Artigo Engenharia de Software 2 - Análise Orientada a Objetos

Saiba como chegar a um modelo OO focado em responsabilidades a partir de casos de uso.

Esse artigo faz parte da revista Engenharia de Software 2 edição especial.

Clique aqui para ler todos os artigos desta edição

Projeto

Análise Orientada a Objetos

Como chegar a um modelo OO focado em responsabilidades a partir de casos de uso

O conceito de orientação a objetos surgiu com o intuito de minimizar os problemas encontrados até então na criação de softwares complexos, projetados por meio de decomposição funcional e sub-rotinas.

Podemos identificar como um dos maiores problemas a não existência de encapsulamento lógico para operações e dados, o que leva a não existência da divisão de tarefas por responsabilidades. O que leva a construção de longos trechos de código, muitas vezes difíceis de compreender devido ao acúmulo de responsabilidade que lhe é atribuído.

Por conseqüência, quanto mais complexo o software se torna, mais difícil se torna também a sua manutenção. Com isso aumentam os custos e o risco de confiabilidade do mesmo.

Nesse artigo veremos como efetuar uma analise orientada a objetos, com base em responsabilidade, extraída a partir das descrições de casos de uso.

Conceituando análise Orientada a Objetos

O foco da análise OO é no mapeamento de uma solução sistêmica para algum processo de negocio.

No inicio da análise OO, elaboramos os casos de uso. Estes juntamente com as descrições dos casos de uso formam uma espécie de ponte funcional entre o processo de negocio e a solução de software a ser produzida.

Outros dois documentos podem ser usados como fontes complementares aos casos de uso na análise OO:

· Glossário: onde estão definidos os significados de todos os termos inerentes ao negócio mapeado nos casos de uso;

· Documento de arquitetura: na utilização de possíveis mecanismos de análise, que são padrões de comportamento ou estrutura de uso recorrente e comprovadamente válido. Iremos falar mais sobre eles mais adiante.

Normalmente, o resultado da análise orientada a objetos se traduz em diagramas UML de seqüência e classe, como mostra a **Figura 1**. Entretanto, outros diagramas UML podem ser usados nessa tarefa, desde que seja verificado algum ganho no entendimento ou mapeamento da solução com seu uso.

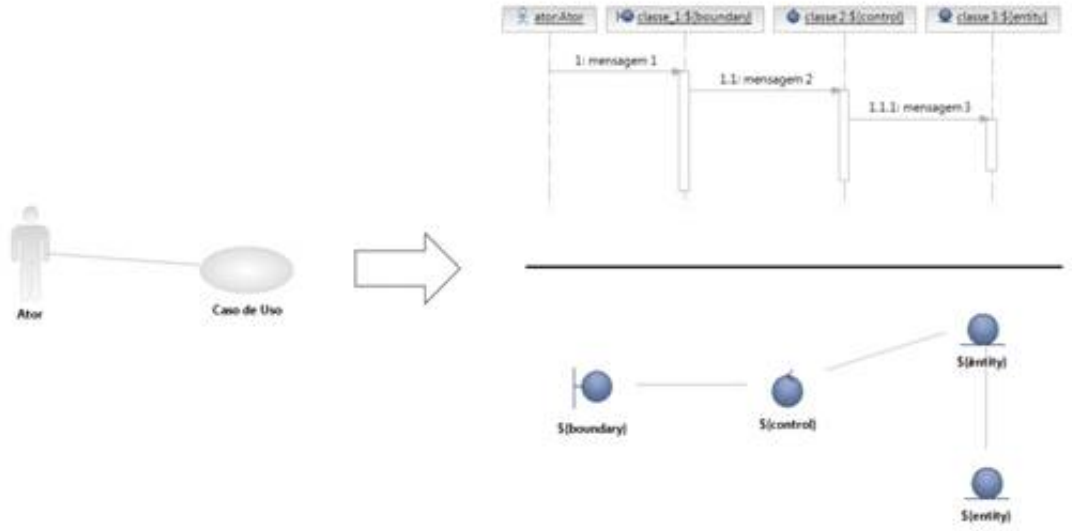


Figura 1. Produtos da análise.

Através da **Figura 1** também percebemos que os estereótipos utilizados no diagrama de classe não são os que costumamos ver normalmente. Pois bem, ali estamos usando os estereótipos de análise.

O uso desses estereótipos nos oferece uma orientação mais específica para o processo de identificação de classes.

Os estereótipos de análise se dividem em: classe de fronteira (ou boundary), classe de controle (ou control) e classe de entidade (ou entity), como mostra a **Figura 2**.

Figura 2. Estereotípos de análise

A classe de fronteira é responsável por modelar a interação entre o ambiente do sistema e seus trabalhos internos. Essa interação envolve transformar e converter

eventos, bem como observar mudanças na apresentação do sistema. É a classe de fronteira que trata das questões relativas à camada mais externa do sistema.

A classe de controle é usada para modelar um comportamento de controle específico de um ou alguns casos de uso. Geralmente estão controlando chamadas a classes de entidade. Por isso seu comportamento é muito ligado à idéia de coordenação, de ponte entre a camada mais externa do sistema (classes de fronteira) e a camada mais interna do sistema (classes de entidade).

E finalmente, com a classe de entidade, modelamos comportamentos e informações que devem ser armazenados. É de responsabilidade das classes de entidade manter e atualizar informações relativas ao negócio do sistema, como: pessoas, eventos, objetos reais, ou qualquer outra informação ligada ao negocio ao qual o sistema está inserido. Por exemplo: em um sistema voltado para a área de ensino, provavelmente teremos uma classe de entidade chamada aluno, outra chamada disciplina e assim por diante.

Efetuando a análise

Extrair classes orientadas a objeto com base em descrições de casos de uso não é uma tarefa simples. Para isso, é necessário ter uma boa capacidade de abstração. Entretanto, se dividirmos esse trabalho em etapas consecutivas e complementares, o processo de análise como um todo se torna mais simples.

Sendo assim, nossa análise OO será construída ao longo dos quatro tópicos a seguir.

Identificando Abstrações Chave

A partir da leitura e entendimento dos casos de uso e do glossário, devemos identificar quais serão os conceitos mais importantes e óbvios do sistema. Esses conceitos são chamados de abstrações chave. Eles devem ser acompanhados de uma breve descrição e dos possíveis relacionamentos entre eles.

O objetivo em se identificar essas abstrações é justamente dar um guia, uma referência primária para toda a análise que virá pela frente.

Vale ressaltar que as abstrações chave encontradas não devem ser vistas como

verdades absolutas e imutáveis. Com o avanço da análise do sistema, e conseqüentemente, com o entendimento mais amplo sobre a solução OO que está sendo desenvolvida, as abstrações podem sofrer alterações.

Criando diagramas de seqüência com base em responsabilidade

Para cada cenário do caso de uso (fluxo principal e fluxos alternativos) devemos executar uma releitura dos passos do cenário. À medida que vamos avançando na leitura, devemos ir identificando as classes de fronteira, controle e entidade envolvidas no texto.

Nesse ponto do trabalho, a identificação de abstrações chave feita anteriormente se torna bastante útil, pois, iremos utilizá-las como base para a identificação das classes de entidade. Tanto as abstrações chave quanto as classes de entidade estão intimamente ligadas aos conceitos de negócio envolvidos no caso de uso. As abstrações chave nos dão uma primeira visão sobre o negocio, uma visão um pouco turva. Começamos a dar mais nitidez a essa visão quando transformamos as abstrações chave em classes de entidade.

A coisa mais importante nesse momento é despender total atenção para a correta divisão de responsabilidade entre as classes encontradas.

A responsabilidade de uma classe representa o conjunto de ações que ela pode desempenhar e o conjunto de informações que ela pode ser solicitada a fornecer.

E porque iremos fazer um diagrama de seqüência com essas classes e não um diagrama de classe?

Porque à medida que vamos lendo gradualmente os passos do caso de uso podemos identificar não só as classes, mas também a interação entre o ator do caso de uso e as classes.

A princípio, essa abordagem pode parecer um pouco complexa demais. Afinal,

normalmente aprendemos a fazer um diagrama de classe com base no caso de uso e só depois passamos para algum diagrama dinâmico, como o de seqüência.

Mas na verdade, começar pelo diagrama de seqüência e não pelo de classe é uma forma bastante natural de se trabalhar. Pois, enquanto estamos lendo o caso de uso e visualizando as interações entre ator e sistema, nada nos impede de já transformarmos essas interações em mensagens trocadas pelas classes do diagrama de seqüência.

Trabalhando dessa forma, as classes vão aparecendo gradualmente no diagrama de seqüência, em decorrência da leitura do passo do caso de uso e sua interpretação como um conjunto de mensagens trocadas entre classes.

Ao final teremos um ou mais diagramas de seqüência de análise para cada cenário descrito no caso de uso. Vale lembrar que se um cenário oferece uma alta complexidade de entendimento ou é longo de mais, podemos criar mais de um diagrama de seqüência focado nele.

Muito provavelmente, teremos também um diagrama de classe de todo o caso de uso. Afinal de contas, na maioria das ferramentas para diagramação UML hoje em dia, quando geramos diagramas de seqüência estamos automaticamente criando um digrama de classe contemplando as classes usadas no diagrama de seqüência e os relacionamentos entre elas.

Preenchendo e refinando as classes

Agora que já criamos diagramas de seqüência para todos os cenários do caso de uso, chegou a hora de arrumar o diagrama de classe resultante desse trabalho e também o de seqüência, se for o caso.

Como tratamos cada cenário de forma isolada, é possível que tenhamos criado classes com responsabilidades muito parecidas, porém, em diagramas de seqüência diferentes. Nesse momento devemos fazer uma varredura no diagrama de classes, objetivando eliminar possíveis redundâncias.

A partir daí passamos a analisar os vínculos entre classes, que são as mensagens trocadas entre elas nos diagramas de seqüência. Essas mensagens representam a

comunicação entre as classes, o relacionamento entre elas. O refinamento aqui é justamente identificar corretamente qual o tipo de relacionamento existente entre duas classes do diagrama de classes. É imprescindível que tenhamos um bom entendimento do caso de uso, pois sem isso, seria difícil, por exemplo, identificar se determinado relacionamento deve ser uma agregação ou uma composição.

Também podemos identificar os atributos das classes com base nas mensagens trocadas no diagrama de seqüência quando a mensagem é criada com o intuito de trafegar informações entre classes. Nesse caso, essas informações se tornam atributos da classe responsável por fornecê-las. Mas tenhamos cuidado com isso! Pois se uma classe A precisa acessar uma classe B para fornecer uma informação, então provavelmente a responsabilidade de guardar essa informação na forma de atributo, é da classe B e não da classe A.

Aliás, desconfie quando as classes de entidade não trocam mensagens entrei si para resolver passos do caso de uso no diagrama de seqüência. Isso **pode** significar que a divisão de responsabilidade entre as classes não foi bem feita, como mostra a **Figura 3**.

Figura 3. Divisão de responsabilidade provavelmente mal feita.

Uma boa divisão de responsabilidade entre classes de negócio (entidades) normalmente se traduz em diagramas como os da **Figura 4**.

Figura 4. Divisão de responsabilidade bem feita.

Observe que na **Figura 3 a classe de controle chamada *Classe2* se responsabiliza pela comunicação com as classes de entidade 3, 4 e 5, as quais não trocam mensagens entre si. Agora dando nome aos bois, imagine que a classe 3 se chama *Pedido*, a classe 4 *ItemPedido* e a classe 5 *Produto*.**

Com essa visão de negócio em mente, chegaríamos facilmente à seguinte constatação: s**omente o Pedido tem conhecimento de quais são os Itens de Pedido que o compõe.**

Após essa constatação, teria sentido pedir para a classe de controle se responsabilizar diretamente pelas informações referentes à *IntemPedido*, por exemplo?

De forma alguma!

O mais coerente, de acordo com a correta divisão de responsabilidade seria somente a classe Pedido ter o direito de se comunicar com o *ItemPedido*. Então, quando alguma classe precisar de informações sobre *ItemPedido*, ela terá de pedir a informação para a classe Pedido.

Um diagrama de seqüência de análise feito com essa mentalidade baseada em divisão de responsabilidades normalmente se aproxima da forma exposta na **Figura 4**, onde as classes de entidade costumam se relacionar com foco na responsabilidade.

Padrões de análise

No início do artigo, o documento de arquitetura foi apresentado como um dos documentos que servem de insumo para se fazer a análise OO devido aos mecanismos de análise que ele pode apresentar. Pois bem, existem mecanismos de análise focados

em divisão de responsabilidade. Eles foram criados de forma bem genérica, para que seu uso seja válido e aconselhável em praticamente qualquer análise OO. Eles são chamados de padrões GRASP (General Responsibility Assignment Software Patterns). Os padrões GRASP representam as boas práticas que podemos usar para atribuir responsabilidades às classes. São eles:

Especialista (Expert)

É o padrão mais usado na atribuição de responsabilidade. Ele determina quem é o especialista em determinada informação. A classe especialista em determinada informação é a classe que tem a informação necessária para satisfazer tal responsabilidade ou que sabe como obter tal informação. Por exemplo: em um sistema de vendas onde existem as classes *Venda* e *ItemVenda*, sendo que *Venda* representa uma composição de *ItemVenda*, qual das duas classes seria a responsável por manter a informação total de uma venda?

De acordo com o padrão especialista, a classe responsável por manter o total de uma venda seria a classe *Venda*, pois a *Venda* é que detém o conhecimento sobre todos os seus *ItemVenda*. Conseqüentemente, é nela que devemos manter o somatório dos valores de seus *ItemVenda*.

A classe *ItemVenda* não deve ter a responsabilidade de guardar o total de uma venda. Afinal, um *ItemVenda* só conhece o valor do seu item e não de todos os itens de uma venda (ver **Figura 5**).

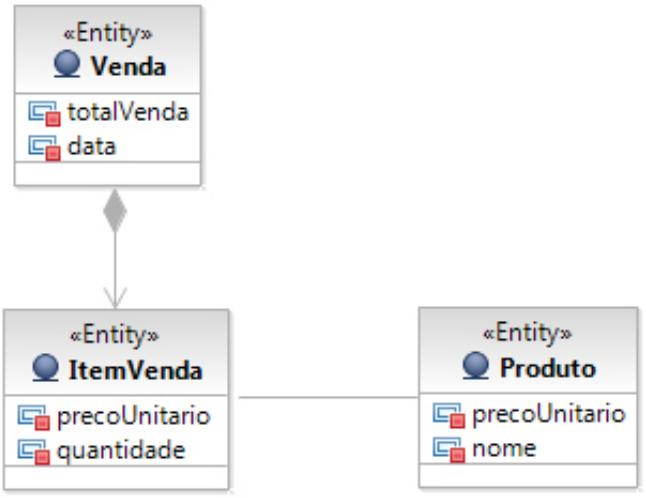


Figura 5. Usando o padrão especialista.

A atribuição de responsabilidades muitas vezes não tem correspondente no mundo real. Nesse caso da venda, por exemplo, no mundo real uma venda não calcula seu próprio total, isso seria feito por uma pessoa. No mundo OO, entidades inertes, como produtos, ou conceitos, como uma venda, podem ter responsabilidades.

Criador (Creator)

Padrão que define qual classe deve ser responsável por criar instâncias de outra classe. A classe criadora deve ser aquela que possui a outra como parte dela ou que esteja fortemente associada a ela. Sendo assim, atribuímos à classe B a responsabilidade de criar uma instância da classe A se uma das seguintes condições for verdadeira:

- B agrega objetos A
- B contém objetos A
- B registra instâncias de objetos A
- B usa de maneira muito próxima objetos de A
- B é um especialista com relação à criação de A

Controlador (Controller)

Padrão que define a classe responsável por tratar um acontecimento externo ao sistema e que desencadeia alguma ação do mesmo. É o padrão responsável por tratar eventos do sistema.

Podemos atribuir essa responsabilidade de duas formas. Uma seria criando uma classe que vá tratar todos os eventos de um caso de uso, nesse caso, estamos criando um "controlador de caso de uso". A outra seria criando uma classe que vá controlar todo o sistema, nesse caso, estamos criando um "controlador de fachada".

Lembra do estereótipo "control" apresentado há pouco? Pois é, qualquer semelhança com o padrão controlador, não é mera coincidência.

Já que voltamos a falar dos estereótipos de análise, vale salientar que as classes de fronteira (ou boundary) não devem ter a responsabilidade de manipular eventos do sistema, elas devem apenas delegar essa tarefa para o controlador. Dessa forma, temos uma maior capacidade de reuso, pois estaremos criando interfaces "plugáveis" para o sistema. Afinal, se a camada do sistema que faz interface com o usuário (classes de fronteira) não se responsabiliza por resolver os eventos gerados, então ela pode ser trocada facilmente. Ou seja, ela oferece um baixo nível de acoplamento, que alias, é o nome de mais um padrão GRASP.

Baixo acoplamento (Low Coupling)

Acoplamento é a medida de quão fortemente uma classe está conectada a outras classes. Essa conexão se mostra em qualquer tipo de relacionamento entre classes, como dependência ou herança, por exemplo. Uma classe com acoplamento baixo (ou fraco) não depende de muitas outras. Já uma com acoplamento alto (ou forte) é mais difícil de compreender isoladamente, mais difícil de reutilizar (seu uso depende da reutilização das outras classes da qual ela depende) e também mais sensível a mudanças nas classes associadas a ela.

Sendo assim, na modelagem de interações, sempre que for possível, evitamos a

criação de mensagens que impliquem em associações redundantes no modelo de classes.

Coesão Alta (High Coesion)

Coesão é a medida do quão fortemente relacionadas e focalizadas são as responsabilidades de uma classe. Uma classe com baixa coesão acaba se responsabilizando por mais coisas do que ela realmente deveria ser responsável. Classes assim acabam oferecendo dificuldade para o seu entendimento e manutenção, além de serem mais difíceis de reutilizar, por não terem um foco bem definido.

Exemplo de execução da análise OO

Para exemplificar de forma rápida o processo de análise OO apresentado até aqui, vamos usar um cenário de caso de uso bastante simplificado, ligado a área de negócio financeira, como mostra a **Figura 6**.

Figura 6. Caso de Uso Consulta Saldo.

Caso de Uso: Consultar Saldo

Descrição: Este caso de uso permite fazer uma consulta simples de saldo em conta bancária.

Ator Principal: Cliente

Pré-Condições: O Cliente deve estar devidamente identificado no sistema, como correntista do banco.

Fluxo de Eventos Básico – Consultar Saldo:

1. Cliente informa agência e conta;
2. Cliente solicita saldo;
3. Sistema solicita senha;
4. Cliente informa senha;
5. Sistema informa nome do cliente, agencia, conta e saldo atual.

Fluxos de Eventos Alternativos: N/A (não se aplica)

Pós-Condições: N/A (não se aplica)

Regras: Caso a conta corrente do cliente esteja vinculada a uma conta poupança, o saldo atual deve apresentar os valores disponíveis nas duas contas.

Fluxos de Exceção das Regras: N/A (não se aplica)

Mensagens dos Fluxos de Exceção: N/A (não se aplica)

Analisando o caso de uso, identificamos as seguintes abstrações chave:

· **Cliente** - pessoa ou empresa que utiliza os serviços do banco;

· **Conta** - representação dos valores monetários do cliente no banco;

· **Agência** - representação de um conjunto de contas.

Com base nos passos do caso de uso e usando os estereótipos de análise, chegamos ao

diagrama de seqüência da **Figura 7** e por conseqüência, ao digrama de classe da **Figura 8.**

Observando esses dois diagramas e lembrando dos conceitos sobre estereótipos de análise, podemos deduzir que:

· Criamos a classe *ConsultaSaldo*, de estereótipo boundary, para que ela sirva de elo entre o usuário do caso de uso e o sistema, responsabilizando-se pela interação entre os dois;

· Criamos a classe *ContaControl*, de estereótipo control, para coordenar as chamadas a classes de entidade, que esse caso de uso necessite;

· Criamos as classes *Agencia* e *Conta*, de estereótipo entity, para armazenar as informações do negócio necessárias ao cumprimento dos passos do caso de uso.

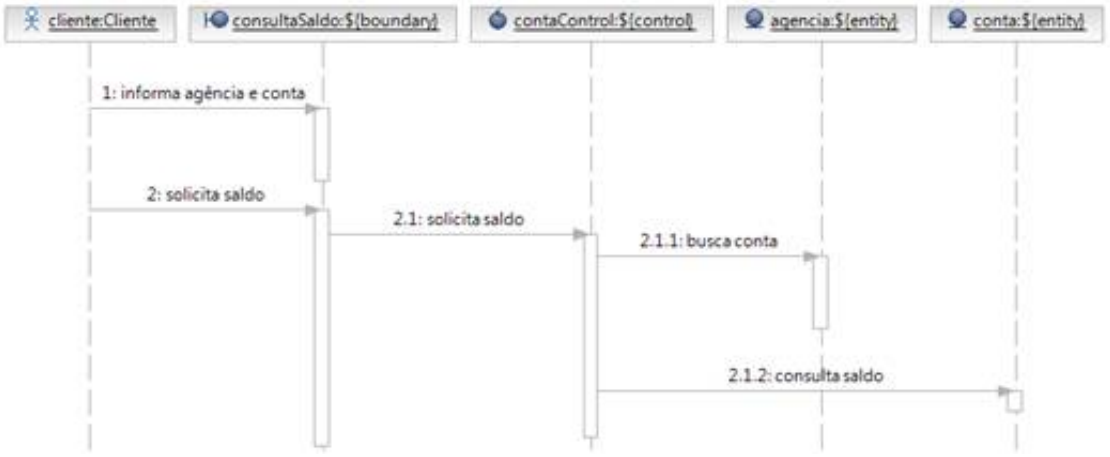


Figura 7. Diagrama de Seqüência Consulta Saldo.

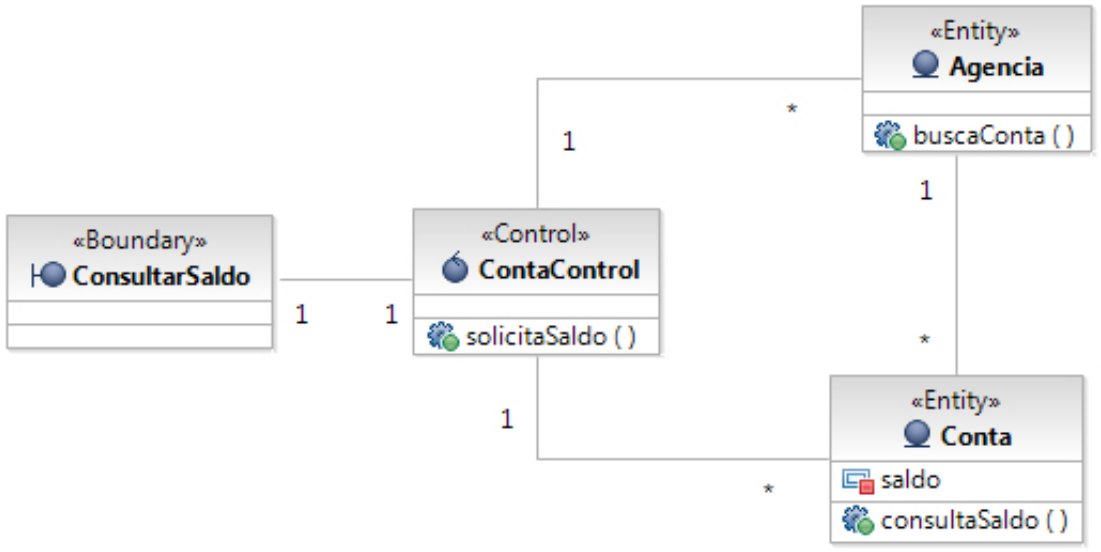


Figura 8. Diagrama de Classe Consulta Saldo.

E finalmente, utilizando os padrões de análise, chegamos a um refinamento nas responsabilidades das classes e também a uma identificação mais precisa dos atributos e operações envolvidos. É muito comum esse refinamento nos levar a descoberta de muitas novas classes, atributos e operações, quando estamos sendo iniciados na análise OO, como mostra o diagrama de seqüência da **Figura 9** e o diagrama de classe da **Figura 10**.

Vejamos então, como as classes e as mensagens trocadas entre elas mudaram depois do refinamento.

Inicialmente a classe *ContaControl* tinha dependência direta para as classes *Agencia* e *Conta*. Ela se responsabilizava por buscar a conta pedindo-a para a classe *Agencia* e por buscar diretamente o saldo contido na classe *Conta*.

Pelo princípio do padrão **Especialista**, quem deveria se responsabilizar pelo acesso às informações da classe *Conta* é a classe *Agencia*. Afinal, a agência é dona de suas contas e essas contas não existem sem sua agência. Sendo assim, criamos uma composição entre as classes *Agencia* e *Conta* e também eliminamos o acesso direto da classe *ContaControl* a classe *Conta*.

Após essas modificações, diminuímos o acoplamento entre as classes, visto que *ContaControl* não está mais acoplada diretamente a *Conta*. Então usamos também o

padrão **Baixo Acoplamento**.

Analisando o caso de uso e as abstrações chave, percebemos algumas coisas, como:

· De acordo com a única regra do caso de uso, a conta pode ser corrente ou poupança;

· A definição da abstração chave Cliente mostra o cliente como uma pessoa ou uma empresa.

Batendo essas informações com o diagrama de classe, vemos que precisaremos melhorar duas coisas: a coesão da classe *Conta* e tratar o conceito de negócio cliente.

A classe *Conta* está se responsabilizando tanto pelas particularidades da conta corrente quanto pelas da conta poupança. Para melhorar sua coesão e assim respeitar o padrão **Coesão Alta**, fizemos o seguinte:

· Tornamos a classe *Conta* abstrata, responsabilizando-a apenas pelo comportamento e informações que forem comuns tanto a conta corrente quanto a conta poupança;

· Criamos as classes concretas *ContaCorrente* e *ContaPoupança*, filhas de *Conta*, para tratar de questões específicas sobre conta corrente e conta poupança.

Analogamente, percebemos que criar apenas uma classe para cliente a levaria ao mesmo problema de coesão baixa. Então ela recebeu o mesmo tratamento que a classe Conta no diagrama de classe.

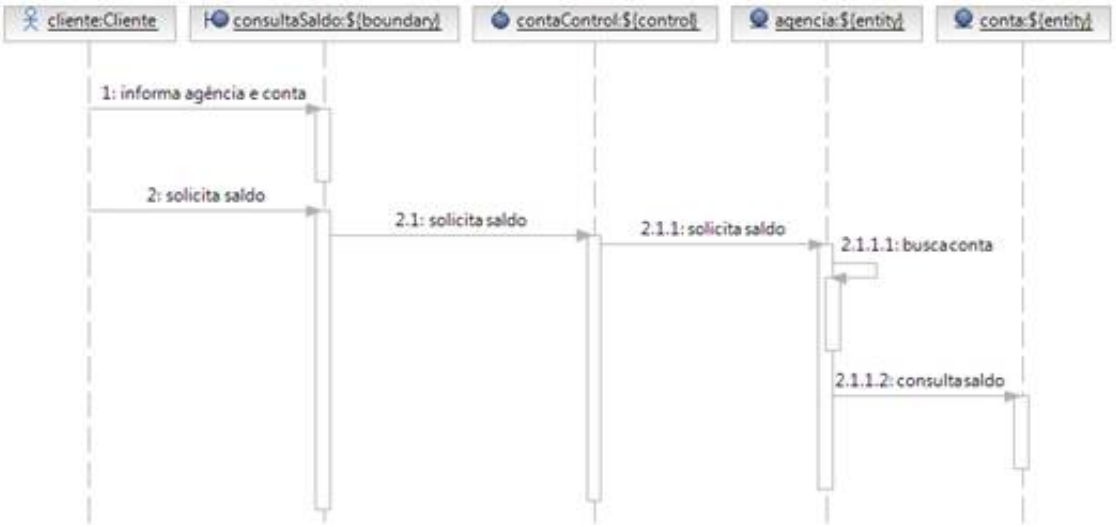


Figura 9. Diagrama de Seqüência Consulta Saldo – Após refinamento.

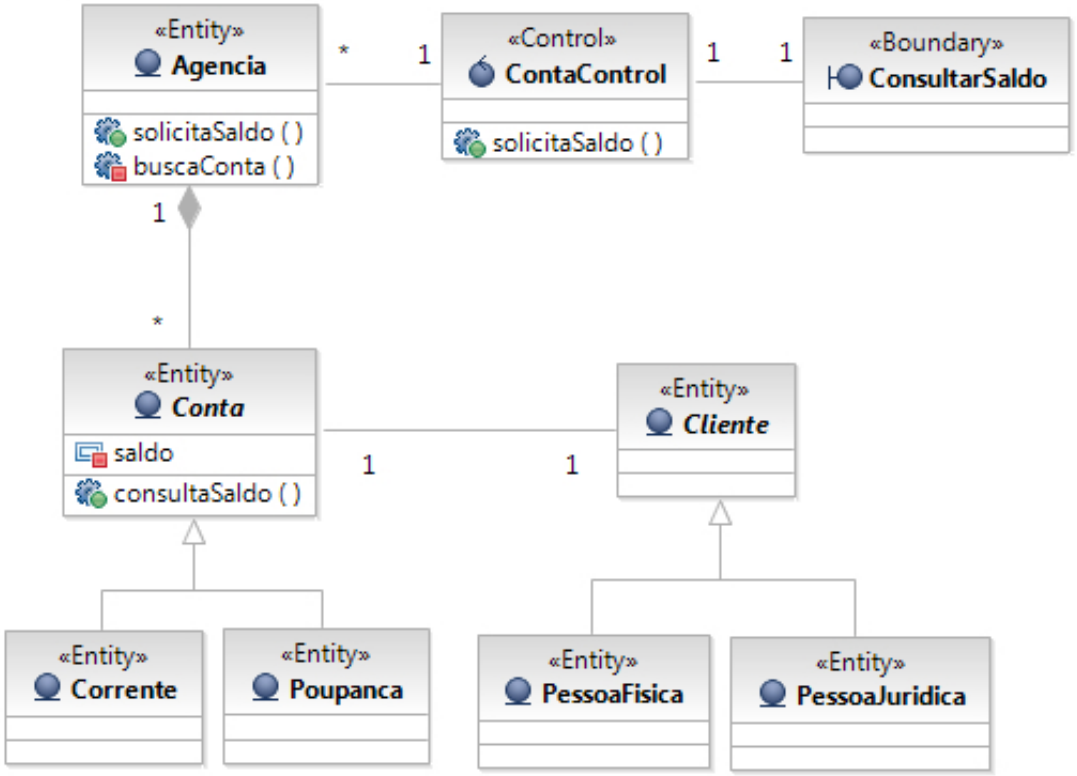


Figura 10. Diagrama de Classe Consulta Saldo – Após refinamento.

Com a experiência no uso dos conhecimentos apresentados nesse artigo, conseguimos definir de forma correta boa parte das responsabilidades de cada classe. Nesse nível de maturidade profissional, a etapa de refinamento acaba sendo mais usada para a eliminação de possíveis redundâncias entre os modelos de classe feitos para cada

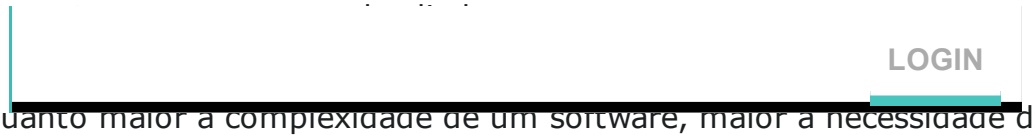
cenário do caso de uso.

Conclusão

Efetuar uma boa análise OO na criação de um sistema é fundamental para o seu sucesso. Por incrível que pareça, muita gente ainda enxerga a análise OO apenas como uma forma de documentar um sistema. Sim, esse é um dos ganhos que se tem com ela. Afinal, com seu uso, o conhecimento sobre o sistema não fica preso apenas na cabeça de quem o construiu, como também não fica submerso em meio a milhares de linhas de código fonte. Isso facilita a reutilização, o entendimento e a manutenção do software. Mas o benefício da documentação não é o único ao se utilizar a análise OO.

A maior vantagem em utilizá-la é justamente poder pensar em uma solução sistêmica focada em responsabilidades, antes de começar a construí-la em linguagens de programação.

É claro que para o sistema que vai controlar seu estoque pessoal de DVDs, a análise OO não se faz tão necessária assim. Mas na medida em que vamos aumentando o nível de complexidade e o tamanho das soluções de software, mais a análise OO se



Quanto maior a complexidade de um software, maior a necessidade de:

· Ter disponível uma visão mais contemplativa possível sobre os problemas que se quer resolver, para assim achar os melhores caminhos a seguir;

· Poder traçar estratégias isoladas ou abrangentes;

· Poder identificar de quem é a responsabilidade sobre determinado conceito.

Unindo UML, estereótipos de análise e padrões de divisão de responsabilidade, a Análise Orientada a Objetos se encaixa como uma boa alternativa para a solução destas necessidades.

Links

OMG - The Object Management Group

www.omg.org

RUP - Rational Unified Process 7.0

http://www-306.ibm.com/software/awdtools/rup/

Bibliografia

UML Essencial

Martin Fowler, Editora Bookman

Utilizando UML e Padrões

Craig Larman, Editora Bookman

Marcelo C. Araújo

Atua no ramo de engenharia de software a 6 anos, trabalhando com customização de metodologias basedas em UP(Unified Process) e participação em varios projetos nos papeis de: analista de processo de negocio, especificador de requi [...]

Comentário | Tire sua dúvida

Luís Gabriel Nascimento Simas

Muito bom o artigo e está marcado como recomendado em minhas redes Sociais, estou utilizando a Análise Orientada a Objetos em todos os meus Sistemas e tem funcionado muito bem. Abraços e obrigado pelo compartilhamento do conhecimento.

[há +1 ano] - Responder

Serviços

- Inclua um comentário
- Adicionar aos Favoritos
- Marcar como lido/assistido
- Incluir anotação pessoal

+Engenharia de software

Mais posts

Artigo

Desenvolvendo Plug-ins para o software Astah

Artigo

Scrum e Planning Poker: Análise de estimativa de software

Video aula

Desenvolvendo a Declaração do Escopo do Projeto - Curso de PMBOK - Gerenciando projetos com Excelência - Aula 48

Video aula

Desenvolvendo o Plano de Gerenciamento dos Requisitos e a Documentação dos Requisitos - Curso de PMBOK - Gerenciando projetos com Excelência - Aula 47

Video aula

Desenvolver o Plano de Gerenciamento do Escopo - Curso de PMBOK - Gerenciando projetos com Excelência - Aula 46

Video aula

Desenvolvendo o Plano de Gerenciamento do Projeto - Curso de PMBOK - Gerenciando projetos com Excelência - Aula 45

Video aula

Criação do Registro das Partes Interessadas - Curso de PMBOK - Gerenciando projetos com Excelência - Aula 44

Artigo

Engenharia Web baseada na UML

Artigo

Integrando XP as principais metodologias agéis

Artigo

Gestão Empresarial com Inteligência Analítica de Negócios

Listar mais conteúdo