Combinação e Aplicação de Técnicas para o Desenvolvimento de Software Orientado a Aspectos

Gabriel Costa Silva, Munif Gebara Junior, Daniela Eloise Flor
Curso de Sistemas de Informação, Universidade Paranaense (UNIPAR)
Paranavaí - PR, CEP 87706-490, Brasil
munif@unipar.br

e

Yandre Maldonado e Gomes da Costa
Departamento de Informática, Universidade Estadual de Maringá (UEM)
Maringá - PR, CEP 87020-900, Brasil
yandre.costa@gmail.com

Abstract

This work describes obtained results with the combination of two techniques for aspect oriented software development. A method to aspects and components development was applied into requisite's document of system, producing documents for a software development. A modeling technique for aspects was combined with method in one of his phases, producing new documents. Both this techniques combination, allows this aspects is detected and modelled during the development process in the modular form, allowing the reuse and considering both components and aspects as "first-class citizens".

Keywords: Software Development Method, Aspect-Oriented Development, Components, Crosscutting Concern, Software Modelling.

Resumo

Este trabalho descreve os resultados obtidos com a combinação de duas técnicas para o desenvolvimento de software orientado a aspectos. Um método para o desenvolvimento baseado em componentes e aspectos foi aplicado sobre o documento de requisitos de um sistema, gerando artefatos para a produção de um software. Uma técnica de modelagem própria para aspectos foi combinada ao método em uma de suas fases, produzindo novos artefatos. A combinação destas técnicas permite que aspectos sejam detectados e modelados durante o processo de desenvolvimento de forma modular, permitindo o reuso e tratando tanto os componentes-base quanto os aspectos como "cidadãos de primeira classe".

Palavras chaves: Método de Desenvolvimento de Software, Desenvolvimento Orientado a Aspectos, Componentes, Interesses Transversais, Modelagem de Software.

1 INTRODUÇÃO

Com o constante aumento da competitividade no mercado mundial, os sistemas computacionais se tornaram indispensáveis nas empresas. Através deles é possível alcançar uma maior produtividade com menor custo. Alguns sistemas permitem que simulações sejam feitas, evitando que erros sejam cometidos e garantindo uma maior qualidade no produto ou serviço oferecido.
Com o aumento da demanda por *software*, a comunidade de desenvolvimento tem buscado meios de garantir uma maior produtividade aliada ao aumento da qualidade no produto computacional. Essa busca por qualidade fez surgir metodologias de desenvolvimento que contemplam o software desde os interesses do usuário até sua implantação na empresa. Técnicas de modelagem, documentação e modularização surgiram no intuito de permitir que softwares sejam alterados de forma simples e rápida [12].
Este trabalho aborda o desenvolvimento de *software* baseado em componentes e aspectos. O uso de componentes torna o desenvolvimento menos complexo, mais ágil e oferece maior qualidade para o produto computacional [5]. A utilização de uma metodologia de desenvolvimento garante que todos os ciclos de vida do *software* serão contemplados, permitindo que ajustes sejam feitos ainda em nível de projeto, diminuindo assim a probabilidade de erros futuros. Uma linguagem de modelagem auxilia na comunicação entre os envolvidos no projeto do sistema, fornecendo notações comuns para todas as áreas envolvidas [1]. Por fim, a separação de interesses promovida pela utilização da orientação a aspectos, permite que o software se torne mais modular a medida que o código referente ao domínio da aplicação é separado do código que implementa outros interesses, como os interesses não-funcionais.
Para a elaboração deste trabalho, um experimento foi conduzido tomando-se por base o documento de requisitos de um sistema para prestadores de serviço. O método para desenvolvimento de *software* orientado a aspectos (DSBC/A), proposto por [5], foi aplicado sobre este documento, gerando artefatos. A linguagem aSideML foi introduzida na etapa de Especificação sobre os modelos que envolvem aspectos, e sobre os modelos que envolvem o relacionamento de aspectos e elementos base. Na etapa de Provisionamento e Montagem, alguns componentes foram providos através de repositórios de componentes, e aqueles não encontrados, foram implementados. Todos os aspectos foram implementados utilizando a linguagem AspectJ.

2 PROGRAMAÇÃO ORIENTADA A ASPECTOS

Para o desenvolvimento de um produto de *software* é necessário que os interesses do usuário sejam decompostos em unidades menores de *software*. Essas unidades colaboram entre si para que um determinado objetivo do sistema seja atingido. Esse princípio recebe o nome de modularização. No entanto, existem interesses que não podem ser modularizados em uma única unidade. Esses interesses atravessam outras unidades, gerando o espalhamento (*scattering*) e entrelaçamento (*tangling*) de código [9]
Para exemplificar, vamos supor o controle de *logging* de um sistema. O controle de *logging* permite armazenar em registros as atividades realizadas pelo sistema como o acesso de usuários, tentativas de invasão e falhas do sistema, por exemplo. Neste caso, a parte do código fonte referente ao domínio do negócio visa permitir que um usuário tenha acesso ao sistema, ou que um recurso seja corretamente compartilhado. Entretanto, para que haja o controle dessa atividade, é necessário que o código seja atravessado por um segundo código, não inerente ao negócio, mas que tem por objetivo o monitoramento dessa atividade. Esse entrelaçamento de interesses é chamado de interesse transversal (*Crosscutting concerns*) [3], e fez surgir a necessidade de formas de se separar código referente ao negócio, dos interesses não inerentes ao negócio de uma forma centralizada [11]. Padrões de desenvolvimento (*design patterns*) são utilizados em alguns casos com o intuito de

diminuir o problema dos interesses transversais. No entanto, mesmo com o uso de padrões, a redundância ainda não é eliminada por completo.
Através do uso da orientação a aspectos, é possível modularizar os interesses transversais em unidades centralizadas, que afetam os componentes desenvolvidos para suprir os interesses de negócio inserindo neles comportamentos ou mesmo atributos, sem que o código referente ao negócio seja entrecortado [9].
Na orientação a aspectos, os interesses de negócio são implementados através de uma linguagem base, como Java, por exemplo, enquanto os interesses não referentes ao negócio, são implementados como aspectos, utilizando-se uma linguagem própria, como AspectJ, por exemplo. Dessa forma, a orientação a aspectos afeta as unidades base, mas estes, não precisam conhecer os aspectos.
A programação orientada a aspectos é composta de uma linguagem para implementar os interesses base, que será utilizada para desenvolver o código referente ao negócio da aplicação. É necessária ainda uma linguagem para o desenvolvimento de aspectos. Com o programa base já escrito, o código referente a aspectos é incorporado ao código base através de um combinador de aspectos (*aspect weaver*), inserindo os novos comportamentos ou atributos, e gerando uma nova aplicação. Este esquema é apresentado com detalhes na figura 1.

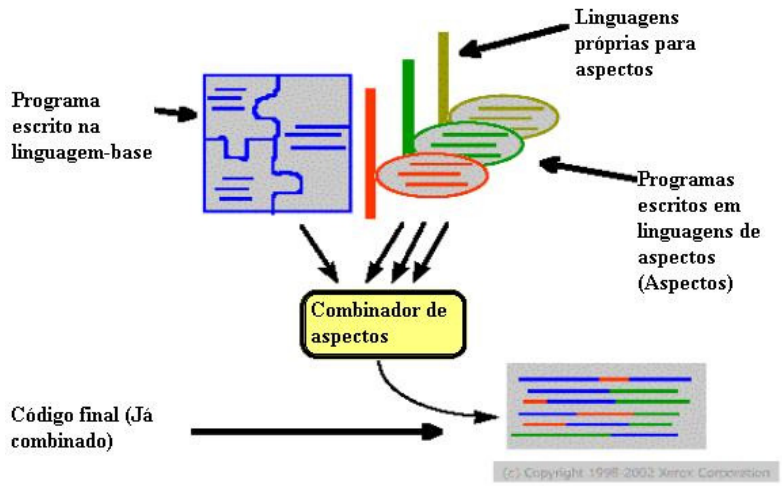

Figura 1: Processo de combinação de aspectos e linguagem-base [9].

3 DESENVOLVIMENTO BASEADO EM COMPONENTES

A modularização está inserida no conceito de engenharia de *software* e faz parte da evolução das linguagens de programação [5]. A diminuição da complexidade do desenvolvimento é um dos fatos que justificam a importância do uso da modularização na elaboração de *software*. Pode-se mencionar um exemplo bastante emblemático relacionado à indústria automobilística. Imagine como seria complexa a criação de um carro no qual todas as peças fizessem parte de uma única estrutura, sem separações. Caso uma das peças apresentasse problemas, toda sua estrutura deveria ser refeita, gerando enormes perdas de tempo e dinheiro. A modularização permitiria a substituição de uma peça sem que as demais sejam afetadas por essa mudança.
Da mesma forma, uma peça, como um volante por exemplo, pode ser usada em vários veículos de diferentes modelos, oferecendo sempre a mesma funcionalidade. Este princípio recebe o nome de reuso e é aplicado no desenvolvimento de *software*. O reuso de uma unidade de *software* já existente, diminui o esforço durante o desenvolvimento, visto que menos unidades deverão ser implementadas. Outra vantagem advinda do reuso é o aumento da qualidade, uma vez que uma unidade que já foi reutilizada algumas vezes, obrigatoriamente já passou pelo teste de outros

desenvolvedores, o que atesta a favor da sua qualidade [8]. Além das vantagens, outro ponto que motiva o reuso de *software* é a diminuição do tempo de desenvolvimento, pois, atualmente, não é mais viável criar um *software* a partir do zero [5].
A idéia de reuso gerou diferentes métodos de desenvolvimento, o desenvolvimento baseado em componentes (DBC) é um deles. No DBC, a principal idéia é a construção de *software* a partir de algo já existente [8]. Um componente encapsula serviços, que são oferecidos através de uma interface bem definida [5]. O uso de interfaces para acessar um componente permite que esse componente seja atualizado ou mesmo substituído sem a necessidade se alterar os demais componentes. No DBC, toda comunicação é feita através das interfaces, sendo que os componentes possuem interfaces oferecidas e requeridas [8]. A figura 2 mostra dois componentes e suas respectivas interfaces oferecidas e requeridas. O componente Credito oferece uma interface para acesso aos seus serviços, chamada IFornece. Por sua vez, esse componente depende de uma interface – IConsulta, fornecida pelo componente Cliente. Nesse exemplo, o componente Cliente poderia passar por alterações, ou mesmo ser substituído por uma versão diferente, sem gerar problemas para o componente Credito, pois o acesso está sendo feito por suas interfaces.

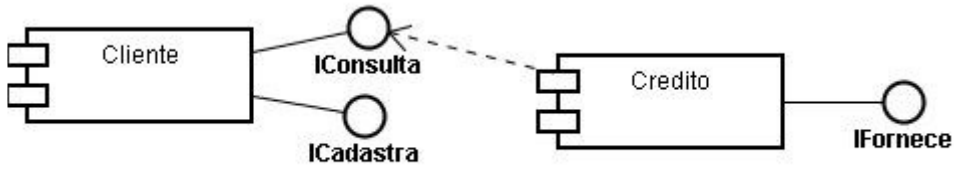

Figura 2: Exemplo de componentes e interfaces oferecidas e requeridas.

Outro ponto a ser considerado sobre componentes são os detalhes de implementação. Um componente deve ser visto como uma "caixa preta" para o usuário (desenvolvedor). Isso acontece porque, ao desenvolvedor da aplicação, cabe conhecer os serviços oferecidos pelo componente, e não a forma como esses serviços são implementados internamente. É importante destacar que componentes são unidades independentes. A união desses componentes, através dos relacionamentos de oferta/dependência de suas interfaces, gera um sistema completo e funcional [5].

3.1 Método Para Desenvolvimento de *Software* Baseado em Componentes e Aspectos

O objetivo do método para desenvolvimento de *software* baseado em componentes e aspectos (DSBC/A) proposto por [5] é, partindo do documento de requisitos do sistema, produzir uma arquitetura formada por componentes base e aspectos. O método DSBC/A opera sobre as interfaces, preservando o encapsulamento dos componentes [5].
A figura 3 apresenta as etapas do método DSBC/A, os principais artefatos gerados e utilizados em cada etapa segundo apresentado por [5]. Cada etapa é subdividida em etapas menores, formadas por algumas atividades. O documento de requisitos é o artefato inicial, utilizado para alimentar a primeira atividade na etapa de Análise de Requisitos. Durante esta etapa, os requisitos do usuário são processados por diversas atividades. Entre as atividades desta etapa, está a identificação dos primeiros requisitos candidatos à implementação com aspectos. Na etapa de Especificação são produzidas as especificações para os componentes base e transversais, bem como as interfaces requeridas e oferecidas. Esta etapa é importante, pois entre outras atividades, é nela que os primeiros candidatos a aspectos são expostos a critérios para a identificação dos componentes transversais. Nas etapas seguintes os componentes que satisfazem os requisitos do sistema e do projeto realizado são providos e por fim, unidos. A interface com o usuário é então implementada

produzindo uma aplicação completa [5]. As etapas de Teste e Implantação não são abordadas por [5] em seu trabalho.

4 MODELAGEM DE *SOFTWARE*

Atualmente, a *Unified Modeling Language* (UML) é um padrão amplamente aceito e empregado para a modelagem de *software* na indústria. Baseada na orientação a objetos (OO), UML é uma linguagem de propósitos gerais, que oferece visões para a descrição de um sistema, desde requisitos até a implementação. Seus diagramas oferecem visões tanto para o comportamento estático, quanto para o comportamento dinâmico do sistema, além de ser uma linguagem independente de processo, ferramenta ou linguagem de programação. A UML ainda apresenta uma forte característica que é a sua extensibilidade. Os mecanismos de extensão da UML permitem que ela seja utilizada para representar novos modelos, que possam surgir com a evolução da complexidade dos sistemas [3].

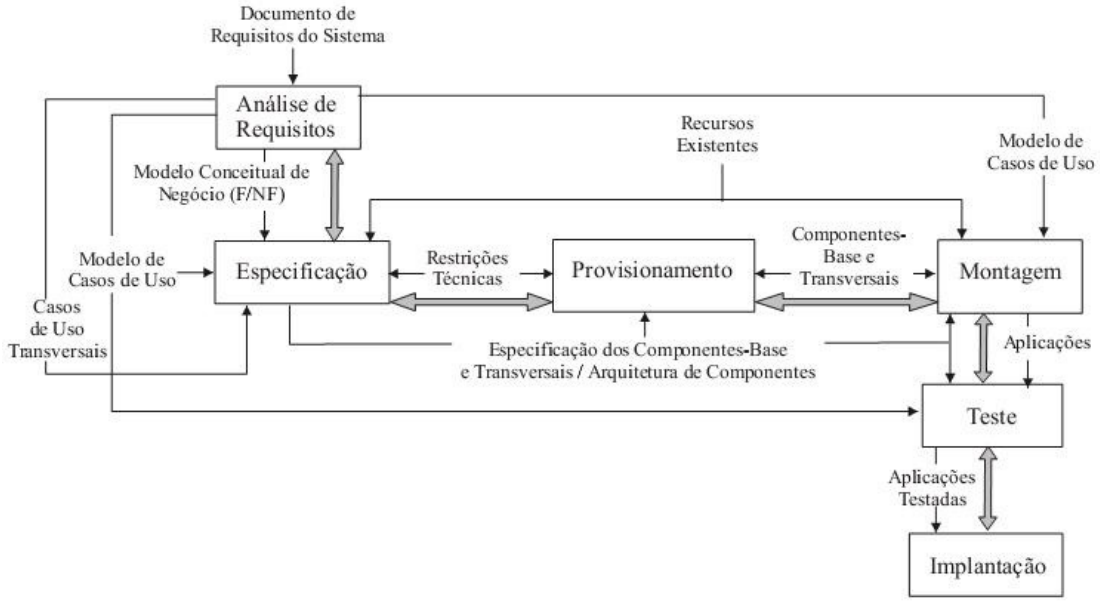

Figura 3: Etapas do método para DSBC/A [1].

Na programação orientada a aspectos (POA), os interesses transversais devem coexistir com os blocos básicos convencionais, independentemente destes serem ou não orientados a objeto. O aspecto deve ser tratado como um bloco adicional, relacionado por um relacionamento de *crosscutting*. Um relacionamento de *crosscutting* especifica a dependência entre um aspecto e um elemento base [3]. A modelagem orientada a aspectos (MOA), incorpora esses conceitos. Segundo [3], "A MOA propõe o uso de outros elementos que nomeiam e localizam interesses transversais e o uso de outras visões a fim de melhorar a compreensão, além de facilitar a reutilização e a evolução". Uma linguagem para MOA deve oferecer compatibilidade com a UML, além de possuir notação independente de linguagem de programação [3].

Existem algumas abordagens que avaliam o uso da UML para a orientação a aspectos, como a UMLaut , AODM e a *Composition Patterns*. De forma geral, essas abordagens geram a necessidade de extensão, os interesses transversais são nomeados, mas não explicitamente especificados, além disso, falta suporte para expressar a semântica de *crosscutting* [3]. Segundo [3], "aspectos não são classes, portanto, a abstração de aspectos deve ser especificada por um novo elemento de modelagem".

4.1 Linguaguem aSideML

A aSideML é uma linguagem baseada na UML que oferece semântica, notação e regras para modelos orientados a aspectos. A aSideML fornece visão para os modelos estáticos e dinâmicos do sistema, além dos modelos composicionais. Os modelos composicionais apresentam o processo de combinação, que envolve a combinação dos elementos base do sistema com os elementos transversais (aspectos). Na aSideML, aspectos e *crosscutting* são tratados como "cidadãos de primeira classe" [3]. Conforme pode ser observado na tabela 1, os elementos de modelagem oferecidos pela aSideML podem ser estruturais ou comportamentais. Os principais elementos estruturais são: o aspecto, elementos base e os relacionamentos. Os principais elementos comportamentais são instâncias de aspecto, interações aspectuais e colaborações aspectuais [3]. A aSideML fornece novos diagramas e enriquece alguns da UML tradicional.

Tabela 1: Modelos, diagramas e elementos da aSideML [4].

Modelo	Diagramas	Perspectivas	Elementos
estrutural	diagrama de aspectos		aspecto, interface transversal, característica transversal
	diagrama de classe estendido	centrado em aspecto	aspecto, *crosscutting*
		centrado na base	interface transversal, *order*
comportamental	diagrama de sequência estendido	ponto de combinação	ponto de combinação dinâmico
	diagrama de colaboração aspectual		instância de aspecto, colaboração aspectual
	diagrama de sequência		instância de aspecto, interação aspectual
processo de combinação	diagrama de classes combinadas		classe combinada
	diagrama de colaboração combinada		colaboração combinada
	diagrama de sequência combinada		interação combinada

5 COMBINAÇÃO E APLICAÇÃO DE TÉCNICAS

Esta seção apresenta os resultados obtidos com a combinação do método para DSBC/A, proposto por [5], e a linguagem de modelagem aSideML, proposta por [3]. O método para DSBC/A é formado por um conjunto de atividades que permitem o desenvolvimento de componentes e aspectos desde o documento de requisitos até a interface com o usuário. Já a linguagem de modelagem aSideML, fornece notações e regras que permitem a construção de modelos orientados a aspectos [3].

O método para DSBC/A é um método completo, no entanto, a modelagem de aspectos e a modelagem que envolve aspectos e componentes base é feita utilizando extensões da UML. Estas extensões não permitem que sejam visualizadas todas as características de aspectos e, em alguns momentos, torna difícil a leitura de alguns modelos. A linguagem aSideML foi projetada para ser independente de linguagem e metodologia de desenvolvimento [3].

A combinação de ambas as técnicas permitem um desenvolvimento orientado a aspectos, onde os aspectos, componentes base e seus relacionamentos, são identificados, modelados e implementados considerando todos suas características, independente da linguagem de programação utilizada.
As etapas de Análise de Requisitos e Especificação são as etapas onde são elaborados os artefatos que definem a arquitetura do sistema. Por este motivo, estas etapas serão abordadas com ênfase neste trabalho. É na etapa de Especificação que a linguagem aSideML é introduzida. Nesta etapa são definidos os elementos base do sistema, bem como a identificação dos elementos transversais e seus relacionamentos com os elementos-base. A aSideML tem um importante papel nesta etapa, pois seus diagramas geram artefatos capazes de comunicar de forma clara o papel dos aspectos, seus relacionamentos e características. As etapas de Teste e Implantação não serão abordadas neste trabalho.

5.1 Análise de Requisitos

A etapa da Análise de Requisitos envolve a identificação e modelagem do domínio da aplicação. É nesta etapa que são identificados os requisitos funcionais e não-funcionais. Esses requisitos são então mapeados e modelados como Casos de Uso. Por fim, são gerados os Modelos Conceituais de Negócio e identificados os casos de uso candidatos a serem implementados como aspectos.
Algumas atividades, como a identificação dos casos de uso, construção do diagrama de casos de uso e descrição dos casos de uso, acontecem de forma individual para os requisitos funcionais e não-funcionais do sistema. Posteriormente, alguns destes elementos são unidos em um diagrama de casos de uso. Uma visão parcial da união dos casos de uso funcionais e não-funcionais pode ser visto na figura 4. A figura apresenta três atores, a saber: Atendente, Gerente e Sistema de Faturamento. Esses atores possuem relacionamentos com casos de uso. Alguns casos de uso, por sua vez, possuem relacionamentos de extensão e inclusão com outros casos de uso. Existe ainda o caso de uso não-funcional “Registrar Operações”, que é identificado pelo estereótipo <<NFR>>, e possui um relacionamento de extensão com outros casos de uso.

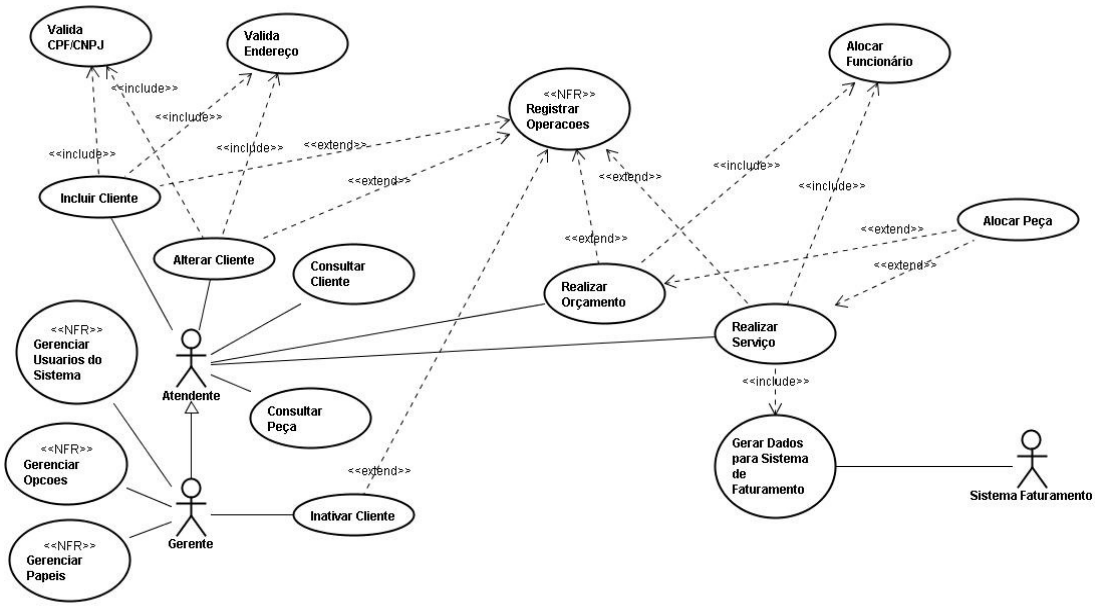

Figura 4: Visão parcial da união de casos de uso funcionais aos não-funcionais.

A última atividade desta etapa envolve a identificação dos primeiros candidatos a aspectos. Os casos de uso candidatos a serem implementados como aspectos são aqueles incluídos, estendidos ou restringidos, por dois ou mais casos de uso. Segundo [5], esse critério se justifica pois, pensando em

nível de implementação, esses casos de uso representam interesses espalhados no sistema. A tabela 2 apresenta os candidatos a aspectos identificados para o sistema em questão.

5.2 Especificação

A etapa de Especificação possui elevada importância no processo de desenvolvimento, visto que é nesta etapa que os componentes e aspectos são especificados para, em uma etapa posterior, serem providos [5].

Tabela 2: Candidatos a aspectos para o sistema para prestadores de serviço.

Caso de Uso	Tipo	Critério
Valida CPF/CNPJ	Funcional	Incluído por dois ou mais casos de uso
Valida Endereço	Funcional	Incluído por dois ou mais casos de uso
Alocar Peça	Funcional	Estende dois ou mais casos de uso
Alocar Funcionário	Funcional	Incluído por dois ou mais casos de uso
Autenticar Usuário	Não-Funcional	Estende dois ou mais casos de uso
Registrar Operações	Não-Funcional	Estende dois ou mais casos de uso
Controlar Acesso dos Usuários	Não-Funcional	Estende dois ou mais casos de uso
Persistir Dados	Não-Funcional	Estende dois ou mais casos de uso

Assim como as demais, esta etapa é compreendida por uma série de atividades. As duas primeiras atividades são relativas aos componentes base e suas interfaces. Essas atividades compreendem a identificação dos componentes e suas interfaces, bem como a interação entre esses componentes. Como resultado destas duas atividades, é produzido um diagrama com a arquitetura parcial dos componentes. A figura 5 apresenta a arquitetura parcial do sistema para prestadores de serviços. Esta figura apresenta apenas a arquitetura dos requisitos funcionais do sistema.

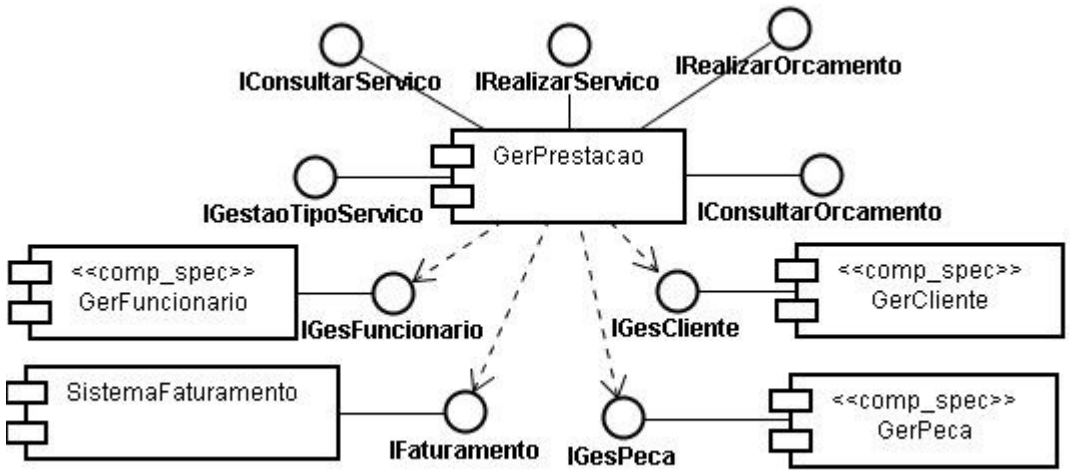

Figura 5: Arquitetura parcial dos componentes funcionais.

5.2.1 Aplicação da aSideML

Segundo o que foi descrito pelo método para DSBC/A, a próxima atividade é a identificação dos componentes transversais. Esta atividade é composta por algumas outras atividades, cujo objetivo é identificar os elementos que serão implementados como aspectos, elaborar suas interfaces e operações e, por fim, definir uma arquitetura para componentes base e transversais. A atividade

seguinte envolve a interação entre componentes-base e transversais. A linguagem aSideML foi aplicada na modelagem destas atividades, permitindo uma maior riqueza de detalhes na apresentação dos modelos.
Os critérios de Atomicidade, Disparo, Integridade, Variabilidade e Custo/Benefício são apresentados por [5] como critérios norteadores que, quando aplicados sobre os casos de uso candidatos a aspectos, auxiliam o analista e decidir pela implementação com aspectos. A tabela 3 apresenta os candidatos a aspectos, os critérios que cada um deles atende e a opção de implementação para cada um deles.
Para os casos de uso que não serão implementados como Aspectos, cabem ainda duas alternativas: ser implementado como uma operação interna do componente ou virar um novo componente.

Tabela 3: Candidatos a aspectos e a decisão de implementação para cada um.

Caso de Uso	Atomicidade	Disparo	Integridade	Variabilidade	Custo/Benefício	Implementação
Valida CPF/CNPJ	X	X	X		X	Aspecto
Valida Endereço	X	X	X		X	Aspecto
Alocar Peça	X	X				Outro
Alocar Funcionário	X	X				Outro
Autenticar Usuário	X	X	X		X	Aspecto
Registrar Operações	X	X	X		X	Aspecto
Controlar Acesso dos Usuários	X	X	X		X	Aspecto
Persistir Dados	X	X				Outro

Identificados os elementos que serão implementados como aspectos, utilizamos um diagrama de aspectos da linguagem aSideML para modelar os aspectos e suas interfaces. A figura 6 apresenta o diagrama de aspectos, na visão completa, para o aspecto "Registrar Operações". É importante observar ainda que na aSideML, um aspecto pode ser apresentado em sua visão completa ou condensada. Nesta figura, o aspecto GerRegistroOp possui uma interface chamada ITRegistrarOperacao. Esta interface transversal possui como característica uma operação de *crosscutting* registrarOperacaoExecutada(), que será utilizada como um refinamento a ser aplicado após a execução de uma determinada operação no componente base. O relacionamento de *crosscutting* de um aspecto com um componente base é apresentado no diagrama de classe estendido, que pode ser visto na figura 7.

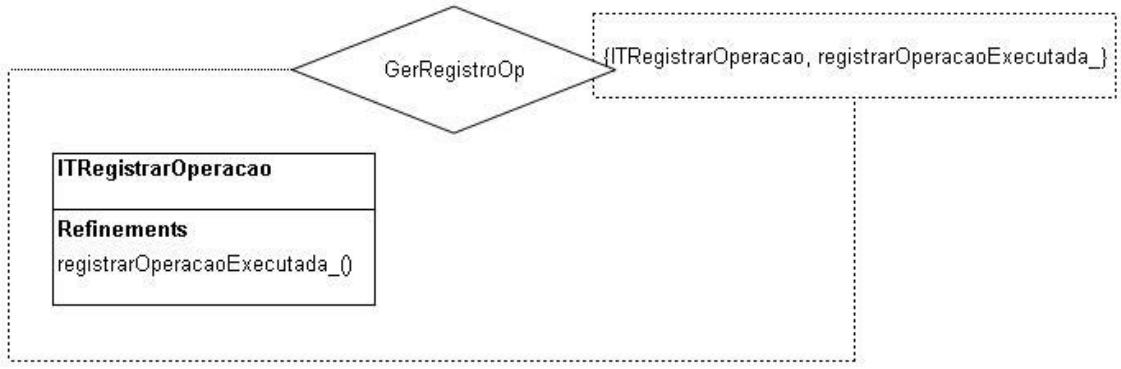

Figura 6: Diagrama de aspecto para o aspecto GerRegistroOp.

Para facilitar a visualização dos relacionamentos de *crosscutting*, [5] apresenta uma atividade que envolve a elaboração de novas interfaces para operações entrecortadas. Nesta atividade, é sugerido que todas as operações entrecortadas por um mesmo aspecto, sejam unidas em uma única interface, desta forma, é possível saber quais operações um aspecto entrecorta. Para exemplificar, vamos considerar as interfaces IRealizarOrcamento e IRealizarServico. A tabela 4 apresenta as operações destas interfaces e sinaliza aquelas que devem ser entrecortadas pelo aspecto GerRegistroOp.

Seguindo o que está descrito pelo método, as operações que são entrecortadas por GerRegistroOp deveriam ser unidas em uma única interface, como uma interface IEntradaDeDados, por exemplo.

Tabela 4: Análise para o agrupamento de operações entrecortadas – suprido pela aSideML.

Interface	Operações	Entrecorte por GerRegistroOP
IRealizarOrcamento	adicionarOrcamento	SIM
	consultarOrcamento	NÃO
IRealizarServico	adicionarServico	SIM
	consultarServico	NÃO
	alterarServico	SIM
	inserirPecas	SIM

O diagrama de classe estendido da aSideML pode ser utilizado para modelar os relacionamentos de *crosscutting*, tornando esta atividade desnecessária. Esse diagrama apresenta o relacionamento de *crosscutting* em nível estático entre um aspecto e um elemento base. O diagrama apresenta notação específica para cada relacionamento, tornando possível a identificação de qual operação será entrecortada por cada operação no aspecto. A figura 7 apresenta o diagrama de classe estendido para a interface IRealizarOrcamento. O relacionamento de *crosscutting* é identificado pelo estereótipo <<crosscutt>>, seguido pela operação transversal do aspecto e a operação que será entrecortada no elemento base. Nessa representação, o aspecto é apresentado em sua visão condensada.

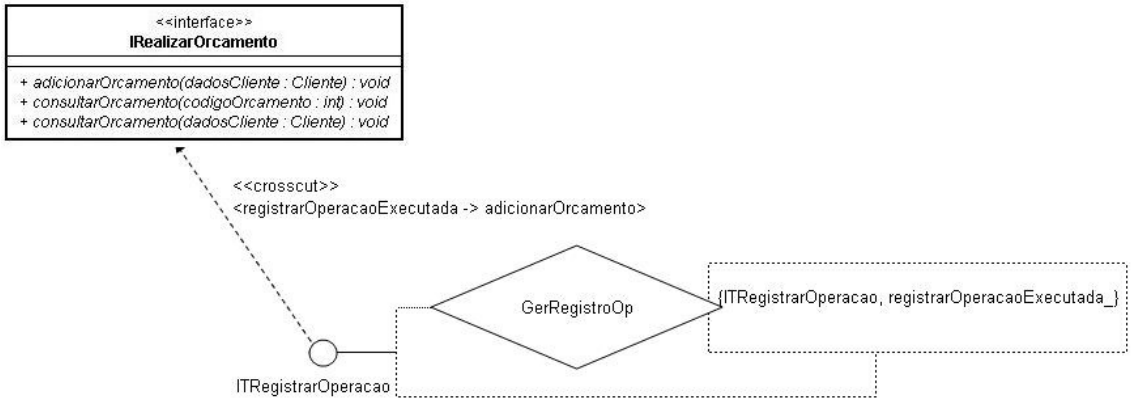

Figura 7: Relacionamento de *crosscutting* entre aspecto e interface.

Alguns diagramas da aSideML oferecem diferentes perspectivas. A figura 8 apresenta uma visão com a perspectiva centrada em ponto de combinação. Nessa perspectiva é possível observar um conjunto de elementos base e quais das suas operações são entrecortadas por determinados aspectos. Uma das últimas atividades é a interação entre componentes base e transversais. Essa interação foi modelada através dos diagramas comportamentais da aSideML

5.3 Provisionamento e Montagem

Nas etapas anteriores, foram criadas as estruturas necessárias para a geração do sistema. Nesta etapa, os componentes que satisfazem as condições especificadas são providenciados e unidos, formando uma aplicação completa. No desenvolvimento baseado em componentes, existem três alternativas de provisionamento. A primeira consiste no reuso de componentes utilizados em outras aplicações, a segunda alternativa é aquisição de componentes de terceiros e, por fim, a implementação de novos componentes.

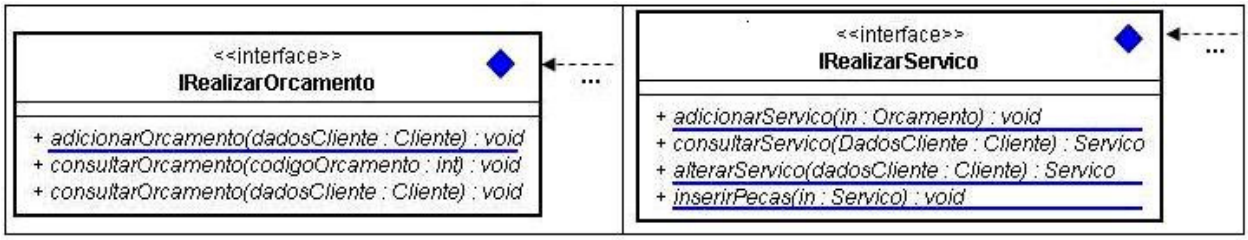

Figura 8: Visualização de *crosscutting* – perspectiva centrada em ponto de combinação.

Para este trabalho, alguns componentes foram adquiridos através de repositórios de componentes na Web e, para aqueles não disponíveis, houve a implementação. Já para os aspectos, todos foram implementados utilizando a linguagem AspectJ.

Para permitir que os aspectos implementados para esta aplicação pudessem ser reutilizados, estes foram implementados de forma genérica. Isso significa que suas operações e interfaces são definidas, porém, os pontos de entrecorte serão definidos apenas no momento da montagem dos componentes e aspectos, através do uso de conectores [5]. A figura 9 apresenta dois quadros. No lado esquerdo, é possível observar uma transcrição genérica para a implementação do aspecto GerRegistroOp. Observe que o aspecto é declarado como abstrato e suas operações internas são definidas. Vale ressaltar que, como a linguagem AspectJ não implementa interfaces, o conjunto de pontos de junção recebe o nome da interface.

Na etapa de Montagem, os componentes devem ser unidos, e a interface com o usuário, implementada. Neste momento, os conectores que unirão os aspectos aos seus pontos de entrecorte, também devem ser desenvolvidos. Na figura 9, o quadro à direita apresenta um conector para o aspecto GerRegistroOp. Este conector é uma extensão do aspecto GerRegistroOp, declarado anteriormente como abstrato. Neste conector, o ponto de entrecorte, antes declarado apenas genericamente, é sobrecarregado por um ponto específico.

```
abstract aspect GerRegistroOp {
        public abstract pointcut ITRegistrarOperacao();
        public registrarOperacaoExecutada () {
                ...
        }
        after(): ITRegistrarOperacao() {
                registarOperacaoExecutada();
                ...
        }
}
```

```
public aspect conectorParaGerRegistroOp
        extends GerRegistroOp {
        public pointcut ITRegistrarOperacao():
                call (* *.adicionarOrcamento (..));
}
```

Figura 9: Implementação de aspecto genérico e conector.

6 CONCLUSÕES

O método para DSBC/A fornece critérios úteis para a identificação e implementação de aspectos durante o processo de desenvolvimento de *software*. As atividades contidas no método fornecem uma forte documentação para os componentes e aspectos, possibilitando o reuso e facilitando a manutenção. A combinação da aSideML para modelagem da especificação de aspectos, suas características e relacionamentos com componentes base, gera modelos de maior entendimento, além de evitar algumas atividades adicionais, propostas anteriormente pelo método. Os modelos gerados pela aSideML explicitam de forma simples e completa as características da orientação a aspectos. A combinação destas técnicas tornou possível o desenvolvimento de uma aplicação

completa baseada em componentes e aspectos, onde ambos são tratados como cidadãos de primeira classe, podem ser reutilizados e facilmente atualizados.

REFERÊNCIAS

[1] Bezerra E. Princípios de Análise e Projeto de Sistemas com UML. Elsevier, Rio de Janeiro, 2002.

[2] Booch G. UML: Guia do usuário. Campus, Rio de Janeiro, 2000.

[3] Chavez C. V. F. G. "Um Enfoque Baseado em Modelos para o Design Orientado a Aspectos," tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

[4] Cheesman, J. e Daniels J. UML Components: A simple process for specifying component-based software. Addison-Wesley, 2000.

[5] Eler M. M. "Um método para o desenvolvimento de software baseado em componentes e aspectos," dissertação de mestrado, Universidade de São Paulo, São Carlos, 2006.

[6] Garcia A. et al. *Relatório do Primeiro Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos*, tech. report, 18º. Simpósio Brasileiro de Engenharia de Software, 2004.

[7] Garcia A. e Chavez C. V. F. G. "Desenvolvimento de Software Orientado a Aspectos", *19º. Simpósio Brasileiro de Banco de Dados e 18º. Simpósio Brasileiro de Engenharia de Software*, Brasília, 2004.

[8] Gimenes I. e Huzita E. H. M. Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. Ciência Moderna, Rio de Janeiro, 2005.

[9] Junior V. G. S. e Winck D. V. e Machado C. A. P. "Programação Orientada a Aspectos Abordando Java e AspectJ", *União de Tecnologia e Escolas de Santa Catarina*, Joinville, [2003?].

[10] Laddad R. AspectJ in Action – Practical Aspect-Oriented Programming. Manning Publications Co., Connecticut, 2003.

[11] Silva G. C. et al. "Uma Introdução a Programação Orientada a Aspectos", CD-ROM, *VIII Semana de Informática e V Mostra de Iniciação Científica do Curso de Sistemas de Informação*, Paranavaí, 2006.

[12] Sommerville I. Engenharia de Software. Addison-Wesley, São Paulo, 2003.