de templates para geração de classes e/ou aspectos específicos que representam instâncias concretas dos pontos de extensão.

2.2. Arquitetura da Ferramenta

A Figura 1 também apresenta uma visão geral da arquitetura de implementação do GenArch. A ferramenta foi implementada como um plug-in da plataforma Eclipse utilizando vários *kits* de desenvolvimento orientado a modelos disponíveis em tal plataforma. Todos os modelos do GenArch foram construídos e são manipulados usando os recursos disponilizados pelo *Eclipse Modeling Framework* (EMF) [2]. EMF é um framework Java/XML que permite a construção de ferramentas orientadas a modelos. Ele possibilita a geração de classes Java para criação e edição de modelos baseadas num dado meta-modelo especificado em XML Schema ou usando diagramas de classe UML. O modelo de característica usado na ferramenta GenArch é especificado pelo plug-in FMP (*Feature Modelling Plug-in*). Os modelos criados no FMP são também modelos baseados no EMF.

A ferramenta GenArch utiliza o plug-in openArchitectureWare (oAW) [11] para especificar seus templates. A linguagem XPand que faz parte do oAW é usada com tal finalidade. XPand é uma linguagem simple e expressiva. Permitindo a especificação de

templates que podem ser customizados usando informações provenientes do modelo de característica.

3. Exemplo Ilustrativo: Framework JUnit

Nesta seção utilizamos o framework JUnit para ilustrar como a ferramenta Genarch pode ser utilizada: (i) na preparação da linha de produtos a partir de código já existente; e (ii) para derivar membros dessa linha de produtos. O propósito principal do framework JUnit é permitir o projeto, implementação e execução de testes de unidade em aplicações Java. A Figura 2 apresenta os principais elementos da arquitetura do JUnit. As suas principais funcionalidades são: a definição de casos de testes ou suítes para serem executadas (`TestCase` e `TestSuite`); a execução de casos de testes ou suítes (`BaseTestRunner` e `TestRunner`); e coleta (`TestResult`) e apresentação visual dos resultados. Além disso, diferentes extensões podem ser implementadas para adicionar novas funcionalidades dentro do núcleo do framework JUnit. Alguns exemplos de extensões simples e que foram implementadas são: (i) permitir ao JUnit executar suítes de teste em threads separadas (`ActiveTestSuite`) e esperar pelo término de todos os teste. Na implementação dessa extensão é necessário observar os eventos que ocorrem quando um caso de teste inicia sua execução, o evento que ocorre quando cada método de teste roda e o evento que ocorre quando o caso de teste pára de rodar; (ii) e permitir o JUnit executar cada método de teste repetidamente (`RepeatedTestGeneric`). Na implementação dessa extensão é necessário observar o inicio e o término da execução de cada método de teste. Nosso estudo de caso, utilizou uma versão do JUnit que considerou a implementação dessas extensões usando programação orientada a aspectos. Detalhes adicionais sobre tal estudo de caso podem ser encontrados em [10].

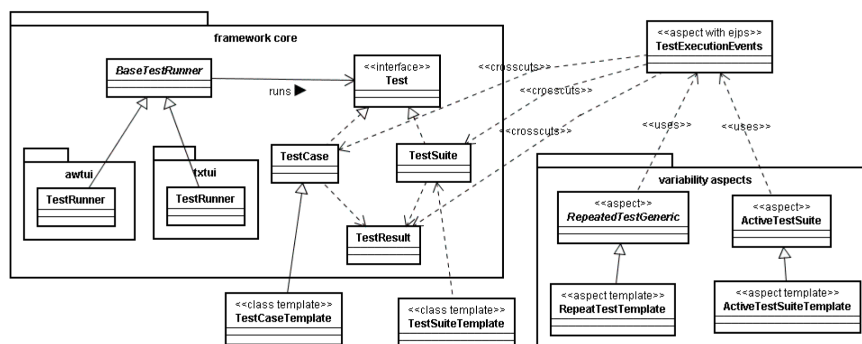


Figura 2. Arquitetura Orientada a Aspectos do JUnit

3.1. Criando Anotações em Classes e Aspectos

O primeiro passo do nosso estudo de caso foi criar anotações GenArch, indicando explicitamente no código-fonte do framework JUnit, relacionamentos entre características e elementos de implementação, assim como suas variabilidades. No JUnit, as classes `TestCase` e `TestSuite` além de serem pontos de extensão (*hotspots*) também estão relacionadas com as características Test Case e Test Suite. Dessa forma, estas classes foram anotadas com `@Feature` para descrever o mapeamento e `@Variability` para indicar a criação de um template para cada classe. O código anotado pode ser visto na Figura 3. Além dessas classes, os aspectos

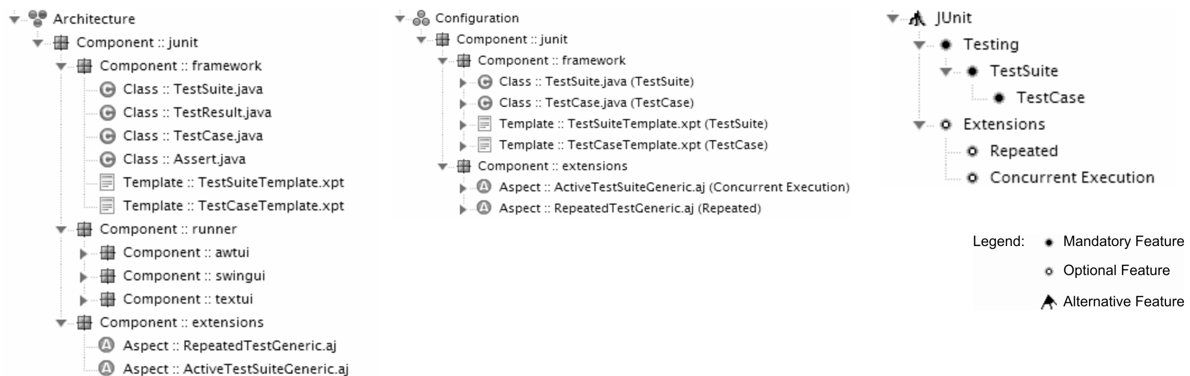
RepeatedTestGeneric e ActiveTestGeneric também foram anotados com @Feature e @Annotation, pois representam aspectos abstratos que são pontos de extensão orientados a aspectos do framework JUnit.

```
@Feature(name="TestCase",parent="TestSuite",type=FeatureType.mandatory)
@Variability(type=VariabilityType.hotSpot,feature="TestCase")

public abstract class TestCase extends Assert implements Test {

 private String fName;
 public TestCase() {
  fName= null;

 }
 public TestCase(String name) {
  fName= name;
 }
 ...
}
```

Figura 3. Código parcial da classe TestCase anotada.

3.2. Gerando e Atualizando os Modelos

O passo seguinte do estudo de caso foi adicionar ao projeto Java/Eclipse em uso a natureza de um projeto GenArch. Essa operação é feita através da opção "*Convert to Genarch project*". Antes do início da importação o plug-in solicita ao engenheiro de domínio que especifique os *source folders* que serão explorados. O módulo de importação é então acionado e a partir do conteúdo dos *source folders* e das anotações descritas no código a ferramenta gera uma versão inicial de cada um dos modelos e dos templates[1]. As versões iniciais dos modelos gerados para o JUnit são apresentadas na Figura 4.



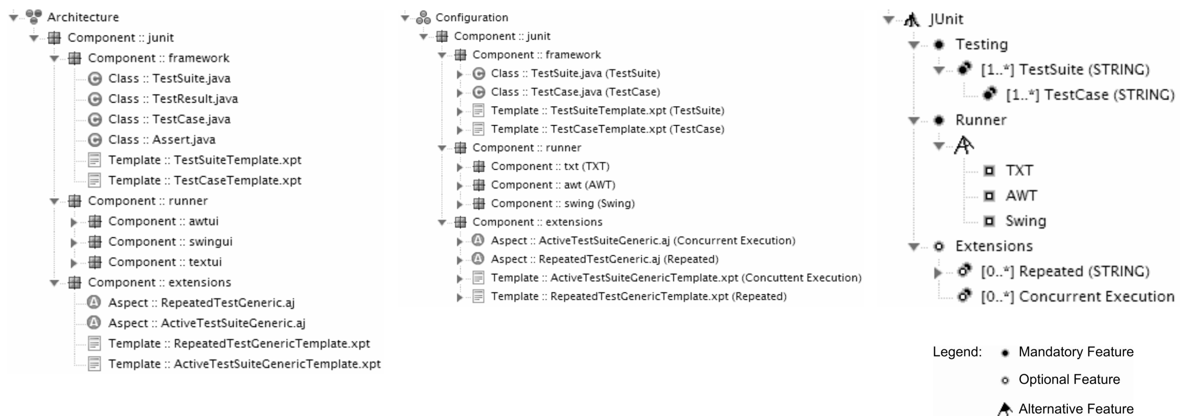
(a) Modelo de Arquitetura **(b) Modelo de Configuração** **(c) Modelo de Características**

Figura 4. JUnit GenArch Models – Versão inicial.

O modelo de arquitetura da Figura 4(a) contém todos os elementos que implementam o framework JUnit. O modelo de configuração, apresentado na Figura

[1] A geração do código inicial dos templates no GenArch é realizada através do processamento das anotações do tipo @Variability.

4(b) possui apenas as configurações que foram representadas no código, através das anotações `@Feature` e `@Variability`. Para o estudo de caso do JUnit foram criadas relações de dependência no modelo de configuração entre as classes `TestCase` e `TestSuite` se relacionando com as características TestCase e TestSuite, respectivamente. Também os templates `TestCaseTemplate` e `TestSuiteTemplate` que foram gerados a partir das anotações `@Variability` das classes `TestCase` e `TestSuite`, respectivamente, são relacionados com as mesmas características. Os aspectos `ActiveTestSuiteGeneric` e `RepeatedTestGeneric` foram relacionados com as características ConcurrrentExecution e Repeated, respectivamente. As características que foram descritas nas classes e no aspectos também possibilitaram a criação de uma versão inicial do modelo de características, mostrada na Figura 4(c).



(a) Modelo de Arquitetura **(b) Modelo de Configuração** **(c) Modelo de Características**

Figura 5. Versão final dos modelos de características e configuração

Como mencionado anteriormente (Seção 2.1), a versões apresentadas na Figura 4 são apenas as versões iniciais dos modelos. Foi necessário o incremento e refinamento dos modelos de configuração e características para tornar possível a derivação automática de instâncias do framework JUnit. A Figura 5 apresenta a versão final dos modelos. Neste caso os componentes que implementam a interface de execução dos testes não foram configurados inicialmente pela ferramenta e tiveram de ser configurados manualmente. No modelo de características, Figura 5(c), as características Runner, TXT, AWT e Swing foram adicionadas. Cada uma dessas características representa um tipo de interface de execução dos testes do JUnit. Já no modelo de configuração, apresentado na Figura 5(b), foram criados relacionamentos entre os componentes (pacotes Java) que implementam cada uma das interfaces de execução e suas respectivas características (TXT, AWT e Swing). Ainda no modelo de configuração os templates `ActiveTestSuiteGenericTemplate` e `RepeatedTestGenericTemplate` foram associados com as características ConcurrentExecution e Repeated. Esses templates representam subaspectos dos aspectos abstratos existentes no mesmo componente.

3.3. Derivando um Membro da Linha de Produto ou Framework

Na última etapa da ferramenta GenArch, um membro da linha de produtos é derivado. A derivação se dá a partir da criação de uma instância do modelo de características, dos

mapeamentos descritos no modelo de configuração e do modelo de arquitetura. O primeiro passo nessa etapa é a seleção e configuração das variabilidades no modelo de características usando o plug-in FMP. Em seguida, o engenheiro de aplicação passa para a ferramenta GenArch a instância do modelo de característica criado, assim como o modelo de configuração e de arquitetura da linha de produto sendo considerada. A partir desses modelos a ferramenta seleciona quais elementos são necessários para implementar o produto a ser gerado. Esses elementos são copiados para suas respectivas pastas em um novo projeto Eclipse, que é especificado pelo engenheiro.

4. Conclusões e Trabalhos Futuros

Este artigo apresentou uma visão geral do GenArch, uma ferramenta baseada em modelos que em combinação com anotações Java permite a derivação automática de membros de uma linha de produtos. A ferramenta é implementada utilizando a plataforma Eclipse, assim como diversas tecnologias de desenvolvimento orientado a modelo disponíveis em tal plataforma. Os recursos da ferramenta foram ilustrados usando o framework JUnit, detalhes adicionais desse e outros estudos de caso podem ser encontrados em [5]. Atualmente, diversas funcionalidades estão sendo incorporadas a ferramenta, entre elas: (i) sincronização entre os modelos, código e anotações; (ii) customização de pontos de corte de aspectos usando o modelo generativo orientado a aspectos proposto em [9,8]. Finalmente, novos estudos com linhas de produto mais complexas estão sendo planejados para validar o uso da ferramenta em cenários reais.

Referências

[1] M. Antkiewicz, K. Czarnecki. FeaturePlugin: Feature modeling plug-in for Eclipse, OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004.

[2] F. Budinsky, et al. Eclipse Modeling Framework. Addison-Wesley, 2004.

[3] P. Clements, L. Northrop. Software Product Lines: Practices and Patterns. 2001: Addison-Wesley Professional.

[4] K. Czarnecki, U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.

[5] GenArch – Generative Architectures Plugin, URL: http://www.teccomm.les.inf.puc-rio.br/genarch/.

[6] J. Greenfield, K. Short. Software Factories: Assembling Applications with Patterns, Frameworks, Models and Tools. 2005: John Wiley and Sons.

[7] C. Krueger. "Easing the Transition to Software Mass Customization". In Proceedings of the 4th International Workshop on Software Product-Family Engineering, pp. 282–293, 2001.

[8] U. Kulesza, et al. Mapping Features to Aspects: A Model- Based Generative Approach. Early Aspects, AOSD'2007, Vancouver, Canada. LNCS 4765, Springer-Verlag.

[9] U. Kulesza, C. Lucena, P. Alencar, A. Garcia. Customizing Aspect-Oriented Variabilites Using Generative Techniques. International Conference on Software Engineering and Knowledge Engineering (SEKE'06). July 2006. San Francisco.

[10] U. Kulesza, R. Coelho, V. Alves, A. C. Neto, A. Garcia, C. Lucena, C.; A. V. Staa, P. Borba. Implementing Framework Crosscutting Extensions with XPIs and AspectJ. XX Simpósio Brasileiro de Engenharia de Software (SBES'2006). 2006. Florianópolis.

[11] openArchitectureWare, URL: http://www.eclipse.org/gmt/oaw/

[12] D. Weiss, C. Lai. Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley Professional, 1999.

[13] S. Deelstra, M. Sinnema, J. Bosch. Product derivation in software product families: a case study. Journal of Systems and Software 74(2): 173-194, 2005.