

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Aplicação web corporativa

Melhores práticas em uma aplicação web de verdade com MVC e Entity Framework

Neste artigo veremos

Boas práticas de arquitetura; Aspectos de testabilidade e Paradigmas de arquitetura, como DDD

Qual a finalidade

Demonstrar o uso de padrões comprovados para a criação de uma aplicação de negócios durável.

Quais situações utilizam esses recursos?

Aplicações que tenham um tempo de vida longo, ou de alta complexidade, vão se beneficiar dos conceitos expostos no artigo, já que terão maiores chances de sucesso.



Resumo do DevMan

Criar aplicações ASP.Net é algo que se faz há 7 anos. Ainda assim, com frequência encontramos aplicações desenvolvidas sem padrões, que dificultam muito sua manutenção, ou nós mesmos desenvolvemos aplicações que, ao longo do projeto, já estão engessadas. Para resolver estes e outros problemas, padrões de arquitetura e de projetos comprovados pelo mercado e pelo tempo, vêm à nossa salvação. Este artigo mostra como é possível construir uma aplicação focada nestes conceitos.

Qual a melhor estrutura para uma solução? Quais os melhores padrões de arquitetura para determinado cenário? Qual estratégia de tratamento de erros utilizar em tal aplicativo? Quais design patterns (padrões de projeto) se encaixam melhor nos requisitos levantado? Estas são apenas algumas perguntas com que nos deparamos quando iniciamos o desenvolvimento de um aplicativo. São

perguntas que o arquiteto de software deve responder, e, se respondidas de maneira errada, ou sem visão de futuro, podem levar ao fracasso do projeto.

Neste artigo vou demonstrar a montagem de uma aplicação em que as preocupações reais do dia a dia do negócio serão atendidas. São requisitos reais, que um artigo focado em tecnologia não consegue normalmente atender. O foco deste artigo será atender requisitos



Giovanni Bassi

giggio@giggio.net

É MCSD .Net e MCPD Enterprise Application Developer e trabalha com a plataforma .Net há sete anos. É consultor especialista em .Net, focado principalmente em arquitetura e melhores práticas. Além da consultoria de arquitetura trabalha auxiliando empresas na adoção de novas tecnologias Microsoft e na definição da estratégia de engenharia de departamentos de TI. Também realiza atividades de mentoring, ministra treinamentos especializados e dá palestras sobre .NET e arquitetura de software. Giovanni fundou recentemente um grupo de estudos sobre arquitetura de software. Acesse seu blog em <http://unplugged.giggio.net>.

que você recebe na montagem de uma aplicação real. Alguns pontos abordados serão:

- 1 - Qual estratégia de mapeamento objeto/relacional utilizar;
- 2 - Como montar uma estratégia de tratamento de erros eficiente;
- 3 - Como preparar a aplicação para aceitar mudanças de maneira flexível, sem obrigar que todo o código seja revisado ou refeito, mas apenas as partes envolvidas;

Todas estas perguntas serão respondidas diante de um cenário, que vou apresentar em seguida. Como se trata de uma aplicação Web, adotaremos o Beta do ASP.Net MVC como front-end (depois você verá porque), e utilizaremos Entity Framework (EF) para obter os dados.

A aplicação

Construiremos uma aplicação de e-commerce, ao estilo E-Bay, onde um usuário pode colocar um produto próprio para vender, e outro usuário pode comprá-lo. Toda aplicação precisa de uma análise do negócio, ou seja, precisa definir seu domínio. Diante disso, os seguintes casos de uso serão atendidos:

1. Cadastrar-se no site;
2. Logar no site;
3. Exibir produtos à venda;
4. Cadastrar um produto para vender, sendo possível cadastrar produtos do tipo livro ou veículo;
5. Criar uma veiculação para um produto;
6. Pagar por uma veiculação;
7. Exibir um produto para possível compra
8. Fazer a proposta de compra de um produto;
9. Exibir propostas de compra para o vendedor do produto;
10. Aceitar ou rejeitar as propostas de compras;
11. Exibir os erros da aplicação.

As Figuras de 1 a 3 mostram os diagramas destes casos de uso. Atente para os números dos casos de uso, porque eles serão utilizados ao longo do artigo.

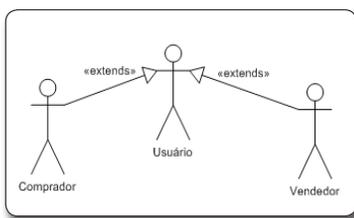


Figura 1. Atores do sistema

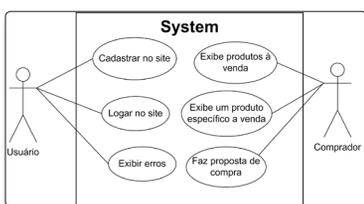


Figura 2. Atores "Usuário" e "Comprador" e seus casos de uso

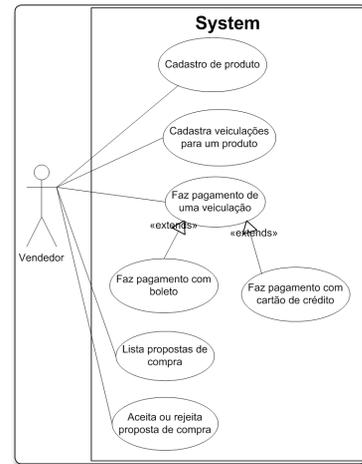


Figura 3. Ator "Vendedor" e seus casos de uso

O aplicativo funcionará da seguinte forma: o usuário acessa o aplicativo, onde vê a página inicial da aplicação, conforme Figura 4. Neste caso, note que o usuário já está logado (veja no canto superior direito na tela o login apresentado), e pode ver os produtos à venda na área principal da aplicação (caso de uso número 3), assim como os produtos que precisam ser verificados, pois têm proposta de vendas, na parte inferior da aplicação (caso de uso número 9).



Figura 4. Home page da aplicação

O usuário poderá então, cadastrar produtos. Ele o faz clicando na aba "Meus produtos", no canto superior direito, onde a tela apresentada na Figura 5 é exibida. Note que todos os seus produtos são exibidos, possibilitando editar os produtos existentes ou cadastrar um novo (caso de uso número 4). Novamente é exibida a lista de produtos com solicitação de compra, na parte inferior da aplicação (caso de uso número 9).

Ao clicar no link de incluir produtos, o usuário é direcionado à tela de inclusão. Além de ser possível incluir livros, é também possível cadastrar produtos do tipo veículos. Em ambos os casos, se algum campo não for informado, ou for inválido, um erro deve ser apresentado, conforme a Figura 6, que apresenta a tela de cadastro de veículos com um erro por falta de preenchimento de campos obrigatórios. A tela de cadastro de livros é semelhante, contendo campos para editora e autor, entre outros.



Figura 5. Tela de produtos

Após cadastrado o produto, o usuário é direcionado a uma tela onde pode criar uma veiculação para o produto (caso de uso número 5), conforme exibido na Figura 7, onde as veiculações são listadas. Note que é possível cadastrar mais de uma veiculação por produto, e que a tabela exibe os pagamentos da veiculação em uma das colunas.

É feito então o pagamento da veiculação, conforme exibido na Figura 8, onde se vê a tela de pagamento (caso de uso número 6). Neste caso trata-se de um pagamento de cartão de crédito, mas poderia ser um pagamento feito com boleto, onde a tela de pagamento é um pouco diferente. Após o pagamento o produto passa a ser exibido na home page da aplicação, na coluna apropriada.

Figura 6. Cadastrando um veículo: há campos obrigatórios.

Caso um usuário se interesse por um produto, ele clica sobre o produto exibido na Figura 4, e uma tela de exibição de detalhes do produto é exibida (caso de uso número 7). Pode então solicitar a compra (caso de uso número 8), e uma tela é exibida com os dados do produto. O usuário comprador pode então clicar em um botão de confirmação de proposta de compra, e o produto é marcado para que o vendedor possa aprovar a compra. O vendedor então poderá verificar quem é o comprador e aprovar ou rejeitar a venda (caso de uso número 10) em uma tela simples com um botão de aprovar ou rejeitar a proposta.

Há ainda a possibilidade de ocorrer um erro na aplicação (caso de uso número 11). Para ver os erros logados, o usuário poderá clicar na aba de erros, no canto superior direito da tela da aplicação (veja na Figura 4), e então uma lista de erros não tratados será exibida, conforme a Figura 9. Apenas os erros não tratados, ou seja, bugs, serão reportados nesta página. Quando um bug acontecer, além da realização do log, a view padrão de

exibição de erros do ASP.Net MVC será exibida.

Figura 7. Exibindo as veiculações e seus pagamentos

Figura 8. Realizando o pagamento com cartão de crédito

O login (caso de uso número 2) é feito da maneira usual do ASP.Net MVC, sem mudanças. Já o cadastro do usuário foi alterado (caso de uso número 1), conforme a Figura 10, para incluir as propriedades nome e sobrenome, que não estão presentes normalmente no ASP.Net MVC. Mais a frente veremos como fazer para armazenar essas informações, mas já adiante que utilizamos os profiles do ASP.Net, mas sem quebrar as regras de encapsulamento e mantendo um padrão de repositório.

Figura 9. Lista de erros

Esse é o cenário que iremos atender. Obviamente não compreende todas as funções de um site de compra e venda profissional, mas atende as operações mais importantes de toda a aplicação, como as operações CRUD, e regras de negócios complexas. Existem interações interessantes, onde precisaremos obter objetos que dependem uns dos outros, fazer buscas de acordo com determinado critério, especificado por um objeto, e até operações que envolvem aplicativos externos ao sistema que está sendo desenvolvido, como as operações de pagamento, que devem se integrar com um aplicativo bancário.

Mais importante do que o negócio apresentado até este instante, ou as regras de negócio apresentadas a seguir, são os padrões aplicados a cada situação. Explicarei os pontos onde os padrões são aplicados na prática e porque foram escolhidos.

Visão de futuro

Antes de prosseguirmos na solução proposta, vamos abordar os motivos que nos levam a montar uma arquitetura mais complexa. Não seria necessário montar uma arquitetura focada em melhores práticas se acreditássemos que a solução teria somente os requisitos apresentados até aqui. Preparamos uma arquitetura fundada em padrões reconhecidos justamente porque sabemos que a aplicação não vai manter os mesmos requisitos durante toda sua existência. Na verdade, aplicativos do mundo real não mantêm os mesmos requisitos nem mesmo durante o desenvolvimento do projeto inicial, quem diria ao longo de toda sua existência.

Figura 10. Cadastro de usuário

Durante toda a vida do aplicativo, o mesmo deverá passar o teste mais difícil de todos: o tempo. A média de investimento em TI de uma empresa, cujo negócio principal não é TI, fica entre 70% e 80% para manutenção de software, e somente o restante para criação de software novos. Isso significa que a maior parte deste investimento vai na manutenção do legado. E só de ouvir a palavra “legado”, muitos já se assustam, porque muitas vezes significa código difícil de manter (às vezes impossível). É esse desafio que uma arquitetura bem montada quer vencer.

Dito isto, nas consultorias de arquitetura que realizo, sempre recomendo que os padrões sejam utilizados, mesmo para aplicações muito pequenas. É muito comum empresas ou usuários afirmarem que um aplicativo é pequeno, não vai ter muitas atribuições, não será alterado no futuro, ou ainda que terá vida curta. É também muito comum que estes aplicativos sobrevivam por muito tempo, não sejam substituídos, e, pior de tudo, cresçam desordenadamente. Já perdi a conta de quantas vezes encontrei um aplicativo, que era para ser um simples aplicativo CRUD e foi desenvolvido na base do “arrastar e soltar”, funcionando depois de anos do que era esperado, e depois de diversas atualizações, onde agora vivem regras de negócios complexas e às vezes fundamentais para o dia a dia da empresa. Não caia nesse erro.

Se este projeto fosse um projeto do mundo real eu também recomendaria que fosse desenvolvido aplicando um ciclo de desenvolvimento ágil, de forma a garantir que os requisitos seriam implementados o mais próximo possível da realidade.

O tipo de arquitetura que será montada ao longo deste artigo, baseada em DDD, como veremos a seguir, e que abraça as mudanças de forma muito flexível (ao invés do mais comum: brigar com as mudanças), facilita muito o desenvolvimento no modelo ágil, e é recomendado pela maior parte dos praticantes do Domain Driven Design.

Camadas: Tiers e Layers

Vou começar apresentando as camadas da aplicação. O desenho das camadas está exposto na Figura 11. Em inglês o termo camada pode ser traduzido como tier, que tem significado que se aproxima mais de camadas físicas, ou como layer, que significa camadas lógicas. Neste caso, estamos vendo uma mistura das duas. Isso não é um problema uma vez que o foco do diagrama é passar uma informação, e de nada adiantaria um diagrama que focasse em tiers ou layers, mas ficasse incompleto. Diante deste foco: resolver um problema, o diagrama atende às necessidades que motivaram sua criação.

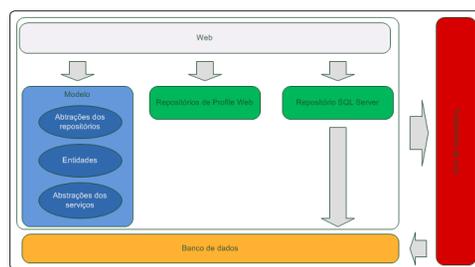


Figura 11. Camadas da aplicação

Teremos como camada de apresentação uma aplicação web, onde utilizaremos, como já foi dito, ASP.Net MVC. O banco será SQL Server, podendo ser utilizada qualquer versão a partir do SQL Server 2000. As camadas intermediárias são bibliotecas, baseadas no template Class Library do Visual Studio. Na Figura 12 você vê a estrutura de projetos do Visual Studio.

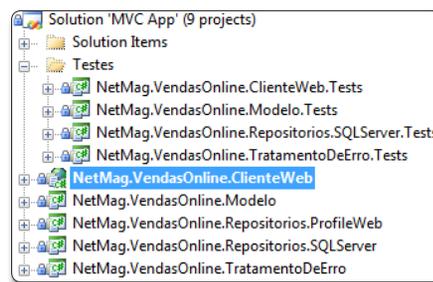


Figura 12. Projetos no Visual Studio

Monte esta estrutura de projetos no Visual Studio, lembrando que todos os projetos são baseados no template Class Library, com exceção do projeto ClienteWeb, que é baseado em ASP.Net MVC, versão beta. Os projetos de testes são baseados no Template Test Project, e são criados automaticamente pelo Visual Studio quando você clica com o botão direito sobre o código de uma classe e seleciona “Create Unit Tests...”. Para

mais referência sobre testes unitários veja a série de três artigos que publiquei na .Net Magazine, da edição 53 a 55.

O projeto será baseado em Domain Driven Design (DDD), por ser uma abordagem que atenderá muito bem às necessidades propostas, como regras de negócios complexas, facilidade de manutenção e testabilidade. Na edição 57, publiquei um artigo onde introduzi o Domain Driven Design, e sugiro que você reveja o texto deste artigo para absorver melhor os conceitos que serão apresentados neste artigo, como serviços, repositórios, entidades e objetos de valor.

Conforme proposto pelo DDD, temos uma separação clara da nossa camada de domínio da aplicação. É lá que colocaremos as regras de negócios. O projeto NetMag.VendasOnline.Modelo é o que representará esta camada e está representado em azul na **Figura 11**. É por lá que o desenvolvimento da aplicação será iniciado, e não pela base de dados, algo mais comum em projetos deste tipo (note que eu disse “mais comum”, e não “melhor”). Há no entanto um motivo para iniciar pelo modelo de domínio, e não de dados. Quando você começa a aplicação pelo MER, ou diretamente modelando as tabelas no banco de dados, há uma tendência de que você trabalhe focado nos dados, o que chamamos Data Driven Development, ou desenvolvimento focado em dados, e não focado no domínio (ou negócio). Os adeptos do TDD começariam pelos testes, mas o propósito aqui não é detalhar TDD, então deixaremos este assunto para outra oportunidade.

A camada de tratamento de erros, apresentada em vermelho na **Figura 11**, e representada pela aplicação NetMag.VendasOnline.TratamentoDeErro é totalmente independente das outras camadas, para que possa ser acessado por todas elas. Ela referencia apenas o banco de dados, que é seu repositório final, ainda que tenha outro repositório de backup (utilizado em caso de falha do banco), como veremos quando falarei de tratamento de erros. Note que esta camada não referencia nem mesmo a camada de repositório do SQL Server, mas é referenciada por ela, já que erros também podem ocorrer na camada que contém os repositórios, e estes precisam ser logados da mesma forma.

Os repositórios estão separados em duas camadas: uma que fará contato com o SQL Server (projeto NetMag.VendasOnline.Repositorios.SQLServer) e outra que fará o contato com o sistema de Profiles do ASP.Net (projeto NetMag.

VendasOnline.Repositorios.ProfileWeb). Ambas estão representadas em verde na **Figura 11**.

Começando o projeto: camada de domínio

Começamos a aplicação desenvolvendo a camada de domínio, como foi falado quando apresentei as camadas da aplicação. Na **Figura 13** você vê o digrama de classes das principais entidades da aplicação: produto e usuário. Note que as entidades são todas representadas por interfaces, e então concretamente construídas com classes. A classe concreta do usuário está na camada de Repositório de profiles web por uma questão de separação entre as camadas que veremos adiante. Note que a entidade produto é representada concretamente pelas classes Livro e Veiculo, que nada fazem além de definir algumas propriedades extras. Na classe base Produto estão apresentados os métodos principais da entidade, como os que aceitam ou rejeitam uma venda, ou que propõem uma nova venda. O código da classe produto é apresentado na **Listagem 1**. Note como as regras de negócios são mantidas perfeitamente pela própria entidade, como pode ser visto no método ProporVenda (linha 24), e como a entidade não possui conhecimento algum sobre como será realizada sua persistência no banco de dados. Esse tipo de objeto é chamado de POCO, que significa Plain Old CLR Object, ou, bom e velho objeto do CLR. Os construtores são parametrizados com os valores das propriedades e, assim como as propriedades (todas implementadas com propriedades automáticas), estão ocultos, mas podem ser vistos no diagrama da **Figura 13**. As classes livro e veiculo não possuem comportamento específico, mas herdam o comportamento da classe base, e suas propriedades também são automáticas, e podem ser vistas também na **Figura 13**.

Os objetos de valor, neste caso, são os status da venda, e as categorias. O diagrama não mostra, mas as propriedades Veiculos e Livros são estáticas, e apresentam as categorias concretas, que na verdade implementam as categorias, conforme pode ser visto na **Listagem 2**, nas linhas 9 e 12. Note também que sobrescrevi os métodos Equals e GetHashCode, e também os operadores == e !=. Isso foi feito para que, quando um objeto de categoria seja comparado com outro com estes operadores ou com os métodos Equals o GetHashCode, eles apenas verifiquem se o Id é o mesmo, e se for, assumam como o mesmo objeto. Isso é muito comum em objetos de valor, porque não nos importamos com a instância que temos (eles são imutáveis), mas apenas com os valores (é um livro ou é um veículo). Note ainda que os objetos Livro e Veiculo são Singletons, e são compartilhados por toda a aplicação. Não precisamos criar um objeto de valor toda vez que ele for utilizado, e por isso



Nota do DevMan

Test Driven Development (TDD)

O TDD é uma técnica de desenvolvimento em que os testes vêm antes do desenvolvimento dos métodos da aplicação em si. Você escreve código que vai testar classes e métodos que às vezes nem existem, sob a perspectiva de usuário da classe.

Depois de construído um teste, escreve-se o código que fará o teste passar da forma mais simples possível. Por fim, refatora-se o código escrito, tendo em mente que o teste deve continuar passando. O processo deve ser realizado a cada nova funcionalidade que for adicionada ao projeto.

Esta técnica vem ganhando adeptos a cada dia, e alguns de seus usuários muitas vezes a defendem com o mesmo fervor de um time de futebol. Vale a pena conferir.

Nota

O desenvolvimento focado em dados é provavelmente o mais utilizado quando se fala em ASP. Net. A maioria das ferramentas do ASP.Net é focada neste paradigma de desenvolvimento, como é o caso dos Datasets e TableAdapters, que são componentes de contato direto e manipulação de dados, e do GridView, DataList, DetailsView, e SqlDataSource, que são componentes de apresentação dos dados na interface gráfica.

o Singleton foi utilizado, o que acaba por poupar memória. Ainda que o benefício da memória poupada seja muito pouco hoje em dia – memória é muito barato – o padrão valeu a pena neste contexto. Não utilize um singleton a não ser que ele seja realmente válido, porque muitas vezes eles trazem problemas, como em casos em que se cruza operações multi-threading e objetos muito complexos.

Não há necessidade de apresentar a interface IUserario, já que o diagrama a define por completo. O código completo da aplicação encontra-se disponível no site da revista para download.

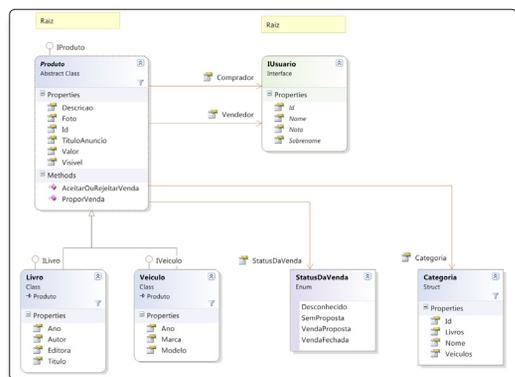


Figura 13. Diagrama de classes das entidades Produto e Usuário

Um ponto importante a notar é o construtor. Note que as entidades possuem sempre dois construtores. O primeiro é o construtor utilizado quando vai se “reidratar” o objeto, ou seja, recompô-lo a partir do banco de dados. Trata-se de um objeto já pré-existente, já havia iniciado seu ciclo de vida, somente não estava na memória, e este construtor permite trazê-lo de volta, incluindo propriedades que não devem ser escritas indiscriminadamente, como o Id. O segundo é utilizado na criação de um objeto novo, que não existia antes, e não possui as propriedades que são configuradas automaticamente na criação como o status da venda de um novo produto, que é sempre “sem proposta”.

A Figura 14 apresenta o diagrama da agregação de veiculação, que tem a classe principal Veiculacao apresentada na Listagem 3. Da mesma forma, esta classe possui os dois construtores principais. Note, no entanto, que ela chama o método AcertarCustoEStatusLiberacao após a construção, onde ela configurará a si mesma, setando a propriedade Custo, se for necessário (veja método AcertarCusto, na linha 107), e também as propriedades EstaLiberada e Inicio, se os pagamentos estão acima do valor do custo da veiculação. Ela verifica sua própria coleção de pagamentos em uma bela expressão LINQ no método ObterValorTotalAprovado da linha 140. No construtor, a coleção de pagamentos sempre estará vazia, porque ela não é passada via construtor, de forma que a veiculação nunca vai estar liberada na construção. Mas conforme os pagamentos são adicionados, ou na reidratação do banco de dados, ou através de um processo de negócio, a propriedade é alterada e fica correta. A classe de lista de pagamento é na verdade uma *ObservableCollection<IPagamento>*, que lança eventos sempre

que um item é adicionado. Ao tratar estes eventos no método pagamentos_CollectionChanged (linha 83), somos capazes de tratar as propriedades internas de maneira bastante simples.

O problema da propriedade ficar desconfigurada durante a construção é esperado e acontece porque há uma relação bidirecional entre a entidade de veiculação e de pagamento, conforme a Figura 14 deixa claro (note que a dependência é, na verdade às suas interfaces). O problema só é realmente real em casos de reidratação de objetos, o que só acontece no repositório que nós mesmos controlamos.

A entidade recebe ainda uma referência para o serviço de cálculo de veiculação, que é responsável por fazer apresentar o custo por determinada veiculação (veja linha 110). O código deste serviço é bastante simples e pode ser visto na Listagem 4. No entanto, encare o serviço como algo mais complexo, que seria responsável por falar com algum sistema legado ou web service, e por isso está fora da responsabilidade da classe de veiculação. Essa composição é importante também para não deixar uma classe de negócio dependente de código de infra-estrutura, e trata-se claramente de um uso do princípio de OO de inversão de dependência.

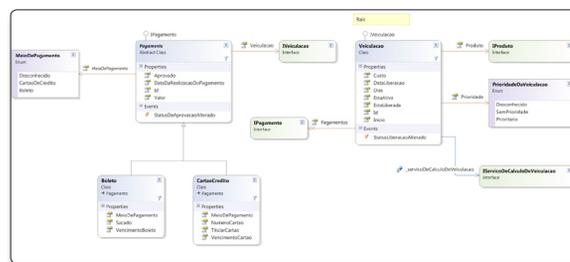


Figura 14. Diagrama de classes das entidades Veiculação e Pagamento

A entidade de pagamento é bastante simples. Está na Listagem 5, e fora suas propriedades base, possui um evento que notifica em caso de sucesso em sua aprovação (linha 44), e uma lógica um pouco mais complexa na propriedade de aprovação (linha 27). Esta propriedade é um *Nullable<bool>*, ou seja, é uma booleana que é anulável. Isso significa que o status do pagamento pode ser aprovado, não aprovado, ou ainda, sem status, que seria antes da realização do pagamento, como por exemplo, antes de solicitar a aprovação de uma conta paga com cartão de crédito. Após a confirmação ou rejeição do pagamento, a propriedade seria então configurada. As classes Boleto e CartaoCredito são simples e possuem apenas as propriedades automáticas demonstradas no diagrama da Figura 14, e os dois construtores, por isso não serão apresentadas.

Serviços: meio no domínio meio na infra-estrutura

Os serviços fazem parte do domínio, mas com frequência realizam atividades que estão fora do escopo de uma atividade de negócio e mais relacionadas a atividades de infra-estrutura, como conectar em um banco de dados, web service, ou sistema legado, mas não obrigatoriamente. Em nossa aplicação temos dois serviços, conforme visto na Figura 15.

Listagem 1. Entidades Produto, Livro e Veiculo

```

1 namespace NetMag.VendasOnline.Modelo.Entidades
2 {
3     public abstract class Produto : NetMag.VendasOnline.
        Modelo. Entidades.IProduto
4     {
5         public Produto(int id, Status DaVenda statusDaVenda,
6             IUsuario comprador, Categoria categoria,
7             IUsuario -vendedor,
8             string tituloAnuncio, decimal valor)
9             : this(categoria,vendedor,tituloAnuncio, valor)
10        {
11            this.Id = id;
12            this.StatusDaVenda = statusDaVenda;
13            this.Comprador = comprador;
14        }
15        public Produto(Categoria categoria, IUsuario vendedor,
16            string tituloAnuncio, decimal valor)
17        {
18            this.Categoria = categoria;
19            this.Vendedor = vendedor;
20            this.TituloAnuncio = tituloAnuncio;
21            this.StatusDaVenda = StatusDaVenda.SemProposta;
22            this.Valor = valor;
23        }
24        public void ProporVenda(IUsuario comprador)
25        {
26            if (this.StatusDaVenda >= StatusDaVenda.VendaPro
27                posta)
28                throw new InvalidOperationException
29                    ("0 produto está vendido ou em processo
30                    de venda.");
31            if (this.Vendedor.Id == comprador.Id)
32                throw new InvalidOperationException
33                    ("0 vendedor não pode ser a mesma pessoa
34                    que o vendedor.");
35            this.Comprador = comprador;
36            this.StatusDaVenda = StatusDaVenda.VendaProposta;
37        }
38        public void AceitarOuRejeitarVenda(IUsuario vendedor,
39            bool aceitar)
40        {
41            if (this.StatusDaVenda != StatusDaVenda.VendaProposta)
42                throw new InvalidOperationException
43                    ("0 produto não tem venda proposta.");
44            if (vendedor.Id != this.Vendedor.Id)
45                throw new InvalidOperationException
46                    ("Somente o vendedor pode aceitar ou rejei
47                    tar a venda de um produto.");
48            if (aceitar)
49                this.StatusDaVenda = StatusDaVenda.VendaFechada;
50            else
51            {
52                this.StatusDaVenda = StatusDaVenda.SemProposta;
53                this.Comprador = null;
54            }
55        }
56        public bool Visivel { get; set; }
57        public int Id { get; set; }
58        public Categoria Categoria { get; private set; }
59        public IUsuario Vendedor { get; private set; }
60        public byte[] Foto { get; set; }
61        public string TituloAnuncio { get; set; }
62        public string Descricao { get; set; }
63        public IUsuario Comprador { get; private set; }
64        public StatusDaVenda StatusDaVenda { get; private set; }
65        public decimal Valor { get; set; }
66    }
67 }
68 namespace NetMag.VendasOnline.Modelo.Entidades
69 {
70     public class Livro : Produto, ILivro
71     {
72         public Livro(IUsuario vendedor, string tituloAnuncio,
73             string titulo, decimal valor)
74             : base(Categoria.Livros, vendedor, tituloAnuncio, valor)
75         {
76             this.Titulo = titulo;
77         }
78         public Livro(int Id, StatusDaVenda statusDaVenda,
79             IUsuario comprador,
80             IUsuario vendedor, string tituloAnuncio, string
81             titulo, decimal valor)
82             : base(Id, statusDaVenda, comprador, Categoria.
83                 Livros, vendedor, tituloAnuncio, valor)
84         {
85             this.Titulo = titulo;
86         }
87         public string Titulo { get; set; }
88         public string Autor { get; set; }
89         public string Editora { get; set; }
90         public short? Ano { get; set; }
91     }
92 namespace NetMag.VendasOnline.Modelo.Entidades
93 {
94     public class Veiculo : Produto, IVeiculo
95     {
96         public Veiculo(IUsuario vendedor, string titulo
97             Anuncio, short ano,
98             string marca, string modelo, decimal valor)
99             : base(Categoria.Veiculos, vendedor, tituloAnuncio, valor)
100        {
101            this.Ano = ano;
102            this.Marca = marca;
103            this.Modelo = modelo;
104        }
105        public Veiculo(int Id, StatusDaVenda statusDaVenda, IU
106            suario comprador,
107            IUsuario vendedor, string tituloAnuncio,
108            short ano, string marca,
109            string modelo, decimal valor)
110            : base(Id, statusDaVenda, comprador, Categoria.
111                Veiculos,
112                vendedor, tituloAnuncio, valor)
113        {
114            this.Ano = ano;
115            this.Marca = marca;
116            this.Modelo = modelo;
117        }
118        public short Ano { get; set; }
119        public string Marca { get; set; }
120        public string Modelo { get; set; }
121    }
122 }

```

Listagem 2. Objeto de valor Categoria

```

1 namespace NetMag.VendasOnline.Modelo.Entidades
2 {
3     public struct Categoria
4     {
5         public string Nome { get; private set; }
6         public int Id { get; private set; }
7
8         private static Categoria _Veiculos = new Categoria()
9             { Id = 1, Nome = "Veiculos" };
10        public static Categoria Veiculos
11            { get { return _Veiculos; } }
12        private static Categoria _Livros = new Categoria()
13            { Id = 2, Nome = "Livros" };
14        public static Categoria Livros
15            { get { return _Livros; } }
16        public override bool Equals(object obj)
17        {
18            if (!(obj is Categoria))
19                return false;
20            return ((Categoria)obj).Nome == this.Nome;
21        }
22        public static bool operator ==(Categoria objA,
23            Categoria objB)
24        { return objA.Equals(objB); }
25        public static bool operator !=(Categoria objA,
26            Categoria objB)
27        { return !objA.Equals(objB); }
28        public override int GetHashCode()
29        { return this.Id; }
30    }

```



Figura 15. Diagrama de classes dos serviços da aplicação

Ambos os serviços foram implementados na camada de domínio, mas dependendo das atividades, poderiam ter ficado fisicamente em outro local, já que o que interessa mesmo são suas interfaces, que definem os padrões de comunicação. Já demonstrei o serviço de cálculo de veiculação na **Listagem 4**, por causa de sua relação com a classe de veiculação. O serviço de cobrança, no entanto, não será utilizado pela camada de negócios, mas será consumido pela camada de interface gráfica. Seu código está na **Listagem 6**, e, assim como o serviço de cálculo, é apenas uma implementação simples do que seria um serviço de cobrança baseado em diversos outros aplicativos.

Listagem 3. Entidade Veiculação

```

1 namespace NetMag.VendasOnline.Modelo.Entidades
2 {
3     public class Veiculacao : NetMag.VendasOnline.Modelo.
4         Entidades.IVeiculacao
5     {
6         public Veiculacao(IServicoDeCalculoDeVeiculacao
7             servicoDeCalculoDeVeiculacao,
8             int id, DateTime? dataLiberacao, IProduto produto,
9             decimal custo,
10            int dias, DateTime inicio, PrioridadeDaVeiculacao
11            prioridade)
12        {
13            var pagamentos = new ObservableCollection<IPagamento>();
14            pagamentos.CollectionChanged +=
15                new NotifyCollectionChangedEventHandler(_
16                pagamentos_CollectionChanged);
17            this.Pagamentos = pagamentos;
18            this._servicoDeCalculoDeVeiculacao =
19                servicoDeCalculoDeVeiculacao;
20            this.Produto = produto;
21            this.Custo = custo;
22            this.Dias = dias;
23            this.Inicio = inicio;
24            this.Prioridade = prioridade;
25            this.Id = id;
26            this.DataLiberacao = dataLiberacao;
27            //após acertar as propriedades manter o status
28            //geral das outras propriedades dependentes
29            AcertarCustoEStatusLiberacao();
30        }
31
32        public Veiculacao(IServicoDeCalculoDeVeiculacao
33            servicoDeCalculoDeVeiculacao,
34            IProduto produto, int dias, DateTime inicio,
35            PrioridadeDaVeiculacao prioridade)
36        {
37            var pagamentos = new ObservableCollection<IPagamento>();
38            pagamentos.CollectionChanged +=
39                new NotifyCollectionChangedEventHandler(_
40                pagamentos_CollectionChanged);
41            this.Pagamentos = pagamentos;
42            this._servicoDeCalculoDeVeiculacao =
43                servicoDeCalculoDeVeiculacao;
44            this.Produto = produto;
45            this.Dias = dias;
46            this.Inicio = inicio;
47            this.Prioridade = prioridade;
48            //após acertar as propriedades manter o status
49            //geral das outras propriedades dependentes
50            AcertarCustoEStatusLiberacao();
51        }
52
53        private IServicoDeCalculoDeVeiculacao _
54            servicoDeCalculoDeVeiculacao;
55
56        public IProduto Produto { get; private set; }
57        public int Id { get; set; }
58        public IList<IPagamento> Pagamentos { get; private set; }
59        public bool EstaLiberada { get; private set; }
60        public DateTime Inicio { get; set; }
61        public DateTime? DataLiberacao { get; set; }
62        public decimal Custo { get; set; }
63
64        private PrioridadeDaVeiculacao _prioridade;
65        public PrioridadeDaVeiculacao Prioridade
66        {
67            get { return _prioridade; }
68            set
69            {
70                _prioridade = value;
71                AcertarCustoEStatusLiberacao();
72            }
73        }
74        private int _dias;
75        public int Dias
76        {
77            get { return _dias; }
78            set
79            {
80                _dias = value;
81                AcertarCustoEStatusLiberacao();
82            }
83        }
84        public bool EstaAtiva
85        {
86            get
87            {
88                if (!this.Estaliberada)
89                    return false;
90                return (this.DataLiberacao.Value <=
91                    DateTime.Now
92                    && this.DataLiberacao.Value.
93                        AddDays(this.Dias) >= DateTime.Now);
94            }
95        }
96        void _pagamentos_CollectionChanged(object sender,
97            NotifyCollectionChangedEventArgs e)
98        {
99            if (e.Action == NotifyCollectionChangedAction.Add)
100            {
101                foreach (IPagamento pgto in e.NewItems)
102                    pgto.StatusDeAprovacaoAlterado +=
103                        new EventHandler(Items_
104                        StatusDeAprovacaoAlterado);
105                AcertarCustoEStatusLiberacao();
106            }
107            if (e.Action == NotifyCollectionChangedAction.Remove)
108            {
109                foreach (IPagamento pgto in e.OldItems)
110                    pgto.StatusDeAprovacaoAlterado -=
111                        new EventHandler(Items_
112                        StatusDeAprovacaoAlterado);
113                AcertarCustoEStatusLiberacao();
114            }
115        }
116        void AcertarCustoEStatusLiberacao()
117        {
118            AcertarCusto();
119            AcertarStatusLiberacao();
120        }
121        private void AcertarCusto()
122        {
123            if (this.Custo == 0)
124                this.Custo = this._servicoDeCalculo
125                    DeVeiculacao.CalcularCusto(this);
126        }
127        void AcertarStatusLiberacao()
128        {
129        }
130    }
131 }

```

Continuação - Listagem 3. Entidade Veiculação

```
115         decimal valorAprovado = this.ObterValor
            TotalAprovado();
116
117         if (this.Custo > valorAprovado)
118         {
119             if (this.EstaLiberada)
120             {
121                 this.EstaLiberada = false;
122                 OnStatusLiberacaoAlterado();
123             }
124             else
125             {
126                 if (!this.EstaLiberada)
127                 {
128                     this.EstaLiberada = true;
129                     this.DataLiberacao = DateTime.Now;
130                     if (this.Inicio < this.DataLiberacao)
131                         this.Inicio = this.DataLiberacao.Value;
132                     OnStatusLiberacaoAlterado();
133                 }
134             }
135         }
136
137         void Item_StatusDeAprovacaoAlterado(object sender,
            EventArgs e)
138         {
139             AcertarStatusLiberacao();
140         }
141         decimal ObterValorTotalAprovado()
142         {
143             decimal valorAprovado = 0;
144             this.Pagamentos.Where(p => p.Aprovado.HasValue
145                 && p.Aprovado.Value).ToList()
146                 .ForEach((pgto) => valorAprovado += pgto.Valor);
147             return valorAprovado;
148         }
149         public event EventHandler StatusLiberacaoAlterado;
150         protected void OnStatusLiberacaoAlterado()
151         {
152             if (StatusLiberacaoAlterado != null)
153                 StatusLiberacaoAlterado.Invoke(this,
154                     EventArgs.Empty);
155         }
156     }
157 }
```

Listagem 4. Serviço de cálculo

```
1 namespace NetMag.VendasOnline.Modelo
2 {
3     public class ServicoDeCalculoDeVeiculacao :
4         NetMag.VendasOnline.Modelo.IServico
5         DeCalculoDeVeiculacao
6     {
7         public decimal CalcularCusto(IVeiculacao veiculacao)
8         {
9             //aquí alguma lógica complexa seria
10            implementada
11            //como um contato com um web service ou legado
12            //a lógica abaixo serve apenas para compor um
13            exemplo simples
14            decimal custo = 0;
15            if (veiculacao.Produto.Categoria == Categoria.
16                Livros)
17                custo = 0.07m;
18            else if (veiculacao.Produto.Categoria ==
19                Categoria.Veiculos)
20                custo = 0.8m;
21
22            custo *= veiculacao.Dias;
23            if (veiculacao.Prioridade ==
24                PrioridadeDaVeiculacao.Prioritario)
25                custo *= 1.2m;
26            return custo;
27        }
28    }
29 }
```

Listagem 5. Entidade Pagamento

```
1 namespace NetMag.VendasOnline.Modelo.Entidades
2 {
3     public abstract class Pagamento : NetMag.VendasOnline.
4         Modelo.Entidades.IPagamento
5     {
6         public Pagamento(int id, bool? aprovado, DateTime?
7             dataDaRealizacaoDoPagamento,
8             IVeiculacao veiculacao, decimal valor)
9         {
10             this.Id = id;
11             this.Aprovado = aprovado;
12             this.DataDaRealizacaoDoPagamento =
13                 dataDaRealizacaoDoPagamento;
14             this.Veiculacao = veiculacao;
15             this.Valor = valor;
16         }
17         public Pagamento(IVeiculacao veiculacao, decimal valor)
18         {
19             this.Veiculacao = veiculacao;
20             this.Valor = valor;
21         }
22         public virtual int Id
23         {
24             get; set;
25         }
26         public virtual DateTime?
27             DataDaRealizacaoDoPagamento { get; set; }
28         public abstract MeioDePagamento MeioDePagamento
29             { get; }
30         public virtual decimal Valor
31         {
32             get; private set;
33         }
34         public virtual IVeiculacao Veiculacao
35         {
36             get; private set;
37         }
38         private bool? _aprovado;
39         public virtual bool? Aprovado
40         {
41             get
42             {
43                 return _aprovado;
44             }
45             set
46             {
47                 if (_aprovado.HasValue && !value.HasValue)
48                     throw new ArgumentException("Não pode
49                         desfazer pagamento.");
50                 _aprovado = value;
51                 if (_aprovado.HasValue)
52                 {
53                     DataDaRealizacaoDoPagamento =
54                         DateTime.Now;
55                     OnStatusDeAprovacaoAlterado();
56                 }
57             }
58         }
59         public virtual event EventHandler
60             StatusDeAprovacaoAlterado;
61         private void OnStatusDeAprovacaoAlterado()
62         {
63             if (StatusDeAprovacaoAlterado != null)
64                 StatusDeAprovacaoAlterado.Invoke(this,
65                     EventArgs.Empty);
66         }
67     }
68 }
```

Repositórios: Ressuscitando as entidades

Os repositórios são os responsáveis por trazer os objetos de volta à vida, ou guardar objetos alterados ou novos. Outras operações dependentes do banco de dados (ou qualquer que seja sua fonte de dados) também são realizadas por eles. Há um repositório para cada raiz de agregação da aplicação, sendo então um repositório para usuários, outro para produtos, e mais um para veiculações. A **Figura 16** mostra suas interfaces.

Mencionei no começo do artigo que o repositório de usuário utilizará o sistema de perfis do ASP.Net, no entanto, note que estamos tratando a interface do repositório totalmente independente desta tecnologia. Para o aplicativo, o que interessa são os objetos de negócio, e nesse caso, este objeto é a entidade representada pela interface `IUsuario`, que não depende absolutamente em nada do ASP.Net. Veremos em seguida como essa diferença foi conciliada. Da mesma forma, os repositórios de produto e veiculação também conhecem apenas os objetos de negócio, mas não os objetos do Entity

Framework, que será utilizado para entrar em contato com o banco de dados.

Iniciaremos então com o repositório de usuários, que fica no projeto `NetMag.VendasOnline.Repositorios.ProfileWeb`. Este projeto referencia `System.Web`, e utiliza profiles para trabalhar integrado ao ASP.Net. Não seria possível reaproveitá-lo de maneira muito simples em uma aplicação desktop, mas, diante do benefício, considero que valeu a pena a implementação, que é bastante simples e atendeu perfeitamente às necessidades.

Listagem 6. Serviço de cobrança

```

1 namespace NetMag.VendasOnline.Modelo
2 {
3     public class ServicoDeCobranca : NetMag.VendasOnline.
4         Modelo.IServicoDeCobranca
5     {
6         public void RealizarCobranca(IPagamento pagamento)
7         {
8             switch (pagamento.MeiodePagamento)
9             {
10                case MeioDePagamento.CartaoDeCredito:
11                    RealizarCobrancaCartaoDeCredito(pagamento);
12                    break;
13                case MeioDePagamento.Boleto:
14                    RealizarCobrancaBoleto(pagamento);
15                    break;
16            }
17        }
18        void RealizarCobrancaCartaoDeCredito(IPagamento
19            pagamento)
20        {
21            //vai até a administradora, obten o retorno de
22            imediato
23            //simulacao: 90% das vezes passa
24            var rnd = new Random().Next(100);
25            pagamento.Aprovado = rnd < 90;
26        }
27        void RealizarCobrancaBoleto(IPagamento pagamento)
28        {
29            //manda o título para o banco
30            //não vamos fazer nada, teríamos que
31            //ter outro serviço para fazer checagens
32            periódicas
33            //e alterar o status de um pagamento
34        }
35    }
36 }

```

Na **Listagem 7** você encontra a implementação concreta do usuário. Note que, como a definição da entidade usuário foi feita através de uma interface, que a classe `UsuarioWeb` implementa, ainda foi possível utilizar outra classe para sua herança. Neste caso, a classe utilizada foi a classe `ProfileBase`, do ASP.Net, que é responsável por toda a infra-estrutura do sistema de profiles. A partir daí, cada propriedade delega seus valores a um indexador na classe base, e o `id` funciona diretamente ligado à propriedade `username` da classe base, que é a responsável pela identificação no sistema de profiles, e é setada através do método `Initialize`, que você vê na linha 16. Mais simples impossível, certo?

Na **Listagem 8** está o repositório de usuários. Toda sua interação é também com a infra-estrutura de profiles do ASP.Net. Note que para obter o usuário por `Id`, basta chamar `ProfileBase.Create` (linha 13). Para salvar, basta chamar o método `Save` do próprio usuário. Também muito simples. Seria possível trocar esta implementação por outra baseada em um banco de dados relacional, mas não acredito que o ganho em tempo

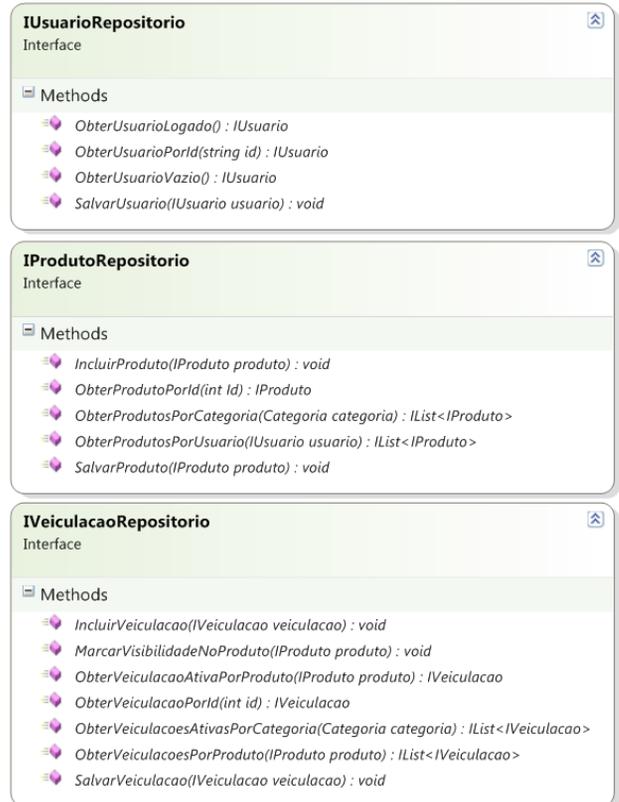


Figura 16. Diagrama de classes dos repositórios

de desenvolvimento seria válido, afinal, são apenas 27 linhas no repositório e 41 no usuário.

Para fazer o repositório baseado em Profiles funcionar, basta configurar o `web.config` do aplicativo web, conforme **Listagem 9**. O ASP.Net MVC também funciona sem problemas com ele. Note na linha 16 a informação ao sistema de profiles que utilizaremos a classe `UsuarioWeb` para os profiles. Na linha 5 também está configurado o atributo `connectionStringName`, que é o nome da sua string de conexão que terá acesso ao banco que você for utilizar. Na listagem também está disponível o código de membership, que também foi ligeiramente alterado.

Você vai precisar ainda instalar a estrutura do banco de dados, utilizando a ferramenta `aspnet_regsql.exe`, que fica no diretório `%windir%\Microsoft.NET\Framework\v2.0.50727\` (onde `%windir%` geralmente é `c:\windows`). A ferramenta é bastante simples, baseada em um wizard, onde você escolhe o banco de dados onde quer criar a estrutura e ela cria para você. Ela também remove, se você precisar.

Como agora vamos tocar nos repositórios de banco de dados, começarei exibindo o MER, na **Figura 17**. Ele reproduz claramente nossos objetos, de forma normalizada, incluindo os objetos de valor, com as tabelas `StatusDaVenda` e `Categoria`, para garantir checagem dupla de chaves (na aplicação e no banco de dados). Já no diagrama gerado no Entity Framework, na **Figura 18**, note que as tabelas de objetos de valor não foram trazidas do banco, assim como a tabela de erros, que é independente do negócio da aplicação, que é o foco do repositório.

No diagrama do EF não estamos alterando absolutamente

Listagem 7. Implementação concreta do usuário

```
1 namespace NetMag.VendasOnline.Repositorios.ProfileWeb
2 {
3     public class UsuarioWeb : ProfileBase, IUsuario
4     {
5         #region IUsuario Members
6         [SettingsAllowAnonymous(false)]
7         public string Id
8         {
9             get
10            {
11                return this.UserName;
12            }
13            set
14            {
15                if (this.UserName == null)
16                    this.Initialize(value, true);
17                else if (this.UserName != value)
18                    throw new InvalidOperationException(
19                        "Não é possível alterar o nome do
20                        usuário.");
21            }
22        }
23        [SettingsAllowAnonymous(false)]
24        public string Nome
25        {
26            get { return base["Nome"].ToString(); }
27            set { base["Nome"] = value; }
28        }
29        [SettingsAllowAnonymous(false)]
30        public string Sobrenome
31        {
32            get { return base["Sobrenome"].ToString(); }
33            set { base["Sobrenome"] = value; }
34        }
35        [SettingsAllowAnonymous(false)]
36        public decimal Nota
37        {
38            get { return Convert.ToDecimal(base["Nota"]); }
39            set { base["Nota"] = value; }
40        }
41    }
42 }
```

nada. Apesar do EF ser um mapeador objeto relacional, e de ser possível criar um objeto único para Livro, que herde de produto (como foi feito na camada de modelo), por exemplo, não faremos isso. O papel do EF será em nos ajudar a não escrever SQL. Os repositórios trabalharão traduzindo o que for necessário dos objetos de negócio para o EF. Obviamente que isso é bastante trabalho, além de trazer mais pontos possíveis de erro durante a manutenção. Para resolver estes problemas há duas soluções. A primeira são testes unitários, que permitem testar banco de dados, e vou tocar neste ponto no final do artigo. A segunda opção seria utilizar o framework que mapeasse diretamente nossos objetos de negócio para o banco. O mapeador O/R mais conhecido e maduro do mercado hoje para esse tipo de tarefa é o NHibernate, que já foi abordado aqui na .Net Magazine, e será abordado novamente em breve. Até lá, seguiremos com o EF, que é mais conhecido de todos. A **Figura 18** mostra o diagrama do Entity Framework, mapeando as tabelas de forma praticamente idêntica.

O repositório de produto tem uma dependência forte com o repositório de usuários. Isso porque precisa criar usuários para associar aos compradores e vendedores dos produtos, e esta ação de criar usuários a partir de sua identificação é responsabilidade do repositório de usuários. Por esse motivo, ele demanda em seu construtor uma instância deste repositório, assim como uma instância da string de conexão com o banco de dados. A string poderia ficar no arquivo de configuração web.config, mas isso tornaria o repositório

Listagem 8. Repositório de Usuários

```
1 namespace NetMag.VendasOnline.Repositorios.ProfileWeb
2 {
3     public class UsuarioRepositorio : IUsuarioRepositorio
4     {
5         #region IUsuarioRepositorio Members
6         public IUsuario ObterUsuarioLogado()
7         {
8             var usuario = (UsuarioWeb)HttpContext.Current.
9             Profile;
10            return usuario;
11        }
12        public IUsuario ObterUsuarioPorId(string id)
13        {
14            var usuario = (UsuarioWeb)ProfileBase.
15            Create(id);
16            return usuario;
17        }
18        public void SalvarUsuario(IUsuario usuario)
19        {
20            var usuarioWeb = (UsuarioWeb)usuario;
21            usuarioWeb.Save();
22        }
23        public IUsuario ObterUsuarioVazio()
24        {
25            return new UsuarioWeb();
26        }
27    }
28 }
```

Listagem 9. Configuração no web.config para utilizar os perfis customizados

```
1 <membership defaultProvider="CustomizedProvider">
2 <providers>
3 <clear/>
4 <add name="CustomizedProvider" type="System.Web.
5 Security.SqlMembershipProvider"
6 connectionStringName="conn"
7 applicationName="VendasOnline"
8 minRequiredPasswordLength="5"
9 minRequiredNonalphanumericCharacters="0"
10 enablePasswordRetrieval="false"
11 enablePasswordReset="true"
12 requiresQuestionAndAnswer="false"
13 />
14 </providers>
15 </membership>
16 <profile defaultProvider="CustomProfileProvider"
17 enabled="true"
18 inherits="NetMag.VendasOnline.Repositorios.
19 ProfileWeb.UsuarioWeb">
20 <providers>
21 <clear/>
22 <add name="CustomProfileProvider" type="System.Web.
23 Profile.SqlProfileProvider"
24 connectionStringName="conn"
25 applicationName="VendasOnline"
26 />
27 </providers>
28 </profile>
```

mais difícil de testar. Além disso, acoplaria o repositório ao cliente, o que não é o ideal.

Note como em cada operação de sua interface, a tarefa é praticamente a mesma: o repositório, disponível na Listagem 10, vai até o EF, obtém uma instância da linha que precisa alterar (linha 24) ou um novo item para incluir (linhas 37, 41 e 58) e delega as operações de alteração ou construção a métodos de auxílio (AtualizarProduto e ConstruirProduto). Então ele realiza as alterações necessárias no contexto do EF, como incluir (linha 47) ou salvar (em várias chamadas a *db.SaveChanges()*). Mesmo nos métodos que retornam coleções, como o método ObterProdutosPorUsuario, a delegação ainda acontece, dentro de um laço de foreach. É importante notar que, nas queries, sempre está se utilizando o método Include para incluir as relações entre tabelas necessárias já na

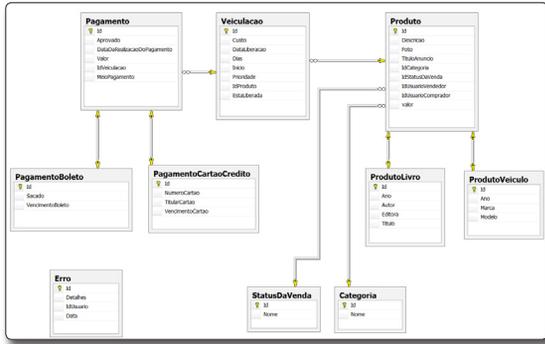


Figura 17. MER da solução, sem incluir tabelas do ASP.Net

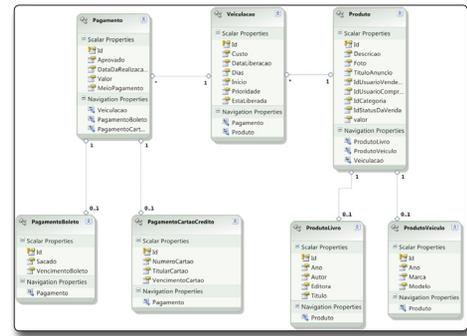


Figura 18. Diagrama do Entity Framework

primeira chamada SQL, não deixando para fazer isso somente quando for necessário, o que resultaria em diversas chamadas ao banco. Desta forma, o EF faz chamadas com inner joins, e já traz tudo que vai precisar de uma vez só.

Os métodos de auxílio, que fazem o mapeamento entre os objetos do domínio e os objetos do EF, são apenas dois, o de criação, ConstruirProduto, e o de alteração, AtualizarProduto. É neste momento que o polimorfismo encerra, e o código passa

Listagem 10. Implementação concreta do repositório de produtos

```

1 namespace NetMag.VendasOnline.Repositorios.SQLServer
2 {
3     public class ProdutoRepositorio : IProdutoRepositorio
4     {
5
6         public ProdutoRepositorio(RepositoriosIUsuarioRepositorio
7             usuarioRepositorio, string connectionString)
8         {
9             this.usuarioRepositorio = usuarioRepositorio;
10            this._connectionString =
11                @"metadata=res://*/Repositorio.csdl|res://*/
12                Repositorio.ssd|res://*/Repositorio.msl;
13                provider=System.Data.SqlClient;provider
14                connection string="" + connectionString + @""";
15        }
16
17        private string _connectionString;
18
19        private IUsuarioRepositorio _usuarioRepositorio;
20
21        #region IProdutoRepositorio Members
22
23        public void SalvarProduto(IProduto produto)
24        {
25            using (var db = new RepositorioInterno.VendasOnline
26                Entities(this._connectionString))
27            {
28                var produtoDB = (from prods in db.Produto.Include
29                    ("ProdutoLivro").Include("ProdutoVeiculo")
30                    where prods.Id == produto.Id
31                    select prods).First();
32                AtualizarProduto(produtoDB, produto);
33                db.SaveChanges();
34            }
35        }
36
37        public void IncluirProduto(IProduto produto)
38        {
39            var produtoDB = new RepositorioInterno.Produto();
40            if (produto.Categoria == Categoria.Livros)
41            {
42                produtoDB.ProdutoLivro = newRepositorioInterno.
43                    ProdutoLivro();
44            }
45            else if (produto.Categoria == Categoria.Veiculos)
46            {
47                produtoDB.ProdutoVeiculo = newRepositorio
48                    Interno.ProdutoVeiculo();
49            }
50            AtualizarProduto(produtoDB, produto);
51        }
52
53        using (var db = new RepositorioInterno.VendasOnline
54            Entities(this._connectionString))
55        {
56            db.AddToProduto(produtoDB);
57            db.SaveChanges();
58        }
59
60        produto.Id = produtoDB.Id;
61    }
62 }
63
64 public IProduto ObterProdutoPorId(int Id)
65 {
66     using (var db = new RepositorioInterno.VendasOnline
67         Entities(this._connectionString))
68     {
69         var produtoDB = (from prods in db.Produto.Include
70             ("ProdutoLivro").Include("ProdutoVeiculo")
71             where prods.Id == Id
72             select prods).First();
73         IProduto produto = ConstruirProduto(produtoDB);
74         return produto;
75     }
76 }
77
78 public IList<IProduto> ObterProdutosPorCategoria
79 (Categoria categoria)
80 {
81     using (var db = new RepositorioInterno.VendasOnline
82         Entities(this._connectionString))
83     {
84         var produtosDB = from prods in db.Produto.Include
85             ("ProdutoLivro").Include("ProdutoVeiculo")
86             where prods.IdCategoria == categoria.Id
87             select prods;
88         var produtos = new System.Collections.Generic.
89             List<IProduto>();
90         foreach (var produtoDB in produtosDB)
91         {
92             produtos.Add(ConstruirProduto(produtoDB));
93         }
94         return produtos;
95     }
96 }
97
98 public IList<IProduto> ObterProdutosPorUsuario
99 (IUsuario usuario)
100 {
101     using (var db = new RepositorioInterno.VendasOnline
102         Entities(this._connectionString))
103     {
104         var produtosDB = from prods in db.Produto.Include
105             ("ProdutoLivro").Include("ProdutoVeiculo")
106             where prods.IdUsuarioVendedor ==
107                 usuario.Id
108             select prods;
109         var produtos = new System.Collections.Generic.
110             List<IProduto>();
111         foreach (var produtoDB in produtosDB)
112         {
113             produtos.Add(ConstruirProduto(produtoDB));
114         }
115         return produtos;
116     }
117 }
118
119 #endregion
120 //continua...

```

Listagem 11. Implementação concreta do repositório de veiculações

```

1 namespace NetMag.VendasOnline.Repositorios.SQLServer
2 {
3     public class VeiculacaoRepositorio : IVeiculacaoRepositorio
4     {
5         public VeiculacaoRepositorio(Repositorios.
6             IProdutoRepositorio produtoRepositorio,
7             IServicoDeCalculoDeVeiculacao servicoDe
8                 CalculoDeVeiculacao, string connectionString)
9         {
10             this._servicoDeCalculoDeVeiculacao = servicoDe
11                 CalculoDeVeiculacao;
12             this._produtoRepositorio = produtoRepositorio;
13             this._connectionString =
14                 @"metadata=res://*/Repositorio.csd|res://*/
15                 Repositorio.ssd|res://*/Repositorio.msl;
16                 provider=System.Data.SqlClient;provider
17                 connection string=''' + connectionString + @''''";
18         }
19
20         private readonly string _connectionString;
21         private readonly Repositorios.IProdutoRepositorio _produto
22             Repositorio;
23         private readonly IServicoDeCalculoDeVeiculacao _servico
24             DeCalculoDeVeiculacao;
25
26         #region IVeiculacaoRepositorio Members
27
28         public void SalvarVeiculacao(IVeiculacao veiculacao)
29         {
30             using (var db = new RepositorioInterno.Vendas
31                 OnlineEntities(this._connectionString))
32             {
33                 var veiculacaoDB = (from veics in db.
34                     Veiculacao.Include("Produto")
35                         .Include("Pagamento.
36                             PagamentoBoleto")
37                         .Include("Pagamento.
38                             PagamentoCartaoCredito")
39                         where veics.Id == veiculacao.Id
40                         select veics).First();
41                 AtualizarVeiculacao(veiculacaoDB, veiculacao, db);
42                 db.SaveChanges();
43             }
44
45             public void IncluirVeiculacao(IVeiculacao veiculacao)
46             {
47                 var veiculacaoDB = new RepositorioInterno.Veiculacao();
48                 using (var db = new RepositorioInterno.Vendas
49                     OnlineEntities(this._connectionString))
50                 {
51                     var produtoDB = (from produto in db.Produto
52                         where produto.Id == veiculacao.
53                             Produto.Id
54                         select produto).First();
55                     veiculacaoDB.Produto = produtoDB;
56                     db.AddToVeiculacao(veiculacaoDB);
57                     AtualizarVeiculacao(veiculacaoDB, veiculacao, db);
58                     db.SaveChanges();
59                     veiculacao.Id = veiculacaoDB.Id;
60                 }
61
62                 public IVeiculacao ObterVeiculacaoAtivaPorProduto
63                     (IProduto produto)
64                 {
65                     IList<IVeiculacao> veiculacoes;
66                     using (var db = new RepositorioInterno.Vendas
67                         OnlineEntities(this._connectionString))
68                     {
69                         var veiculacoesDB = from veics in db.Veiculacao.
70                             Include("Produto")
71                                 .Include("Pagamento.
72                                     PagamentoBoleto")
73                                 .Include("Pagamento.
74                                     PagamentoCartaoCredito")
75                                 where veics.Produto.Id == produto.Id
76                                 select veics;
77                         veiculacoes = ObterVeiculacoes(veiculacoesDB.
78                             ToList());
79                     }
80
81                     var veiculacoesAtivas = from veics in veiculacoes
82                         where veics.EstaAtiva
83                         select veics;
84                     return veiculacoesAtivas.ToList();
85                 }
86
87                 public IList<IVeiculacao> ObterVeiculacoesPorProduto
88                     (IProduto produto)
89                 {
90                     IList<IVeiculacao> veiculacoes;
91                     using (var db = new RepositorioInterno.Vendas
92                         OnlineEntities(this._connectionString))
93                     {
94                         var veiculacoesDB = from veics in db.Veiculacao.
95                             Include("Produto")
96                                 .Include("Pagamento.
97                                     PagamentoBoleto")
98                                 .Include("Pagamento.
99                                     PagamentoCartaoCredito")
100                                 where veics.Produto.Id == produto.Id
101                                 select veics;
102                         veiculacoes = ObterVeiculacoes(veiculacoesDB.
103                             ToList());
104                     }
105
106                     return veiculacoes;
107                 }
108
109                 public IVeiculacao ObterVeiculacaoPorId(int id)
110                 {
111                     using (var db = new RepositorioInterno.Vendas
112                         OnlineEntities(this._connectionString))
113                     {
114                         var veiculacaoDB = (from veics in db.Veiculacao.
115                             Include("Produto")
116                                 .Include("Pagamento.
117                                     PagamentoBoleto")
118                                 .Include("Pagamento.
119                                     PagamentoCartaoCredito")
120                                 where veics.Id == id
121                                 select veics).First();
122                         var veiculacao = ConstruirVeiculacao(veiculacaoDB);
123                         return veiculacao;
124                     }
125                 }
126
127                 public void MarcarVisibilidadeNoProduto(IProduto produto)
128                 {
129                     if (this.ObterVeiculacaoAtivaPorProduto(produto) == null)
130                         produto.Visivel = false;
131                     else
132                         produto.Visivel = true;
133                 }
134             }
135         }
136     }
137 }

```

a tratar diretamente com instâncias reais. Ainda que a classe Livro implemente a interface ILivro, que por sua vez implementa IProduto, o repositório precisa saber com que tipo de objeto está lidando, e precisa de uma classe concreta para construir um novo objeto, e não de uma interface. Então constrói estas

classes mapeando as classes do Entity Framework para as classes POCO de livro e veículo. Há também uma chamada ao repositório de usuários para obter o usuário comprador e o usuário vendedor do produto. Há um mapeamento parecido para as classes de veiculação, que será exibido a seguir.

Listagem 12. Métodos de auxílio do repositório de veiculações

```

125 //continua
126 #region Métodos de auxílio
127
128 private IList<IVeiculacao> ObterVeiculos(IList
129 <RepositorioInterno.Veiculacao> veiculosDB)
130 {
131     var veiculos = new List<IVeiculacao>();
132     foreach (var veiculacaoDB in veiculosDB)
133     {
134         var veiculacao = ConstruirVeiculacao(veiculacaoDB);
135         veiculos.Add(veiculacao);
136     }
137     return veiculos;
138 }
139
140 private IVeiculacao ConstruirVeiculacao
141 (RepositorioInterno.Veiculacao veiculacaoDB)
142 {
143     var produto = this._produtoRepositorio.
144     ObterProdutoPorId(veiculacaoDB.Produto.Id);
145     var veiculacao = new Veiculacao(_servicoDe
146     CalculoDeVeiculacao,
147     veiculacaoDB.Id,
148     veiculacaoDB.DataLiberacao,
149     produto,
150     veiculacaoDB.Custo,
151     veiculacaoDB.Dias,
152     veiculacaoDB.Inicio,
153     (PrioridadeDaVeiculacao)veiculacaoDB.Prioridade);
154     foreach (var pgtoDB in veiculacaoDB.Pagamento)
155     {
156         IPagamento pgto = null;
157         if (pgtoDB.Meiopagamento == (short)MeioDe
158         Pagamento.Boleto)
159         {
160             pgto = new Boleto(veiculacao, pgtoDB,
161             Valor, pgtoDB.PagamentoBoleto.Sacado,
162             pgtoDB.PagamentoBoleto.Vencimento
163             Boleto);
164         }
165         else if (pgtoDB.Meiopagamento == (short)
166         MeioDePagamento.CartaoDeCredito)
167         {
168             pgto = new CartaoCredito(veiculacao,
169             pgtoDB.Valor,
170             pgtoDB.PagamentoCartaoCredito.
171             NumeroCartao,
172             pgtoDB.PagamentoCartaoCredito.
173             VencimentoCartao,
174             pgtoDB.PagamentoCartaoCredito.
175             TitularCartao);
176         }
177         pgto.Aprovado = pgtoDB.Aprovado;
178         pgto.DataDaRealizacaoDoPagamento = pgtoDB.
179         DataDaRealizacaoDoPagamento;
180         pgto.Id = pgtoDB.Id;
181         veiculacao.Pagamentos.Add(pgto);
182     }
183     return veiculacao;
184 }
185
186 private void AtualizarVeiculacao(RepositorioInterno.
187 Veiculacao veiculacaoDB,
188 IPagamento pgto,
189 RepositorioInterno.
190 VendasOnlineEntities db)
191 {
192     veiculacaoDB.Custo = veiculacao.Custo;
193     veiculacaoDB.DataLiberacao = veiculacao.
194     DataLiberacao;
195     veiculacaoDB.Dias = veiculacao.Dias;
196     veiculacaoDB.Estaliberada = veiculacao.
197     Estaliberada;
198     veiculacaoDB.Inicio = veiculacao.Inicio;
199     veiculacaoDB.Estaliberada = veiculacao.
200     Estaliberada;
201     veiculacaoDB.Prioridade = (short)veiculacao.
202     Prioridade;
203     foreach (var pgto in veiculacao.Pagamentos)
204     {
205         var pgtoDB = veiculacaoDB.Pagamento.Where
206         ((p) => p.Id == pgto.Id).SingleOrDefault();
207         if (pgtoDB == null)
208         {
209             pgtoDB = new RepositorioInterno.Pagamento();
210             veiculacaoDB.Pagamento.Add(pgtoDB);
211             switch (pgto.Meiopagamento)
212             {
213                 case MeioDePagamento.CartaoDe
214                 Credito:
215                     var pgtoCC = (CartaoCredito)pgto;
216                     pgtoDB.PagamentoCartaoCredito =
217                     new RepositorioInterno.
218                     PagamentoCartaoCredito()
219                     {
220                         NumeroCartao = pgtoCC.
221                         NumeroCartao,
222                         TitularCartao = pgtoCC.
223                         TitularCartao,
224                         VencimentoCartao = pgtoCC.
225                         VencimentoCartao
226                     };
227                     break;
228                 case MeioDePagamento.Boleto:
229                     var pgtoBoleto = (Boleto)pgto;
230                     pgtoDB.PagamentoBoleto = new
231                     RepositorioInterno.
232                     PagamentoBoleto()
233                     {
234                         Sacado = pgtoBoleto.Sacado,
235                         VencimentoBoleto =
236                         pgtoBoleto.VencimentoBoleto
237                     };
238                     break;
239             }
240             pgtoDB.Aprovado = pgto.Aprovado;
241             pgtoDB.Meiopagamento = (short)pgto.
242             MeioDePagamento;
243             pgtoDB.Valor = pgto.Valor;
244             if (pgto.DataDaRealizacaoDoPagamento.
245             HasValue)
246                 pgtoDB.DataDaRealizacaoDoPagamento =
247                 pgto.DataDaRealizacaoDoPagamento.Value;
248             db.SaveChanges();
249             pgto.Id = pgtoDB.Id;
250         }
251     }
252 }
253 #endregion
254 }

```

O repositório de veiculações segue a mesma linha do repositório de produtos. Ele está disponível na **Listagem 11** para a implementação da interface, e na **Listagem 12** para os métodos de auxílio de mapeamento. Os seguintes pontos são interessantes ressaltar:

1 - A dependência deste repositório é sobre o repositório de produtos, o serviço de cálculo de veiculação (que na verdade é uma dependência do objeto de veiculação criado posteriormente – veja linha 142 da **Listagem 12**), e a string de conexão.

2 - As chamadas do método `Include` do EF para carregar as informações adiantado neste caso traz uma diferença: como as veiculações precisam vir associadas com seus pagamentos, ele precisa também destes dados. No entanto, os pagamentos podem

vir como boletos ou cartão de crédito, que é mais uma relação a ser feita. Isso é entregue por chamadas como a seguinte:

```
db.Veiculacao.Include("Produto").Include("Pagamento.PagamentoBoleto")
```

3 - Este repositório possui uma ação interessante, que é realizada pelo método `MarcarVisibilidadeNoProduto`. Ele verifica se há alguma veiculação disponível e ativa, e se houver marca o produto como visível. Isso poderia ser feito no repositório de produtos, mas para isso ele precisaria de uma referência ao repositório de veiculações, e isso criaria uma referência circular, tornando impossível de criar qualquer dos dois objetos.

Para resolver esse caso precisaríamos de uma factory, mas isso poderia trazer uma complexidade adicional que não caberia neste projeto. Desta forma funciona bem, até aqui.

Com isso, as camadas de negócios estão concluídas. Falta apenas o tratamento de erro e a camada web.

Interface gráfica: ASP.Net MVC

Todo o investimento feito em indirectionamento até agora vai ser pago. Com a arquitetura montada até este ponto somos capazes de montar uma interface gráfica completamente testável. Para isso, precisamos fazer forte uso de interfaces nas camadas de repositórios e modelo. Não podemos, por exemplo, em um determinado controlador do ASP.Net MVC, chamar diretamente o repositório de produtos NetMag.VendasOnline.Repositorios.SQLServer.ProdutoRepositorio. Isso acoplaria o controlador ao banco de dados, o que seria muito complicado de testar. Como fazer então?

Há duas opções. A primeira é receber as dependências destes objetos, como os repositórios e os serviços, em construtor, sempre utilizando interfaces ou classes de base. O AccountController, que já vem no projeto, faz exatamente isso. Veja na **Listagem 13** a técnica de injeção de dependência utilizada. Os comentários que estão no código são os comentários padrão desta classe do ASP.Net MVC. A infra-estrutura do ASP.Net MVC sempre chama o construtor padrão, que acaba chamando o construtor parametrizado passando somente nulos, o que resulta na criação dos componentes padrão. Na hora de estar, é só utilizar diretamente o construtor parametrizado, e injetar objetos de mocks (veja edições 53 a 55, já mencionadas no princípio do artigo).

Listagem 13. Construtores do account controller

```
1 // This constructor is used by the MVC framework to
  instantiate the controller using
2 // the default forms authentication and membership providers.
3 public AccountController()
4     : this(null, null, null)
5 {
6 }
6 // This constructor is not used by the MVC framework but is
  instead provided for ease
7 // of unit testing this type. See the comments at the end of
  this file for more
8 // information.
9 public AccountController(IFormsAuthentication formsAuth,
  MembershipProvider provider, Repositorios.IUsuarioRepositorio
  usuarioRepositorio)
10 {
11     this.UsuarioRepositorio = usuarioRepositorio
12     ?? new NetMag.VendasOnline.Repositorios.ProfileWeb.
13     UsuarioRepositorio();
14     FormsAuth = formsAuth ?? new FormsAuthenticationWrapper();
15     Provider = provider ?? Membership.Provider;
16 }
```

A única diferença entre a versão da **Listagem 13** e a versão original é a presença do repositório de usuários nos construtores, se aproveitando do mesmo padrão. Mais à frente, no método Register, de criação dos usuários, o repositório é então utilizado. Veja na **Listagem 14** um trecho desta função, alterada para tirar proveito desta função e criar o usuário.

A outra opção é utilizar o padrão de arquitetura Registry, e

Listagem 14. Utilizando o repositório para criar o usuário na função Register do AccountController

```
1 if (ViewData.ModelState.IsValid)
2 {
3     // Attempt to register the user
4     MembershipCreateStatus createStatus;
5     MembershipUser newUser = Provider.CreateUser(username,
6     password, email, null, null, true, null, out createStatus);
7
8     if (newUser != null)
9     {
10        var usuario = this.UsuarioRepositorio.ObterUsuario
11        Vazio();
12        usuario.Id = username;
13        usuario.Nome = nome;
14        usuario.Sobrenome = sobrenome;
15        usuario.Nota = 0;
16        this.UsuarioRepositorio.SalvarUsuario(usuario);
17
18        FormsAuth.SetAuthCookie(username, false /* create
19        PersistentCookie */);
20        return RedirectToAction("Index", "Home");
21    }
22    else
23    {
24        ModelState.AddModelError("_FORM", ErrorCodeToString
25        (createStatus));
26    }
27 }
```

foi o padrão que eu adotei por também permitir testabilidade e por facilitar bastante o trabalho, como será visto a seguir. Um Registry é um objeto global, bem conhecido, que todas as aplicações podem utilizar para obter novos objetos. A classe ConfigurationManager, por exemplo, é um registry que permite acessar objetos de configuração, na verdade, strings. Vamos criar um Registry que permite acessar os repositórios e os serviços, sempre focados na testabilidade. Veja a **Listagem 15** para o código desta classe. É uma das classes mais importantes de toda a solução, e também uma das mais inteligentes.

A interface parece um tanto confusa, por essa mistura entre chamadas estáticas e objetos não estáticos. O que acontece é o seguinte: há um singleton do mesmo tipo do objeto, Registry, declarado como protected static, de nome _instancia (linha 22), e uma propriedade também protected static para acessá-lo, que cria uma instância da própria classe na primeira chamada (linha 23). Esta instância está sendo criada com o construtor parametrizado que recebe um booleano para indicar se o objeto deve ser configurado. Nesse momento, todos os objetos concretos, como os repositórios SQL Server, e o repositório de profiles web, são criados. Note que este singleton não está disponível para consumidores da classe, porque ele é protected.

Há também métodos estáticos que retornam os objetos que queremos (repositórios, serviços, etc). Veja, por exemplo, o repositório de usuários que possui uma chamada para ser obtido no método ObterUsuarioRepositorio da linha 54. Este método solicita ao singleton (que também é estático) que retorne o objeto _usuarioRepositorio, que é protected, ou seja, não é acessível externamente à classe, mas é acessível ao método ObterUsuarioRepositorio. E o objeto é então retornado.

Para testar esta classe, basta criar uma classe stub que herde dela e que sobrescreva o campo protected, e sobrescreva também os outros campos dos objetos de serviços. Mais à frente vou mostrar esta classe. É um código um pouco complexo, por isso volte a ele quando eu abordar testes no artigo.

Para utilizar o registry acaba ficando muito simples. Veja

o código do HomeController, por exemplo, na **Listagem 16**. Como toda a complexidade foi abstraída da camada de interface gráfica, tudo o que o controller tem que fazer é... controlar! Ele utiliza os objetos de repositório, acessados através do Registry, para obter as informações que precisa. O código chega a ser tão simples que fala por si só.

Veja, por exemplo, que o método ObterProdutosVisiveis faz uma solicitação ao repositório de produtos pelos produtos de determinada categoria passada por parâmetro (linha 20), tudo em uma única linha. Em seguida, solicita ao repositório de veiculações que marque os produtos visíveis, na linha 22 (este é o método que comentei que poderia estar no repositório de produtos, quando tratei de repositórios). Depois, com Linq to Objects os produtos são filtrados por visibilidade e por status, e somente os 5 últimos são utilizados.

O resultado é utilizado na ação Index do controlador, onde ele cria um dicionário de categorias e lista de produtos (linha 12) e passa como modelo à view. A classe ModeloBase, exibida na **Listagem 17**, é uma classe que permite compor um modelo com informações constantes. Neste caso, as views sempre conhecem o usuário logado e os produtos com proposta, conforme pode ser visto nas linhas 26 e 27, mas conhece também o modelo principal, retornado na propriedade Modelo da classe genérica ModeloBase<T>.

A View Index aparece na **Listagem 18** que mostra aspx, exibindo o seu conteúdo dentro do ASP:Content indexContent, e também o code behind, que exhibe o modelo em uso na linha

41, e duas propriedades para auxiliar na utilização do modelo. Você pode ver estas propriedades em uso nas linhas 7 e 23. Ela também faz uso de uma renderização parcial, conforme pode ser visto na penúltima linha da aspx. O controle, responsável por exibir os itens com solicitação de compra (veja **Figura 4**, parte inferior), está disponível na **Listagem 19**, e recebe uma lista de produtos como modelo.

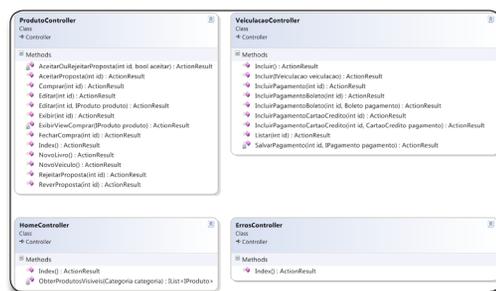


Figura 19. Diagrama de classes dos controllers

Depois de todo o tratamento na camada de negócios e infraestrutura, a aplicação deve seguir o modelo da classe HomeController: chamadas de coordenação às APIs de negócio. A **Figura 19** mostra os controladores e suas ações. Note que eles reproduzem as ações exibidas no começo do artigo, como *IncluirPagamento* e *SalvarPagamento*, no caso do controlador de veiculação, e *FecharCompra* e *RejeitarProposta*, no caso do controlador de produtos. O controlador de erros somente lista os erros com a ação padrão *Index*.

Listagem 15. Objetos de registry para permitir o acesso aos repositórios de forma indireta

```

1 namespace NetMag.VendasOnline.ClienteWeb.Models
2 {
3     public class Registry
4     {
5         public Registry()
6         {
7         }
8         public Registry(bool setup)
9         {
10            if (setup)
11            {
12                _stringDeConexao = System.Configuration.
13                    ConfigurationManager.ConnectionStrings["conn"].
14                        ConnectionString;
15                _servicoDeCalculoDeVeiculacao = new ServicoDe
16                    CalculoDeVeiculacao();
17                _servicoDeCobranca = new ServicoDeCobranca();
18                _usuarioRepositorio = new Repositorios.ProfileWeb.
19                    UsuarioRepositorio();
20                _produtoRepositorio = new Repositorios.SQLServer.
21                    ProdutoRepositorio(_usuarioRepositorio, _
22                        stringDeConexao);
23                _veiculacaoRepositorio = new Repositorios.SQLServer.
24                    VeiculacaoRepositorio(_produtoRepositorio, _
25                        servicoDeCalculoDeVeiculacao, _stringDeConexao);
26                _tratadorDeErro = new TratamentoDeErro.Tratador
27                    DeErro();
28            }
29        }
30        protected static Registry _instancia;
31        protected static Registry Instancia
32        {
33            get
34            {
35                if (_instancia == null)
36                    _instancia = new Registry(true);
37                return _instancia;
38            }
39        }
40        protected string _stringDeConexao;
41        public static string StringDeConexao()
42        {
43            return Instancia._stringDeConexao;
44        }
45        protected TratamentoDeErro.ITratadorDeErro _tratadorDeErro;
46        public static TratamentoDeErro.ITratadorDeErro Obter
47            TratadorDeErro()
48        {
49            return Instancia._tratadorDeErro;
50        }
51        protected IServicoDeCalculoDeVeiculacao _servico
52            DeCalculoDeVeiculacao;
53        public static IServicoDeCalculoDeVeiculacao Obter
54            ServicoDeCalculoDeVeiculacao()
55        {
56            return Instancia._servicoDeCalculoDeVeiculacao;
57        }
58        protected IServicoDeCobranca _servicoDeCobranca;
59        public static IServicoDeCobranca ObterServicoDeCobranca()
60        {
61            return Instancia._servicoDeCobranca;
62        }
63        protected IUserarioRepositorio _usuarioRepositorio;
64        public static IUserarioRepositorio ObterUsuarioRepositorio()
65        {
66            return Instancia._usuarioRepositorio;
67        }
68        protected IVeiculacaoRepositorio _veiculacaoRepositorio;
69        public static IVeiculacaoRepositorio ObterVeiculacao
70            Repositorio()
71        {
72            return Instancia._veiculacaoRepositorio;
73        }
74        protected IProdutoRepositorio _produtoRepositorio;
75        public static IProdutoRepositorio ObterProdutoRepositorio()
76        {
77            return Instancia._produtoRepositorio;
78        }
79    }

```

Listagem 16. Home Controller

```
1 namespace NetMag.VendasOnline.ClienteWeb.Controllers
2 {
3     [InterceptorDeErrosParaLog]
4     [HandleError]
5     public class HomeController : Controller
6     {
7         public ActionResult Index()
8         {
9             ViewData["Title"] = "Compra e venda";
10            var livrosVisiveis = ObterProdutosVisiveis(Categoria.
11                Livros);
12            var veiculosVisiveis =
13                ObterProdutosVisiveis(Categoria.Veiculos);
14            var produtosAExibir = new Dictionary<Categoria,
15                IList<IProduto>>();
16            produtosAExibir.Add(Categoria.Livros, livrosVisiveis);
17            produtosAExibir.Add(Categoria.Veiculos,
18                veiculosVisiveis);
19            var modelo = new ModeloBase<IDictionary<Categoria,
20                IList<IProduto>>>(produtosAExibir);
21            return View(modelo);
22        }
23        private IList<IProduto> ObterProdutosVisiveis(Categoria
24            categoria)
25        {
26            var produtos = Registry.ObterProdutoRepositorio().
27                ObterProdutosPorCategoria(categoria);
28            var veiculacaoRepositorio = Registry.
29                ObterVeiculacaoRepositorio();
30            foreach (var produto in produtos)
31            {
32                veiculacaoRepositorio.MarcarResultosVisibilidadeNoProduto
33                    (produto);
34            }
35            var produtosVisiveis = (from produto in produtos
36                where produto.Visivel == true
37                && produto.StatusDaVenda == Status
38                    DaVenda.SemProposta
39                select produto).Take(5);
40            return produtosVisiveis.ToList();
41        }
42    }
43 }
```

Listagem 17. Classe de base do modelo

```
1 namespace NetMag.VendasOnline.ClienteWeb.Models
2 {
3     public class ModeloBase
4     {
5         public ModeloBase()
6         {
7             var usuarioRepositorio = Registry.
8                 ObterUsuarioRepositorio();
9             this.UsuarioLogado = usuarioRepositorio.
10                 ObterUsuarioLogado();
11             this.ProdutosComProposta = this.
12                 ObterProdutosComProposta();
13         }
14         public ModeloBase(IUsuario usuarioLogado, IList<IProduto>
15             produtosComProposta)
16         {
17             this.UsuarioLogado = usuarioLogado;
18             this.ProdutosComProposta = produtosComProposta;
19         }
20         private IList<IProduto> ObterProdutosComProposta()
21         {
22             if (string.IsNullOrEmpty(Helper.UsuarioLogado.Id))
23                 return new List<IProduto>();
24             var produtos = Registry.ObterProdutoRepositorio().
25                 ObterProdutosPorUsuario(Helper.UsuarioLogado);
26             var produtosVisiveis = from produto in produtos
27                 where produto.StatusDaVenda ==
28                     StatusDaVenda.VendaProposta
29                 select produto;
30             return produtosVisiveis.ToList();
31         }
32         public IUsuario UsuarioLogado { get; private set; }
33         public IList<IProduto> ProdutosComProposta { get; private
34             set; }
35     }
36     public class ModeloBase<T> : ModeloBase
37     {
38         public ModeloBase(T modelo)
39             : base()
40         {
41             this.Modelo = modelo;
42         }
43         public T Modelo { get; private set; }
44     }
45 }
```

As Listagens 20 e 21 mostram os controladores de Produtos e Veiculações, que são os maiores fluxos da aplicação. A Listagem 20 contém também um vinculador, ou binder, que cria uma veiculação sob demanda. Há também um para produtos que segue a mesma linha. Não precisamos listar o código das views, porque são extremamente simples e não atrapalham no entendimento do artigo, e você pode obtê-lo nos downloads no site da Devmedia. Da mesma forma, as listagens contém apenas as ações mais expressivas de modo a demonstrar como fica fácil trabalhar com um modelo de domínio bem planejado, enquanto outras ações estão ocultas. Note na Listagem 21, linha 16, a chamada ao método *MarcarVisibilidadeNoProduto* da Listagem 11, do repositório de veiculação. É um bom exemplo da chamada sendo realizada para exibir somente os produtos visíveis.

Tratamento de erros

Uma das questões mais importantes no tratamento de erros é: qual será a estratégia de tratamento de erros? Quando falamos de erros, estamos dizendo erros não esperados, ou seja, bugs. Para estes erros, a aplicação deve ser capaz de reagir da melhor forma possível. Como estamos tratando de uma aplicação que não requer recomposição em caso de falha, significa que a estratégia pode se restringir a logar o erro. Para isso, já vimos, temos uma camada independente de tratamento de erros.

Outra questão importante é a escolha sobre em que camada o erro não esperado será finalmente endereçado. Se ele não for endereçado aquela tela amarela do ASP.Net será exibida, o que nunca é agradável. Se fosse uma aplicação desktop seu processo seria finalizado. Dessa forma, eu geralmente recomendo: sempre trate pelo menos na última camada, seja ela uma interface gráfica ou uma interface de serviços. Você pode também, opcionalmente, tratar nas camadas superiores, mas não é obrigatório. Você sempre deve tratar quando precisar se desfazer de algum objeto com Dispose, mas o log em si, não é necessário, ou seja, crie um trecho try... finally, em vez de try...catch...finally.

Esta camada possui dois componentes importantes: a entidade de erros e o tratador de erros, disponíveis respectivamente nas Listagens 22 e 23. Note que o tratador de erros utiliza-se de um table adapter para manipular as informações da tabela Erro, exibida na Figura 17. O mapeamento é direto também com o objeto Erro, que é utilizado para retornar as listagens de erro à aplicação. O paradigma desta camada da aplicação saiu de DDD para um padrão de arquitetura chamado de Record Set, do qual a DataTable, a DataRow e o DataSet são os maiores representantes. Isso não é um problema, uma vez que o componente será independente do resto da aplicação.

Há também um tratamento de fallback, em caso de

Listagem 18. View Index.aspx e Index.aspx.cs

Index.aspx:

```

1 <%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.
  Master" AutoEventWireup="true"
2 CodeBehind="Index.aspx.cs" Inherits="NetMag.VendasOnline.
  ClienteWeb.Views.Home.Index" %>
3 <@ Import Namespace="NetMag.VendasOnline.Modelo.Entidades" %>
4 <asp:Content ID="indexContent" ContentPlaceHolderID="Main
  Content" runat="server">
5 <h2>Produtos disponíveis:</h2>
6 <table cellpadding="0" cellspacing="0">
7 <tr valign="top">
8 <td>
9 <div class="Livros">
10 <h3><strong>Livros:</strong></h3>
11 <% foreach (ILivro livro in this.Livros)
12 { %>
13 <div class="produtoHome">
14 <strong><% =Html.ActionLink(livro.
  TituloAnuncio, "Exibir",
15 "Produto", new { id =
  livro.Id },
16 (de <% =livro.Vendedor.Nome + " " +
  livro.Vendedor.Sobrenome %><br />
17 <% =livro.Titulo %><br />
18 <% =livro.Descricao %><br />
19 <% =livro.Valor.ToString("f") %>
20 </div>
21 <% } %>
22 </div>
23 </td>
24 <td>
25 <div class="Veiculos">
26 <h3><strong>Veiculos:</strong></h3>
27 <% foreach (IVeiculo veiculo in this.Veiculos)
28 { %>
29 <div class="produtoHome">
30 <strong><% =Html.
  ActionLink(veiculo.
  TituloAnuncio, "Exibir",
31 "Produto", new { id = veiculo.Id },
  null)></strong>
32 (de <% =veiculo.Vendedor.Nome +
  " " + veiculo.Vendedor.Sobrenome %><br />

```

```

33 <% =veiculo.Marca %> <% =veiculo.
  Modelo %> <% =veiculo.Ano.
  ToString() %><br />
34 <% =veiculo.Descricao %><br />
35 <% =veiculo.Valor.ToString("f") %>
36 </div>
37 <% } %>
38 </div>
39 </td>
40 </tr>
41 </table>
42 <% =Html.RenderPartial("ProdutosComProposta", this.
  ViewData.Model.ProdutosComProposta); %>
43 </asp:Content>

```

Index.aspx.cs

```

44 namespace NetMag.VendasOnline.ClienteWeb.Views.Home
45 {
46 public partial class Index : ViewPage<ModeloBase<IDictionary
  <Categoria, IList<IProduto>>>
47 {
48 protected IList<IProduto> Livros
49 {
50 get
51 {
52 return this.ViewData.Model.Modelo[Categoria.Livros];
53 }
54 }
55 protected IList<IProduto> Veiculos
56 {
57 get
58 {
59 return this.ViewData.Model.Modelo[Categoria.Veiculos];
60 }
61 }
62 protected string ObterClasseCSS(IProduto produto)
63 {
64 return "";
65 }
66 }
67 }
68 }

```

Listagem 19. Controle ProdutosComProposta utilizado em renderização parcial

ProdutosComProposta.ascx

```

1 <%@ Control Language="C#" AutoEventWireup="true"
  CodeBehind="ProdutosComProposta.ascx.cs"
2 Inherits="NetMag.VendasOnline.ClienteWeb.Views.Shared.
  ProdutosComProposta" %>
3 <% if (this.ViewData.Model.Count > 0)
4 { %>
5 <hr />
6 <div>
7 Você tem os seguintes produtos com propostas
  pendentes:<br />
8 <ul>
9 <% foreach (var produto in this.ViewData.Model)
10 { %>
11 <li><% =Html.ActionLink(produto.TituloAnuncio,
  "ReverProposta",
12 "Produto", new { id = produto.Id },
  null)></li>
13 <% } %>
14 </ul>
15 </div>
16 <% } %>

```

Trecho do arquivo ProdutosComProposta.ascx.cs

```

public partial class ProdutosComProposta : ViewUserControl<IList
  <IProduto>>

```

falha do banco de dados. Imagine que o banco de dados caiu, você perde a aplicação e o tratamento de erros? Com o tratamento, dado a partir da linha 22 da Listagem 23, a estrutura do ASP.Net passa a ser responsável por receber os objetos de erro, até que você consiga solucionar o problema no banco de dados.

Para tratar o erro no ASP.Net MVC, em vez de ficar

inserindo trechos de try...catch em todo código, utilize um interceptador. A Listagem 24 mostra o código do interceptador, utilizando o registry para obter acesso ao tratador de erro, e logando-o de acordo. Sua utilização está disponível na Listagem 16, na linha 3, onde o Home Controller o aplica em toda a classe, em forma de atributo. Desta forma, sempre que um erro acontecer, ele interceptará e logará. Use também o atributo HandleError, para garantir que a tela amarela do ASP.Net não aparecerá, e o usuário receberá uma mensagem mais amigável.

Testes unitários

Testes são a garantia de qualidade de sua aplicação. É um tema muito polêmico, porque geralmente os testes não constroem os testes até o final do desenvolvimento da aplicação, o que, muitas vezes, demora muitos meses. Quando este ponto é atingido boa parte da equipe já se foi e os testes acabam não sendo escritos. O erro está no processo. O testes devem ser escritos junto aos métodos. O processo é simples: escreve um método, escreve seu teste, confirme que o teste passa verificando as condições de negócio, e repita até o fim do projeto.

Para testar a aplicação, vamos precisar, antes de mais nada, resolver a questão do Registry, que eu havia tocado anteriormente (veja Listagem 15). O código do stub do Registry está na Listagem 25, e note como ele consegue substituir por completo o objeto do qual herda.

Este objeto está utilizando Rhino Mocks para criar os objetos

Listagem 20. Controlador de veiculações

Controlador:

```

1 namespace NetMag.VendasOnline.ClienteWeb.Controllers
2 {
3     [InterceptorDeErrosParaLog]
4     public class VeiculacaoController : Controller
5     {
6         public ActionResult Listar(int id)
7         {
8             var produto = Models.Registry.Obter
9             ProdutoRepositorio().ObterProdutoPorId(id);
10            var veiculos = Models.Registry.ObterVeiculacao
11            Repositorio().ObterVeiculosPorProduto(produto);
12            var vp = new Models.VeiculosEProduto() { Produto =
13            produto, Veiculos = veiculos };
14            var modelo = new Models.ModeloBase<Models.Veiculos
15            EProduto>(vp);
16            return View(modelo);
17        }
18        [AcceptVerbs(HttpVerbs.Get)]
19        public ActionResult Incluir()
20        {
21            var produtos = Models.Registry.Obter
22            ProdutoRepositorio().ObterProdutosPorUsuario(Models.
23            Helper.UsuarioLogado);
24            var modelo = new Models.ModeloBase<IList<IProduto>>
25            (produtos);
26            return View(modelo);
27        }
28        [AcceptVerbs(HttpVerbs.Post)]
29        public ActionResult Incluir(IVeiculacao veiculacao)
30        {
31            if (veiculacao == null)
32            {
33                return Incluir();
34            }
35            Models.Registry.ObterVeiculacaoRepositorio().Incluir
36            Veiculacao(veiculacao);
37            TempData["veiculacao"] = veiculacao;
38            return RedirectToAction("IncluirPagamento", new { id =
39            veiculacao.Id });
40        }
41        [AcceptVerbs(HttpVerbs.Get)]
42        public ActionResult IncluirPagamento(int id)
43        {
44            IVeiculacao veiculacao ;
45            if (TempData["veiculacao"] != null)
46                veiculacao = (IVeiculacao)TempData["veiculacao"];
47            else
48                veiculacao = Models.Registry.Obter
49                VeiculacaoRepositorio().ObterVeiculacaoPorId(id);
50            return View("IncluirPagamento", new Models.ModeloBase
51            <IVeiculacao>(veiculacao));
52        }
53        [AcceptVerbs(HttpVerbs.Get)]
54        public ActionResult IncluirPagamentoCartaoCredito(int id)
55        {
56            var veiculacao = Models.Registry.Obter
57            VeiculacaoRepositorio().ObterVeiculacaoPorId(id);
58            return View(new Models.ModeloBase<IVeiculacao>(veiculacao));
59        }
60        [AcceptVerbs(HttpVerbs.Get)]
61        public ActionResult IncluirPagamentoBoleto(int id)
62        {
63            var veiculacao = Models.Registry.Obter
64            VeiculacaoRepositorio().ObterVeiculacaoPorId(id);
65            return View(new Models.ModeloBase<IVeiculacao>(veiculacao));
66        }
67        [AcceptVerbs(HttpVerbs.Post)]
68        public ActionResult IncluirPagamentoBoleto(int id, Boleto
69        pagamento)
70        {
71            return SalvarPagamento(id, pagamento);
72        }
73        [AcceptVerbs(HttpVerbs.Post)]
74        public ActionResult IncluirPagamentoCartaoCredito(int id,
75        CartaoCredito pagamento)
76        {
77            return SalvarPagamento(id, pagamento);
78        }
79        [NonAction]
80        private ActionResult SalvarPagamento(int id, IPagamento

```

```

66        pagamento)
67        {
68            Models.Registry.ObterServicoDeCobranca().Realizar
69            Cobranca(pagamento);
70            var veiculacaoRepositorio = Models.Registry.Obter
71            VeiculacaoRepositorio();
72            var veiculacao = veiculacaoRepositorio.Obter
73            VeiculacaoPorId(id);
74            veiculacao.Pagamentos.Add(pagamento);
75            veiculacaoRepositorio.SalvarVeiculacao(veiculacao);
76            return RedirectToAction("Listar", new { id = veiculacao.
77            Produto.Id });
78        }
79    }
80 }
81
82 Binder:
83 1 public class VeiculacaoBinder : IModelBinder
84 2 {
85 3     #region IModelBinder Members
86 4     public ModelBinderResult BindModel(ModelBindingContext binding
87     Context)
88     {
89         var request = bindingContext.HttpContext.Request;
90
91         var IdProduto = Convert.ToInt32(request.Form["Produto.Id"]);
92         IProduto produto = Registry.ObterProdutoRepositorio().
93         ObterProdutoPorId(IdProduto);
94         int dias = 0;
95         try
96         {
97             dias = Convert.ToInt32(request.Form["Dias"]);
98         }
99         catch (Exception)
100        {
101            bindingContext.ModelState.AddModelError("Dias",
102            request.Form["Dias"], "Não é um valor válido para o
103            número de dias.");
104        }
105        DateTime inicio = DateTime.MinValue;
106        try
107        {
108            inicio = Convert.ToDateTime(request.Form["Inicio"]);
109        }
110        catch (Exception)
111        {
112            bindingContext.ModelState.AddModelError("Inicio",
113            request.Form["Inicio"], "Não é uma data válida para o
114            início.");
115        }
116        PrioridadeDaVeiculacao prioridade = PrioridadeDaVeiculacao.
117        Desconhecido;
118        try
119        {
120            prioridade = (PrioridadeDaVeiculacao)Convert.ToInt32
121            (request.Form["Prioridade"]);
122        }
123        catch (Exception)
124        {
125            bindingContext.ModelState.Add
126            ModelError("Prioridade", request.Form
127            ["Prioridade"], "Não é uma data válida para a
128            prioridade.");
129        }
130        if (bindingContext.ModelState.IsValid)
131        {
132            IVeiculacao veiculacao = new Veiculacao(Registry.Obter
133            ServicoDeCalculoDeVeiculacao(),
134            produto,
135            dias,
136            inicio,
137            prioridade);
138            var result = new ModelBinderResult(veiculacao);
139            return result;
140        }
141        else
142            return new ModelBinderResult(null);
143    }
144    #endregion
145 }

```



Listagem 21. Controlador de produtos

```

1 namespace NetMag.VendasOnline.ClienteWeb.Controllers
2 {
3     [InterceptorDeErrosParaLog]
4     [HandleError]
5     public class ProdutoController : Controller
6     {
7         [Authorize]
8         public ActionResult Index()
9         {
10            ViewData["Title"] = "Produto";
11            var usuarioLogado = Helper.UsuarioLogado;
12            var produtos = Models.Registry.Obter
13                ProdutoRepositorio().ObterProdutosPorUsuario(usuario
14                Logado);
15            var veiculacaoRepositorio = Models.Registry.Obter
16                VeiculacaoRepositorio();
17            foreach (var produto in produtos)
18            {
19                veiculacaoRepositorio.MarcavisibilidadeNoProduto
20                (produto);
21            }
22            var modelo = new Models.ModeloBase<IList<IProduto>>
23                (produtos);
24            return View(modelo);
25        }
26        [AcceptVerbs(HttpVerbs.Get)]
27        public ActionResult Editar(int id)
28        {
29            ViewData["Title"] = "Produto > Edição";
30            var produto = Models.Registry.Obter
31                ProdutoRepositorio().ObterProdutoPorId(id);
32            if (produto.Categoria == Categoria.Livros)
33                return View("EditarLivro", new Models.Modelo
34                    Base<ILivro>((ILivro)produto));
35            if (produto.Categoria == Categoria.Veiculos)
36                return View("EditarVeiculo", new Models.Modelo
37                    Base<IVeiculo>((IVeiculo)produto));
38            throw new InvalidOperationException("Produto
39                desconhecido.");
40        }
41        [AcceptVerbs(HttpVerbs.Post)]
42        public ActionResult Editar(int id, IProduto produto)
43        {
44            if (produto == null)
45                if (id > 0)
46                    return Editar(id);
47            else
48            {
49                if (Convert.ToInt32(Request.Form["Modelo.
50                    Categoria.Id"]) == Categoria.Veiculos.Id)
51                    return NovoVeiculo();
52                else if (Convert.ToInt32(Request.Form["Modelo.
53                    Categoria.Id"]) == Categoria.Livros.Id)
54                    return NovoLivro();
55            }
56            var produtoRepositorio = Registry.ObterProduto
57                Repositorio();
58            if (produto.Id <= 0)
59                produtoRepositorio.IncluirProduto(produto);
60            else
61                produtoRepositorio.SalvarProduto(produto);
62            return RedirectToAction("Index");
63        }
64        [AcceptVerbs(HttpVerbs.Get)]
65        public ActionResult NovoLivro()
66        {
67            ViewData["Title"] = "Produto > Novo livro";
68            return View("EditarLivro", new Models.ModeloBase
69                <ILivro>(null));
70        }
71        [AcceptVerbs(HttpVerbs.Get)]
72        public ActionResult NovoVeiculo()
73        {
74            ViewData["Title"] = "Produto > Novo veiculo";
75            return View("EditarVeiculo", new Models.ModeloBase
76                <IVeiculo>(null));
77        }
78        [AcceptVerbs(HttpVerbs.Get)]
79        public ActionResult Exibir(int id)
80        {
81            ViewData["Title"] = "Produto > Exibindo";
82            var produto = Models.Registry.Obter
83                ProdutoRepositorio().ObterProdutoPorId(id);
84            return View(new Models.ModeloBase<IProduto>(produto));
85        }
86        [AcceptVerbs(HttpVerbs.Get)]
87        public ActionResult Comprar(int id)
88        {
89            var produto = Models.Registry.Obter
90                ProdutoRepositorio().ObterProdutoPorId(id);
91            return ExibirViewComprar(produto);
92        }
93        [AcceptVerbs(HttpVerbs.Post)]
94        [ActionName("Comprar")]
95        [Authorize]
96        public ActionResult FecharCompra(int id)
97        {
98            var produto = Models.Registry.Obter
99                ProdutoRepositorio().ObterProdutoPorId(id);
100            if (produto.Vendedor.Id == Helper.UsuarioLogado.Id)
101            {
102                this.ModelState.AddModelError("Vendedor", "0
103                    vendedor não pode ser o mesmo que o comprador.");
104                return ExibirViewComprar(produto);
105            }
106            if (produto.StatusDaVenda != StatusDaVenda.SemProposta)
107            {
108                this.ModelState.AddModelError("StatusDaVenda",
109                    "Produto já com proposta de compra.");
110                return ExibirViewComprar(produto);
111            }
112            ViewData["Title"] = "Produto > Compra realizada";
113            produto.ProrporVenda(Helper.UsuarioLogado);
114            Models.Registry.ObterProdutoRepositorio().Salvar
115                Produto(produto);
116            var compra = new Models.ProdutoSendoComprado() {
117                Produto = produto, CompraRealizada = true };
118            return View("Comprar", new Models.ModeloBase<Models.
119                ProdutoSendoComprado>(compra));
120        }
121        [NonAction]
122        private ActionResult ExibirViewComprar(IProduto produto)
123        {
124            ViewData["Title"] = "Produto > Comprando";
125            var compra = new Models.ProdutoSendoComprado() {
126                Produto = produto, CompraRealizada = false };
127            return View("Comprar", new Models.ModeloBase<Models.
128                ProdutoSendoComprado>(compra));
129        }
130        [Authorize]
131        [AcceptVerbs(HttpVerbs.Get)]
132        public ActionResult ReverProposta(int id)
133        {
134            var produto = Models.Registry.ObterProduto
135                Repositorio().ObterProdutoPorId(id);
136            if (produto.Vendedor.Id != Helper.UsuarioLogado.Id)
137                throw new InvalidOperationException("Somente o
138                    vendedor pode rever este produto.");
139            var revisao = new ProdutoComPropostaSendoRevista()
140                { Produto = produto, PropostaRevisada = false };
141            return View(new ModeloBase<ProdutoComPropostaSendo
142                Revista>(revisao));
143        }
144        [Authorize]
145        [AcceptVerbs(HttpVerbs.Post)]
146        public ActionResult AceitarProposta(int id)
147        {
148            return AceitarOuRejeitarProposta(id, true);
149        }
150        [Authorize]
151        [AcceptVerbs(HttpVerbs.Post)]
152        public ActionResult RejeitarProposta(int id)
153        {
154            return AceitarOuRejeitarProposta(id, false);
155        }
156        [NonAction]
157        private ActionResult AceitarOuRejeitarProposta(int id,
158            bool aceitar)
159        {
160            var produto = Registry.ObterProdutoRepositorio().
161                ObterProdutoPorId(id);
162            if (produto.Vendedor.Id != Helper.UsuarioLogado.Id)
163                throw new InvalidOperationException("Somente o
164                    vendedor pode rever este produto.");
165            produto.AceitarOuRejeitarVenda(Helper.UsuarioLogado,
166                aceitar);
167            Registry.ObterProdutoRepositorio().SalvarProduto
168                (produto);
169            var revisao = new ProdutoComPropostaSendoRevista()
170                { Produto = produto, PropostaRevisada = true, Proposta
171                Aceita = aceitar };
172            return View("ReverProposta", new ModeloBase<Produto
173                ComPropostaSendoRevista>(revisao));
174        }
175    }
176 }

```

fake de testes. Dessa forma, um teste, antes de chamar um controlador, chama primeiro o RegistryStub, conforme o teste disponível na **Listagem 26**.

O teste utiliza mocks com o RegistryStub, e as chamadas do HomeController acabam nunca batendo na base de dados.

Listagem 22. Estrutura de erros

```
1 namespace NetMag.VendasOnline.Tratamento
  DeErro
2 {
3     public struct Erro
4     {
5         public int Id { get; set; }
6         public string Detalhes { get;
7             set; }
8         public string IdUsuario
9             { get; set; }
10        public DateTime Data { get;
11            set; }
12    }
13 }
```

Veja que ele verifica tudo o que está acontecendo na chamada ao método Index do HomeController. Ele verifica o título da página, na variável Title do ViewData, o modelo, e os valores retornados no modelo. Também cria algumas expectativas lançadas nas linhas 18 e 21, e depois as verifica na linha 49. Qualquer alteração no método Index que passe a não mais atender os requisitos deste teste vai levá-lo a falhar.

Testes no modelo são mais simples. Veja na **Listagem 27** onde está sendo testado o método ProporVenda, e verifica-se que, após a chamada do método, as propriedades Comprador e VendaProposta tenham sido alteradas da maneira correta (linhas 12 e 13). Mais uma regra de negócio testada sem tocar no banco de dados.

Baixe o código da aplicação completo e veja os outros testes realizados. Há mais de 30 testes unitários testando todas as camadas, vale a pena ver como isso está sendo feito. Ainda que existam tantos, nem de perto toda a aplicação está sendo testada.

Conclusão

Apresentei uma aplicação completa, do começo ao fim, focada em padrões e boas práticas de arquitetura. É uma aplicação simples, mas que coloca princípios que podem ser seguidos em boa parte das aplicações reais que você vai desenvolver no seu dia a dia. É claro que cada aplicação é diferente, e aí está o valor do arquiteto, por isso o importante é absorver os princípios e não a aplicação. Nas consultorias que realizo, o foco com frequência é este: adaptar o conhecimento destes princípios à realidade de um novo projeto.

Caso você se interesse, há alguns pontos interessantes em que você pode estender e melhorar esta aplicação, como os seguintes:

- O conceito de Unit Of Work pode auxiliar para integrar melhor os repositórios e as operações de salvamento;
- O conceito de Identity Map pode evitar recuperar objetos no banco de dados que já foram recuperados antes, mantendo uma espécie de Cache;
- A implementação de um contêiner de injeção de dependência, como o Unity ou o Windsor, poderiam substituir assumir as funções do Registry.
- Substituir o Entity Framework pelo NHibernate e comparar

Listagem 23. Tratador de erros

```
1 namespace NetMag.VendasOnline.TratamentoDeErro
2 {
3     public class TratadorDeErro : NetMag.VendasOnline.Tratamento
4         DeErro.ITratadorDeErro
5     {
6         public void TratarErro(Exception ex, string idUsuario)
7         {
8             TratarErro(ex.ToString(), idUsuario);
9         }
10        public void TratarErro(string detalhes, string
11            idUsuario)
12        {
13            try
14            {
15                DatasetDeErros ds = new DatasetDeErros();
16                var erro = ds.Erro.NewErrorRow();
17                erro.Data = DateTime.Now;
18                erro.Detalhes = detalhes;
19                erro.IdUsuario = idUsuario;
20                ds.Erro.AddErrorRow(erro);
21                var ta = new DatasetDeErrosTableAdapters.Erro
22                    TableAdapter();
23                ta.Update(erro);
24            }
25            catch (Exception ex)
26            {
27                var erro = new Erro()
28                {
29                    Data = DateTime.Now,
30                    Detalhes = detalhes,
31                    Id = -1,
32                    IdUsuario = idUsuario
33                };
34                ErrosFallback.Add(erro);
35                var erro2 = new Erro()
36                {
37                    Data = DateTime.Now,
38                    Detalhes = ex.ToString(),
39                    Id = -1,
40                    IdUsuario = null
41                };
42                ErrosFallback.Add(erro2);
43            }
44        }
45        public IList<Erro> ObterErros()
46        {
47            var ta = new DatasetDeErrosTableAdapters.Erro
48                TableAdapter();
49            try
50            {
51                var linhasDeErros = ta.GetData();
52                var erros = new List<Erro>();
53                foreach (var linhaDeErro in linhasDeErros)
54                {
55                    var erro = new Erro()
56                    {
57                        Data = linhaDeErro.Data,
58                        Detalhes = linhaDeErro.Detalhes,
59                        Id = linhaDeErro.Id,
60                        IdUsuario = linhaDeErro.IsIdUsuarioNull()
61                            ? string.Empty : linhaDeErro.IdUsuario
62                    };
63                    erros.Add(erro);
64                }
65                return erros;
66            }
67            catch (Exception ex)
68            {
69                TratarErro(ex, null);
70                return ErrosFallback;
71            }
72        }
73        private IList<Erro> ErrosFallback
74        {
75            get
76            {
77                if (System.Web.HttpContext.Current.Application
78                    ["erro"] == null)
79                    System.Web.HttpContext.Current.Application
80                        ["erro"] = new List<Erro>();
81                return (IList<Erro>)System.Web.HttpContext.
82                    Current.Application["erro"];
83            }
84        }
85    }
86 }
```

Listagem 24. Interceptador de erros do ASP.Net MVC

```

1 namespace NetMag.VendasOnline.ClienteWeb.Controllers
2 {
3     public class InterceptadorDeErrosParaLogAttribute : Filter
4         Attribute, IExceptionFilter
5     {
6         public void OnException(ExceptionContext filter
7             Context)
8         {
9             var ex = filterContext.Exception;
10            var usuarioLogado = Helper.UsuarioLogado;
11            Registry.ObterTratadorDeErro().TratarErro(ex,
12                usuarioLogado.Id);
13        }
14    }
15 }

```

Listagem 25. Stub do Registry feito para testar com Rhino Mocks

```

1 namespace NetMag.VendasOnline.ClienteWeb.Tests
2 {
3     class RegistryStub : Registry
4     {
5         public RegistryStub(MockRepository mocks)
6         {
7             Registry._instancia = this;
8             this._produtoRepositorio = mocks.CreateMock<IProduto
9                 Repositorio>();
10            this._servicoDeCalculoDeVeiculacao = mocks.CreateMock
11                <IServicoDeCalculoDeVeiculacao>();
12            this._servicoDeCobranca = mocks.CreateMock<IServico
13                DeCobranca>();
14            this._stringDeConexao = "abc";
15            this._usuarioRepositorio = mocks.CreateMock<IUsuario
16                Repositorio>();
17            this._veiculacaoRepositorio = mocks.CreateMock
18                <IVeiculacaoRepositorio>();
19        }
20
21        public IProdutoRepositorio ProdutoRepositorio
22        { get { return this._produtoRepositorio; } }
23        public IServicoDeCalculoDeVeiculacao Servico
24        DeCalculoDeVeiculacao
25        { get { return this._servicoDeCalculoDeVeiculacao; } }
26
27        public IServicoDeCobranca ServicoDeCobranca
28        { get { return this._servicoDeCobranca; } }
29        public IUsuarioRepositorio UsuarioRepositorio
30        { get { return this._usuarioRepositorio; } }
31        public IVeiculacaoRepositorio VeiculacaoRepositorio
32        { get { return this._veiculacaoRepositorio; } }
33    }
34 }

```

Listagem 26. Teste da ação Index do HomeController

```

1 namespace NetMag.VendasOnline.ClienteWeb.Tests.Controllers
2 {
3     [TestClass]
4     public class HomeControllerTest
5     {
6         [TestMethod]
7         public void Index()
8         {
9             // Arrange
10            var mocks = new MockRepository();
11            var registryStub = new RegistryStub(mocks);
12            var usuarioLogadoStub = mocks.Stub<IUsuario>();
13            usuarioLogadoStub.Id = "giovanni";
14            Expect.Call(registryStub.UsuarioRepositorio.Obter
15                UsuarioLogado())
16                .Return(usuarioLogadoStub).Repeat.Any();
17
18            var livrosStub = new List<IProduto>();
19            Expect.Call(registryStub.ProdutoRepositorio
20                .ObterProdutosPorCategoria(Categoria.Livros))
21                .Return(livrosStub);
22            var veiculosStub = new List<IProduto>();
23            Expect.Call(registryStub.ProdutoRepositorio
24                .ObterProdutosPorCategoria(Categoria.Veiculos))
25                .Return(veiculosStub);
26            registryStub.VeiculacaoRepositorio.MarcasVisibilidade
27                NoProduto(null);
28            LastCall.IgnoreArguments().Repeat.Any();
29            var produtosPorUsuarioStub = new List<IProduto>();

```

Listagem 27. Teste da entidade produto

```

1 [TestMethod]
2 public void Consegue_Propor_Uma_Venda_A_Um_Produto_Sem_Proposta()
3 {
4     var mocks = new MockRepository();
5     var vendedor = mocks.Stub<IUsuario>();
6     vendedor.Id = "giovanni";
7     Livro target = new Livro(1, StatusDaVenda.SemProposta,
8         null, vendedor, "0 mundo é plano. Novo!!!", "0 mundo é
9         plano", 20m);
10
11     IUsuario comprador = mocks.Stub<IUsuario>();
12     comprador.Id = "guinther";
13     target.ProporVenda(comprador);
14     Assert.AreEqual(comprador, target.Comprador);
15     Assert.AreEqual(StatusDaVenda.VendaProposta, target.
16         StatusDaVenda);
17 }

```

os prós e contras.

- Criar testes unitários para as partes não testadas.

A arquitetura de software traz grandes oportunidades a quem a estuda. Encabeço um grupo de estudos sobre arquitetura de software totalmente aberto e gratuito sobre o assunto desde 2008 que também pode interessar. Você pode obter mais informações sobre o grupo de estudos no meu blog: [http://unplugged.giggio.net/?tag=/grupo de estudos](http://unplugged.giggio.net/?tag=/grupo+de+estudos) ou no site do próprio grupo <http://www.dotnetarchitects.net/> . ●

Dê seu feedback sobre esta edição!

A .NET Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/netmagazine/feedback

