3
GenArch: Uma Ferramenta Baseada em Modelos para Derivação de Produtos de Software

Este capítulo apresenta a abordagem de derivação de LPS implementada pela ferramenta GenArch. A abordagem tem como objetivo principal simplificar o processo de derivação de LPS, permitindo com isso que a comunidade de desenvolvimento de software utilize os conceitos e fundamentos de LPS no desenvolvimento de sistemas de software e artefatos, como, *frameworks* e bibliotecas customizáveis, sem a necessidade do entendimento de conceitos e modelos complexos presentes nas ferramentas existentes.

3.1.
Visão Geral da Abordagem

A abordagem proposta nessa dissertação busca incentivar o uso das técnicas de LPS através da estratégia de adoção extrativa e facilitar o gerenciamento da evolução da LPS. Para isso, são fornecidos mecanismos que permitem a criação (módulo de importação) e a sincronização (módulo de sincronização) automática do conteúdo dos modelos de derivação a partir do código-fonte de produtos já existentes. A Figura 4 mostra uma visão geral da abordagem. As subseções que se seguem apresentam detalhadamente cada passo da abordagem destacado na Figura 4.

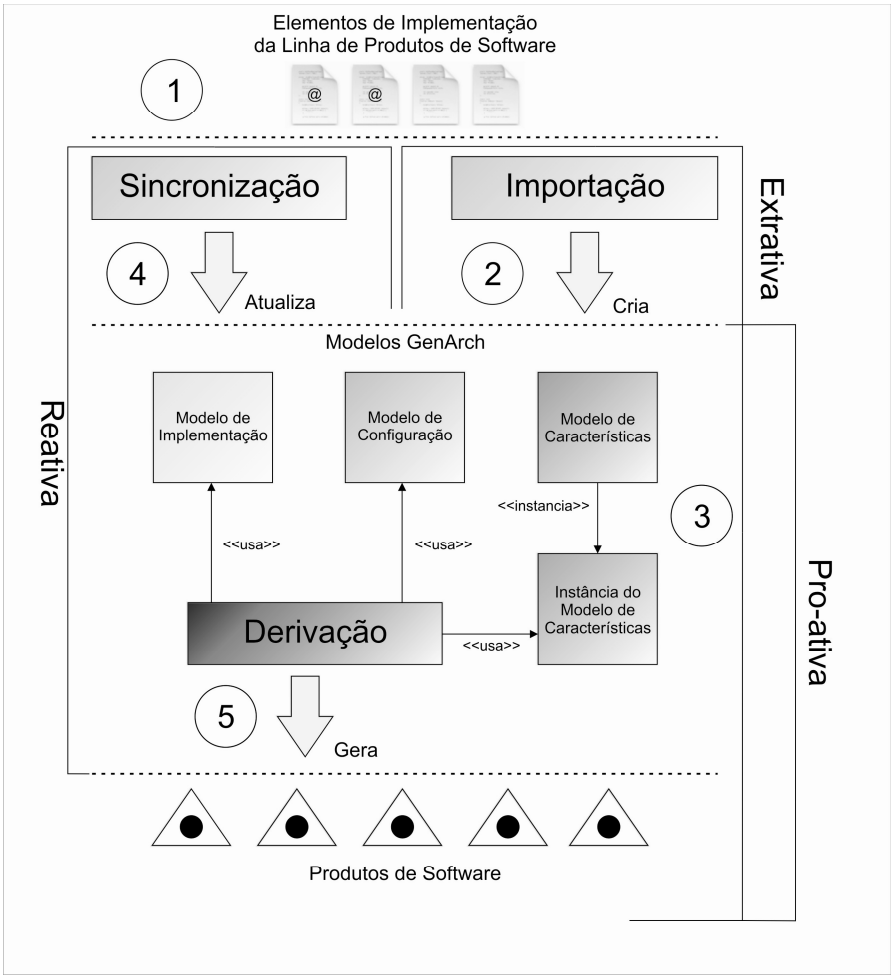

Figura 4. Visão geral da abordagem

3.1.1. Anotação do código-fonte com características e variabilidades

Inicialmente (passo 1), durante a extração dos modelos de derivação a partir de produtos já existentes, o engenheiro de domínio é responsável por anotar o código existente da arquitetura da LPS. Um conjunto de anotações Java é utilizado pelo engenheiro de domínio para anotar elementos de implementação (classes, interfaces e aspectos), com o objetivo de: (i) especificar qual(is) elemento(s) de implementação corresponde(m) a determinada(s) característica(s); e (ii) indicar qual elemento (classe abstrata, interface ou aspecto abstrato) representa um ponto de extensão (*hot-spot)*.

A mostra os dois tipos de anotações definidos pela abordagem: (i) `@Feature` – essa anotação é utilizada para indicar que um dado elemento de implementação endereça uma característica específica. Esta anotação também permite especificar o tipo da característica (obrigatória, alternativa, ou opcional)

que está sendo implementada e seu respectivo pai, caso exista; e (ii) `@Variability` – esta anotação é utilizada para indicar que um determinado elemento de implementação representa um ponto de extensão (*hotspot*) na arquitetura da LPS.

Tabela 1. Anotações GenArch e seus atributos

@Feature	
Atributos	
Name	Nome da característica
Parent	O nome da característica-pai
Type	*alternative*, *optional* ou *mandatory*
@Variability	
Atributos	
Type	*hotspot* ou *hotspotAspect*
Feature	Contém a característica associada com a variabilidade

3.1.2. Modelos de Derivação

Após a anotação do código de implementação de uma LPS, a ferramenta GenArch processa as anotações e gera uma versão inicial dos modelos de derivação (passo 2). Três modelos precisam ser especificados para permitir a derivação automática de membros de uma LPS, sendo eles: (i) o modelo de implementação; (ii) o modelo de características; e (iii) o modelo de configuração. O modelo de implementação define uma representação visual dos elementos de implementação da LPS (componentes, classes, aspectos, templates, pastas e arquivos extras) com o objetivo de facilitar o mapeamento entre os elementos de implementação (espaço de solução) e características presentes no modelo de características (espaço de problema). Esse modelo pode ser automaticamente criado a partir da leitura de diretórios existentes, indicados pelo engenheiro de domínio, e que contém os elementos de implementação da arquitetura da LPS (passo 2). Durante esse processo de importação, cada pacote Java é convertido para um componente, com o mesmo nome¸ no modelo de implementação. Já os outros tipos de elementos de implementação (classes, interfaces, aspectos, templates ou arquivos) possuem uma representação correspondente no modelo de implementação.

O modelo de característica (Kang et al. 1990) é utilizado na abordagem para representar os pontos de variabilidade presentes na arquitetura da LPS.

Uma versão inicial do modelo de característica pode ser criada a partir das anotações do tipo `@Feature` encontradas no código-fonte dos elementos de implementação. Cada anotação `@Feature` demanda a criação de uma nova característica no modelo de característica com seu respectivo tipo, descrito no atributo `type`. Se a anotação processada possui o atributo `parent`, uma característica pai é criada, desde que ela ainda não exista no modelo, para agregar a nova característica.

Finalmente, o modelo de configuração define um conjunto de mapeamentos entre elementos do modelo de implementação da LPS e características do modelo de características. Este modelo representa o conhecimento de configuração proposto em desenvolvimento generativo (Czarnecki et al. 2000), discutida na seção 2.2, sendo fundamental para conectar o espaço do problema (características) ao espaço da solução (elementos de implementação). Uma versão inicial do modelo de configuração pode ser criada a partir das anotações `@Feature` presentes no código-fonte dos elementos de implementação. Para cada anotação `@Feature`, a ferramenta GenArch adiciona um mapeamento entre a(s) característica(s) anotadas(s) e o respectivo elemento de implementação anotado. A versão atual do modelo de configuração mostra os mapeamentos em formato de árvore (similar ao modelo de implementação), mas com uma indicação explícita das características que cada um dos elementos depende. Apenas elementos que dependem explicitamente de alguma característica para serem instanciados são apresentados no modelo de implementação. Todos os outros são considerados mandatórios e farão parte de qualquer instância (produto) da LPS.

3.1.3. Criação Automática da Estrutura de Templates

Templates são usados pela ferramenta GenArch para codificar elementos de implementação (classes, interfaces, aspectos e arquivos) que necessitam ser customizados durante a derivação de um produto. Exemplos de elementos que podem ser implementados usando templates são: (i) instâncias concretas de pontos flexíveis da arquitetura de LPS; e (ii) arquivos de configuração parametrizados. Os templates podem utilizar informação coletada dos modelos de característica e implementação para customizar código variável presente em uma classe, aspecto, interface ou arquivo de configuração. Versões iniciais de templates são automaticamente criadas a partir das anotações do tipo

`@Variability` definidas no código-fonte de classes, interfaces e aspectos. Cada anotação `@Variability` demanda a criação de um template que representa instâncias concretas do elemento de implementação extensível que foi anotado. A ferramenta GenArch adota a linguagem XPand (openArchitectureWare 2008) para especificar o código dos templates. Uma representação visual dos templates é automaticamente criada no modelo de implementação, para que esses possam ser relacionados com características. A Figura 5 contém exemplo de um template implementado na linguagem XPand. Detalhes sobre a implementação de templates utilizando a linguagem XPand podem ser encontrados na seção 4.1.3.

```
01 «IMPORT br::pucrio::inf::les::genarch::models::instance»
02 «EXTENSION br::pucrio::inf::les::genarch::models::Model»
03 «DEFINE Main FOR Instance»
04
05 «LET feature("TestSuite",featureElements) AS
06     feature»
07
08 «FILE feature.attribute + ".java"»
09 package junit.framework;
10 public class «feature.attribute» {
11
12 }
13 «ENDFILE»
14 «ENDFOREACH»
15 «ENDDEFINE»
```

Figura 5. Exemplo de template criado automáticame

3.1.4. Refinamento e Sincronização dos Modelos de Derivação

Após a criação automática da versão inicial dos modelos de derivação e dos templates, o engenheiro de domínio deve refiná-los de forma que eles modelem todas as variabilidades presentes na LPS (passo 3). Durante o processo de refinamento, novas características podem ser introduzidas no modelo de características ou as já existentes podem ser reorganizadas. No modelo de implementação, novos elementos também podem ser incluídos ou reorganizados, e a implementação de um template pode ser acrescida com código comum ou variável. Finalmente, novos mapeamentos entre características e elementos de implementação podem ser criados no modelo de configuração.

No contexto de evolução da LPS, os modelos de derivação também podem ser revisitados para incorporar novas modificações ou requisições. Tais modificações podem ser associadas ao modelo de adoção reativo, apresentado na Seção 2.1, e sincronizações são realizadas para garantir a consistência entre os diferentes modelos. O módulo de sincronização é responsável por observar modificações nos elementos de implementação e automaticamente refletí-las nos modelos (passo 4). A sincronização entre código, anotações e modelos também é fundamental para garantir a compatibilidade entre os diversos artefatos que compõem a LPS e evitar inconsistência durante todo o processo de derivação. O módulo de sincronização da ferramenta GenArch busca a resolução automática das seguintes inconsistências/revisões: (i) remoção de características do modelo de característica que não são mais utilizadas pelo modelo de configuração ou por alguma anotação no código; (ii) remoção de mapeamentos no modelo de configuração que referenciam características ou elementos de implementação que não existem mais; (iii) remoção de elementos do modelo de implementação que referenciam arquivos de implementação que não existem mais; (iv) criação automática de anotações do tipo `@Feature` em elementos de implementação, baseada na existência de mapeamentos existentes ou recentemente criados no modelo de configuração; e (v) modificação no caminho original de um elemento de implementação (mover o elemento para outro diretório), implica na mesma modificação no elemento relacionado no modelo de implementação e configuração.

3.1.5. Documentação de Variabilidades

Os modelos de característica e configuração podem ser vistos, respectivamente, como uma documentação útil das variabilidades existentes na LPS e do mapeamento dessas variabilidades para elementos de implementação. Essa documentação também é um primeiro passo para endereçar a rastreabilidade de características obrigatórias e variáveis de LPS, do modelo de características para os modelos de implementação e respectivos artefatos de código. A rastreabilidade de características comuns e variáveis no desenvolvimento de LPS pode apoiar as atividades de engenheiros de software no que se refere à análise de cobertura das características na arquitetura de LPS e artefatos de implementação e análise do impacto de mudanças de uma dada característica em artefatos de implementação. Tal rastreabilidade também é

utilizada pelo módulo de derivação para descidir quais artefatos vão compor o produto derivado. A funcionalidade de sincronização garante que esta documentação se mantenha sempre atualizada e sincronizada com os artefatos reais.

3.1.6. Processo de Derivação de Produtos

Após todos os modelos serem refinados e representarem a implementação e as variabilidades da arquitetura da LPS, a ferramenta GenArch está pronta para automaticamente derivar instâncias/produtos da LPS (passo 5). Durante a derivação, o engenheiro de aplicação deve, primeiramente, criar uma instância do modelo de características (também chamado de configuração (Czarnecki et al. 2004)) para decidir quais variabilidades farão parte da aplicação final que será gerada (passo 4) e fornecê-la para a ferramenta. Posteriormente, a ferramenta percorre o modelo de implementação e processa cada elemento de implementação encontrado verificando no modelo de configuração se este depende de uma ou mais características, neste caso, a ferramenta apenas instância este elemento (e processa os respectivos sub-elementos), se a avaliação da expressão booleana de características associada for verdadeira. A avaliação da expressão booleana é feita da seguinte maneira: (i) para cada característica na expressão, sua ocorrência é verificada na configuração do modelo de característica, se a característica ocorre selecionada na configuração do modelo de característica seu valor é “true” (verdadeiro), se não, seu valor é “false” (falso); e então (ii) a expressão é avaliada. A ferramenta produz, como resultado do processo de derivação, um projeto Eclipse contendo somente os elementos de implementação correspondentes ao produto expressa na configuração do modelo de características e especificada pelo engenheiro de aplicação.

3.2. Arquitetura e Implementação da Ferramenta GenArch

Essa seção apresenta a arquitetura, os modelos adotados e as tecnologias utilizadas no desenvolvimento da ferramenta GenArch. Primeiramente, a Seção 3.2.1 apresenta a arquitetura de implementação adotada em termos de tecnologias. As seções seguintes (3.2.2, 3.2.3 e 3.2.4) apresentam uma breve descrição dos plug-ins *Eclispe Modeling Framework* (Budinsky et al. 2003), *Open*

Architecture Ware (openArchitectureWare 2008) e *Feature Modeling Plug-in* e como esses foram utilizados na implementação da ferramenta.

3.2.1. Visão Geral da Arquitetura

A ferramenta GenArch foi desenvolvida como um plug-in da plataforma Eclipse (Shavor et al. 2003). Eclipse é uma IDE para desenvolvimento de aplicações que fornece uma plataforma extensível e diversos recursos que facilitam o desenvolvimento de plug-ins. A plataforma Eclipse oferece um motor de execução, responsável pela instanciação de toda a plataforma. As demais funcionalidades são estruturadas por um conjunto de subsistemas implementados por um ou mais plug-ins. O núcleo da plataforma Eclipse implementa uma arquitetura que permite dinamicamente descobrir, carregar, e executar plug-ins. Um plug-in adiciona novas funcionalidades à plataforma, ou seja, se pluga a plataforma, através da realização de pontos de extensão espostos pela mesma. Um plug-in também pode ser extendindo através da esposição de um ou mais pontos de extensão.

A Figura 6 mostra a estrutura geral da arquitetura de implementação da ferramenta GenArch. No desenvolvimento da ferramenta GenArch, quatro plug-ins extensíveis da plataforma Eclipse foram fundamentais para suportar: (i) a construção de wizards, a navegação pela árvore de diretórios de um projeto e funções para manipulação de recursos (arquivos e diretórios); (ii) a manipulação dos modelos; e (iii) a geração de código.

Para manipulação de código-fonte Java e AspectJ, a ferramenta GenArch utiliza os plug-is Java Development Toolkit (JDT) (Shavor et al. 2003) e AspectJ Development Toolkit (AJDT) (Colyer 2004), respectivamente. Tais plug-ins fornecem um conjunto de ferramentas para gerenciamento da área de trabalho, visões, editores, compiladores e manipuladores de código. A API de árvore de sintaxe abstrata disponível nesses plug-ins foi utilizada na manipulação (leitura, escrita e remoção) das anotações GenArch e na captura da estrutura de cada elemento de implementação (classe, interface ou aspecto). Como foi apresentado anteriormente, é através das anotações no código que a ferramenta consegue gerar uma versão inicial dos modelos de derivação.

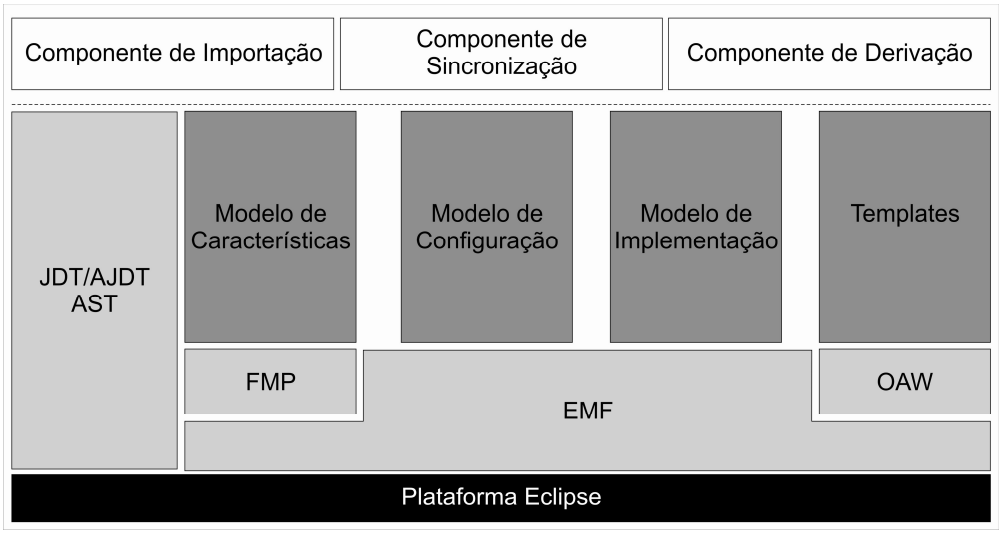

Figura 6. Arquitetura da ferramenta GenArch

A infra-estrutura para manipulação dos modelos foi implementada usando os recursos do *Eclipse Modelling Framework* (EMF). O EMF (Budinsky et al. 2003) fornece um conjunto de plug-ins que suportam o desenvolvimento de ferramentas de modelagem e geradores de código. Ele é estruturado como um framework Java que oferece diversas facilidades para: (i) criação e manipulação de modelos; (ii) criação e persistência de instâncias de um dado modelo; e (iii) geração de código a partir de instâncias de um dado modelo. O EMF foi utilizado na geração das classes que permitem a criação e manipulação dos modelos de derivação. O modelo de características, em particular, é implementado pelo plug-in *Feature Modelling Plugin* (FMP). Esse plug-in permite a modelagem do modelo de característica proposto por Czarnecki et al (Czarnecki et al. 2000). A notação proposta por Czarnecki et al suporta a especificação de características obrigatórias, opcionais e alternativas, e suas respectivas cardinalidade. Boa parte da infra-estrutura para modelagem de características oferecida pelo plug-in FMP é também implementada usando o EMF.

O plug-in openArchitectureWare (oAW) (openArchitectureWare 2008) provê um conjunto de ferramentas que explora de forma completa a abordagem de desenvolvimento baseado em modelos (Stahl et al. 2006). Tal plug-in oferece um conjunto de componentes que podem ser utilizados para leitura e instanciação de modelos, verificação de regras, transformação entre modelos e geração de código. Atualmente, a ferramenta GenArch faz uso somente do componente de geração de código. A geração se dá a partir do processamento

de templates escritos em uma linguagem própria, denominada XPand. O plug-in oAW também utiliza como base de sua implementação o plug-in EMF.

As próximas seções apresentam, brevemente, as principais características dos plug-ins e como esses foram utilizados no desenvolvimento da ferramenta GenArch.

3.2.2. Eclipse Modeling Framework (EMF)

EMF é um framework Java/XML muito utilizado na construção de ferramentas dirigidas por modelos estruturados. O EMF é parte integrante da implementação de *Model Driven Development* (MDD) (Stahl et al. 2006) do projeto Eclipse (Shavor et al. 2003) e tem como principal objetivo fornecer um conjunto de ferramentas que facilitam a especificação e construção de modelos utilizados como base para o desenvolvimento de aplicativos.

O EMF oferece para construção automática de ferramentas orientadas a modelos: (i) um conjunto de classes Java que representam um meta-modelo e permitem o acesso a esse em tempo de execução; (ii) um conjunto de classes adaptadoras que possibilitam a visualização e edição dos modelos; e (iii) um editor visual básico. Todos esses elementos são gerados automaticamente a partir da especificação de um meta-modelo eCore (Budinsky et al. 2003). A geração automática dos elementos citados acima é feita em três níveis: (i) no nível de modelo – um conjunto de interfaces Java e classes que as implementam são geradas em conjunto com as classes `Factory` e `Package` (meta-dados); (ii) no nível de adaptação – é gerado um conjunto de classes que adaptam as classes do modelo para serem editadas e visualizadas; e (iii) no nível de edição – é produzido um editor estruturado em forma de árvore que pode ser facilmente customizado.

O EMF foi utilizado nesse trabalho na construção dos modelos de implementação e configuração da nossa abordagem. Além desses dois modelos, um terceiro também foi definido com o objetivo de conter as informações que serão utilizadas pelos templates para realizar a customização de um elemento de implementação contendo variabilidades. Tal modelo agrega informações tanto do modelo de implementação, quanto do modelo de características. Essa decisão de implementação foi necessária, pois templates XPand só recebem como parâmetro de entrada um único meta-modelo EMF.

O primeiro passo para construção da infra-estrutura de manipulação dos modelos foi a criação da meta-modelos dos modelos usados pela ferramenta GenArch. As Figura 7, 8 e 9 mostram, respectivamente, os meta-modelos dos modelos de implementação, de configuração e de derivação..

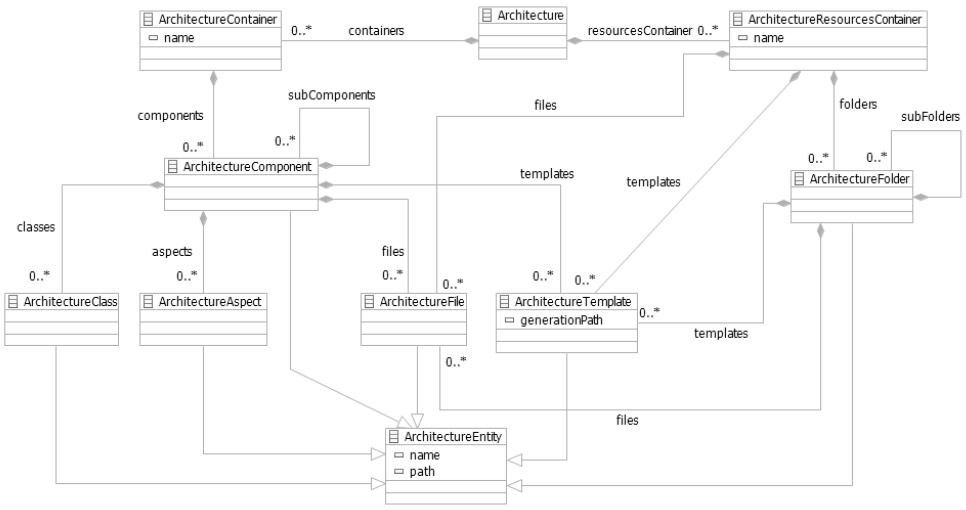

Figura 7. Meta-modelo do modelo de implementação

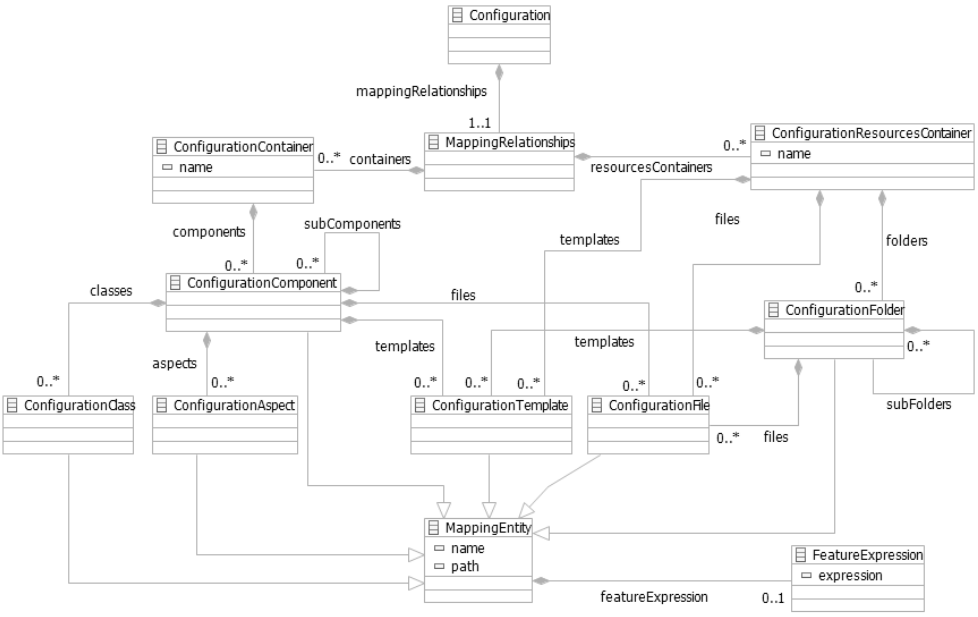

Figura 8. Meta-modelo do modelo de configuração

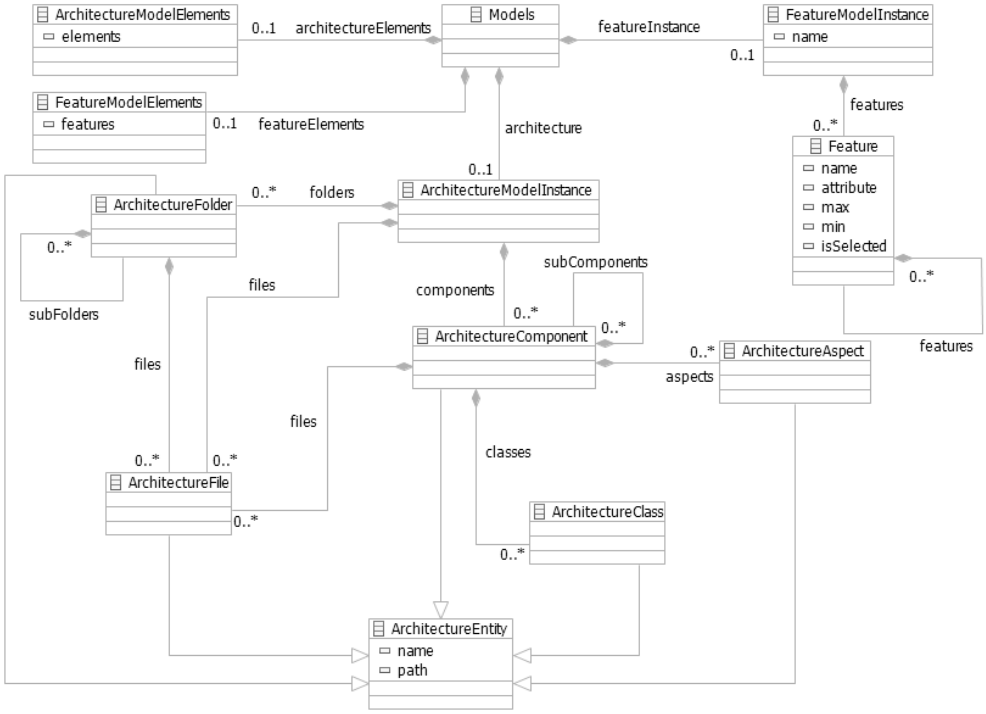

Figura 9. Meta-modelo do modelo de derivação

Como pode ser visto na Figura 9, o modelo de derivação é uma combinação do modelo de implementação com o modelo de características. A partir dos meta-modelos criados, o EMF foi capaz de gerar um conjunto de classes que implementa toda a infra-estrutura necessária para leitura, armazenamento, edição e manipulação dos modelos. A Tabela 2 apresenta todos os pacotes gerados pelo EMF e suas respectivas descrições.

Tabela 2. Pacotes gerados automaticamente pelo EMF

`br.pucrio.inf.les.genarch.models.instance`
Esse pacote contém as classes que implementam o meta-modelo da instância do modelo de características que é utilizada para a customização dos templates durante o processo de derivação.

`br.pucrio.inf.les.genarch.models.implementation`
`br.pucrio.inf.les.genarch.models.implementation.impl`
`br.pucrio.inf.les.genarch.models.implementation.presentation`
`br.pucrio.inf.les.genarch.models.implementation.provider`
`br.pucrio.inf.les.genarch.models.implementation.util`

Esses pacotes contém: (i) as classes que implementam o meta-modelo do modelo de implementação; e (ii) as classes que implementam um editor para o modelo de implementação.

br.pu-rio.inf.les.genarch.models.configuration br.pucrio.inf.les.genarch.models.configuration.impl br.pucrio.inf.les.genarch.models.configuration.presentation br.pucrio.inf.les.genarch.models.configuration.provider br.pucrio.inf.les.genarch.models.configuration.util
Esses pacotes contém: (i) as classes que representa o meta-modelo do modelo de configuração e (ii) as classes que implementam um editor para o modelo de configuração.

3.2.3. openArchitectureWare (oAW)

O *openArchitectureWare* (oAW) (openArchitectureWare 2008) é um plug-in para o ambiente Eclipse, desenvolvido com objetivo de fornecer uma infra-estrutura completa para o desenvolvimento dirigido por modelos (Stahl et al. 2006).

O oAW é baseado em um processador que permite a definição do fluxo de execução dos geradores/transformadores (Figura 10). Juntamente com o processador, o oAW fornece um conjunto de componentes prontos que podem ser facilmente utilizados para a leitura e instanciação de modelos, validação de modelos, transformação entre modelos e geração de código. No que diz respeito a transformação entre modelos, o oAW permite definir um processo configurável de transformação: (i) de modelos para textos; (ii) de modelos para modelos; (iii) de textos para modelos. A Figura 10 mostra um exemplo de um fluxo de execução abstrato que envolve todos os componentes do oAW. Inicialmente os modelos são carregados e instanciados. Após a instanciação, cada modelo é verificado de acordo com regras específicas no meta-modelo. Se alguma regra for violada o processo é interrompido. Estando de acordo com as regras, os modelos podem ser transformados ou servirem de entrada para geradores de código que podem ser integrados com códigos escritos a mão.

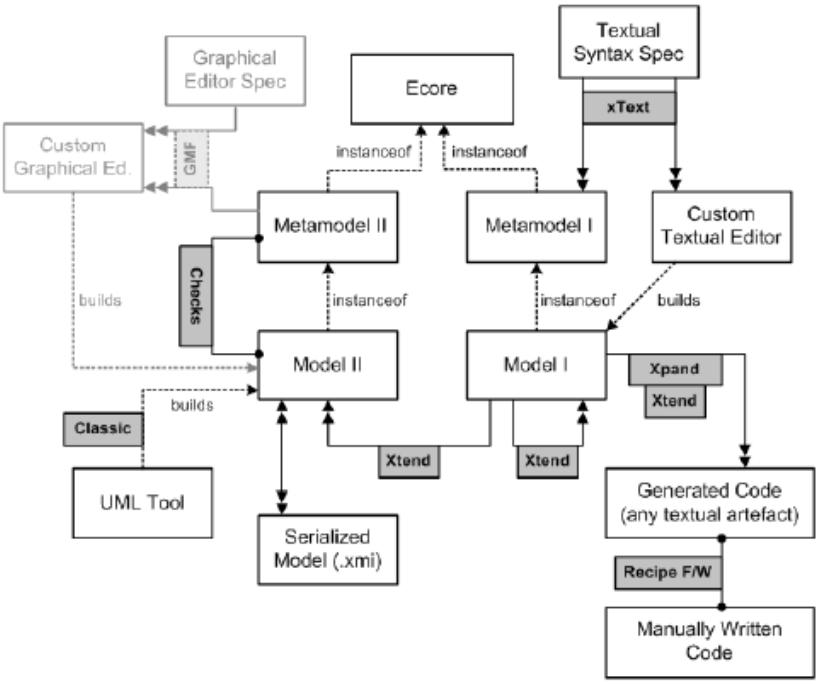

Figura 10. Fluxo de execução do plug-in oAW (Stahl et al. 2006)

Os componentes do oAW podem ser utilizados de forma integrada, como mostrado na Figura 10, ou individualmente em aplicações isoladas. No caso da ferramenta GenArch, somente a linguagem XPand foi utilizada. Ela é utilizada na especificação de templates e, consequentemente, para geração de código no momento da derivação.

oAW está preparado para trabalhar com modelos EMF e UML2, além de ser integrado com diversas ferramentas para suporte a modelagem UML, tais como, MagicDraw, e Rational Rose.

3.2.3.1. Estendendo as funcionalidades da linguagem XPand

Para facilitar o acesso a elementos específicos dos modelos, foi definido um conjunto de funções que estendem as características base da linguagem XPand. A Tabela 3 apresenta as funções criadas e a respectiva descrição.

Tabela 3. Funções de extensão

Função	Descrição
`Feature feature(Object name,Object model)`	Função responsável pela busca e recuperação de uma dada característica no modelo de características. Deve-se passar

	como parâmetro: o nome da característica; e o modelo de características.
`ArchitectureEntity element(Object name,Object model)`	Função responsável pela busca e recuperação de um dado elemento de implementação no modelo de implementação. Deve-se passar como parâmetro: o nome do elemento; e o modelo de implementação.
`EList configurations(Object name,Object model)`	Função responsável pela busca e recuperação de todas as configurações de uma dada características numa instância do modelo de características. Deve-se passar como parâmetro: o nome da característica; e a instância do modelo de características.

3.2.4. Feature Modeling Plug-in

Feature Modeling Plug-in (FMP) é um plug-in Eclipse que permite a modelagem de características baseada em cardinalidade. Cardinalidade é um intervalo que determina quantas vezes uma dada característica pode ser clonada. O modelo de característica (Czarnecki et al. 2004) suportado pelo plug-in estende o modelo originalmente proposto pela metodologia FODA (Kang et al. 1990) permitindo, além da modelagem de características obrigatórias, opcionais e alternativas, a modelagem de cardinalidade, atributos, referências entre características e anotações definidas pelo usuário.

O modelo da descreve as características do framework JUnit, que será apresentado na seção 4.1, utilizando o plug-in FMP. Esse modelo consiste de um diagrama de característica contendo uma característica raiz, denominada **JUnitFramework** e indicada pelo símbolo . A característica **Testing** possui uma sub-característica **TestSuite** que por sua vez possui uma sub-característica **TestCase**. O símbolo indica que a característica **Testing** possui cardinalidade [1..1], ou seja, é uma característica obrigatória. A característica **Extension** possui cardinalidade [0..1] e indica que essa é uma característica opcional. O símbolo é utilizado para representar esse tipo de característica. O modelo de característica permite também a modelagem de um agrupamento de características. Agrupamento pode ser de três tipos: (i) agrupamentos que

requerem a seleção de no mínimo uma característica, ou seja, agrupamento com cardinalidade <1-k>, indicado pelo símbolo ; (ii) agrupamentos que permitem a seleção de somente uma característica do grupo, ou seja, a cardinalidade da seleção é <1-1>, indicado pelo símbolo . Esse tipo de agrupamento corresponde à modelagem de características alternativas; e (iii) agrupamento com cardinalidade <x-k> especificada pelo engenheiro de domínio. Esse tipo de agrupamento é indicado pelo símbolo . As características **TestSuite** e **TestCase** possuem cardinalidade [1..*]. O * indica que não existe um limite para o número de vezes que tais características podem ser clonadas. No diagrama, o tipo do atributo que pode ser especificado em uma característica aparece ao lado do nome da característica, como, por exemplo, nas características **Test Suite**, **Test Case**, **Repeated Test** e **Concurrent Test**. O valor desse atributo deve ser configurado na instância do modelo de característica.

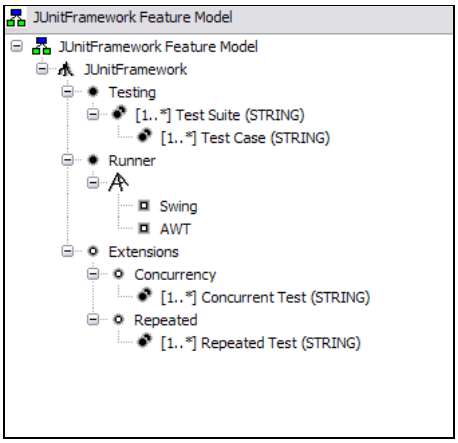

Figura 11. Modelo de característica do Framework JUnit

Em alguns casos, um modelo de características pode demandar a especificação de regras que não são possíveis de serem expressas a partir de características ou cardinalidade. O plug-in FMP possui um mecanismo que permite a criação de regras lógicas, aritméticas, conjuntivas e de operação sob *strings* (Antkiewicz et al. 2004). O processo de execução dessas regras é feito pela linguagem XPath (Kay 2004).

3.3. Conclusão

Esse capítulo apresentou detalhadamente o processo de derivação implementado pela ferramenta GenArch. O processo é baseado na definição de

três modelos (característica, configuração e implementação). Através de anotações Java específicas, presentes no código de implementação, é possível a criação automática de uma versão inicial dos modelos de derivação. As anotações também permitem uma engenharia de *round-trip* entre modelos e código, onde modificações nas anotações podem ser refletidas nos modelos e alterações no modelos podem ser refletidas no código.

O capítulo apresentou também a arquitetura de implementação e as tecnologias usadas na concepção da ferramenta GenArch. A plataforma Eclipse, por fornecer um ambiente de desenvolvimento extensível trouxe várias facilidades para o desenvolvimento da ferramenta. Aliado a plataforma Eclipse, os plug-ins oAW, FMP e EMF possibilitaram a construção da infra-estrutura para manipulação dos modelos e templates.