Frameworks Orientados a Aspectos

Valter Vieira de Camargo[1], Paulo Cesar Masiero

Instituto de Ciências Matemáticas e de Computação (ICMC)

Universidade de São Paulo (USP) – Av. do Trabalhador São-carlense, 400

Cep 13.560-970 – São Carlos – SP

{valter, masiero}@icmc.usp.br

Abstract. *This paper presents a definition for aspect-oriented frameworks and classifies them into two types: aspect-oriented application frameworks (AOAF) and crosscutting frameworks (CFs). Several CFs are briefly presented and one of them, namely table-based calculation, has its design discussed in detail. The development of an application is shown to exemplify the instantiation and composition of this type of frameworks.*

Keywords*: Aspects, Frameworks, Aspect-Oriented Frameworks*

Resumo. *Este artigo apresenta uma definição para frameworks orientados a aspectos e os classifica em dois tipos: frameworks de aplicação orientados a aspectos (FAOA) e frameworks transversais (FTs). Vários FTs são brevemente apresentados e um deles, o de cálculo baseado em tabela, tem seu projeto discutido com detalhes. O desenvolvimento de uma aplicação é mostrado para exemplificar a instanciação e a composição de frameworks desse tipo.*

Palavras chave*: Aspectos, Frameworks, Frameworks Orientados a Aspectos*

1 – Introdução

Frameworks orientados a objetos (FOO) são coleções de classes organizadas em uma arquitetura (*design*) abstrata com o objetivo de implementar uma família de problemas relacionados [Johnson, 1997] [Fayad et al, 1999] [Johnson and Foot, 1998]. É extensa a literatura sobre o uso de frameworks, o que mostra sua importância como uma ferramenta de reúso. Eles podem ser vistos como aplicações incompletas que devem ser especializadas para o desenvolvimento de aplicações concretas. Uma outra característica importante dessa tecnologia é que o fluxo de controle da aplicação é invertido, isto é, não é responsabilidade do engenheiro de aplicações determinar esse fluxo, mas do próprio framework [Pree, 1999].

Com o aparecimento da Programação Orientada a Aspectos (POA) [Kiczales et al, 1997] e da linguagem AspectJ [Gradecki and Lesiecki, 2003] como uma extensão orientada a aspectos da linguagem Java, surgiu também o interesse em investigar como esses novos conceitos e mecanismos podem ser utilizados no desenvolvimento de frameworks. Vários autores têm usado o conceito intuitivo de framework orientado a aspectos (FOA) – um framework orientado a objetos que também usa estruturas de

[1] Trabalho realizado com o apoio financeiro da CAPES.

aspectos em sua implementação – sem apresentar uma definição precisa [Constantinides et al, 2000], [Pinto et al, 2002], [Rashid and Chitchyan, 2003], [Rausch et al, 2003], [Soares et al, 2002], [Vanhaute et al, 2001]. Hanenberg foi o único autor encontrado na literatura que define um FOA como uma coleção de aspectos concretos e abstratos [Hanenberg, 2000]. Ele afirma que além dos métodos-gancho (*hook methods*) e métodos-gabarito (*template methods*), conceitos amplamente utilizados em FOO [Gamma et al, 1995], os conjuntos de junção (*pointcuts*) também podem ser especializados, permitindo que um interesse seja acoplado a inúmeras estruturas primárias [Hanenberg, 2000] [Hanenberg e Schmidmier, 2003]. A maioria dos artigos encontrados na literatura sobre frameworks orientados a aspectos trata de interesses transversais (*crosscuting concerns*) não-funcionais, os quais são discutidos brevemente abaixo. Adicionalmente, há na literatura algumas menções ao uso de aspectos para a codificação de regras de negócios [Suvée et al, 2005], [Cibrán et al, 2003], mas não foram encontradas publicações que apresentam exemplos de implementação de aspectos desse tipo na forma de frameworks.

Cibrán mostra como encapsular com aspectos o código que conecta regras de negócio à funcionalidade básica do sistema, com o objetivo de reusar esses "conectores" em outros contextos [Cibrán et al, 2003]. Neste trabalho, além de modularizar os conectores, busca-se também generalizar o comportamento das regras de negócio.

Hanenberg et al apresentam um "framework de aspectos" que oferece alternativas de algoritmos para percurso em grafos [Hanenberg et al, 2004]. Os autores mostram que os mecanismos de composição da programação orientada a objetos (POO) não são adequados para desenvolvimento de frameworks quando a granularidade dos ganchos é pequena e sugerem o uso de aspectos para resolver o problema. Da mesma forma, um "framework de aspectos" (*aspect framework*) para concorrência, desenvolvido em Java puro, foi proposto por Constantinides et al com o objetivo de inserir o interesse transversal de concorrência em uma aplicação [Constantinides et al, 2000]. Os autores apresentam o framework e comentam sobre sua estrutura. Vanhaute et al relatam o desenvolvimento e discutem a arquitetura de um "framework baseado em aspectos" para o interesse de segurança, além disso discutem como o framework pode ser composto com o código-base. Esse framework inclui os sub-interesses de controle de acesso, autenticação e confidencialidade [Vanhaute et al, 2001]. Nesse artigo não está claro se o utilizador do framework possui recursos que permitem instanciar variabilidades relacionadas às regras de segurança implementadas no framework, além de sua simples composição com o código-base.

Rashid e Chitchyan, por sua vez, propõem um FOA para persistência de dados [Rashid and Chitchyan, 2003]. A parte variável possui métodos e mecanismos de junção abstratos que precisam ser concretizados por aspectos especializados para retornar valores específicos da aplicação, como por exemplo, a localização do banco de dados e o nome do *driver* de conexão. Os autores não discutem variabilidades relacionadas à persistência de dados e concentram-se na composição do framework com a aplicação. Isso é feito definindo-se as classes da aplicação como sub-tipos de uma classe do framework por meio de declarações inter-tipo da linguagem AspectJ [Gradecki and Lesiecki, 2003]. Rausch, Rumpe e Hoogendoorn também usam persistência para discutir a necessidade de criar modelos que permitam estudar a solução a ser adotada antes de implementá-la [Rausch et al, 2003]. Eles modelaram esse requisito como um framework e discutem como representar os pontos variáveis como aspectos. Soares et al

propõem implementações dos interesses de persistência e distribuição com aspectos e procuram generalizar o código para reusá-los em outros contextos, contudo como esse não é o foco principal do trabalho, não há definições precisas a respeito de frameworks orientados a aspectos [Soares et al, 2002].

Este artigo tem por objetivo definir e mostrar o uso de frameworks orientados a aspectos e discutir características de seus projetos. As definições apresentadas são baseadas em um estudo prospectivo de desenvolvimento e reutilização de treze frameworks orientados a aspectos, implementados em AspectJ. A discussão e as técnicas apresentadas neste trabalho são gerais e podem ser aplicadas a outras implementações orientadas a aspectos, como por exemplo, AspectS [Hirschfeld, 2002].

A estrutura deste artigo é a seguinte: Na seção 2 são apresentadas as definições e classificações de FOAs. Na Seção 3 são apresentados a biblioteca de frameworks desenvolvida e o projeto de um FOA de regra de negócio. Na Seção 4 exemplifica-se o desenvolvimento de uma aplicação com o framework de regra de negócio apresentado na seção anterior e na Seção 5 encontram-se as considerações finais.

2 – Frameworks Orientados a Aspectos

Do ponto de vista da estrutura, um FOA é um conjunto formado por unidades básicas de programação OO (classes), cuja presença não é obrigatória, e unidades básicas de programação OA (aspectos). A não obrigatoriedade de classes significa que um FOA pode ser composto apenas por aspectos, e embora isso não seja comum, é possível que ocorra em situações especiais. Esse conjunto representa o projeto abstrato de soluções para uma família de problemas relacionados [Johnson and Foot, 1998].

Do ponto de vista do propósito, assim como os FOO [Fayad et al, 1999], um FOA pode ser definido como um sistema semi-completo e reutilizável que pode ser instanciado por um desenvolvedor de aplicações. A arquitetura de um FOA possui uma parte fixa e uma parte variável, sendo que esta deve ser adaptada para que o framework seja acoplado a uma aplicação já existente ou produza uma nova aplicação. A adaptação geralmente envolve a concretização de um mecanismo de composição abstrato, no caso dos aspectos, e de classes e métodos abstratos, no caso das classes.

Quanto à natureza, há dois tipos de FOAs: Frameworks Transversais e Frameworks de Aplicação Orientados a Aspectos. Os dois tipos são discutidos a seguir, dando-se ênfase aos frameworks transversais por serem o foco deste trabalho.

Um **Framework Transversal (*Crosscuting Framework*) (FT)** é um FOA que possui mecanismos de composição abstratos e variabilidades correspondentes a um único interesse transversal, como por exemplo: persistência, distribuição, segurança e regras de negócio. Como um interesse pode ser particionado em sub-interesses, fica a critério do projetista implementar o sub-interesse com aspectos. Caso isso seja feito, também é sua responsabilidade determinar o nível de granularidade desses interesses e se cada um será projetado ou não na forma de um FT. Uma outra característica desse tipo de framework é que ele só pode ser utilizado se for acoplado a algum código-base existente, isto é, seu reúso não produz uma aplicação. Sendo assim, seu processo de reúso possui duas etapas "semanticamente" distintas: instanciação e composição.

A instanciação é o processo convencional de reúso dos FOOs tradicionais e consiste em especializar o código que foi especialmente projetado para isso. É o processo pelo qual implementam-se os ganchos do framework, escolhe-se alguma

funcionalidade alternativa ou implementa-se uma nova. Isso é feito normalmente sobrepondo-se métodos que retornam valores da aplicação específica.

A etapa de composição por sua vez, consiste em duas atividades: identificação dos pontos de junção e fornecimento de regras de composição. A primeira atividade consiste em identificar no código-base o(s) ponto(s) de junção adequado(s) ao acoplamento e deve ser feita com base nas "alternativas de composição" disponíveis no framework. É interessante que os FTs sejam projetados com alternativas de composição, principalmente aqueles que necessitam de dados da aplicação em seu processamento. Essas alternativas aumentam as chances de acoplamento com códigos-base já existentes, além de diminuir a complexidade das regras de composição que precisam ser fornecidas. A necessidade de alternativas de composição torna-se mais evidente quando o framework deve ser acoplado a um código-base já existente. No caso de um novo desenvolvimento, o código-base já pode ser projetado com vistas ao acoplamento que será feito. Contudo, esse "desenvolvimento orientado às alternativas de composição" pode tornar o código-base confuso e difícil de manter, pois pode ser que pontos de junção fictícios tenham que ser criados apenas para o acoplamento.

A segunda atividade da etapa de composição consiste em fornecer regras que unem as variabilidades escolhidas/implementadas do framework com o código-base, e para isso tarefas orientadas a aspectos devem ser realizadas como, por exemplo, a concretização de um mecanismo de composição abstrato ou a utilização de declarações inter-tipo, no caso da linguagem AspectJ [Gradecki and Lesiecki, 2003]. Em alguns casos, a etapa de composição depende de informações que são determinadas na etapa de instanciação, o que determina uma ordem de realização: primeiro a instanciação e depois a composição. Mas também pode ocorrer de não haver essa dependência, fazendo com que essas etapas possam ser feitas em qualquer ordem ou em paralelo.

Embora essas duas etapas sejam semanticamente distintas, sua separação "física" pode não existir durante o reúso, pois isso depende do projeto do framework e da linguagem orientada a aspectos utilizada. Visto que algumas abordagens OA tendem a separar as regras de composição do comportamento transversal [Tarr et al, 2002] [JAML, 2004], infere-se que o projeto de um framework deste tipo deve ser elaborado com o objetivo de separar as duas etapas tanto quanto possível para que a separação semântica continue existindo fisicamente. Contudo, como AspectJ é uma abordagem que mantém as regras de composição no mesmo módulo do comportamento transversal, um bom projeto precisa ser elaborado para separar fisicamente essas duas etapas.

Quando se utiliza FTs não há inversão de controle do ponto de vista da aplicação, pois é ela que determina o fluxo principal. Porém, do ponto de vista interno do framework isso acontece, pois, como qualquer framework orientado a objetos, ele possui código genérico que chama métodos abstratos implementados na aplicação específica. Em outras palavras, não é responsabilidade do engenheiro de aplicações chamar métodos do framework. Pode-se dizer assim que o fluxo de controle principal é da aplicação, enquanto que nos FTs o fluxo de controle é deles. Essa é outra característica diferente das bibliotecas de classes, pois quando elas são utilizadas, o fluxo de controle principal é sempre da aplicação. Nesse sentido, os FTs são similares aos *framelets* [Pree, 1999], pois não assumem o controle da aplicação, têm interface de composição simples e bem definida, e geralmente, mas não necessariamente, têm um número pequeno de unidades de programação (classes e aspectos).

Um **Framework de aplicação orientado a aspectos (FAOA)** é um FOA que implementa uma arquitetura genérica de um domínio e sua instanciação produz uma aplicação desse domínio. A arquitetura do FAOA é projetada com classes e aspectos (implementando provavelmente interesses transversais e regras de negócios) de forma integrada dentro da arquitetura do framework. Eventualmente podem ser acoplados outros FTs que implementem interesses que estejam fora de seu escopo. Os FAOA são bastante parecidos com os frameworks de aplicação OO com relação ao seu propósito e suas diferenças estão na sua arquitetura, que usa aspectos abstratos e concretos, bem como classes abstratas e concretas para implementar partes variáveis de seu código e que, portanto, são concretizados quando ocorre a instanciação de uma aplicação. Nesses frameworks o princípio de inversão de controle continua válido.

2.1 - Formas de Reúso

O reúso de FOAs geralmente é mais abrangente do que de um FOO convencional, já que pode ser utilizado em diferentes domínios. Ele pode ser acoplado a aplicações existentes ou em desenvolvimento, a frameworks OO e a outros frameworks transversais. Há três formas possíveis de reúso quando se trata de frameworks desse tipo e muitas vezes, sua natureza (FT ou FAOA) determina essa forma. As formas possíveis são: 1) apenas instanciação, 2) apenas composição, e 3) instanciação e composição.

A primeira forma ocorre somente com FAOAs porque esses já possuem o código-base embutido em sua estrutura, não necessitando ser acoplados a nenhum outro código-base. Além disso, a maioria de seus interesses transversais já está codificada internamente, assim como suas regras de composição. A segunda ocorre somente com FTs de uma única funcionalidade. Um exemplo é um FT de rastreamento (*tracing*), em que a única funcionalidade é o armazenamento das informações em arquivos de texto. Nesse caso, a única etapa de reúso é a composição. Frameworks desse tipo são muito simples e limitados com relação aos benefícios do reúso. A terceira forma de reúso é um pouco mais complexa porque utiliza as duas etapas. Essa forma ocorre com mais freqüência com FTs que possuem mais de uma variabilidade, mas também pode ocorrer com FAOAs. Um exemplo é um FT de conexão em que suas variabilidades funcionais são relacionadas com a escolha do banco de dados e com o *driver* de conexão. Reusar esse framework consiste em escolher a variabilidade desejada na etapa de instanciação e também fazer a composição com algum código-base.

Nas duas últimas formas de reúso, a composição pode ocorrer entre dois ou mais FTs e regras de prioridade de atuação precisam ser definidas. Os FTs são os que ocorrem mais comumente nessa categoria, contudo, nada impede a existência de FAOA cujo processo de instanciação também inclua a etapa de composição.

A classificação mostrada nesta sub-seção é válida para frameworks caixa-branca. Quando o framework é caixa-preta, não importa como foi projetada a arquitetura da parte variável, pois todas as variabilidades já foram concretizadas e trata-se apenas de um processo de escolha que geralmente é feito com auxílio automatizado.

3. Projeto de um Framework Transversal de Regra de Negócio

As definições apresentadas na Seção 2 são baseadas na experiência de desenvolvimento de treze FTs que formam uma estrutura que pode ser utilizada no desenvolvimento de aplicações. Esta seção apresenta sucintamente esses frameworks e mostra com mais

detalhes o projeto de um deles para ilustrar os principais pontos das definições apresentadas anteriormente.

A Tabela 1 apresenta os treze frameworks desenvolvidos e suas principais características. A terceira coluna classifica-os como de produção (*P*) ou de desenvolvimento (*D*). Os frameworks de produção são aqueles que acompanham a aplicação final e fazem parte de sua funcionalidade, enquanto que os de desenvolvimento apóiam atividades de desenvolvimento, como teste e depuração, mas não fazem parte do código final. Apenas dois frameworks são de desenvolvimento: o de rastreamento (*tracing*) e o de garantia de políticas (*policy enforcement*). A quarta coluna mostra a classificação de cada FT em relação ao reúso, conforme Seção 2.1, e informa se a composição deve ser feita antes da instanciação (separação por virgulas) ou em paralelo (separação por uma barra). A quinta coluna mostra as alternativas de composição que alguns frameworks possuem. Elas ampliam as chances de acoplamento com inúmeros códigos base e permitem o fornecimento de regras de composição mais simples durante a instanciação. Existe um número mais ou menos conhecido de alternativas de composição, contudo apenas com a reutilização do framework em novos contextos é possível descobrir outras alternativas. A sexta coluna mostra as variabilidades funcionais do framework, e a sétima é relacionada com a granularidade do interesse transversal, mostrando sub-interesses que foram aspectualizados. A maioria foi transformada em FTs que tratam desse sub-interesse.

Em relação à quinta coluna da Tabela 1, os interesses que possuem mais de uma alternativa de composição geralmente possuem uma alternativa *default* (representadas por um asterisco), que será utilizada caso nenhuma outra seja escolhida. Isso normalmente ocorre com FTs que são sub-interesses utilizados por outros frameworks. Por exemplo, como *Pooling* é um sub-interesse do framework de persistência [Camargo et al, 2003] e é desejável sua utilização conjunta, já existe uma alternativa de composição escolhida e conseqüentemente, uma regra de composição pré-definida com o framework de persistência. Isso implica que ao utilizar o sub-interesse de *pooling* em uma aplicação, que já usa o framework de persistência, não será necessário fornecer regras de composição para o acoplamento. Ressalta-se que o reúso em outros contextos pode ocorrer, desde que novas regras de composição sejam definidas. Na sexta coluna também é possível encontrar asteriscos representando variabilidades *default*.

Os frameworks de 1 a 9, 12 e 13 tratam exclusivamente de requisitos não-funcionais e podem ser utilizados em domínios mais amplos. Os frameworks 10 e 11 modularizam interesses transversais funcionais, que implementam regras de negócio que ficam espalhadas e/ou entrelaçadas com a funcionalidade básica do sistema quando são implementadas com técnicas orientadas a objetos.

Para exemplificar o projeto de FTs, é mostrado o "framework de cálculo baseado em tabela". Esse framework modulariza uma regra de negócio que tem o objetivo de incrementar ou decrementar um valor capturado do código-base por meio de um cálculo percentual, o qual depende da faixa em que o valor capturado se enquadra. Esse cálculo pode fazer uso de um redutor, que é um valor fixo que será subtraído do cálculo percentual. Regras de negócio típicas que podem ser tratadas por esse framework são, por exemplo: um valor de vale-refeição que pode ou não ser acrescido ao salário do funcionário dependendo de sua faixa salarial; um desconto na conta de água para usuários cujo consumo esteja, por exemplo entre 11 e 15 m^3 e um bônus para

funcionários cujo número de horas extras mensal é superior a 30. A estrutura da tabela utilizada no cálculo é fixa, mas os valores podem variar conforme a regra de negócio em uso. A Tabela 2 mostra a estrutura da tabela e valores para o cálculo do imposto de renda, que também é uma das regras de negócio que pode ser tratada por esse framework e que faz uso do redutor.

Tabela 1 – Frameworks Transversais

No.	*Framework*	*Tipo*	*Forma de reúso*	*Alternativas de composição*	*Variabilidades funcionais*	*Sub-interesses aspectualizados*
1	Persistência	*P*	Instanciação/ composição	Padrão	Consciência total*	Conexão, Memória auxiliar, pooling e controle de objetos sujos
					Consciência parcial	
2	Conexão	*P*	Instanciação, composição	Padrão	Driver nativo	
					Driver de ODBC*	
					mySql*	
					Interbase	
					SyBase	
3	Memória auxiliar "Carregador de Objetos"	*P*	Composição	Com retorno*	Não	
				Sem retorno		
4	Memória auxiliar "Todas Tuplas"	*P*	Composição	Com retorno*	Não	
				Sem retorno		
5	Pooling	*P*	Composição	Com retorno	Não	
				Sem parâmetros (this)*		
				Sem parâmetros(target)		
				Sem parâmetros (args)		
6	Controle de acesso	*P*	Composição	Padrão	Não	Autenticação, registro de acesso e persistência.
7	Registro de acessos	*P*	Instanciação, Composição	Sem parâmetro (this)	Em arquivo	persistência
				Sem parâmetro (target)	Em tabela*	
				Sem parâmetro (args)	Escolha dos dados	
				Com parâmetros*		
8	Autenticação	*P*	Instanciação, Composição	Padrão	Não	Políticas de autenticação
9	Políticas de autenticação	*P*	Instanciação, Composição	Com retorno*	Erros sucessivos*	
				Sem retorno	Erros não sucessivos	
10	Reajuste de valor	*P*	Instanciação, Composição	Padrão	Reajuste linear para todas categorias	
					Reajuste linear para categorias específicas	
					Reajuste Não – linear	
11	Cálculo baseado em tabela	*P*	Instanciação, Composição	Com retorno	Incremento com redutor	Persistência
				Sem parâmetro (this)		
				Sem parâmetro (target)	Incremento sem redutor	
				Sem parâmetro (args)		
					Decremento com redutor	
				Com parâmetros	Decremento sem redutor	
12	Garantia de Políticas	*D*	Composição	Padrão	Não	
13	Rastreamento	*D*	Instanciação/ composição	Padrão	Saída em arquivo convencional*	persistência
					Saída em arquivo XML	
					Saída na tela	

Tabela 2 – Tabela do Framework de Regra de Negócio para o Cálculo do Imposto de Renda

Base de cálculo	Porcentagem	Redutor
Até R$ 1.164,00	0	0
De R$ 1.164,01 a R$ 2.326,00	15%	R$ 176,60
Acima de R$ 2.326,00	27,5%	R$ 465,35

O modelo de projeto do framework de cálculo baseado em tabela pode ser visto na Figura 1. Os aspectos estão sendo representados por classes com estereótipo `<<aspect>>` apenas para facilitar a compreensão. Os métodos que possuem o estereótipo `<<hook>>` são abstratos e devem ser estendidos pelo engenheiro da aplicação. Esse estereótipo auxilia a documentar o framework, facilitando a identificação dos pontos de extensão [Fontoura et al, 2002]. Os conjuntos de junção e os adendos são representados com notas anexadas aos aspectos.

Como pode ser visto na Tabela 1, o framework número 11 possui quatro variabilidades funcionais e cinco alternativas de composição. A instanciação deve ser feita por meio das classes abstratas `WithoutDeductible` e `WithDeductible`, a primeira deve ser escolhida quando a regra de negócio possui um redutor e a segunda quando não possui. Para fazer a instanciação o engenheiro de aplicação deve criar uma subclasse que representa a regra de negócio que se deseja tratar, definir seus construtores e implementar o método-gancho `isIncrement()`, o qual permite escolher o algoritmo de incremento ou de decremento. Deve ser criada uma subclasse e uma tabela no banco de dados para cada regra de negócio a ser tratada pelo framework. Caso seja de interesse do engenheiro de aplicação, uma nova variabilidade pode ser adicionada na hierarquia especializando-se a classe `Calculation`. Um exemplo de uma instanciação é apresentado na seção seguinte.

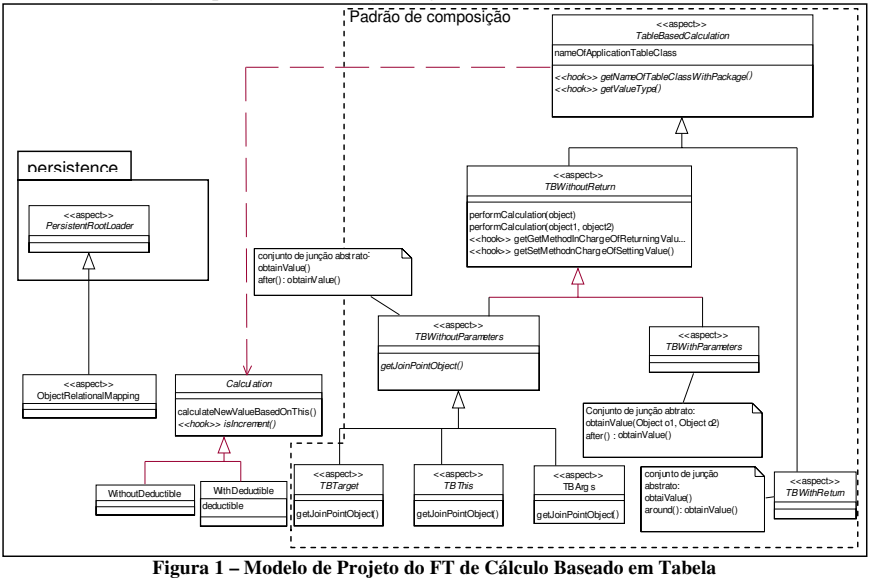

Figura 1 – Modelo de Projeto do FT de Cálculo Baseado em Tabela

Como descrito anteriormente, a etapa de composição consiste em duas atividades: identificação dos pontos de junção e fornecimento das regras de composição. Para identificar o(s) ponto(s) de junção mais adequado(s) deve-se analisar as especificações do framework, analisar o que ele requer do ponto de junção e verificar quais são as alternativas de composição possíveis. No caso deste framework, o importante é que por meio do ponto de junção escolhido seja possível obter o valor que será submetido ao cálculo da regra de negócio. Como há várias formas de se obter esse valor deve-se analisar as alternativas de composição do framework para identificar como cada uma delas obtém o valor e procurar um ponto de junção adequado. Com isso, pode-se decidir que ponto de junção será fornecido na regra de composição e, assim, qual será a forma de captura do valor, se por meio de um método que retorna o valor, se por meio de um objeto que possui o valor, ou alguma outra forma. As cinco alternativas de composição disponíveis podem ser vistas na Tabela 3. Elas constituem um padrão (de projeto) de composição para FTs que está destacado com uma figura

geométrica tracejada na Figura 1. Esse padrão, ou parte dele, se repete também nos frameworks de *Pooling* e de Registro de acessos, como pode ser visto na Tabela 1.

O fornecimento das regras de composição deve ser feito criando-se aspectos especializados que correspondem a cada regra de negócio tratada, isto é para cada subclasse de `WithoutDeductible` e/ou `WithDeductible`. Os aspectos que podem ser estendidos pelo engenheiro de aplicação são: `TBTarget`, `TBThis`, `TBArgs`, `TBWithParameters` e `TBWithReturn`, os quais representam cada uma das alternativas de composição do padrão mostrado na Tabela 3. A criação desses aspectos também exige a implementação de alguns métodos-gancho, como por exemplo `getNameOfTableClassWithPackage()`, que é um método-gancho do aspecto `TableBasedCalculation`. Esse método deve retornar o nome da classe que representa a regra de negócio tratada, fazendo uma correspondência com a subclasse que foi criada na etapa de instanciação. A implementação desse método gancho é um exemplo de um fator que determina a ordem do processo de reúso deste framework. Como o nome da classe retornada é determinado na etapa de instanciação, não seria possível realizar a composição antes da instanciação.

Tabela 3 – Alternativas de Composição do FT de Cálculo Baseado em Tabela

Nome	Descrição
Com Retorno	Esta alternativa consiste em estender o aspecto `TBWithReturn` e fornecer um ponto de junção cujo retorno é o valor ser modificado. Esse ponto de junção pode ser tanto uma chamada, quanto a execução de um método.
Sem parâmetro (this)	Esta alternativa consiste em estender o aspecto `TBThis` e fornecer um ponto de junção que a partir do seu objeto `this` pode-se obter o valor a ser modificado. Também deve ser possível atribuir o valor modificado a esse mesmo objeto. Esse ponto de junção pode ser uma chamada ou execução de um método, ou o acesso a um atributo.
Sem parâmetro (target)	Semelhante à alternativa anterior, porém o aspecto a ser estendido é o `TBTarget` e objeto `target` do ponto de junção fornecido é que será utilizado para obtenção e atribuição do valor modificado.
Sem parâmetro (args)	Semelhante às duas alternativas anteriores, porém o aspecto a ser estendido é o `TBArgs` e o primeiro argumento do ponto de junção fornecido é que será utilizado para obtenção e atribuição do valor modificado.
Com parâmetros	Esta alternativa consiste em estender o aspecto `TBWithParameters` e fornecer um ponto de junção com dois parâmetros, em que o primeiro parâmetro é um objeto que pode ser utilizado para obter o valor a ser modificado, e o segundo objeto deverá poder receber o valor modificado.

A necessidade de várias alternativas de composição ocorre com mais freqüência em FTs que precisam de dados do código-base para realizar suas responsabilidades. Como a obtenção desses dados pode ocorrer de várias formas, o framework deve fornecer antecipadamente várias alternativas de captura desses dados. Por exemplo, o código-base pode possuir um ponto de junção que simplesmente retorna o valor desejado, facilitando sua captura. Essa é a situação ideal, porém nada garante que sempre será assim, pois pode ocorrer que o valor desejado não seja tão explícito no ponto de junção. Neste caso, muitas vezes é necessário utilizar reflexividade para a obtenção dos dados. Assim, o engenheiro da aplicação deve fornecer, além da regra de composição, informações mais detalhadas da aplicação, como por exemplo, o nome de métodos. Nesse caso, o framework deve invocar esses métodos reflexivamente para a obtenção e tratamento dos valores desejados. Por exemplo, o aspecto `TBWithoutReturn` mostrado na Figura 2 utiliza os métodos abstratos `getGetMethodInChargeOfGettingValue` e `setSetMethodInChargeOfSettingValue` para obter o valor a ser modificado e atribuir o valor modificado, respectivamente. Esses métodos-gancho devem ser concretizados pelo engenheiro da aplicação, retornando o nome de métodos que podem ser utilizados para este fim.

Como o framework de cálculo mostrado na Figura 1 necessita do interesse de persistência para manipular as tabelas de cálculo, o aspecto

`ObjectRelationalMapping` mantém um relacionamento de herança com o framework de persistência (representado pelo pacote `persistence` na Figura 1) [Camargo et al, 2003] para permitir manipular objetos no banco de dados. Para completar a atividade de composição, o engenheiro da aplicação deve criar um aspecto que especialize o aspecto `ObjectRelationalMapping` e utilizar declarações inter-tipo da linguagem AspectJ, fazendo com que as classes filhas de `WithoutDeductible` e `WithDeductible` sejam persistentes.

É interessante destacar que o projeto mostrado na Figura 1 evidencia algumas características da definição de FTs apresentada na Seção 2. Por exemplo, a existência de mecanismos de composição abstratos e conjuntos de junção, no caso da linguagem AspectJ, é representada pelas notas. As variabilidades são tratadas por classes normais. A divisão semântica entre escolha de variabilidades e composição mantém-se fisicamente separada neste projeto, em que a primeira etapa é feita por meio de classes e a segunda por meio de aspectos. Além disso, pode-se perceber que não é possível instanciar este framework sem um código-base, o que não ocorre com os FAOAs.

```
public abstract aspect TBWithoutReturn extends TableBasedCalculation {
  public void performCalculation(Object object)
  {  ...
    float valueToBeModified =
     object.getClass().getMethod(getGetMethodInChargeOfGettingTheValue(),…).invoke(…);

    modifiedValue = //perform calculation

    object.getClass.getMethod(getSetMethodInChargeOfSettingTheValue(),…).invoke(
       …,…modifiedValue);
    ...
   }
    public abstract String getSetMethodInChargeOfSettingTheValue ();
    public abstract String getGetMethodInChargeOfGettingTheValue ();
```

Figura 2 – Uso de Reflexividade para fornecer alternativas de composição

Ressalta-se também que o processo de escolha de alternativas de composição pode incluir uma atividade de exclusão das alternativas que não foram escolhidas, deixando a aplicação apenas com o código que é estritamente necessário. Por exemplo, caso a alternativa de composição `TBWithReturn` tenha sido utilizada, toda a hierarquia do aspecto `TBWithoutReturn` e inclusive ele, poderiam ser eliminados, facilitando manutenções e não sobrecarregando o código objeto com trechos não executáveis.

4. Desenvolvimento de uma Aplicação com Frameworks Transversais

Para ilustrar o reúso dos FTs, foi desenvolvido um sistema de oficina de aparelhos eletrônicos. Esse sistema inclui um sub-sistema de pessoal que controla a remuneração dos funcionários, além de realizar o fechamento mensal e gerar a folha de pagamento. O salário base varia conforme a função que exercem na oficina. O cálculo do salário mensal ocorre com base em eventos variáveis, eventos fixos e deduções legais que ocorrem durante o mês. Os eventos variáveis podem ser, por exemplo: uma falta, algumas horas de atraso, determinadas comissões e horas extras. Um exemplo de evento fixo é o cálculo do valor do vale-refeição e um exemplo de uma dedução legal é o imposto de renda. O cálculo do imposto de renda segue as regras definidas na Tabela 2 e o cálculo do vale-refeição as definidas na Tabela 4. A diferença entre essas regras é que o cálculo do imposto de renda é uma regra de decremento com redutor, enquanto que o do vale-refeição é de incremento sem redutor.

Os onze FTs de produção apresentados pela Tabela 1 foram utilizados nesta aplicação. A Figura 3 mostra um diagrama de pacotes que representa a arquitetura de todo o sistema após o processo de instanciação.

Tabela 4 – Tabela para Cálculo do Vale-Refeição

Base de cálculo	Porcentagem
Até R$ 600	10%
De R$ 600,01 a R$ 1.000,00	8%
Acima de R$ 1.000,01	5%

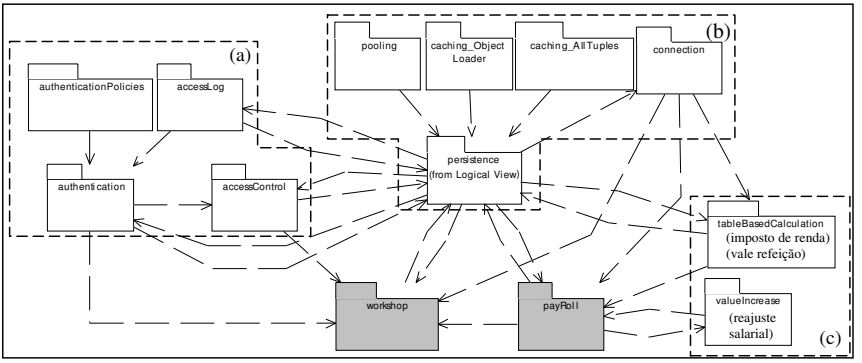

Figura 3 – Diagrama de Pacotes do Sistema de Oficina Eletrônica

As dependências existentes entre os pacotes apresentados na Figura 3 denotam que um aspecto criado pelo engenheiro de aplicação possui regras de composição que adicionam comportamento nos pontos de junção do código-base de outro pacote. Por exemplo, após instanciar o framework `connection`, regras de composição foram definidas para adicionar comportamento nos pontos de junção dos pacotes `workshop` e `payRoll`, criando assim uma dependência do pacote `connection` com esses dois pacotes, os quais representam a aplicação. Foge ao escopo deste trabalho discutir o processo de análise e projeto que levou a essa solução.

As figuras geométricas com linhas tracejadas que envolvem alguns frameworks mostram agrupamentos semânticos. Por exemplo, o agrupamento representado com a letra (a) é semanticamente relacionado com o interesse de segurança, o representado com a letra (b) com persistência e o com a letra (c) com regras de negócio. Esse agrupamento aumenta o nível de abstração e mostra uma estrutura de três famílias de FTs, uma para segurança, uma para persistência e uma para regras de negócio.

O framework de persistência, representado pelo pacote `persistence` na Figura 3, possui uma natureza diferente dos demais porque os pacotes que o utilizam dependem dele. Isso ocorre porque durante o desenvolvimento da aplicação, as classes que representam o código cliente são desenvolvidas com consciência de algumas operações de persistência. Operações relativas a armazenamento e atualização ("`save()`" e "`update()`) não precisam ser conhecidas durante o desenvolvimento desde que a variabilidade funcional "consciência parcial" seja utilizada (Tabela 1). Porém, o mesmo não ocorre com as operações de remoção e busca ("`remove()`" e "`findLike()`), pois elas devem ser explicitamente chamadas por meio de interfaces em momentos definidos pelo engenheiro da aplicação [Rashid and Chitchyan, 2003].

Não há nenhum problema em ter consciência do interesse transversal durante o desenvolvimento, visto que isso é uma característica desejável, mas não obrigatória, da orientação a aspectos [Filman and Friedman, 2004].

O framework de cálculo baseado em tabela, representado pelo pacote `TableBasedCalculation`, foi utilizado para tratamento das regras de negócio referentes ao vale-refeição e ao imposto de renda. A Figura 4 mostra esse framework instanciado para o sub-sistema de pessoal. Para economizar espaço, a parte superior da figura mostra apenas uma parte do framework apresentado pela Figura 1. A parte intermediária mostra as classes e aspectos criados pelo engenheiro de aplicação durante o processo de reúso e a parte inferior mostra as classes do sistema de pessoal.

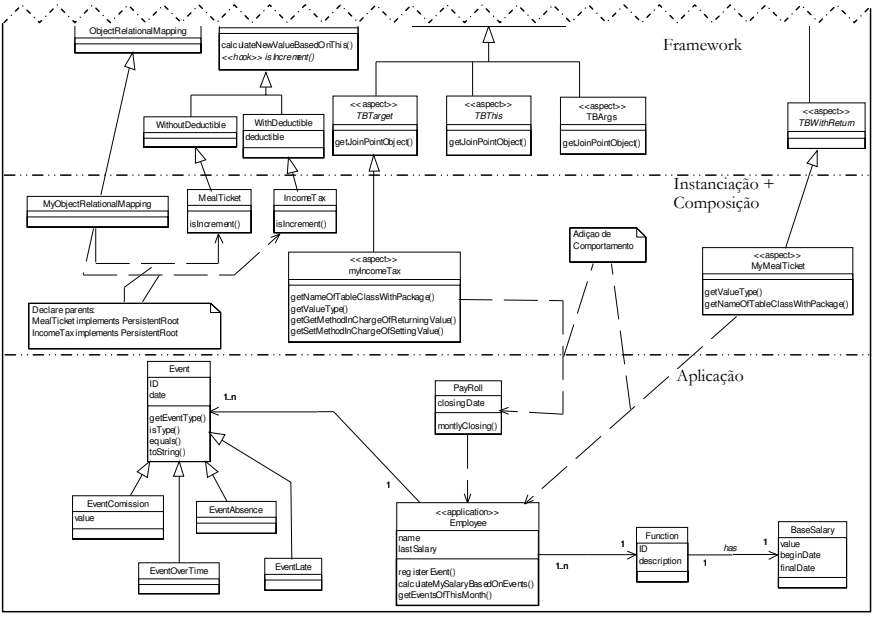

Figura 4 – Framework de Cálculo Baseado em Tabela Instanciado

A instanciação é feita estendendo-se as classes `WithoutDeductible` e `WithDeductible`. Sendo assim, as classes `IncomeTax` (Imposto de renda) e `MealTicket` (vale-refeição) foram criadas para representar as regras de negócio que devem ser tratadas. A Figura 5 mostra o código fonte desta segunda classe com os métodos que ela deve conter, isto é, com dois construtores, um vazio e um normal, e o método `isIncrement()`, que permite escolher um dos algoritmos alternativos do framework. O nome dessas subclasses é de escolha do instanciador, porém, deve ser o mesmo nome das tabelas do banco de dados que elas representam. O código da classe `IncomeTax` não é mostrado porque é similar ao da classe `MealTicket`.

Como mencionado anteriormente, a etapa de composição consiste em identificar os pontos de junção apropriados no código-base e escolher a alternativa de composição mais adequada. O ponto de junção que foi identificado no código-base para a

composição da regra de cálculo do vale-refeição é mostrado na Figura 6. O método apresentado é responsável por efetuar o fechamento mensal com base nos eventos fixos e variáveis do funcionário. O cálculo realizado por este método inicia obtendo o valor do salário base da função exercida pelo funcionário e prossegue realizando acréscimos e deduções baseados nos eventos variáveis ocorridos durante o mês. Assumindo-se que o valor do vale-refeição deve ser acrescido ao salário do funcionário antes dos descontos legais, este é um bom momento para realização do cálculo. Portanto, foi identificado que por meio da chamada do método `getValue()` o valor do salário base poderia ser capturado com a alternativa de composição "com retorno", já que esse método retorna o valor sujeito ao cálculo. Assim, foi criado o aspecto `MyMealTicket` (Figura 7), que estende o aspecto `TBWithReturn` e que permite compor a regra de negócio de cálculo do vale-refeição (representada pela classe `MealTicket`) com o código-base.

```
public class MealTicket extends WithoutDeductible {

       public MealTicket()          {               }
       public MealTicket(...)       {  super (...); }
       public boolean isIncrement(){  return true; }
```

Figura 5 – Escolha de Variabilidades

```
public void calculateMySalaryBasedOnEvents(int month, int year){
     ...
     this.lastSalary = this.getFunction().getBaseSalary().getValue();

     for (int i =0; i < eventsOfThisMonth.size(); i++){

            event = (Event)eventsOfThisMonth.elementAt(i);
            ...
 }
```

Ponto de Junção Escolhido

Figura 6 – Ponto de Junção na Aplicação

A Figura 7 mostra o código do aspecto `MyMealTicket`. Como definido na especificação do framework, quando se utiliza a alternativa de composição "Com Retorno", o ponto de junção fornecido pelo engenheiro de aplicação deve ser uma chamada ou execução de um método que retorne o valor a ser modificado. Portanto, a regra de composição definida entrecorta chamadas ao método `getValue()` da classe `BaseSalary` originadas do método `calculateMySalaryBasedOnEvents`() da classe `Employee`, como representado pela dependência entre o aspecto `MyMealTicket` e a classe `Employee` na Figura 4. Essas chamadas retornam o valor do salário base do funcionário, que é o valor que deve ser modificado com base na tabela de vale-refeição. Nesse mesmo aspecto, o método `getNameOfTableClassWithPackage()` também deve ser implementado para informar ao framework o nome da classe que representa a regra de negócio a ser considerada. Isto é, quando a chamada ao método `getValue()` acontecer, a regra de negócio que deve ser aplicada é definida pela classe `MealTicket`. A implementação do método `getValueType()` também é obrigatória pois define o tipo primitivo do valor que será modificado.

A existência de um ponto de junção que retorne o valor desejado é a situação ideal de acoplamento, como ocorreu no exemplo acima. Contudo, isso pode não ocorrer e, neste caso, uma outra alternativa de composição deve ser escolhida. Por exemplo, o acoplamento da regra de negócio de cálculo do imposto de renda foi feito com a alternativa "sem parâmetro (*target*)", pois não havia um ponto de junção no código-base

que retornasse o salário que é base para esse cálculo. A Figura 8 mostra o ponto de junção que foi utilizado para essa composição. Essa linha de código pertence a um método do sistema de folha de pagamento responsável por efetuar o fechamento mensal. Utilizando a alternativa de composição "sem parâmetro (target)" o framework captura o objeto `employee` e usa reflexividade para capturar e alterar o valor do salário.

```
public aspect MyMealTicket extends TBWithReturn {

       public pointcut obtainValue():
          call (* BaseSalary.getValue())
          && withincode (* Employee.calculateMySalaryBasedOnEvents(int, int));
       public String getNameOfTableClassWithPackage(){
              return "tableBasedCalculation.instantiation.MealTicket";
       }
       public String getValueType(){ return "float";}
 }
```

Figura 7 – Aspecto da Regra de Negócio de Cálculo do Vale-Refeição

Note-se que a utilização de reflexividade é fundamental para bons projetos de acoplamento, principalmente para frameworks que necessitam obter valores da aplicação. Geralmente a obtenção desses valores não é tão simples e o responsável pelo fornecimento das regras de composição deve ter conhecimento dos pontos de junção adequados e de como esses valores podem se capturados.

```
...
employee.calculateMySalaryBasedOnEvents(initialDate.getMonth(),2004);
...
```

Figura 8 – Ponto de Junção para o Acoplamento do Imposto de Renda

Os cálculos do imposto de renda e do vale-refeição são apenas exemplos concretos de possíveis instanciações do framework de cálculo baseado em tabela. Em sistemas de grande porte o cálculo generalizado por este framework pode ser utilizado por outras regras de negócio, com características próprias e que podem ou não ser temporárias. O uso do mesmo cálculo a partir de vários pontos do sistema torna o código de diferentes interesses espalhado e entrelaçado. Além disso, como geralmente as regras de negócio tendem a se alterar com muito mais freqüência do que a funcionalidade do sistema, mantê-las em módulos separados traz benefícios para modularidade e, conseqüentemente, para a manutenção.

5. Considerações Finais e Trabalhos Futuros

Este trabalho apresentou uma definição e uma classificação para Frameworks Orientados a Aspectos (FOAs), concentrando-se em Frameworks Transversais (FTs). Vários frameworks desse tipo foram brevemente apresentados e um deles, o de cálculo baseado em tabela, foi mostrado com maiores detalhes. Mostrou-se também como esses frameworks podem ser instanciados e compostos com o código-base de uma aplicação.

Adicionalmente, mostrou-se como projetar o framework de uma forma que mantenha as diferenças semânticas entre instanciação e composição também no projeto. O uso do padrão de composição auxilia nesse sentido, além de definir uma ordem no processo de reúso, facilitar a manutenção do framework e diminuir a complexidade das regras de composição. Este último item é particularmente importante no processo de reúso, pois diminui a ocorrência de erros do engenheiro de aplicação. Um trabalho anterior destacou algumas diretrizes que podem ser utilizadas para se projetar variabilidades em frameworks transversais [Camargo e Masiero, 2004]. O objetivo foi

tanto permitir que regras de composição mais simples fossem fornecidas, quanto restringir a abrangência de atuação dessas regras, preservando o comportamento que não deveria ser afetado pelo framework. O padrão de composição apresentado aqui complementa esse trabalho anterior em um nível de abstração mais alto, pois as diretrizes podem ser utilizadas para projetar os mecanismos de composição abstratos dentro da hierarquia de classes do padrão.

Famílias de FTs para vários domínios pode ser uma alternativa à utilização de grandes frameworks, ou mesmo uma forma de modularizá-los, assim como ocorre com *framelets* [Pree, 1999]. A principal característica dos FTs, que é a capacidade de se (des)acoplar facilmente ao código-base, pode trazer grandes benefícios para atividades de manutenção. Um sistema desenvolvido por meio de um processo apoiado por FTs conterá em sua estrutura apenas os interesses estritamente necessário à sua funcionalidade final. Novos requisitos podem ser tratados adicionando-se FTs à estrutura do sistema, sem a necessidade de modificações invasivas. Os FTs também pode ser projetados como opções configuráveis para o desenvolvimento de um sistema, ou mesmo como uma característica (*feature*) de uma linha de produtos.

Em continuação a este trabalho, estão sendo desenvolvidas novas aplicações para testar os frameworks e para definir um processo de desenvolvimento de aplicações orientado a aspectos apoiado por FTs. Duas aplicações já foram desenvolvidas, uma de loja de venda de CDs e outra de gerenciamento de contas bancárias.

Referências Bibliográficas

Camargo, V.V., Ramos, R.A., Penteado, R.A.D. and Masiero, P.C. (2003) "Projeto Orientado a Aspectos do Padrão Camada de Persistência". In: Anais do 17º Simpósio Brasileiro de Engenharia de Software (SBES), Manaus-Amazonas, Brasil, outubro.

Camargo, V.V., Ramos, R.A. and Masiero, P.C. (2004) "Implementação de Variabilidades em Frameworks Orientados a Aspectos desenvolvidos em AspectJ". In: Relatório do 1º Workshop de Desenvolvimento de Software Orientado a Aspectos (WASP'04) – realizado em conjunto com o SBES'2004, Brasília, DF, Brasil, outubro.

Cibrán, M., D'Hondt, M. and Jonckers, V. (2003) "Aspect-Oriented Programming for Connecting Business Rules". In: Proc. of the 6th International Conference on Business Information Systems (BIS'03). Colorado Springs, USA, June.

Constantinides, C.A., Bader, A., Elrad, T.H. and Fayad, M.F. (2000) "Designing an Aspect-Oriented Framework in an Object-Oriented Environment". ACM Computing Surveys, v.32, march.

Fayad, M. E., Schimidt, D. C. and Johnson, R. (1999) Building application frameworks: Object-oriented foundations of framework design. John Wiley & Sons.

Filman, R. and Friedman, D. (2004) Aspect-Oriented Programming is Quantification and Obliviousness in Aspect-Oriented Software Development. Addison-Wesley.

Fontoura, M., Pree, W. and Rumpe, B. (2002) The UML Profile for Framework Architectures. Addison Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Gradecki, J.D. and Lesiecki, N. (2003) Mastering AspectJ – Aspect Oriented Programming in Java. Wiley Publishing.

Hanenberg, S. (2000) "Multi-Design Application Frameworks". In: Proc. of the Generative and Component-Based Software Engineering Young Researchers Workshop, Erfurt, October 10.

Hanenberg, S., Hirschfeld, R., Unland, R. and Kawamura, K. (2004) "Applying Aspect-Oriented Composition to Framework Development – A Case Study". In: Proc. of the 1st International Workshop on Foundations of Unanticipated Software Evolution, Barcelona, Spain, march 28.

Hanenberg, S., Schmidmeier. Idioms for Building Software Frameworks in AspectJ. 2nd AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Boston, MA, March 17, 2003.

Hirschfeld, R. Aspect-Oriented Programming with AspectS. In: Proceedings of Net.ObjectDays (NODe), Erfurt – Germany, 2002.

Johnson, R. E. (1997) "Components, frameworks, and patterns". In: Proc. of the ACM Symposium on Software Reusability, (SST' 97), Boston, may, 17-20.

Johnson, R. E. and Foot B. (1998) "Designing Reusable Classes". Journal of Object-oriented Programming, 1 (2), 22-35.

JAML. (2004) http://www.ics.uci.edu/~trungcn/jaml. Último acesso em 3/4/2005.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. and Irving, J. "Aspect Oriented Programming". In: Proceedings of 11 ECOOP. pp. 220-242, 1997.

Tarr, P., Ossher, H. and Sutton, S. (2002) Hyper/J™ : Multi-dimensional Separation of Concerns for Java**.** In: Proc. of the 24th International Conference on Software Engineering. Orlando, Florida.

Pinto, M., Fuentes, L., Fayad, M.E. and Troya, J.M. (2002) "Separation of Coordination in a Dynamic Aspect Oriented Framework". In: Proc. of the 1st International Conference on Aspect-Oriented Software Development, April.

Pree, W. Hot-spot-driven development in M. Fayad, R. Johnson, D. Schmidt. (1999) Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 379–393.

Rashid, A. and Chitchyan, R. (2003) "Persistence as an Aspect". In: Proc. of the 2nd International Conference on Aspect Oriented Software Development (AOSD) Boston–USA, March.

Rausch, A., Rumpe, B. and Hoogendoorn, L. (2003) "Aspect-oriented Framework Modeling". In: Proc. of the 4th AOSD Modeling with UML Workshop (UML Conference 2003) October.

Soares, S., Laureano, E. and Borba, P. (2002) "Implementing Distribution and Persistence Aspects with AspectJ". In: Proc. the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pp 174-190, november.

Suvée, D., Fraine, D.B. and Vanderperren, W. (2005) "FuseJ: An Architectural description language for unifying aspects and components". In: Proc. of the 1st Workshop on Software Engineering Properties of Languages for Aspect Technologies (SPLAT'05), Chicago.

Vanhaute, B., Win, B. and Decker, B. (2001) "Building Frameworks in AspectJ". In: Proc. of the 15th European Conference on Object-Oriented Programming (ECOOP), Separation of Concerns Workshop. pp. 1-6, June.