# AspectJ em 20 minutos

Diogo Vinícius Winck (diogo.winck@gmail.com)
Vicente Goetten (goetten@gmail.com)

## 1. Introdução

A Orientação a Objetos está consolidada. Entretanto, nem todas as promessas que este paradigma se propunha a cumprir foram atendidas. Algumas soluções foram propostas na tentativa de cumprir estas promessas pendentes, entre elas a Orientação a Aspectos.

A Arientação a Aspectos (OA) é um paradigma que estende a Orientação a Objetos (e outros, como o paradigma estruturado) introduzindo novas abstrações. Estes novos elementos são destinados a suprir deficiências na capacidade de representação de algumas situações.

Este artigo foi criado baseado no **capítulo 3** e outros fragmentos do livro: ***AspectJ* – Programação Orientada a Aspectos com Java**, escrito por nós e lançado no início de julho pela Editora Novatec. O nosso objetivo com este artigo é *sobrevoar* os assuntos ligados com a AO e com o *AspectJ*.

Sobre o livro:
**http://www.novateceditora.com.br/livros/poa/**

**http://www.submarino.com.br/books_productdetails.asp?Query=ProductPage&ProdTypeId=1&ProdId=1572973&ST=SE**

## 2. Interesses e Aspectos

Um dos elementos centrais da OA é o conceito de Interesse, que são as características relevantes de uma aplicação. Um interesse pode ser dividido em uma série de aspectos que representam os requisitos. Os aspectos podem ser agrupados no domínio da aplicação, compondo os interesses funcionais, que formam a lógica de negócio. Ou podem ser agrupados em elementos que prestam suporte aos interesses funcionais nomeados por interesses sistêmicos, e também chamados de ortogonais ou transversais.

Segundo Piveta, quando duas propriedades devem ser compostas de maneira diferente e ainda se coordenarem, diz-se que elas são ortogonais entre si. A tentativa de implementar um interesse sistêmico com a aplicação da Orientação a Objetos tem como resultado códigos que se espalham por todo o programa. Para isso dá-se o nome de código espalhado (*Scattering Code*).

O código espalhado pode ser classificado em:

1. Bloco duplicado.
2. Bloco complementar.

A implementação de vários interesses sistêmicos e funcionais em um mesmo módulo com a orientação a objeto resulta no código chamado de emaranhado (*Tangled Code*). O código espalhado e emaranhado dificulta da manutenção e a reutilização de código.

## 3. Orientação a Aspectos

Christina von Flach G. Chavez, importante pesquisadora brasileira em POA (Programação Orientada a Aspectos), afirmou que o desenvolvimento de um novo paradigma de engenharia de software freqüentemente progride da metodologia de programação em direção às metodologias de projeto e análise, fornecendo um caminho completo pelo ciclo de vida de desenvolvimento de software, alcançando assim a maturidade. A programação orientada a aspectos (POA) após uma década de pesquisas encontra-se nesse ponto, resultando em ferramentas eficientes, uma comunidade de usuários capacitados e aplicações iniciando a sua trajetória para o mercado.

Os paradigmas atuais (como o POO) não atendem às necessidades para a implementação dos requisitos de um sistema completo sem que parte dos conceitos desses paradigmas sejam quebrados. Na tentativa de resolver esses problemas, o desenvolvimento orientado a aspectos emerge, promovendo a separação avançada de perspectivas, desde o nível da implementação até os outros estágios do processo de desenvolvimento, incluindo especificação de requisitos, análise e projeto.

Nesse contexto, um dos criadores da POA, Gregor Kiczales, sugere que “para um paradigma de programação conseguir uma ampla aceitação, ele necessita ser expressivo, eficiente, intuitivo, compatível e possuir uma boa ferramenta que o suporte”. A POA já apresenta essas características, como poderemos observar a partir de agora.

## 4. O que é a programação orientada a aspectos?

A programação orientada a aspectos foi criada no fim da década de 1990, mais precisamente no ano de 1997, em Palo Alto, nos laboratórios da Xerox, por Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier e John Irwin. O objetivo era construir uma abordagem que fosse um conjunto não necessariamente homogêneo, que permitisse à linguagem de programação, composta por linguagem central e várias linguagens específicas de domínio, expressar de forma ideal as características sistêmicas (também chamadas de ortogonais ou transversais) do comportamento do programa. A essa abordagem dá-se o nome de meta-programação.

Uma forma de caracterizar uma linguagem de meta-programação é pelo seu compilador. A princípio, os compiladores destinados à orientação a aspectos não geram um produto final (um programa compilado e executável ou interpretável), mas um novo código. Nessa primeira compilação são acrescidos elementos ao código, para dar suporte às novas abstrações. O código resultante necessita ser novamente compilado para que seja, então, gerado o produto final. Outra característica da meta-programação presente na POA é a reflexão computacional, em que parte do código gerado é destinada a alterar características do próprio programa.

A POA estende outras técnicas, como a POO ou programação estruturada, propondo não apenas uma decomposição funcional, mas também sistêmica do problema. Isso permite que a implementação de um sistema seja separada em requisitos funcionais e não-funcionais, disponibilizando a abstração de aspectos para a decomposição de interesses sistêmicos, além dos recursos já oferecidos pelas linguagens de componentes, como Java, por exemplo.

## 5. Resolvendo velhos problemas

O principal objetivo da POA consiste em separar o código referente ao negócio do sistema dos interesses transversais, de uma forma bem definida e centralizada. Como já visto anteriormente, interesses são as características relevantes de uma aplicação. Um interesse pode ser dividido em uma série de aspectos que representam os requisitos da aplicação. Os aspectos que podem ser agrupados no domínio da aplicação compõem os interesses funcionais, referenciados também como lógica de negócio.

A separação e a centralização propiciada pela POA são indicadas pela Figura 1. Gregor Kiczales, no artigo que lançou a POA no mundo acadêmico, deixa claro esse objetivo. Ele observou que as linguagens da época podiam fazer um ótimo trabalho ao representar uma das muitas facetas de um sistema. Entretanto, para as demais, o resultado era sempre apenas satisfatório ou mesmo sofrível. Pode-se citar como exemplo linguagens da família do C (incluindo o C++). O seu trabalho em lidar com características de hardware é louvável, tanto que uma grande quantidade dos drivers para hardwares presentes hoje no mercado foram desenvolvidos nessa linguagem. Já ao lidar com persistência em banco de dados, por exemplo, exige-se um trabalho maçante. Em conseqüência, o resultado são códigos difíceis de serem reaproveitados, dificuldade de inteligibilidade de código, etc.

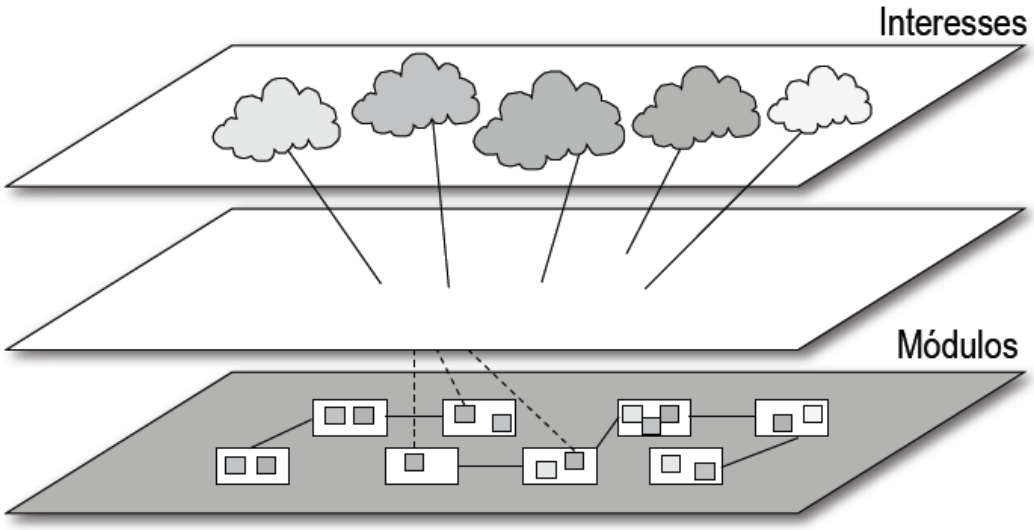



**Figura 1 – Separação de interesses com POA.**

A orientação a aspectos trabalha em paralelo com outro paradigma, portanto, os conceitos aprendidos, por exemplo, na orientação a objetos, são amplamente

reaplicáveis. A POA possibilita um nível maior de abstração no desenvolvimento de software. Esse nível maior de abstração e, conseqüentemente, a separação bem definida de cada um dos componentes, consistem no princípio fundamental da engenharia de software, pois estes estarão em um local bem definido no sistema. Na Figura 1, cada nuvem representa um interesse sistêmico implementado no sistema, como auditoria (log), tratamento de exceções, persistência, distribuição, dentre outros. Por estarem bem separados e em locais bem definidos, os componentes podem ser melhor reutilizados e a sua manutenção e legibilidade torna-se mais agradável.

Ao disponibilizar mecanismos para decomposição não apenas dos elementos e das relações pertinentes aos problemas, mas também para separar interesses sistêmicos, a POA promete simplificar a engenharia de software. Por exemplo, uma classe para implementação do controle de produtos, conforme a Figura 2 atende apenas a um interesse, não infringindo nenhum conceito da orientação a objetos.

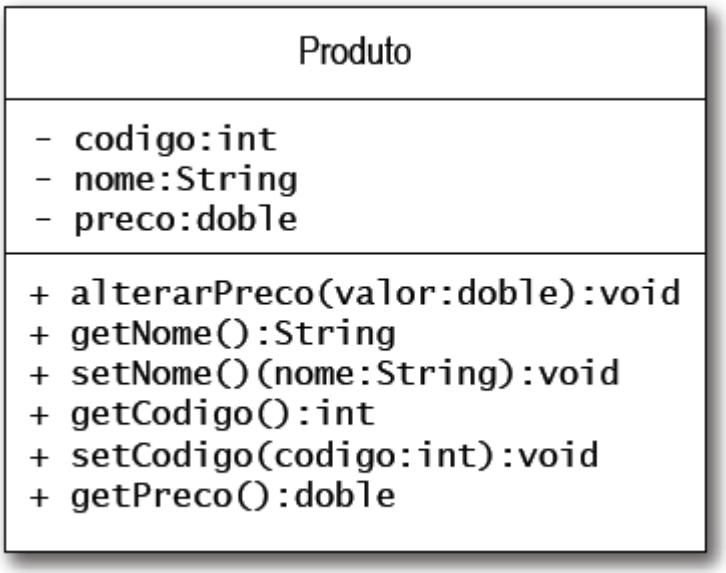



**Figura 2 – Classe Produto.**

Posteriormente, surge a necessidade de implementação do controle de auditoria (log). Sem utilizar POA, o interesse seria implementado na própria classe Produto, como pode ser observado na Figura 3. Caso fosse necessário implementar o controle de exceções, esse também seria feito juntamente na classe, como pode ser observado na Figura 4. A conseqüência disso é um acréscimo gradativo da complexidade no código.

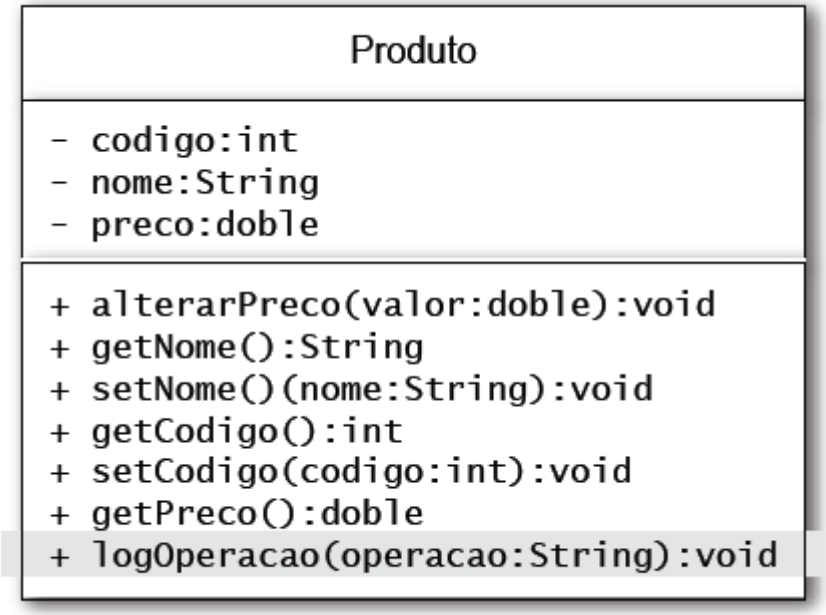



**Figura 3 – Implementação do controle de auditoria com OO.**

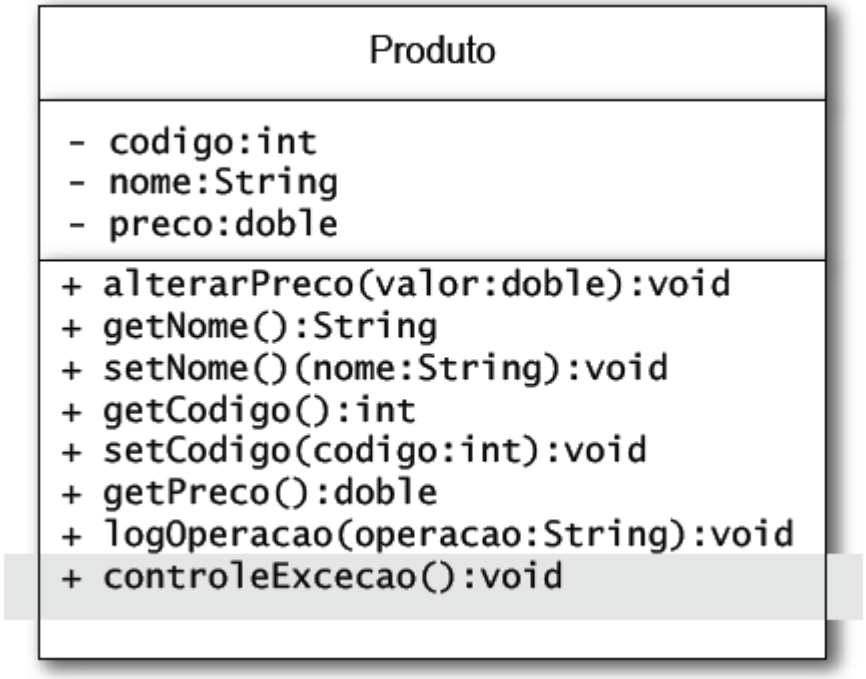



**Figura 4 – Controle de exceção da classe Produto.**

Na programação orientada a aspectos, por sua vez, os interesses são programados em módulos separados (classes e aspectos, por exemplo). Após a programação, ocorre a combinação entre as classes e os aspectos, conforme indicado pela Figura 5. Isso leva à simplificação do problema, pois o programador pode focar cada interesse de forma independente.

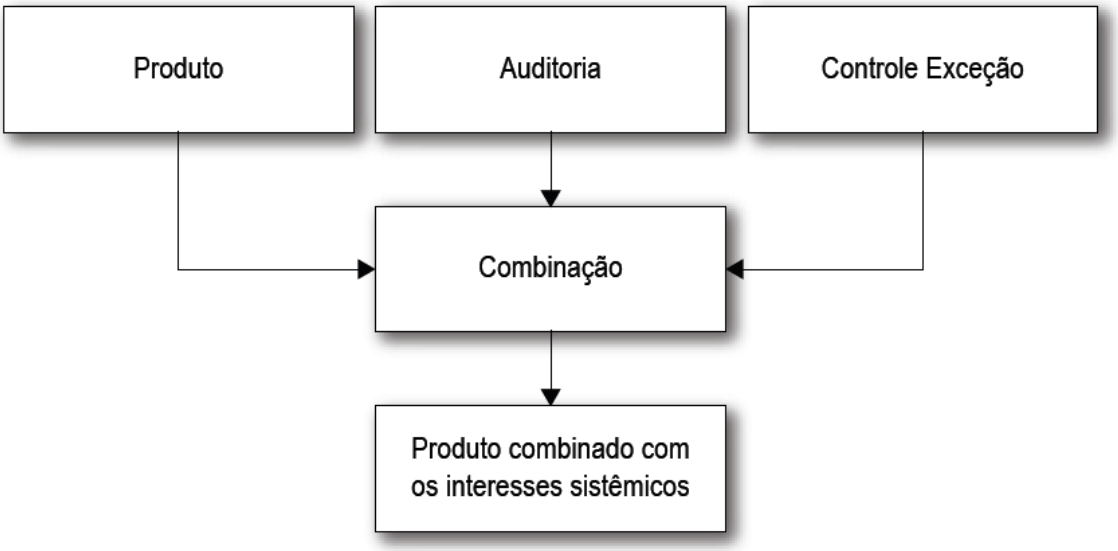



**Figura 5 – Combinação entre a classe Produto e os demais interesses.**

## 6. Composição de um sistema orientado a aspectos

Eduardo Piveta (2001) afirma que um sistema que utiliza a programação orientada a aspectos é composto pelos seguintes componentes:

- Linguagem de componentes: A linguagem de componentes deve permitir ao programador escrever programas que implementem as funcionalidades básicas do sistema, ao mesmo tempo em que não prevêem nada a respeito do que deve ser implementado na linguagem de aspectos. Exemplo de linguagem de componentes: Java, C++, C#, PHP etc.

- Linguagem de aspectos: A linguagem de aspectos deve suportar a implementação das propriedades desejadas de forma clara e concisa, fornecendo construções necessárias para que o programador crie estruturas que descrevam o comportamento dos aspectos e definam em que situações eles ocorrem.

- Combinador de aspectos: A tarefa do combinador de aspectos (aspect weaver) é combinar os programas escritos em linguagem de componentes com os escritos em linguagem de aspectos. No livro, abordamos a linguagem Java como linguagem de componentes e o AspectJ como linguagem de Aspectos.

- Programas escritos em linguagem de componentes: Os componentes podem ser considerados as unidades funcionais do sistema. Num sistema para controle bancário, pode-se definir como componentes os clientes, as contas, os funcionários, dentre outros. No contexto da programação orientada a aspectos, os componentes são abstrações providas pela linguagem, que permite a implementação da funcionalidade do sistema.

- Programas escritos em linguagem de aspectos para atender a interesses sistêmicos.

A composição de um programa orientado a aspectos é exemplificada na Figura 6, que mostra os programas escritos em linguagem de componentes, a linguagem de aspectos, os programas escritos em linguagem de aspectos e o combinador aspectual. Como saída obtém-se, então, o código combinado entre programas escritos em linguagem de componentes e programas escritos em linguagem de aspectos.

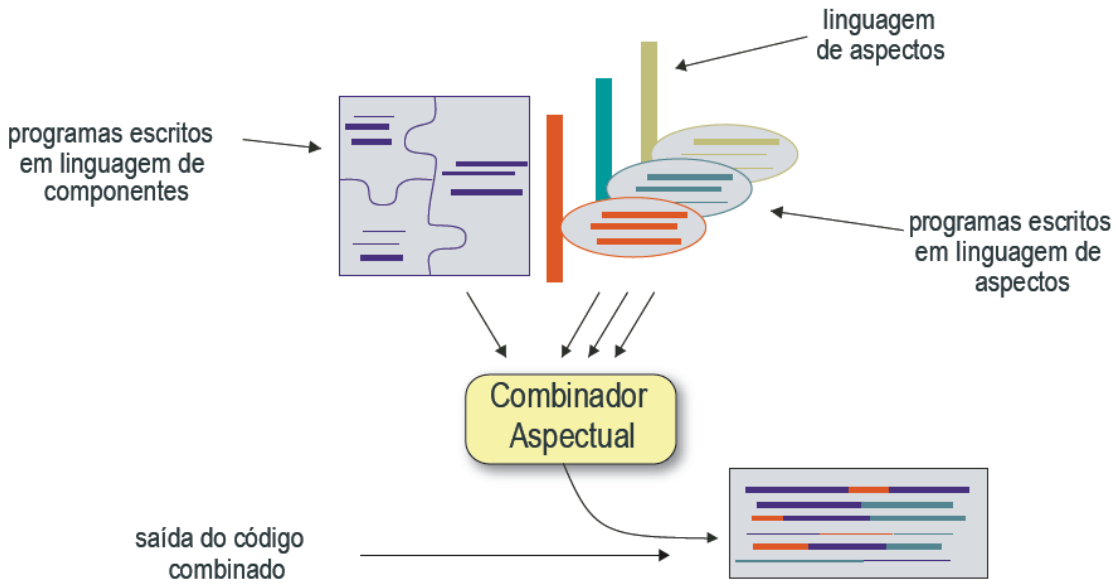



**Figura 6 – Composição de um sistema orientado a aspectos.**

## 6.1 Combinação aspectual

A combinação aspectual é o processo responsável por combinar os elementos escritos em linguagem de componentes com os elementos escritos em linguagem de aspectos. É um processo que antecede a compilação, gerando um código intermediário na linguagem de componentes capaz de produzir a operação desejada, ou de permitir a sua realização durante a execução do programa.

As classes referentes ao código do negócio nos sistemas não sofrem qualquer alteração para suportar a programação orientada a aspectos. Isso é feito no momento da combinação entre os componentes e os aspectos. Esse processo também pode ser definido como recompilação aspectual.

A Combinação aspectual pode ser:

- Estática: Um sistema orientado a aspectos utilizando combinação estática pode trazer agilidade ao mesmo, já que não há necessidade de que os aspectos existam em tempo de compilação e execução. O uso de uma combinação estática previne que um nível adicional de abstração cause um impacto negativo na performance do sistema.

- Dinâmica: Para que a combinação possa ser dinâmica é indispensável que os aspectos existam tanto em tempo de compilação quanto em tempo de

execução. Utilizando uma interface reflexiva, o combinador de aspectos tem a possibilidade de adicionar, adaptar e remover aspectos em tempo de execução.

O combinador utiliza os aspectos e os componentes para criar um novo código. Em um ambiente de desenvolvimento orientado a aspectos hipotéticos, capaz de combinar os aspectos às classes de modo similar a um programador, os códigos gerados no fim do processo de desenvolvimento de um sistema, antes de serem compilados, seriam semelhantes, tanto orientado a objetos quanto a aspectos.

## 7. Conceitos fundamentais da orientação a aspectos

A orientação a aspectos possui quatro conceitos fundamentais, que serão abordados nos sub-tópicos a seguir.

## 7.1 Pontos de junção (join points)

Os pontos de junção são locais bem definidos da execução de um programa, como, por exemplo, uma chamada a um método ou a ocorrência de uma exceção, dentre muitos outros.

A partir dos pontos de junção, são criadas as regras que darão origem aos pontos de atuação. Todos os pontos de junção possuem um contexto associado. Por exemplo, a chamada para um método possui um objeto chamador, o objeto alvo e os argumentos do método disponível como contexto.

## 7.2 Pontos de atuação (pointcuts)

Pontos de atuação são elementos do programa usados para definir um ponto de junção, como uma espécie de regra criada pelo programador para especificar eventos que serão atribuídos aos pontos de junção.

Os pontos de atuação têm como objetivo criar regras genéricas para definir os eventos que serão considerados pontos de junção, sem precisar defini-los individualmente (o que tornaria a POA quase sem sentido). Outra função dos pontos de atuação é apresentar dados do contexto de execução de cada ponto de junção, que serão utilizados pela rotina disparada pela ocorrência do ponto de junção mapeado no ponto de atuação.

Após a identificação dos pontos de junção para um determinado interesse, é necessário agrupá-los em um ponto de atuação. A combinação do ponto de junção e atuação torna evidente a ocorrência de interesses sistêmicos, já que esse interesse muito provavelmente estará propagado em diversas classes no sistema, e estas classes, estarão, portanto, implementando mais de um interesse.

## 7.3 Adendo (advice)

Adendos são pedaços da implementação de um aspecto executados em pontos bem definidos do programa principal (pontos de junção).

Os adendos são compostos de duas partes: a primeira delas é o ponto de atuação, que define as regras de captura dos pontos de junção; a segunda é o código que será executado quando ocorrer o ponto de junção definido pela primeira parte. O adendo é um mecanismo bastante similar a um método (quando comparamos com a programação orientada a objetos), cuja função é declarar o código que deve ser executado a cada ponto de junção em um ponto de atuação.

## 7.4 Aspectos

As propriedades de um sistema que envolvem diversos componentes funcionais não podem ser expressas utilizando a notação e as linguagens atuais de uma maneira bem localizada. Propriedades, tais como sincronização, interação entre componentes, distribuição e persistência, são expressas em fragmentos de código espalhados por diversos componentes do sistema.

Um aspecto é o mecanismo disponibilizado pela programação orientada a aspectos para agrupar fragmentos de código referente aos componentes não-funcionais em uma unidade no sistema. Para desenvolvimento de um sistema de agenda, poderiam ser relacionados os seguintes interesses a serem implementados:

- Gerenciador de contatos
- Gerenciador de compromissos
- Persistência dos dados
- Controle de acesso
- Auditoria (log), dentre outros

Do exemplo citado, os interesses para controle de acesso, auditoria de operações e persistência de dados são necessários, mas não inerentes ao negócio. Caso um desses interesses fosse implementado com a utilização somente das técnicas da orientação a objeto, ocorreria o código emaranhado e espalhado, conforme a Figura 7, onde as barras verticais representam as classes do sistema e os traços, a ocorrência do código para implementação do interesse.

Na programação orientada a aspectos, esses interesses não inerentes ao negócio, denominados interesses sistêmicos, são agrupados em aspectos, evitando o código espalhado e emaranhado. Dessa forma, o aspecto não pode existir isoladamente para implementação dos interesses do sistema (tanto funcionais quanto sistêmicos). Os objetos continuam existindo e neles são implementados os interesses funcionais. Já nos aspectos são tratados os interesses sistêmicos. Podemos definir, então, que o par aspecto-objeto é a unidade principal em um programa orientado a aspectos, assim como o objeto seria o principal em um programa orientado a objetos.

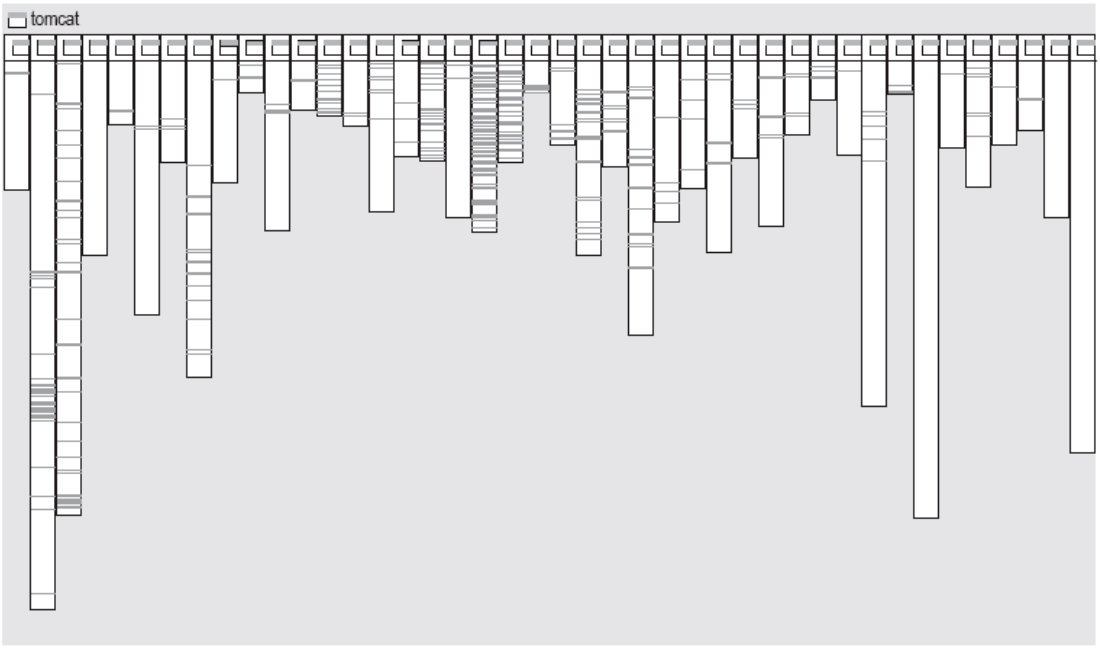



**Figura 7 – Ocorrência de código emaranhado.**

## 8. Onde e por que aplicar a orientação a aspectos

A POA contribui na implementação dos interesses sistêmicos, que, mesmo não sendo inerentes ao negócio, são, na maioria dos casos, indispensáveis para o sucesso da aplicação. Por meio do uso da orientação a aspectos, esses interesses podem ser agrupados em unidades modulares, tornando a manutenção mais simples e facilitando o reúso por reduzir a interdependência entre os interesses. Uma revisão da lista dos interesses sistêmicos, sob a perspectiva da POA, ficaria da seguinte forma:

- Sincronização de objetos concorrentes: a POA disponibiliza um mecanismo eficaz para modularizar a política de sincronização.

- Distribuição: com a programação orientada a aspectos, pode-se tornar a distribuição transparente por meio da criação de um aspecto destinado a representar esse interesse, tornando desnecessária a refatoração. O aspecto mapeia todo o interesse de distribuição e, por meio do combinador aspectual, é gerado um código “distribuível” similar ao que seria o código refatorado, conforme a Figura 8.

- Tratamento de exceções: Por meio do uso da POA é possível centralizar as políticas para tratamento de exceções em unidades elementares, o que facilita tanto a manutenção e legibilidade das classes quanto à abordagem para o tratamento de exceções.

- Coordenação de múltiplos objetos: A POA permite modularizar a lógica destinada a sincronizar a integração entre os objetos ativos em busca de um objetivo global.

- Persistência: Por meio do uso de aspectos é possível centralizar e abstrair a implementação desse interesse da camada de negócio, facilitando, assim, a manutenção e a obtenção de melhorias nas abordagens aplicadas.

- Auditoria: Com a POA, o interesse de auditoria pode ser implementado independentemente da regra de negócio. Dessa forma, o código estaria em um único local centralizado e bem definido no sistema, sendo retirado das classes.

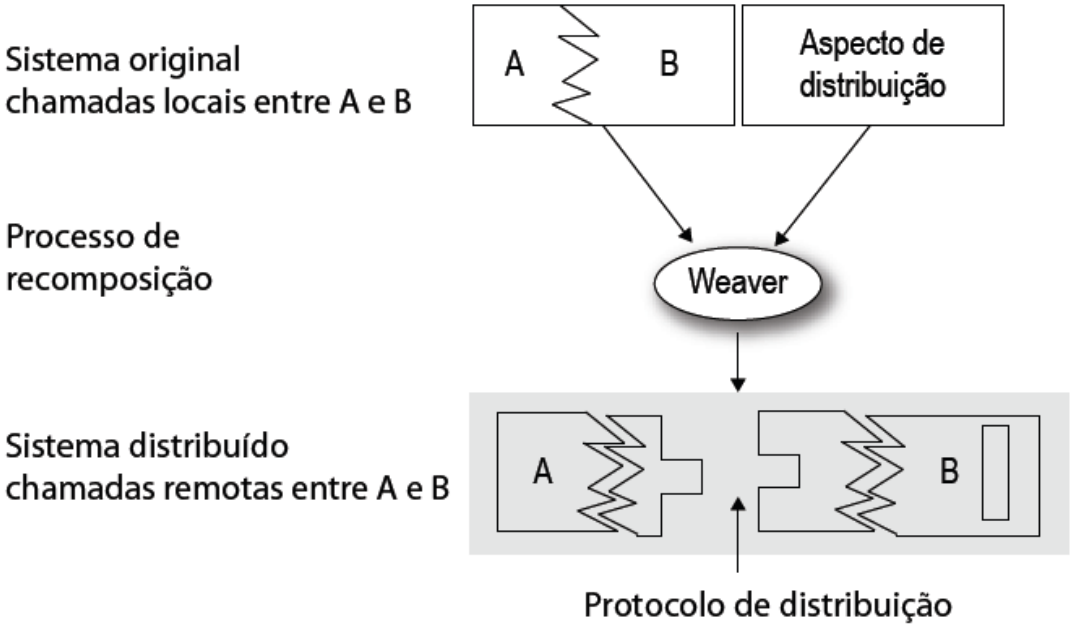



**Figura 8 – Acrescentado distribuição por meio de aspectos.**

## 6. AspectJ – Orientação a Aspectos com Java

O AspectJ é uma extensão para a linguagem de programação Java. Por isso, houve e ainda há uma preocupação pela compatibilidade de quatro itens extremamente importantes e essenciais:

- Compatibilidade total - todo programa Java válido é também um programa AspectJ válido;

- Compatibilidade de plataforma - todo programa AspectJ pode ser executado em uma máquina virtual java (JVM);

- Compatibilidade de ferramentas - deve ser possível estender ferramentas existentes para suportar o AspectJ de uma forma natural; isto inclui IDE's (Integrated Development Environments), ferramentas de documentação e ferramentas de projeto;

- Compatibilidade para o programador - Ao programar com AspectJ, o programador deve sentir-se como se estivesse utilizando uma extensão do linguagem Java.

Existe uma divisão do AspectJ em duas partes: a linguagem de especificação e a linguagem de implementação. A parte da linguagem de especificação define a linguagem na qual o código é escrito; com AspectJ, os interesses funcionais são implementados em Java, e para implementação da combinação de interesses sistêmicos são utilizadas as extensões disponibilizadas pelo próprio AspectJ. A parte da linguagem de implementação fornece ferramentas para compilação, debug e integração com ambientes integrados de desenvolvimento.

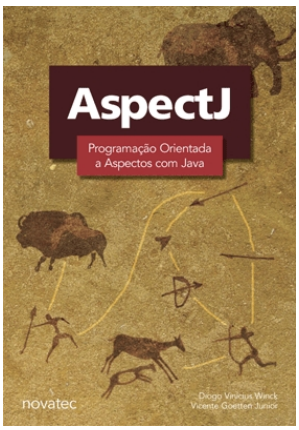
O AspectJ implementa dois tipos de interesses sistêmicos, os estáticos e dinâmicos. São nestes interesses que ocorrem a combinação aspectual. A combinação aspectual pode atravessar diversos módulos de um sistema para que o interesse em questão possa ser devidamente implementado.

## 7. Conclusão

Certamente a Orientação a Aspectos (OA) ainda não é o paradigma ideal, e acreditamos que em breve surgirão outros paradigmas capazes de melhor abstrair/expressar soluções, e por tal, permitindo a criação de soluções melhores. Mas o que fica claro para nós é que, a OA é um grande passo rumo ao paradigma ideal. Neste artigo, e mesmo no livro, trabalhamos apenas a parte da Orientação a Aspectos ligada à programação, entretanto as vantagens da OA quando aplicadas a análise e, principalmente, ao projeto podem ser impressionantes, mas isto é um tema para um outro artigo.

No livro são apresentados diversos exemplos práticos da utilização da programação orientada a aspectos, como:

- Auditoria;
- Registro e rastreamento de operações;
- Persistência de dados;
- Tratamento de exceções;
- Políticas de desenvolvimento;
- Autenticação, identificação e autorização de usuários;
- Etc.




## 8. Bibliografia

- **GOETTEN Junior, WINCK Diogo, 2006. AspectJ – Programação Orientada a Aspectos com Java. São Paulo – SP: Novatec Editora, 2006.**
- ASPECTJ Team. Disponível em: <http: / /www.eclipse.org/ aspectj>.
- BÖLLERT, Kai. On Weaving Aspects. In the Proceedings of the Aspect - Oriented Programming Workshop at ECOOP'99, 1999.
- CHAVEZ, Christina. GARCIA, Alessandro. LUCENA, Carlos. Desenvolvimento Orientado a Aspectos. Anais do XVII Simpósio Brasileiro de Engenharia de Software, Manaus - Amazonas. Universidade Federal do Amazonas. 2003.
- CHAVEZ, Christina. LUCENA, Carlos. A Theory of Aspects for Aspect-Oriented Software Development. Anais do XVII Simpósio Brasileiro de Engenharia de Software. Manaus - Amazonas. Universidade Federal do Amazonas. 2003.
- GOETTEN Junior, WINCK Diogo e MACHADO Caio, 2004. Programação Orientada a Aspectos - Abordando Java e aspectJ. Workcomp Sul – SBC.
- KICZALES,G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C. et al. Aspect-oriented programming. Para apresentação em: ECOOP'97, LNCS 1241. Springer, 1997.
- PIVETA, Eduardo. Um modelo de suporte a programação orientada a aspectos . UFSC. Dissertação submetida como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação, 2001.


## Sobre os autores


**Diogo Vinícius Winck** é professor universitário na UTESC e sócio da Kugel Soft Informática, empresa de desenvolvimento de ERPs. Possui título de bacharel e de mestre em Ciência da Computação. Trabalha com desenvolvimento de software orientado a objetos desde 2001.

**Vicente Goetten Junior** é formado em Tecnologia da Informação pela UTESC e possui MBA em Gestão Empresarial pela FGV. Atua como docente na faculdade e colégio FCJ/Elias Moreira e como consultor de Tecnologia na Datasul. Possui grande experiência com desenvolvimento de software orientado a objetos e nos últimos anos tem coordenado pesquisas sobre programação orientada a aspectos. Site: www.goetten.org.