8
Trabalhos Relacionados

Neste capítulo a abordagem OA para desenvolvimento de frameworks é comparada com diversos trabalhos desenvolvidos pela comunidade. A abordagem é confrontada com: (i) abordagens para implementação de famílias de software que buscam uma melhor modularização de características; (ii) abordagens para instanciação de linhas de produto; e (iii) abordagem baseada em interfaces transversais.

8.1.
Interfaces Transversais (XPIs)

Nosso conceito de EJPs é inspirado no trabalho proposto por Sullivan et al [115] de especificação de interfaces transversais (XPIs). XPIs permitem abstrair comportamento transversal, isolando o projeto de aspectos do código base, e vice-versa. A proposta inicial de XPIs foi documentar *design rules*, através de um formato descritivo contendo os seguintes elementos: nome, razão da existência, dependências, escopo do código afetado, um conjunto de contratos que são oferecidos ou requeridos do núcleo da aplicação ou aspectos. O uso das XPIs é ilustrado por tais pesquisadores, através de um *middleware* de redes sobrepostas, e comparada com a abordagem baseada no princípio *Obliviousness*. Como continuidade do trabalho, Griswold et al [56] mostram como representar XPIs como construções sintáticas em AspectJ.

EJPs podem ser vistos como a especialização do conceito das XPIs para o contexto de desenvolvimento de frameworks. EJPs estão para XPIs, assim como pontos flexíveis (*hot-spots*) OO de frameworks estão para classes abstratas e interfaces. O objetivo central das EJPs é expor um conjunto de eventos/estados do framework que podem ser facilmente estendidos ou observados por aspectos de extensão. Nossa proposta para especificação sintática de EJPs em AspectJ é uma extensão a aquela proposta em [56], com a introdução do conceito de especialização. Para a especificação da parte semântica de EJPs, entretanto, uma

metodologia foi definida a qual explicita diferentes tipos de contratos que podem ser estabelecidos para regular as interações entre o núcleo do framework, os EJPs e os aspectos de extensão.

8.2. Abordagens para Implementação de Famílias de Software

8.2.1. Programação Orientada a Características

Programação orientada a características (FOP – *feature oriented programming*) foi proposta [15, 111] para lidar com o encapsulamento de características que podem ser usados para estender a funcionalidade de programas base existentes. A idéia central proposta por FOP é permitir a geração de aplicações através da composição de características. Uma característica pode ser vista como um incremento na funcionalidade do programa. FOP propõe uma abordagem de refinamento incremental onde características representadas por unidades de implementação modular são gradativamente aplicadas a uma base inicial de código-fonte. *Mixin-Layers* [111] é uma das tecnologias que podem ser usadas para implementar características em FOP. Um módulo *Mixin-Layer* permite o encapsulamento de refinamentos de diversas classes.

Batory et al [14] argumentam as vantagens que abordagens de FOP têm em relação a frameworks OO para projetar e implementar linhas de produto. O módulo *Layer* proposto por abordagens de FOP, permite encapsular várias abstrações que refinam uma base pré-definida, juntas tais abstrações determinam a implementação de uma característica dentro de uma única unidade modular. Tal mecanismo permite uma melhor gerência de características em frameworks, através da adição, remoção ou modificação das diferentes *Layers* definidas para a linha de produto. Mezini & Osterman [94] identificam, entretanto, que FOP é uma abordagem capaz apenas de prover suporte para modularização de características hierárquicas, não sendo capaz de endereçar o projeto e implementação de características transversais. Em especial, FOP não oferece mecanismos para suportar modularização transversal: (i) estática – existindo casos em que os refinamentos a serem definidos para uma dada característica não possuem correspondência direta para estruturas modulares do código base; e (ii) dinâmica –

abordagens FOP só permitem interceptar chamadas de métodos, não havendo suporte em tais abordagens para expressar/especificar um conjunto de pontos de junção do código base a serem afetados por um dado refinamento definido por uma *Layer*.

Recentes estudos [8, 88, 94] têm explorado o uso integrado das abordagens de POA e FOP. Mezini & Osterman [93] propõem CaesarJ, uma linguagem orientada a aspectos que incorpora conceitos de ambas as abordagens visando lidar com as deficiência de ambas. CaesarJ propõe o conceito de *family class* para agregar um conjunto de refinamentos a serem aplicados a uma dada base existente com o objetivo de implementar uma dada característica. Cada refinamento é definido por uma *virtual class*, a qual por sua vez pode definir extensões para serem aplicadas à classes existentes, como por exemplo, redefinir ou sobrepor métodos. Pontos de corte e adendos podem também ser definidos dentro do escopo de uma *family class*, permitindo assim que mecanismos OA possam ser usados na implementação de características sempre que necessário. Mecanismos para facilitar a reutilização assim como a instalação dinâmica de implementações de características estão também disponíveis em CaesarJ.

Apel et al [8] apresentam a abordagem *Aspectual Mixin Layers* (AMLs), a qual também integra os conceitos de POA e FOP, propondo mecanismos no nível de linguagem de programação que estendem a abstração de *Mixin Layers* [111], e para permitir o encapsulamento de *mixins* e aspectos. Cada AML pode implementar papéis de extensão a uma classe, assim como aspectos que contribuem para a implementação de uma característica do sistema. Os autores propõem a estruturação de arquiteturas OA em 2 níveis: (i) aspectos devem ser usados no nível de classes para implementar características transversais; (ii) enquanto *mixins* são responsáveis pela organização da arquitetura como um todo oferecendo mecanismos para estruturar aspectos e classes em módulos de mais alto nível que endereçam determinadas características do sistema. Apel & Batory [7] apresentam os benefícios e potencial da abordagem AMLs numa linha de produto de redes sobrepostas. Os resultados de tal estudo de caso demostram: (i) a utilidade de FOP para endereçar extensões localizadas, tais como, a adição de novas classes e novos membros em classes existentes; e (ii) a utilidade da adoção de aspectos para melhorar a modularidade transversal de determinadas características e para reduzir a quantidade de código redundante.

Nossa abordagem propõe a modularização de características transversais encontradas em frameworks usando programação orientada a aspectos. AspectJ foi a linguagem adotada usada em nossos estudos de caso, por ser a de maior amadurecimento até o momento. De forma distinta aos trabalhos apresentados de combinação de uso de FOP e aspectos, nossa abordagem utiliza EJPs como um mecanismo que permite isolar o projeto do núcleo do framework e de seus aspectos de extensão, dando dessa forma mais disciplina ao desenvolvimento do framework ou arquitetura de família de sistemas. Nos estudos de caso apresentados nesta tese, os mecanismos de POA em combinação com a técnica de frameworks OO foram suficientes para endereçar a modularização adequada de suas características. Explorar o potencial do uso combinado de FOP e POA na implementação de características encontradas em arquiteturas de famílias de sistemas é um tópico interessante para explorar, de forma a derivar diretrizes para potenciais cenários de aplicação de cada uma das tecnologias. Também o uso de técnicas generativas de instanciação baseada em modelos de características ou DSLs em combinação com técnicas de FOP também merece ser investigado.

8.2.2. Método de Decomposição Horizontal

Zhang e Jacobsen [118] propõem o método de Decomposição Horizontal, um conjunto de princípios que auxiliam na modularização de arquiteturas orientadas a aspectos, compostas de um núcleo e um conjunto de customizações do núcleo. Tais customizações são implementadas usando programação orientada a aspectos. Os seguintes princípios norteiam o método: (i) reconhecimento da relatividade de aspectos – que indica que a semântica de aspectos é determinado pela funcionalidade principal da aplicação; (ii) estabelecer a decomposição coerente do núcleo – a base de uma decomposição OA é o estabelecimento da funcionalidade de um núcleo coerente; (iii) definir a semântica de um aspecto de acordo com a decomposição do núcleo – usando o núcleo como referência, uma funcionalidade é considerada transversal se ambos sua semântica e implementação não estão localizados em um único componente do núcleo; (iv) manter uma arquitetura *class-directional* – aspectos devem referenciar/interceptar as classes do núcleo, mas não o contrário; e (v) aplicar refatorações incrementais –

reconhece a dificuldade de estabelecimento de um núcleo e propõe a realização de refatorações incrementais ao longo da evolução do sistema.

O método proposto por Zhang & Jacobsen organiza um conjunto de boas práticas utilizadas para desenvolvimento de arquiteturas OA e o combina com a técnica de refatoração orientadas a aspectos [96] de forma a facilitar a evolução e customização do núcleo. O método foi aplicado com sucesso no domínio de *middlewares* para sistemas distribuídos [118] e também ao sistema de banco de dados Prevayler [52]. Os principais benefícios do método em tais estudos foram: (i) obter uma versão mais simples e reduzida do núcleo do sistema sendo refatorado; e (ii) aumento do grau de configurabilidade e adaptabilidade do sistema, através da resolução de problemas de combinação e conflito entre características.

Nossa abordagem para desenvolvimento de frameworks também motiva a modularização de sua arquitetura através da definição de um núcleo central contendo a funcionalidade obrigatória a todas as instâncias do framework, e um conjunto de extensões OA que são usadas para implementar características transversais opcionais ou alternativas a esse núcleo. Nossa abordagem também é centrada no princípio *class-directional*, proposto inicialmente por Kerten & Murphy [67]. Ao contrário do método de Decomposição Horizontal, o qual adota o princípio de Inconsciência (*Obliviousness*) [43], nossa abordagem é centrada na definição de EJPs, os quais buscam uma diminuição do acoplamento entre o núcleo e as extensões do framework. O uso de EJPs no desenvolvimento de frameworks ou arquiteturas flexíveis OA traz os seguintes benefícios adicionais: (i) facilidade de entendimento da forma como cada extensão OA afeta o núcleo; (ii) possibilita a identificação de conflitos entre extensões OA agindo sobre os mesmos conjuntos de pontos de junção; e (iii) maior estabilidade do núcleo e dos aspectos de extensão durante a evolução do sistema.

8.2.3. Frameworks Transversais

Camargo & Masiero [21, 22] propõem uma classificação para Frameworks Orientados a Aspectos (FOAs). De acordo com tais pesquisadores, um FOA é definido como um conjunto de classes e aspectos que formam uma estrutura semi-

completa que necessita ser completada com detalhes de uma aplicação específica. Os FOAs são classificados como sendo de dois tipos: (i) Framework Transversal (FTs) – que são os FOAs que endereçam apenas um interesse transversal; e (ii) Frameworks de Aplicação Orientado a Aspectos (FAOAs) – que são aqueles FOAs que definem uma arquitetura genérica para um dado domínio de aplicação e cuja instanciação produzem uma aplicação. A arquitetura dos FAOAs é projetada usando as abstrações de classes e aspectos. O foco principal do trabalho desses pesquisadores é no desenvolvimento de Frameworks Transversais. Eles propõem uma arquitetura de referência para o projeto de FTs e apresentam três famílias de FTS que endereçam os seguintes interesses transversais: persistência de dados, segurança e regras de negócio. Finalmente, um processo para desenvolvimento de software apoitado pelo uso de FTs também é proposto.

Uma das diferenças centrais de nosso trabalho com o proposto por tais autores, é que o foco principal dessa tese é nos Frameworks de Aplicação Orientado a Aspectos, em contraposição aos Frameworks Transversais. Essa tese propõe diretrizes explícitas para a modularização de características transversais em FAOAs. Nossa abordagem é também centrada na exposição de EJPs dos FAOAs na forma de interfaces transversais, não sendo baseada no princípio de Inconsciência (*Obliviousness*). Os aspectos do núcleo e de extensão (variabilidade e integração) da nossa abordagem podem também a princípio serem projetados na forma de FTs. Um ponto importante de discussão na categorização de FOAs proposta por tais autores é identificar as diferenças entre bibliotecas de aspectos e FTs. Uma biblioteca de aspectos [70] define um conjunto de aspectos abstratos que modularizam um dado interesse transversal e que podem ser especializados para serem usados no contexto específico de uma aplicação. Diversas linguagens OA têm disponibilizado bibliotecas de aspectos [66]. A diferença central entre ambos é sutil, e indica que os FTs precisam não apenas ser compostos com a aplicação (através da concretização de pontos de corte abstratos), mas também oferece variabilidades funcionais a serem instanciadas (através da implementação de métodos abstratos dos aspectos ou da agregação de uma diferente implementação para um dado tipo abstrato - classe ou interface).

8.2.4. Aspectos de Especialização

Santos et al [108] propõem o conceito de *Aspectos de Especialização* com o objetivo de possibilitar a modularização de pontos flexíveis (*hot-spots*) de frameworks. De acordo com tais pesquisadores, a instanciação de frameworks ocasiona o entrelaçamento e espalhamento de código relacionado diretamente a cada um dos pontos de extensão do framework. Os autores propõem a criação de diversos aspectos abstratos de especialização internos ao framework, os quais são especializados por aspectos da aplicação, durante a instanciação do framework. Diversos benefícios são relatados a partir do uso da abordagem em um estudo de caso, tais como, aumento da configurabilidade e coesão de cada característica relacionada aos pontos flexíveis, redução da complexidade de uso do framework, facilidade de reúso e evolução. Uma das principais desvantagens da abordagem é a criação de uma série de aspectos de especialização, a qual cresce de forma proporcional à quantidade de pontos flexíveis definidos pelo framework.

A principal diferença entre nossa abordagem e a proposta por tais autores é que embora ambas busquem a melhoria da configurabilidade do framework através da gerência de suas variabilidades, nossa abordagem endereça a modularização de características variáveis internas ao framework, enquanto os aspectos de especialização endereçam características oriundas da especialização de cada ponto flexível do framework, e que ocorrem durante a sua instanciação. Assim, as abordagens podem ser usadas de forma complementar, embora isso possa ocasionar a codificação de uma quantidade excessiva (e talvez desnecessária) de aspectos. Nossa abordagem propõe um modelo generativo que habilita a instanciação automática dos pontos flexíveis até mesmo na presença de entrelaçamento e espalhamento de código relacionados à instanciação. Estratégias de uso de modelos de característica ou linguagens específicas de domínio vêm sendo adotadas por várias ferramentas disponíveis na indústria.

8.2.5. Framed Aspects

Frame [11] é uma tecnologia que permite particionar o código-fonte de um sistema ou componente em uma hierarquia de unidades menores, denominadas

frames. Cada *frame* especifica a funcionalidade comum e variável de um dado elemento de implementação. Desenvolvedores podem então definir uma especificação (conjunto de parâmetros) a ser manipulada por um processador de *frames* que então se responsabiliza pela customização da parte varíavel dos frames.

Framed Aspects [89] é uma abordagem que combina as tecnologias de programação orientada a aspectos e *frames* para facilitar a customização automática de aplicações. Aspectos são usados para modularizar interesses transversais e *frames* habilitam a parametrização e configuração de variabilidades encontradas em aspectos. O papel desempenhado pela tecnologia de frames na abordagem *Framed Aspects* é endereçado na nossa abordagem pelo uso de templates de aspectos. Tais templates podem ser customizados baseado em informação coletada por modelos de características.

Outra diferença fundamental entre as abordagens, é que *Framed Aspects* define vários passos de decisão do processo de instanciação no código de templates de *frames* por meio de meta-tags. Na abordagem proposta nesse trabalho, boa parte dessas decisões é especificada separadamente no modelo de configuração. Isso facilita a adaptação ou evolução das decisões relacionadas a customização da arquitetura. Finalmente, nossa abordagem também propõe um conjunto explícito de diretrizes para a modularização de características transversais em frameworks ou arquiteturas de famílias de aplicações os quais não são endereçados pela abordagem *Framed Aspects*.

8.2.6. Abordagem Extrativa de Linha de Produtos

Alves et al [2, 4, 5] definem um método sistemático para a criação e evolução de linhas de produto de software (LPSs). A abordagem propõe a extração de LPSs a partir de produtos existentes, usando refatorações. Essas refatorações são usadas para expor variabilidades requeridas por uma dada LPS. Durante sua evolução, a LPS é então estendida para endereçar novas variabilidades e/ou produtos. O conjunto de refatorações proposto pode ser aplicado tanto no nível de código quanto no modelo de característica da LP.

Aspectos são usados para modularizar características transversais encontradas na implementação de uma LPS.

Nossa abordagem é complementar àquela proposta por Alves et al [2, 4, 5]. As diretrizes apresentadas para modularização de características transversais podem ser usadas durante a extração ou evolução de LPSs. Os EJPs podem também auxiliar os desenvolvedores durante a evolução da LPS, oferecendo um conjunto de pontos de junção candidatos a serem estendidos. Nossa abordagem também oferece uma proposta para especificação do conhecimento de configuração, o qual não é plenamente endereçado por aquele trabalho. Por outro lado a abordagem proposta por tais autores pode trazer mais sistematização a refatoração e evolução de frameworks e linhas de produto, permitindo acompanhar e sincronizar simultaneamente as mudanças aplicadas aos modelos de arquitetura, característica e configuração.

8.3. Abordagens para Instanciação de Frameworks e Linhas de Produto

8.3.1. Abordagem para Instanciação de Frameworks OO

Filho et al [41] propõem uma abordagem para instanciação de frameworks centrada no uso do modelo de característica e da linguagem UML. A abordagem é estruturada em três passos:

(i) *processo de documentação do framework* – um diagrama de classes do framework é relacionado com o seu respectivo modelo de característica. Elementos de projeto (classes, métodos, atributos) que representam pontos de extensão do framework são explicitamente anotados no diagrama de classes representando o framework. Tais elementos de projeto foram definidos como extensões do meta-modelo de UML. Finalmente, relações de dependência UML são definidas para relacionar elementos de ambos os modelos;

(ii) *processo de geração de script de instanciação* – nesse primeiro passo da instanciação do framework, desenvolvedores selecionam as características desejadas para a aplicação que irão criar, e uma ferramenta de posse do modelo de característica e do diagrama de classes representando o framework, gera um script

na linguagem RDL (*Reuse Description Language*) [97] contendo os passos necessários para instanciar a aplicação; e

(iii) *instanciação do framework* – no último passo da instanciação, o script RDL gerado pela atividade anterior é processado para estender o diagrama de classes do projeto original do framework de forma a inserir as classes, métodos e atributos que implementam a concretização dos seus pontos de extensão. Esse processo é semi-automático, intervenções humanas podem ocorrer de forma a solicitar dos desenvolvedores informações adicionais, tais como, nomes de classes, métodos e atributos. Trabalhos recentes têm estendido a abordagem proposta para lidar com a instanciação de frameworks orientados a aspectos [99].

Existem similaridades e diferenças entre nosso modelo generativo para instanciação de frameworks e a abordagem proposta por Filho et al [41]. Ambas as abordagens são centradas na definição de três modelos: de característica, de configuração (ou dependência) e de arquitetura. Nosso modelo de característica, entretanto, incorpora duas propriedades (`<<crosscutting>>` e `<<join point>>`) de forma a oferecer uma representação de aspectos do espaço de solução no espaço de problema. O modelo de arquitetura usado por tais autores é o diagrama de classes UML, enquanto na nossa abordagem tal modelo é uma representação dos artefatos de implementação. As relações de dependência definidas em nosso modelo de configuração têm um propósito similar àquelas propostas por tais autores. Nosso modelo de configuração incorpora informações adicionais, tais como, restrições específicas para definição de relações transversais entre características, assim como especifica o mapeamento de features `<<join point>>` para pontos de junção concretos no espaço de solução. Outra diferença fundamental entre as abordagens, é que são usados templates na nossa abordagem para expressar variabilidades ocorrendo em elementos de implementação da arquitetura, enquanto tais autores usam anotações explícitas em modelos de projeto para representar os pontos de extensão.

8.3.2. Pure::Variants

Existem algumas ferramentas, tais como pure::variants[24] e Gears[25], que foram desenvolvidas para lidar com a derivação e instanciação de famílias de sistemas usando como base modelos de característica. Pure::variants é uma das principais ferramentas disponíveis no mercado. Ela permite especificar linhas de produto ou famílias de sistemas, através da definição de: (i) um modelo de característica; (ii) um modelo da família – que é responsável pela representação dos componentes da arquitetura do sistema; e (iii) modelos do espaço de configuração – definem configurações e transformações, através do modelo de descrição de variantes, que serão utilizadas na instanciação da linha de produto. A ferramenta propõe um modelo que é independente tanto de tecnologias de implementação da arquitetura da linha de produto quanto das técnicas de transformação e geração de código utilizadas.

Existem similaridades e diferenças entre o nosso modelo generativo e a ferramenta pure::variants. Ambos se propõem a instanciar linhas de produto ou famílias de sistemas a partir do modelo de característica. A especificação de relações de dependência entre elementos de implementação e características na ferramenta pure::variants é definida no próprio modelo da família, através de um conjunto de regras de restrição lógica, enquanto na nossa abordagem essas relações são especificadas separadamente no modelo de configuração. Essa especificação separada permite adaptar ou modificar o conhecimento de configuração de forma independente dos modelos de característica e arquitetura. Também é possível que 2 (ou mais) linhas de produto que usem o mesmo conjunto de artefatos (expresso pelo modelo de arquitetura) sejam especificada cada uma com o seu próprio modelo de configuração. Nosso modelo generativo também propõe dois elementos adicionais não contemplados em pure::variants que são as relações válidas entre características to tipo `<<crosscutting>>` e `<<join point>>`, além do mapeamento de características `<<join point>>` para pontos de junção concretos. Tais elementos são diretamente responsáveis pela instanciação das variabilidades OA.

[24] Pure::Variants. URL: http://www.pure-systems.com/, 2007.
[25] Big Lever. Gears. URL: http://www.biglever.com. 2007.