UNIVERSIDADE ESTADUAL DO CEARÁ
DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

THIARLEY FONTENELE MARQUES

OpenBizDAL: OPEN SOURCE BUSINESS AND DATA ACCESS FRAMEWORK

FORTALEZA
2010

THIARLEY FONTENELE MARQUES

OpenBizDAL: OPEN SOURCE BUSINESS AND DATA ACCESS FRAMEWORK

Monografia submetida à coordenação do Curso de Ciências da Computação da Universidade Estadual do Ceará no ano 2010, como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação

Orientador(a) Prof(a). Dr(a) Mariela I. Cortés

FORTALEZA
2010

Dedico à Sofia

AGRADECIMENTOS

À minha família que sempre me incentivou nos estudos e ajudou no meu tempo de faculdade.

À Sofia, por sempre me amar e cuidar quando eu estava muito estressado depois de ficar várias horas escrevendo esse trabalho.

Aos meus melhores amigos Teófilo, Akemi, Bosco, Janaina, Dácio, Vanessa e Paulo Alexandre pelo apoio e por serem ótimos amigos.

Aos meus amigos do Atlântico que sempre estiveram lá quando tinha dúvidas sobre arquitetura.

À minha orientadora Mariela que sempre teve paciência comigo, até mesmo quando eu ficava mandando emails tarde da noite pedindo para ler minha monografia.

Aos professores do Curso de Computação da UECE que me ajudaram a me tornar um bom profissional.

Ao arquiteto Felipe Oquendo que me incentivou a fazer um framework que todos poderiam utilizar e usufruir.

Ao Instituto Atlântico por ter me ajudado a crescer profissionalmente.

“*Together, we can rule the galaxy*”

Darth Vader em *The Empire Strikes Back*

RESUMO

Este trabalho tem como objetivo apresentar um *framework open source* que facilita o desenvolvimento de sistemas multicamadas para a plataforma *Microsoft .NET.* Foi projetado para propiciar a criação de camadas de negócio e de acesso a dados utilizando vários padrões de projeto e influenciando o desenvolvedor a seguir um padrão de programação uniforme, eficiente e extensível. Também mostra alguns *frameworks* com objetivos semelhantes destacando suas vantagens e desvantagens. A arquitetura e o funcionamento do *framework* são apresentados ao longo deste trabalho assim como várias características de configuração. Finalmente, é demonstrado através de um estudo de caso com deve-se utilizar a ferramenta.

Palavras – chave: *Framework*, Padrões de projeto, Arquitetura, OpenBizDAL

ABSTRACT

This paper aims to present an open-source framework that helps the development of multilayer systems for Microsoft. Net platform. It is designed to facilitate the creation of business and data access layers using some design patterns and influencing the developer to follow a uniform pattern of programming, efficient and extensible. It also shows some frameworks with similar goals, highlighting their advantages and disadvantages. The architecture and operation of the framework are presented throughout this work as well as various configuration characteristics. Finally, it is demonstrated through a case study must use the tool.

Key – words: Framework, Design Patterns, Architecture, OpenBizDAL

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Produtividade é um dos fatores-chave para o sucesso no desenvolvimento de um projeto. Atualmente, um dos ambientes mais produtivos para criações de soluções em TI é o *Microsoft .NET Framework 3.5*, permitindo criar soluções para uma ampla gama de problemas, utilizando diferentes linguagens de programação e uma maior velocidade de programação utilizando a ferramenta *Visual Studio* (Krieser, 2009).

O fator negativo presente no desenvolvimento com .NET é a falta de frameworks para tratar das mais comuns necessidades de uma solução, como organização das regras de negócios em camada, integração de sistemas de banco de dados distribuídos e acesso a base de dados, de forma produtiva e utilizando as mais modernas técnicas de projeto orientado à objetos e ao mesmo tempo, ser flexível e adaptável.

Percebe-se uma grande necessidade entre a maioria dos desenvolvedores que empregam a tecnologia da Microsoft, de um framework que atenda os itens citados anteriormente e que possa padronizar a maneira de programar uma solução. O *OpenBizDAL veio para* suprir essa necessidade, criando para a comunidade uma biblioteca otimizada, produtiva, extensível e de fácil utilização.

OpenBizDAL é um projeto Open Source que pode vir a beneficiar desde pequenos desenvolvedores até grandes empresas de TI, pois através dos benefícios de um produto de código aberto, o *framework* atrairá mais colaboradores que ajudarão a incorporar mais experiência e contínua manutenção em versões futuras.

O projeto baseia-se no conceito de arquitetura em multicamadas, que é definido por uma separação de domínios diferentes e complexos, com papeis bem definidos, que se comunicam através de interfaces e objetos de entidades, como por exemplo, camadas de negócio e camada de acesso a dados (SAMPAIO, 2008). Faz uso dos mais utilizados padrões de projetos (GAMMA, 1994) como *Singleton, Factory, Abstract Factory, Dal* e utiliza regras de uso que forçam o programador a desenvolver, de maneira organizada, rápida e extensível.

O *framework* é configurável através de *XML, variando* configurações simples, como conexão a banco de dados, até configurações complexas, como mapeamento de classes empregadas para fabricar objetos.

Com o *OpenBizDAL* é possível utilizar acesso a vários sistemas de banco de dados (SGBD) diferentes e transformar operações entre eles multitransacionais, assim permitindo total flexibilidade na manipulação de dados e controle total das transações. Os SGBDs suportados pelo *framework* são: *Microsoft SQL Server, SQLite.*

1.1 Objetivos

O objetivo principal do presente trabalho é desenvolver um framework utilizando a tecnologia *Microsoft .NET Framework 3.5* que irá suprir algumas das necessidades apresentadas no item anterior para projetos que utilizam o conceito de multicamadas. Para isso é preciso fazer um estudo sobre padrões de projetos, aperfeiçoá-los e aplicá-los para utilizar o maior reuso que a programação orientada a objetos oferece. O Framework deverá oferecer produtividade, desempenho, adaptabilidade e extensibilidade, utilizando para seu desenvolvimento ferramentas e linguagens de programação disponíveis para .NET, como o *Visual Studio* e C#.

1.2 Estrutura da Monografia

Além deste capítulo introdutório, o presente trabalho consiste em mais 5 capítulos. O Capítulo 2 mostra os principais padrões de projeto utilizado amplamente pela comunidade, e destaca aqueles utilizados pelo *OpenBizDAL* e também os padrões que deverão ser empregados ao utilizar-se do *framework*. O Capítulo 3 é focado nos atuais *frameworks* utilizados para *Microsoft.NET* relacionados com o presente trabalho. O Capítulo 4 apresenta de forma mais completa, a arquitetura e os padrões de projeto usados na codificação desse produto, apresentando também, as suas vantagens, desvantagens e limitações. O Capítulo 5 mostra em detalhes,

um estudo de caso demonstrando como o *OpenBizDAL* deve ser incorporado a uma solução e como fazer as principais configurações. Finalmente, o Capítulo 6 apresenta as conclusões e uma breve discussão sobre trabalhos futuros.

2 PADRÕES DE PROJETO

O termo “Padrões” foi utilizado pela primeira vez, na década de 70, por Christopher Alexander, arquiteto e urbanista, através de seus principais livros: *A Pattern Language: Towns, Buildings, Construction* e *The Timeless Way of Building* (FRANTZ, 2007). Alexander mostra em seus livros alguns padrões e técnicas utilizados na construção civil, que representam soluções para os principais problemas em uma obra. Ele descreve- o como (ALEXANDER, 1977):

> “Cada padrão descreve um problema que ocorre sempre no nosso ambiente, e então descreve o núcleo da solução para aquele problema, de uma maneira que você poderá usar esta solução um milhão de vezes, sem ao menos fazê-la do mesmo jeito duas vezes”

Assim, um engenheiro não precisaria desperdiçar tempo procurando por uma saída para problemas ordinários, basta buscar nos registros de padrões por algo que se encaixasse melhor.

Tais idéias, mais tarde em 1987, foram utilizadas pelos programadores *Ward Cunningham* e *Kent Beck*, enquanto trabalhavam no projeto de interfaces gráficas com a linguagem *Smalltalk,* eles aproveitaram as idéias de Alexander para desenvolver cinco padrões de projetos, para serem utilizados por desenvolvedores daquela linguagem de programação (APPLETON, 2000).

Posteriormente, vários outros programadores e arquitetos de software também começaram a utilizar padrões de projeto em suas aplicações, mas somente em 1994 o seu uso seria disseminado para o mundo da computação, pelo lançamento do livro: *“Design Patterns: Elements of Reusable Object-Oriented Software”,* escrito por Erich Gamma, Richard Helm, Ralf Johnson e Josh Vllissides, que ficaram conhecidos como Gang of Four (GoF) (APPLETON, 2000). GoF define padrões como: “Descrições de objetos e classes que são customizadas para resolver um problema geral em um contexto particular”.

Padrões de Projeto ou *Design Patterns* atualmente são considerados um dos tópicos mais importante quando se discute a programação orientada a objetos (FOWLER, 2002). O principal objetivo dessa disciplina é criar um catálogo com soluções de problemas mais comuns e repetitivos durante o desenvolvimento de

software. Outra definição para complementar o entendimento sobre este conceito é citada por Martin Fowler em seu livro "*Patterns of Enterprise Application Arquitecture*" (FOWLER, 2002):

> "Padrões não são originais, eles são muito mais que observações do que acontece no campo de trabalho. E como resultado, nós, autores de padrões, não falamos que 'inventamos', mas sim que 'descobrimos' um".

Nos próximos tópicos são discutidos os principais padrões de projeto, tendo como foco o desenvolvimento em camadas e padrões documentados por GoF.

2.1 Desenvolvimento em camadas

No início do desenvolvimento de software, onde os computadores eram desconectados e independentes de outros, um programa era utilizado para funcionar em apenas uma máquina. Neste contexto, os aplicativos geralmente funcionavam exclusivamente com um módulo, ou seja, toda lógica de apresentação, negócios e acesso a dados estavam contidos em apenas uma camada. Tal arquitetura, dependendo do tamanho do sistema, poderia causar vários problemas, principalmente na manutenção e nas alterações de requisitos, pois para qualquer mudança era necessário substituir todo o aplicativo. A Figura 1 mostra como era o projeto de aplicativos em máquinas *stand alone* (independentes) tendo toda a lógica de negócios, acesso aos dados e interface gráfica juntas:

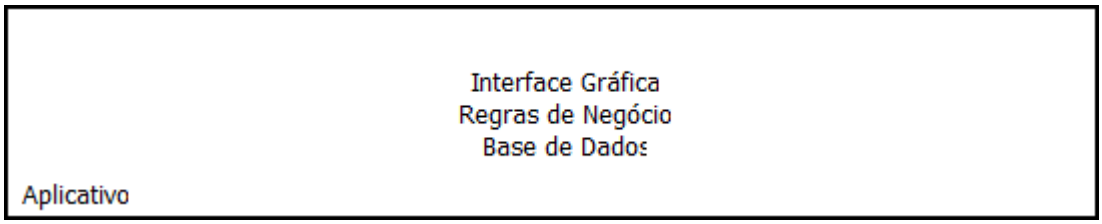

Figura 1 – Arquitetura de sistemas desconectados

Mais tarde, ante a necessidade de vários usuários compartilharem os mesmos dados, surgiu a separação em duas camadas, sendo a primeira relativa à lógica de apresentação e negócio, enquanto que a segunda ao acesso aos dados. A

Figura 2 exemplifica essa separação mostrando à camada de acesso a base de dados separada da interface gráfica e regras de negócio:

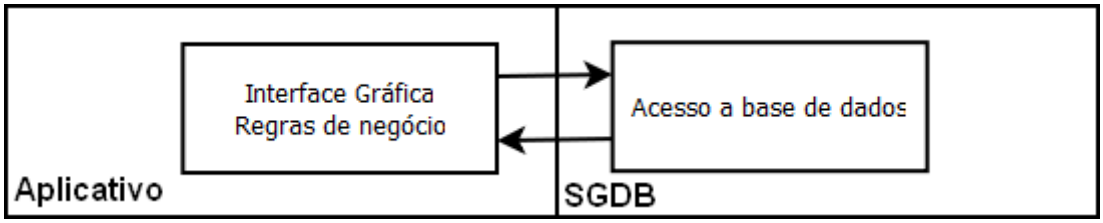

Figura 2 – Arquitetura em duas camadas

A noção de camadas somente mostrou-se mais aparente nos anos 90 com a popularidade dos sistemas cliente-servidor (FOWLER, 2002). A princípio eram apenas aplicativos que mostravam e manipulavam pequenos bancos de dados relacionais. Os problemas começaram a surgir ante a necessidade de manipular cálculos, validações e complexas regras de negócio. Como apenas banco de dados era compartilhado, a manutenção de tais regras aumentava o custo. Uma alternativa foi levar essas validações e normas para a camada de dados utilizando "*Stored Procedures*", entretanto, isso gerava outras dificuldades como conseqüência das limitações decorrentes da linguagem SQL.

As soluções para esses problemas chegaram com a popularidade da programação orientada a objetos, e com isso surgiram os sistemas de três camadas, que são constituídas por uma de apresentação, com a interface gráfica, uma camada de regras de negócio e a camada de dados, exemplificado pela Figura 3.

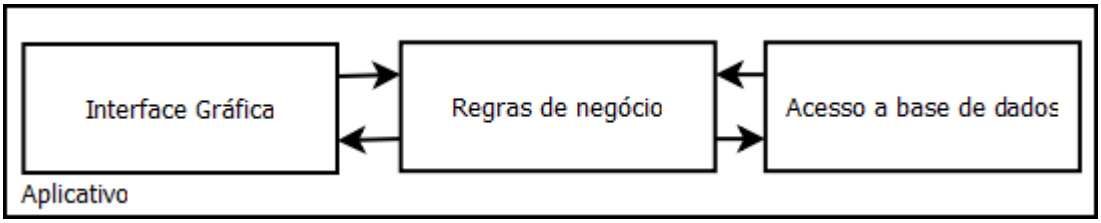

Figura 3 – Arquitetura em três camadas

O grande foco no desenvolvimento de camadas é o gerenciamento de mudanças as quais se tornam mais baratas e rápidas. Uma ótima definição para essa questão é dada por Fowler (2002):

> “Aplicações que manipulam uma grande quantidade de dados (que devem ser persistidos, as vezes por muitos anos), são acessados concorrentemente por uma grande quantidade de pessoas e possuem uma quantidade significante de códigos que se destinam a mostrar esses dados de diferentes maneiras e com diferentes propósitos, para os vários usuários que as utilizam. Geralmente precisam se integrar com outros sistemas, construídos em diferentes tempos com diferentes tecnologias e trabalham com as regras de negócio da instituição para a qual estão sendo desenvolvidos, que geralmente estão em constante mudança”.

Programar em camadas é um padrão de projeto que objetiva separar domínios de diferentes responsabilidades em módulos, de forma que esses módulos tenham baixo acoplamento, portanto eles podem ser trocados facilmente sem necessidade de que outros módulos necessitem de mudanças. A maioria dos padrões de projeto baseia-se nesse conceito.

A forma de separação em camadas não é única e depende de cada sistema a ser desenvolvido. Por exemplo, sistemas simples podem ser separados em três classes simples, cada uma com o seu propósito, mas já para sistemas mais complexos faz-se necessário a divisão em três pacotes.

2.2 *Data Access Object* (DAO)

DAO consiste em um padrão de projeto que tem como objetivo abstrair todo o mecanismo de persistência utilizado pela aplicação. Com isso, uma camada de negócios consegue acessar dados sem saber se foram persistidos em um banco relacional ou em um arquivo de texto. Este padrão oculta os detalhes da origem da base de dados (SUN MICROSYSTEMS, 2002). A Figura 4, retirada da documentação da SUN, mostra como funciona o padrão:

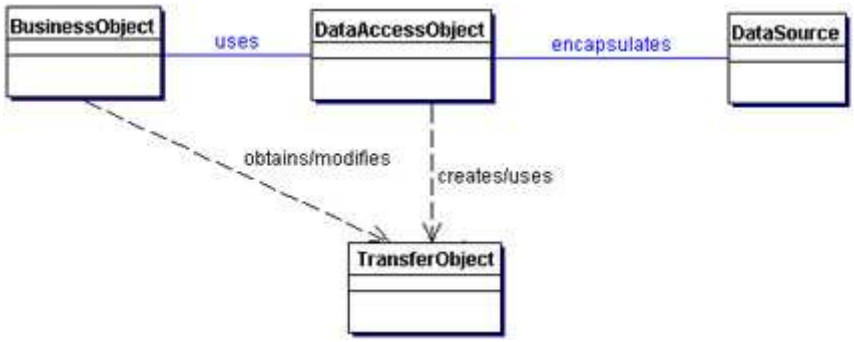

Figura 4 – Padrão de projeto DAO (SUN MICROSYSTEMS, 2002)

A figura mostra um objeto de negócio (*BusinessObject*) que se comunica com o objeto da acesso a dados sem saber o tipo de base que está sendo empregada. A comunicação é feita através de um objeto de transferência, ou *Transfer Objet.*

2.3 Gang of Four

Os padrões Gang of Four (GoF) (GAMMA, 1994) descrevem os modelos para resolução de problemas orientados a objetos. Depois de mapeados, cada padrão é associado, de acordo com a área, em três categorias:

- Padrões de Criação
- Padrões Estruturais
- Padrões Comportamentais

No decorrer do capítulo, cada categoria será estudada, mais com ênfase nos padrões de criação, pois são usados com intensidade pelo OpenBizDAL.

2.3.1 Padrões de Criação

Padrões de criação têm como objetivo abstrair todo o processo de instanciação de um objeto, fazendo com que o sistema se torne independente do

funcionamento interno da fabricação de instâncias. A lista abaixo mostra os padrões relacionados a essa categoria:

- *Factory Method*
- *Abstract Factory*
- *Singleton*
- *Builder*
- *Prototype*

2.3.1.1 Factory Method

O Factory method abstrai a criação de um objeto definindo uma interface e deixando as subclasses decidirem que classe instanciar para satisfazer a atual necessidade do cliente. O livro “Head First – Design Patterns” (FREEMAN, 2007) mostra um exemplo da utilização do *Factory Method* representado pela figura abaixo:

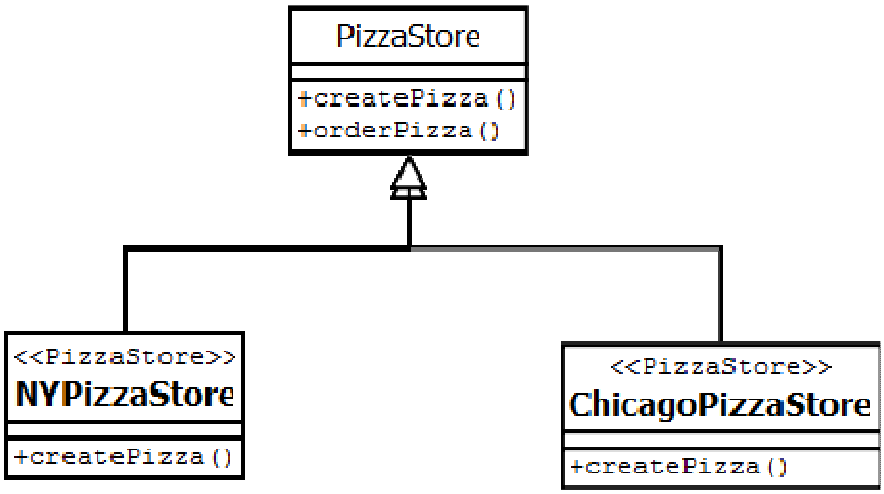

Figura 5 – Padrão *Factory Method* (FREEMAN, 2007)

O modelo descreve uma interface “*PizzaStore*” contendo o *factory method createPizza()*. Percebe-se também que há duas classes herdando da interface, assim, cada classe irá criar uma pizza de acordo com suas necessidades.

2.3.1.2 Abstract factory

Abstract factory fornece uma interface para criar famílias de objetos, relacionados ou dependentes, sem especificar suas classes concretas (GAMMA, 1994). Na figura abaixo é mostrado um exemplo de como um *Abstract Factory* é modelado:

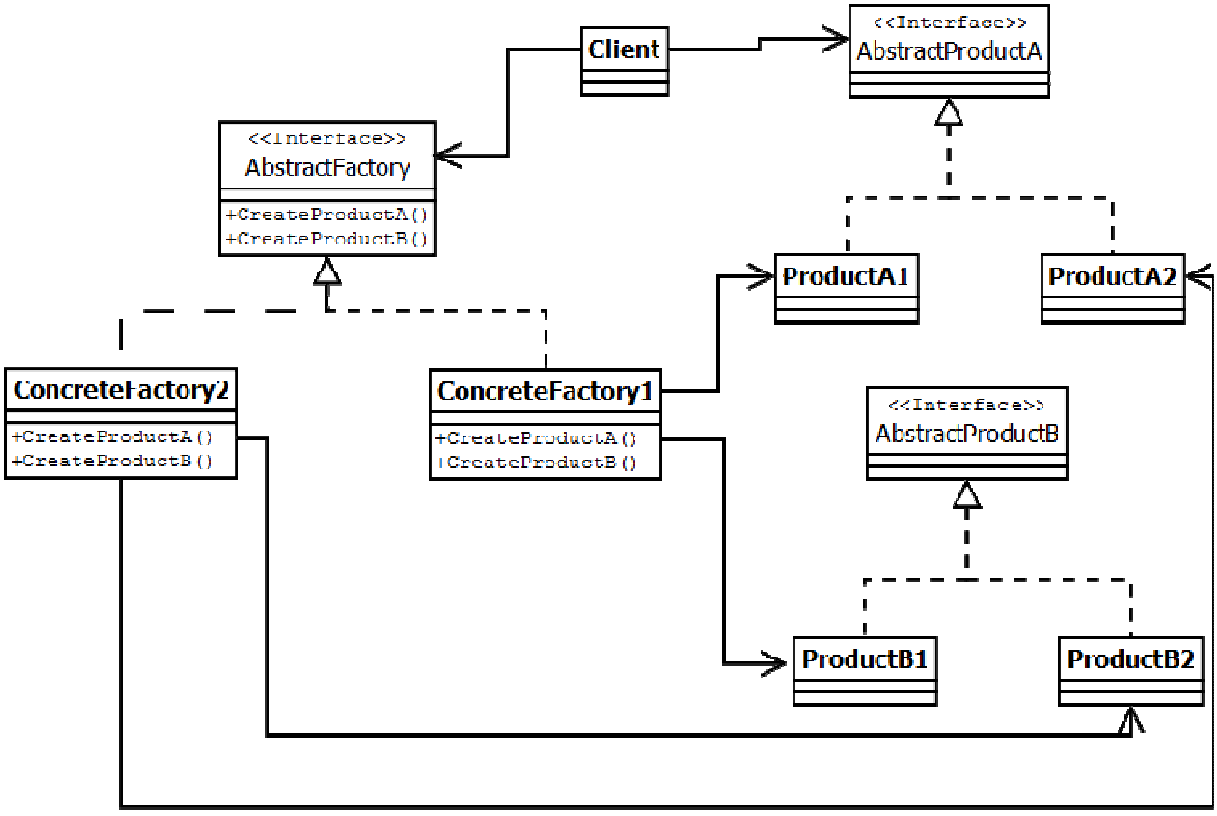

Figura 6 – O Padrão *Abstract Factory* (FREEMAN, 2007)

Percebe-se que existe uma interface (*AbstractFactory*), utilizada para criar subclasses de *fatories* (*ConcreteFactory1 e ConcreteFactory2*), e cada uma irá criar objetos do tipo "*AbstractProductA*", ou "*AbstractProductB*", de maneiras diferentes.

Através desse padrão é possível isolar o cliente da classe que contém o código concreto, deste modo facilitando a integração de novas famílias de objetos e mantendo a transparência entre quem usa a interface e a família existente (utilizando a interface padrão *AbstractFactory*).

2.3.1.3 Singleton

Pelo meio desse padrão é possível garantir que uma classe tenha apenas uma instância e forneça um ponto global de acesso a ela (FREEMAN, 2007). A figura abaixo demonstra melhor o padrão:

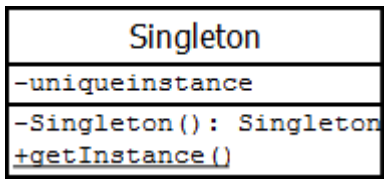

Figura 7 – O Padrão *Singleton* (FREEMAN, 2007)

É importante destacar, que a criação de uma instância se dá apenas pelo seu método estático "*getInstance()*", e o construtor dessa classe deve ser privado. Desta forma, os clientes terão acesso somente pela operação descrita, ou seja, a criação fica controlada e encapsulada.

2.3.2 Padrões Estruturais

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores (GAMMA, 1994). Descrevendo como classes são herdadas de outras, e como são compostas de outras, dependendo do contexto. Os principais padrões nesta categoria são os seguintes:

- *Adapter*
- *Brigde*
- *Composite*

- *Facade*
- *Flyweight*
- *Proxy*

2.3.3 Padrões Comportamentais

Padrões comportamentais descrevem a forma de interação e atribuição de responsabilidade entre classes e objetos e também, a forma de comunicação entre eles. Os padrões de classe utilizam herança, entretanto, os de objetos utilizam a composição. Os principais são:

- *Template Method*
- *Chain of Responsability*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- *Observer*
- *State*
- *Strategy*
- *Visitor*

3 FRAMEWORKS PARA DESENVOLVIMENTO ORIENTADO A OBJETOS

OpenBizDAL é um *framework* que guia o usuário a desenvolver, utilizando camadas de negócio e acesso a dados de maneira padronizada. Atualmente, não existe outro que contemple esses dois domínios juntos, a grande maioria trata apenas do acesso a dados, deixando o usuário criar, ou não, seu próprio domínio de negócio.

Nesse capítulo é definido formalmente o conceito de "*framework*" e são mostrados três produtos para desenvolvimento de aplicativos, orientados a objetos, utilizando a tecnologia *Microsoft® .NET Framework*. Essas bibliotecas tratam exclusivamente em criar um módulo de acesso à base de dados e objetos, que representam os dados contidos no sistema de gerenciamento de banco de dados (SGDB) utilizado pelo cliente. Os *frameworks* analisados no presente capitulo são:

- *Microsoft Linq to SQL*
- *Microsoft Entity Framework*
- *NHibernate Framework*

3.1 Framework

Na área de desenvolvimento de software um *framework* consiste em um aplicativo que pode ser estendido e reusado para resolver problemas complexos (JOHNSON, 2000). Também podemos defini-lo um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação (FAYAD, 1999).

Carlos J. P. Lucena (LUCENA, 2001), Professor do departamento de ciência da computação da PUC-Rio, cita que:

> "*Frameworks* orientados a objetos são a pedra chave na moderna engenharia de software, promovendo reuso de código fonte e arquitetura."

Ao utilizar um framework, geralmente, faz-se uso da inversão de controle, isso significa que a aplicação cliente irá implementar interfaces ou estender classes abstratas e que serão chamadas no momento apropriado (JOHNSON, 2000).

Podemos classificar frameworks em "horizontal" ou "vertical". Vertical define abstrações para resolver problemas de um domínio específico, como imobiliário, financeiro. Já o Horizontal é independente de domínio, podendo ser usados por diferentes áreas de desenvolvimento (SAUVÉ, 2000).

Os frameworks apresentados nos itens seguintes e assim como o OpenBizDAL são frameworks que utilizam o princípio de inversão de controle e são "*frameworks* horizontais".

3.2 Microsoft LINQ to SQL

LINQ to SQL (MSDN, 2007) é apenas uma parte de outro grande *framework* lançado junto com o *Microsoft .NET 3.0* chamado de apenas LINQ, que significa "*Language Integrated Query*" ou linguagem integrada de consultas.

3.2.1 LINQ (Language Integrated Query)

LINQ (MSDN, 2007) consiste em uma linguagem universal de consulta de coleções, por exemplo, *arrays, hashs, XML* e SQL. O autor do livro "*ADO .NET 3.5 with LINQ and the Entity Framework, 2009*", Roger Jennings, define como:

> *"LINQ é uma metodologia de programação revolucionaria que transforma as relações entre programas e dados. Define uma API e um conjunto de extensões para as linguagens de programação Visual Basic e C# que permite fazer consultas de diversos tipos com uma simples sintaxe que é similar ao Structured Query Language (SQL)."*

Na Figura 8 segue a definição de uma consulta genérica:

C# 3.0

```
var query = from element in collection[, from element2 in collection2]
            where condition _
            orderby orderExpression [ascending|descending][, orderExpression2 ...]
            select [alias = ]columnExpression[,[alias2 = ]columnExpression2]
```

VB 9.0

```
Dim Query = From Element In Collection[,Element2 In Collection2] _
            Where Condition _
            Order By OrderExpression [Ascending|Descending][, _
                OrderExpression2 ...] _
            Select [Alias = ]ColumnExpression[,[Alias2 = ]ColumnExpression2]
```

Figura 8 – Exemplo de uma consulta LINQ (JENNINGS, 2009)

Nota-se na figura uma semelhança da consulta LINQ com uma de banco de dados relacional genérico, utilizando elementos como "*from*", "*where*", "*order by*" e "*select*".

A consulta anterior poderia ser traduzida para SQL, mostrando elementos semelhantes ao LINQ, como mostrada na Figura 9:

SQL

```
SELECT [Element.ColumnExpression[, Element2.ColumnExpression2]
FROM Collection AS Element[, Collection2 AS Element2]
WHERE Condition
ORDER BY OrderExpression [ASCENDING|DESCENDING][, OrderExpression2 ...]
```

Figura 9 – Consulta SQL semelhando ao LINQ (JENNINGS, 2009)

O framework define que é possível fazer qualquer consulta para qualquer tipo de informação que programe interfaces do tipo *IEnumerable<T>* ou *IQueryable<T>*. Sendo assim, quase todas as coleções existentes no .NET 3.5 e versões anteriores já implementam *IEnumerable<T>*. Mas para coleções que são derivadas do tipo *IQueryable<T>* possuem características de linguagens dinâmicas e

outras funcionalidades. Assim é possível desenvolver LINQ para qualquer tipo de informação ou serviços. As implementações mais conhecidas são:

- *LINQ to SQL* – Traduz consultas para SQL.
- *LINQ to XML* – Maneira simples de ler arquivos XML.
- *LINQ to Amazon* – Retorna listas de livros de acordo com a consulta feita.
- *LINQ to Flickr* – Faz consultas à coleção de imagens que o serviço oferece.
- *LINQ to NHibernate* – Traduz consultas LINQ para consultas no NHibernate.

3.2.2 LINQ to SQL

Como já descrito anteriormente, LINQ to SQL (MSND, 2007) é uma variação de LINQ (MSDN, 2007), porém, fornece um mecanismo de aquisição de dados de um banco relacional de maneira mais simples e intuitiva. Mas não é apenas uma implementação, ele também inclui uma interface gráfica integrada como um plugin para o *Visual Studio,* IDE (*Integrated Development Environment* ou Ambiente de desenvolvimento integrado) para .NET, utilizado para realizar mapeamento objeto relacional (JENNINGS, 2009). Com essa ferramenta é possível gerar classes de entidade para cada tabela e automaticamente mapeando colunas, associações, chaves e até "*stored procedures*" de bancos altamente normalizados.

LINQ to SQL apenas é compatível para SGDB *Microsoft SQL Server*. Para utilizá-lo basta apenas arrastar e soltar as tabelas, e o *Visual Studio* cria toda a estrutura e tenta automaticamente fazer o mapeamento objeto relacional, criando todas as classes e objetos de acesso a dados. Como mostrado na figura abaixo retirada de (JENNINGS, 2009):

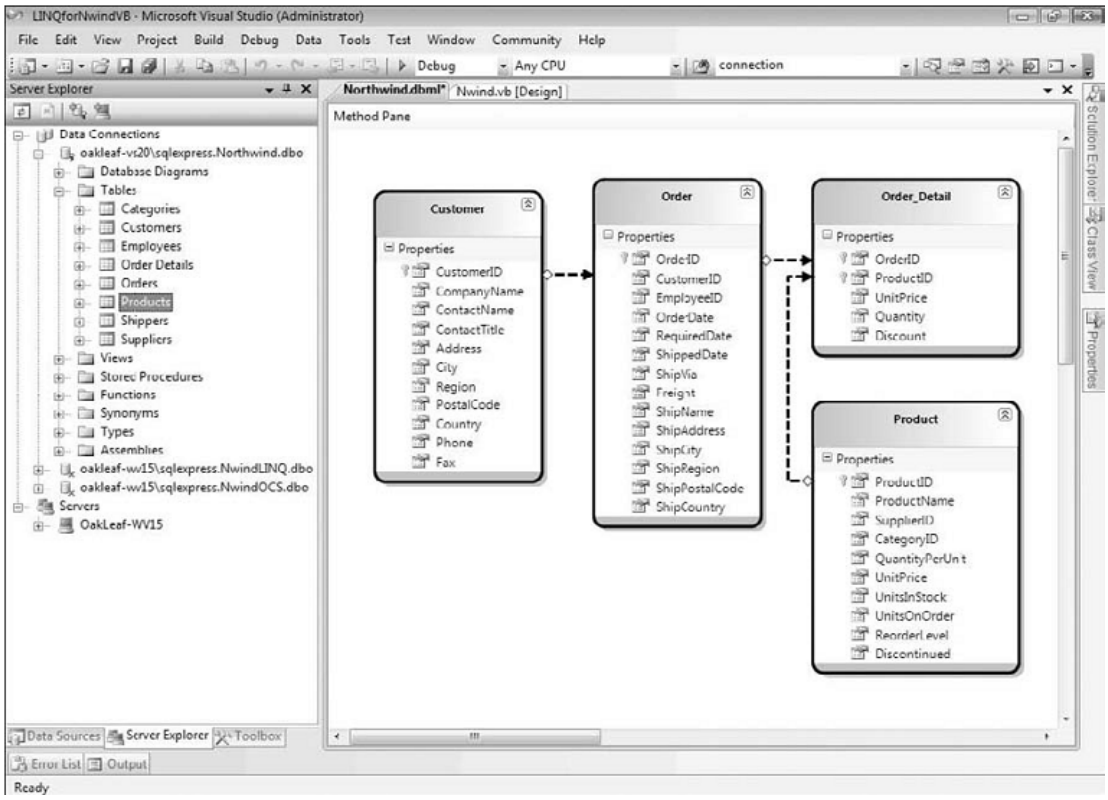

Figura 10 – Ferramenta gráfica para mapeamento objeto relacional (JENNINGS, 2009)

Na parte esquerda da figura é mostrado o banco de dados relacional, e na parte direita o resultado do mapeamento de quatro tabelas. O resultado propriamente dito é colocado em apenas um arquivo, contendo todas as classes declaradas, e também um objeto chamado *DataContext,* cujo papel é gerenciar a comunicação entre a aplicação e o SGDB.

Através do DataContext é possível inserir, alterar, apagar ou consultar qualquer tabela, utilizando a simples linguagem LINQ.

3.2.3 Vantagens

Fazer uso de uma interface gráfica intuitiva para gerar classes de maneira autônoma é um grande ganho de produtividade, economizando varias horas necessárias para fazer manualmente.

Usar LINQ torna essa alternativa ainda mais atraente pela facilidade de gerar consultas, que de outra maneira seria extremamente complexa. Outra vantagem seria abstrair o banco de dados relacional e utilizar orientação a objetos (Agregação, composição, herança e outros).

3.2.4 Desvantagens

Com apenas um ano de vida a *Microsoft* deixou de dar manutenção ao *LINQ to SQL* em detrimento ao *Entity Framework*, como citado por Tim Mallalieu, Gerente de programas da Microsoft (MALLALIEU, 2008):

> "*Nós estamos fazendo grandes investimentos no Entity Framework, assim como no .NET 4.0. Entity Framework será nossa recomendação para acesso a dados utilizando LINQ.*"

Ele opera apenas com o *Microsoft SQL Server*, ou seja, bancos como *Oracle, MySQL, PostGreSQL e SQLite* não são suportados (JENNINGS, 2009).

Foi desenvolvido para o projeto rápido de aplicações, tornando menos flexível e extensível (WORTZEL, 2008).

Não é possível, por sua arquitetura, suportar multitransações entre banco de dados diferentes e também, não é possível alterar consultas sem precisar recompilar o sistema, já que consultas são feitas dentro do código.

3.3 Microsoft Entity Framework

O desenvolvimento do *Entity Framework* (EF) começou em paralelo com o LINQ to SQL, mas, sem comunicação entre as duas equipes, e como dito anteriormente, a Microsoft abandonou o desenvolvimento do LINQ to SQL.

EF é também considerado um framework de mapeamento objeto relacional, mas é o primeiro framework a desenvolver os conceitos de modelo entidade-relacionamento (MER) descrito por Dr. Peter Pin-Shan Chen em 1976 no seu artigo *"The Entity - Relationship Model — Toward a Unified View of Data*" (JENNINGS, 2009). MER consiste em descrever o modelo de dados de um sistema com alto nível de abstração, portanto, ao utilizar o EF, o desenvolvedor pode abstrair o esquema físico do banco de dados relacional e trabalhar utilizando o esquema conceitual (JENNINGS, 2009).

Sendo o *Entity Framework* implementado para trabalhar com o modelo conceitual de dados, ele se torna independente de SGDB, sendo possível trabalhar com *Oracle, MySQL, SQLite, SQL Server* e outros (JENNINGS, 2009).

Para se trabalhar com EF são utilizadas às seguintes formas de consulta:

- *LINQ to Entities* – Dá suporte a utilização de consultas LINQ.
- *Entity SQL* – Como EF é independente de SGDB, ele utiliza sua própria linguagem de consulta, o *Entity SQL* ou apenas eSQL.
- Métodos *Query Builder* – Utiliza *Entity SQL* com o estilo de métodos de consulta LINQ.

EF também fornece para o desenvolvedor um provedor de dados, chamado de *EntityClient*, que gerencia conexões e traduz consultas de entidades em consultas específicas para o SGDB utilizado, analogamente ao *DataContext* no *LINQ to SQL* (MACORATTI, 2009).

Assim como o *LINQ to SQL*, o EF utiliza-se de uma ferramenta com interface gráfica amigável para gerar o modelo entidade relacionamento. Exemplo é mostrado na figura abaixo (JENNINGS, 2009), onde mostra que através do banco de dados foi gerado entidades:

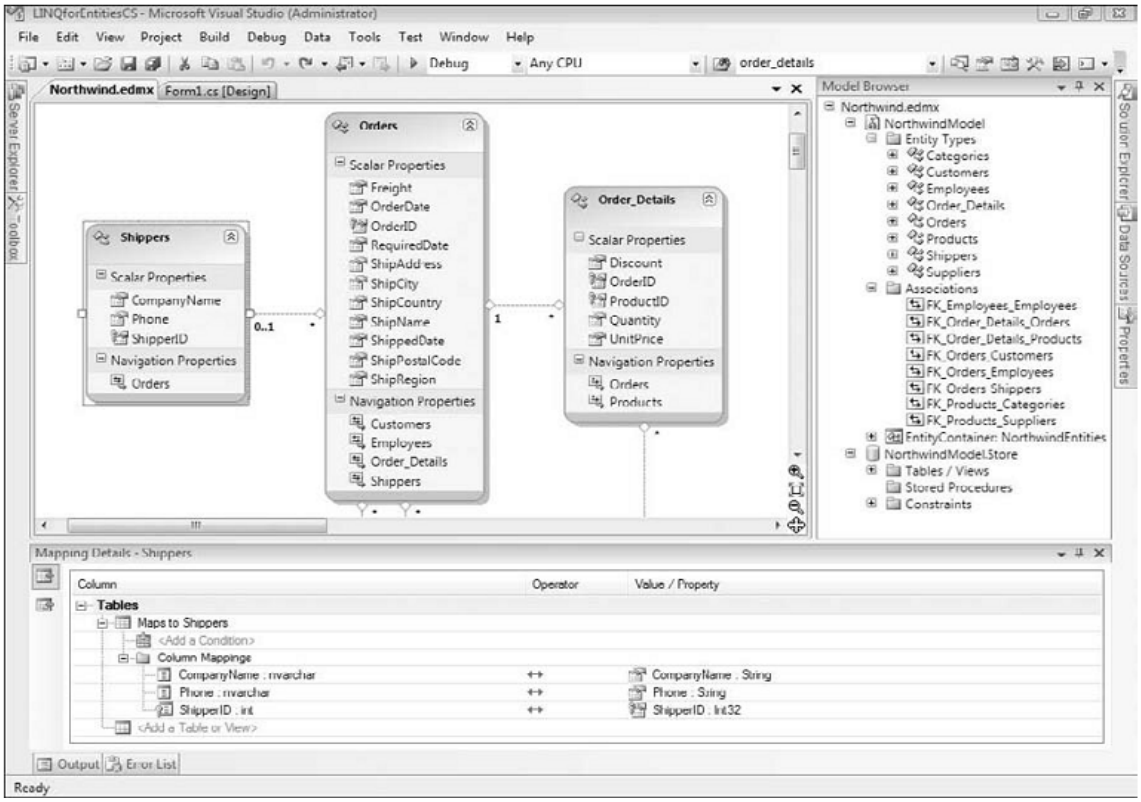

Figura 11 – Ferramenta de mapeamento do *Entity Framework* (JENNINGS, 2009)

3.3.1 Vantagens

EF oferece um alto grau de abstração do modelo físico, deixando o programador ou arquiteto trabalhar de maneira mais fácil e conceitual, além, é claro, de poder empregar a poderosa linguagem LINQ.

Fornece suporte para vários tipos de sistemas de bando de dados, assim, uma migração de sistema torna-se mais rápida e econômica. Adicionalmente, fornece uma interface gráfica amigável para geração automática de entidades e objetos de acesso ao banco.

3.3.2 Desvantagens

Durante o desenvolvimento deste trabalho o EF somente estava disponível na versão beta. A versão final do produto só estará disponível junto com o lançamento do Microsoft .NET Framework 4.0. Por ser ainda uma versão em desenvolvimento, seu suporte e documentação é escasso.

Como o LINQ to SQL, não há suporte para multitransação entre SGDBs diferentes.

3.4 NHibernate Framework

NHibernate é uma versão do famoso *Hibernate* para Java, escrito e compilado para ser usado com *.NET Framework.* É uma ferramenta *open source* mantida atualmente pela comunidade (NHIBERNATE, 2010).

Bauer e King, autores do livro "*Hibernate in Action, 2005*" definem Hibernate como sendo um framework de persistência que tem como objetivo armazenar objetos Java em base de dados relacionais ou fornecer uma visão orientada a objetos de dados relacionais.

Como *Hibernate*, o *NHibernate* é também uma ferramenta de mapeamento objeto relacional, mas os objetos utilizados são .NET (NHIBERNATE, 2010), para isso é utilizado um arquivo XML para mapear os tabelas e colunas para classes e atributos. Em sua versão original, não é prevista a utilização de ferramentas de geração automática de código ou mapeamento automatizado, todo trabalho deve feito manualmente pelo programador ou ferramentas desenvolvidos por pessoas fora da comunidade do NHibernate. A arquitetura básica da utilização do NHibernate é mostrada abaixo:

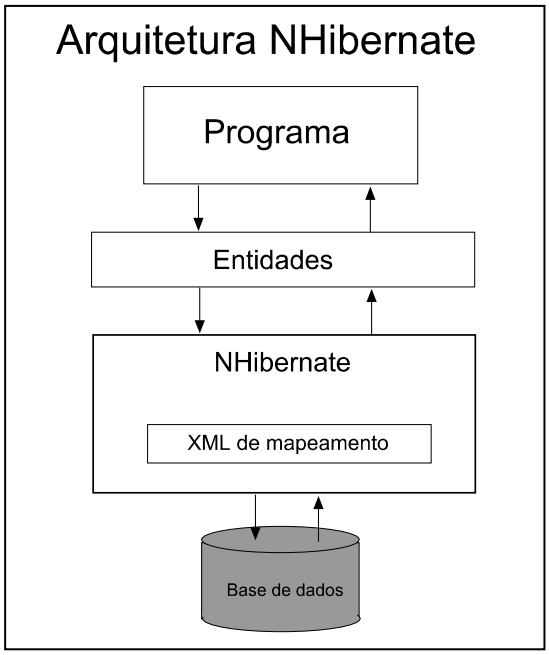

Figura 12 – Arquitetura de um aplicativo empregando o NHibernate (BAUER, 2008)

A Figura 12 mostra como é organizado um sistema que utiliza o NHibernate como camada de acesso aos dados. O programa principal se comunica através de entidades com o NHibernate e este faz interface com o banco de dados.

Seguindo o mesmo conceito de entidades mostrado em outros *frameworks*, essa ferramenta trafega entre camadas, instâncias, classes que representam tabelas ou conjunto de tabelas.

O primeiro passo para o desenvolvedor utilizar a ferramenta é criar um XML de configuração contendo detalhes sobre o banco de dado utilizado. Abaixo segue um exemplo (Figura 13) de um arquivo de configuração que mostra trechos de como configurar o banco de dados:

```
<configuration>
   <configSections>
        <section
           name="nhibernate"
           type="System.Configuration.NameValueSectionHandler, System,
                        Version=1.0.5000.0,Culture=neutral,
                        PublicKeyToken=b77a5c561934e089" />
   </configSections>

   <nhibernate>
        <add
           key="hibernate.connection.provider"
           value="NHibernate.Connection.DriverConnectionProvider"
        />
        <add
           key="hibernate.dialect"
           value="NHibernate.Dialect.MsSql2000Dialect"
        />
        <add
           key="hibernate.connection.driver_class"
           value="NHibernate.Driver.SqlClientDriver"
        />
        <add
           key="hibernate.connection.connection_string"
           value="Server=localhost;initial catalog=nhibernate;User
                        ID=someuser;Password=somepwd;Min Pool Size=2"
        />
        </nhibernate>
</configuration>
```

Figura 13 – Arquivo de configuração do NHibernate (BAUER, 2008)

Sem a geração de código, o desenvolvedor precisa modelar sua solução orientada a objetos e em seguida criar outro XML contendo o mapeamento do objeto para a tabela relacional. Em sistemas complexos essa tarefa pode ser muito trabalhosa e passível de erros, mesmo assim, o ganho em produtividade em manutenções futuras é alto.

3.4.1 Vantagens

A utilização do NHibernate torna o desenvolvimento mais rápido, por não haver necessidade de escrever códigos SQL, uma vez que todo procedimento de criação de consultas é realizado pelo framework baseado no mapeamento objeto relacional escrito pelo programador. Também é possível, para o desenvolvedor que deseja manter suas consultas fora do código- fonte e colocá-las em um arquivo XML, facilitando a manutenção e não precisando recompilar.

Por se tratar de uma ferramenta *open source* com licença LPGL (*Lesser General Public License*) (GNU, 2009) é permitido o uso em aplicações comerciais. A ferramenta continua sendo mantida pela comunidade e atualizado freqüentemente.

O programador pode usar herança e polimorfismo em seus objetos e programar *Lazy loading,* que consiste em que partes do objeto sejam preenchidas pelo servidor apenas quando necessário.

3.4.2 Desvantagens

Para sistemas complexos, o mapeamento manual em XML pode levar tempo e propensão a erros.

Por usar mapeamento em XML e gerar SQL automaticamente, o NHibernate sofre com perda de velocidade.

4 OPENBIZDAL

Este capítulo foca no desenvolvimento, arquitetura e características do OpenBizDAL, assim como suas vantagens e desvantagens.

OpenBizDAL oferece ao desenvolvedor uma maneira de utilizar as boas práticas de programação, meios de obter uma maior produtividade e uma arquitetura que ajuda a criar sistemas divididos em camadas e com baixo acoplamento. O framework faz uso do conceito de inversão de controle, já definido no capítulo anterior, ou seja, a aplicação cliente deverá implementar interfaces ou estender classes abstratas. A utilização do framework possibilita que o arquiteto ou programador seja capaz de separar sua aplicação em, no mínimo, três camadas:

- Interface gráfica
- Regras de negócio
- Camada de acesso a base de dados

Assim como as outras ferramentas apresentadas anteriormente, o OpenBizDAL apresenta meios para se criar uma camada de acesso aos dados sem precisar se preocupar com código de acesso propriamente dito. O que difere dos outros é a possibilidade de também criar uma camada que fica responsável pelas regras de negócio, validações e gerenciamento transacional, ficando o desenvolvedor com a obrigação de apenas escrever código para validar seus objetos antes da inserção, atualização ou remoção.

4.1 Ambiente de desenvolvimento

Todo o desenvolvimento do *framework* foi realizado utilizando apenas ferramentas gratuitas. A implementação do framework foi realizada utilizando a linguagem de programação C# na versão 3.0 com o Microsoft .NET 3.5. A IDE empregada foi *o Visual Studio 2008 Express Edition* com o *plugin Team Explorer*

para acessar o servidor de controle de versão. Como servidor de banco de dados, foi empregado o Microsoft SQL Server 2005 Express, Oracle 10 e SQLite 3.5.

O projeto foi hospedado gratuitamente pelo serviço *Codeplex* oferecido pela *Microsoft* para desenvolvimento de softwares *open source*. Através desse serviço é possível utilizar algumas ferramentas de apoio, tais como:

- *Microsoft Team Foundation Server* – Acessado utilizando a ferramenta *Visual Studio Team Explorer*, oferece um completo sistema de controle de versões.
- *Issue Tracker* – Oferece á comunidade maneiras de registrar *bugs* e melhorias. Qualquer pessoa cadastrada no serviço *CodePlex* poderá criar pendências para serem resolvidas pelo time de desenvolvimento do projeto.
- *Documentation* – Cria um espaço para o desenvolvedor adicionar a documentação ao projeto online para ser acessado por qualquer pessoa.
- *Discussions* – É um fórum criado para o projeto com o intuito de criar discussões sobre o framework.

O portal do OpenBizDAL foi aberto à comunidade no dia 16 de fevereiro de 2009 e todo seu desenvolvimento durou aproximadamente cinco meses. Pode ser acessado pelo endereço: http://openbizdal.codeplex.com.

4.2 Arquitetura

Todo o projeto foi baseado no principio de inversão de controle e a definição de framework horizontal, e para isso, é dividido em três pacotes ou camadas, sendo dois responsáveis por expor as interfaces e classes abstratas a serem usadas pelo cliente, e uma com classes utilitárias e a classe base para as entidades. As camadas envolvidas são:

- *Business* – Manipula regras de negócio e Transações.

- *DAL – Data Access Layer*, camada de acesso a dados.
- *Commons* – Pacote com classes utilitárias e entidades.

Nas próximas seções essas camadas são detalhadas, mostrando os principais padrões de projeto utilizados e como o modelo de desenvolvimento em camadas é usado.

4.2.1 Business

A camada Business é responsável por tratar dos objetos de negócios e gerenciar transações entre várias outras instâncias diferentes. Cada objeto emprega uma entidade específica (corresponde a uma tabela da base de dados) e contem métodos de inserção, remoção, atualização e pesquisa. As classes implementadas nesse pacote são mostradas no diagrama de classes abaixo na Figura 14:

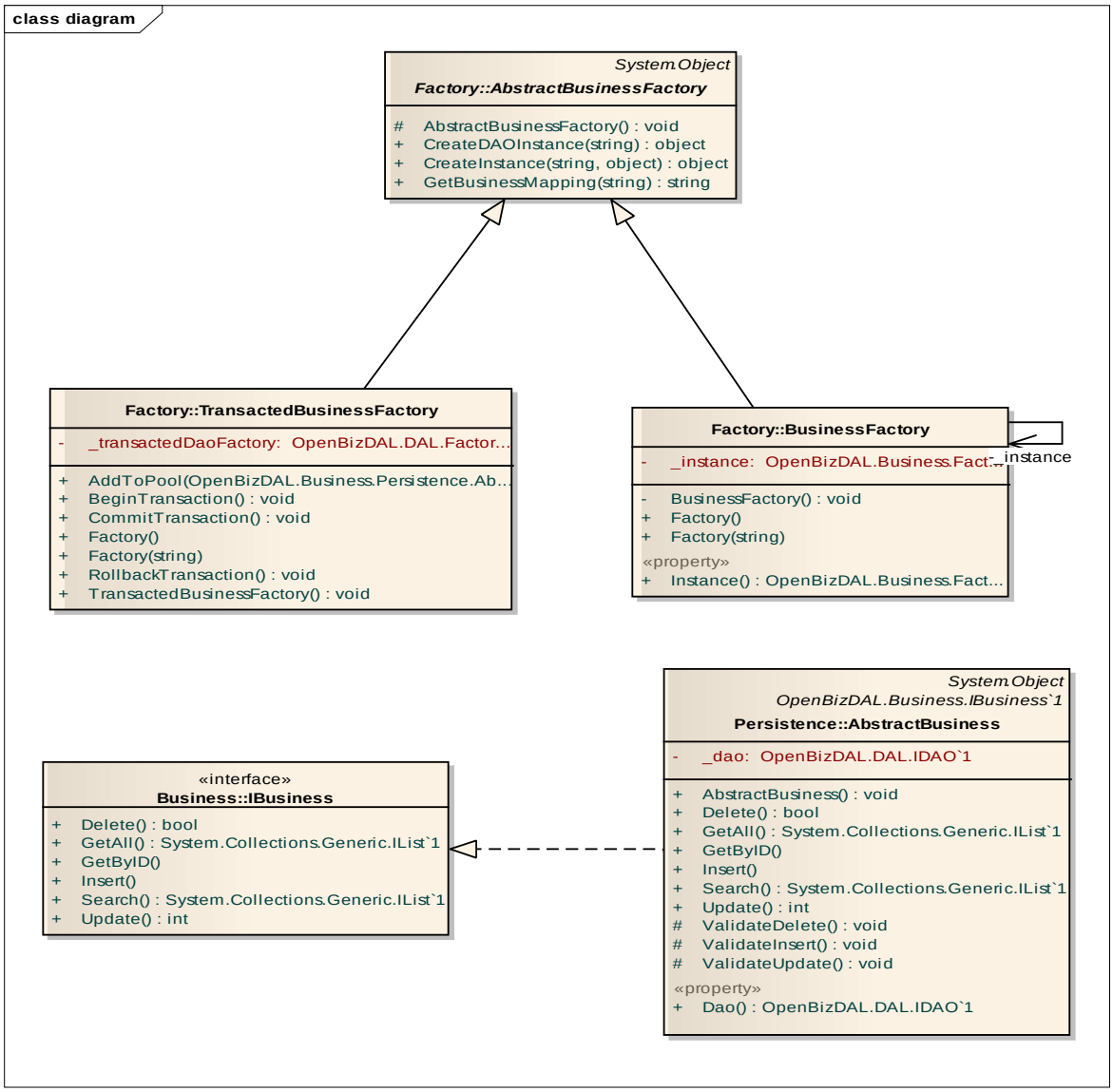

Figura 14 – Diagrama de classes da camada *Business*

A Interface *IBusiness* é o contrato concretizado pela classe abstrata *AbstractBusiness*, que contém todos os métodos concretos necessários para inserção, remoção, atualização e consultas, e métodos abstratos para validações de um determinado tipo de entidade. Todos os objetos de negócio criados pelo usuário devem ser herdados da classe abstrata *AbstractBusiness.*

A classe AbstractBusinessFactory usa o padrão de projeto "*Abstract Factory*" para ser generalizada em mais duas classes, a *TransactedBusinessFactory* e *BusinessFactory* que fazem uso dos padrões *"Singleton" e "Factory Method"*. Ele contém métodos usados principalmente pelos seus filhos para fabricar objetos de acordo com o tipo especificado. O método *CreateBusinessInstance* cria objetos de

negócio a partir do nome de uma classe, para isso é utilizando um mecanismo de mapeamento entre o nome da classe ou interface e sua implementação concreta, empregando o método *GetBusinessMapping*, que recebe o nome da classe e devolve o seu caminho completo para instanciação.

O mapeamento é feito através da utilização de arquivos XML criados de acordo com o padrão estabelecido pelo *OpenBizDAL,* onde o programador deve criar um arquivo contendo o nome da classe simples e também o nome completo (corresponde ao caminho completo). Um exemplo é mostrado no código abaixo, onde a classe *ProductBusiness* tem seu código concreto em *ExampleBusiness.Persistence.ProductBusiess.* Isso permite um ganho de flexibilidade, pois em tempo de execução, o desenvolvedor pode mudar a classe concreta a partir da qual irá implementar um tipo específico:

```
<mappings>
  <class name="ProductBusiness" fullClassName="ExampleBusiness.Persistence.ProductBusiness"/>
  <class name="EmployeeBusiness"
fullClassName="ExampleBusiness.Persistence.EmployeeBusiness"/>
  <class name="CategoryBusiness"
fullClassName="ExampleBusiness.Persistence.CategoryBusiness"/>
</mappings>
```

Caso o programador não queira utilizar mapeamento manual, ele pode utilizar o processo automático. Para isso basta adicionar duas linhas no arquivo de configuração da sua aplicação, como no trecho de XML abaixo:

```
<add key="BusinessMappingPath" value="ExampleBusiness.Persistence"/>
<add key="UseUniqueBusinessMapping" value="true"/>
```

A chave *UseUniqueBusinessMapping* indica se o mapeamento de classes é único, ou seja, para um tipo somente existirá uma classe concreta. O caminho padrão para essas classes é indicado pela entrada *BusinessMappingPath.*

TransactedBusinessFactory corresponde à parte mais importante desse pacote, pois representa a funcionalidade de fabricar objetos de negócio que compartilham uma transação em comum, ou seja, se os objetos forem criados

utilizando essa fábrica, todas as operações realizadas estarão unidas por uma transação, e só serão concretizadas apenas depois do método *commit* ser chamado. Caso algum problema ocorra, todas as alterações poderão ser desfeitas chamando o método *rollback.*

4.2.2 DAL

O pacote DAL corresponde à camada que gerencia toda a comunicação com a base de dados. Atualmente, o OpenBizDAL suporta os bancos relacionais *Microsoft SQL Server* e *SQLite.*

O DAL baseia-se no padrão de projeto DAO (SUN, 2002), já descrito no Capítulo 2, para abstrair das outras camadas detalhes sobre a base de dados. Assim, o pacote Business, por exemplo, não sabe que tipo de banco de dados é utilizado. Abaixo segue o diagrama de classes de parte dessa camada:

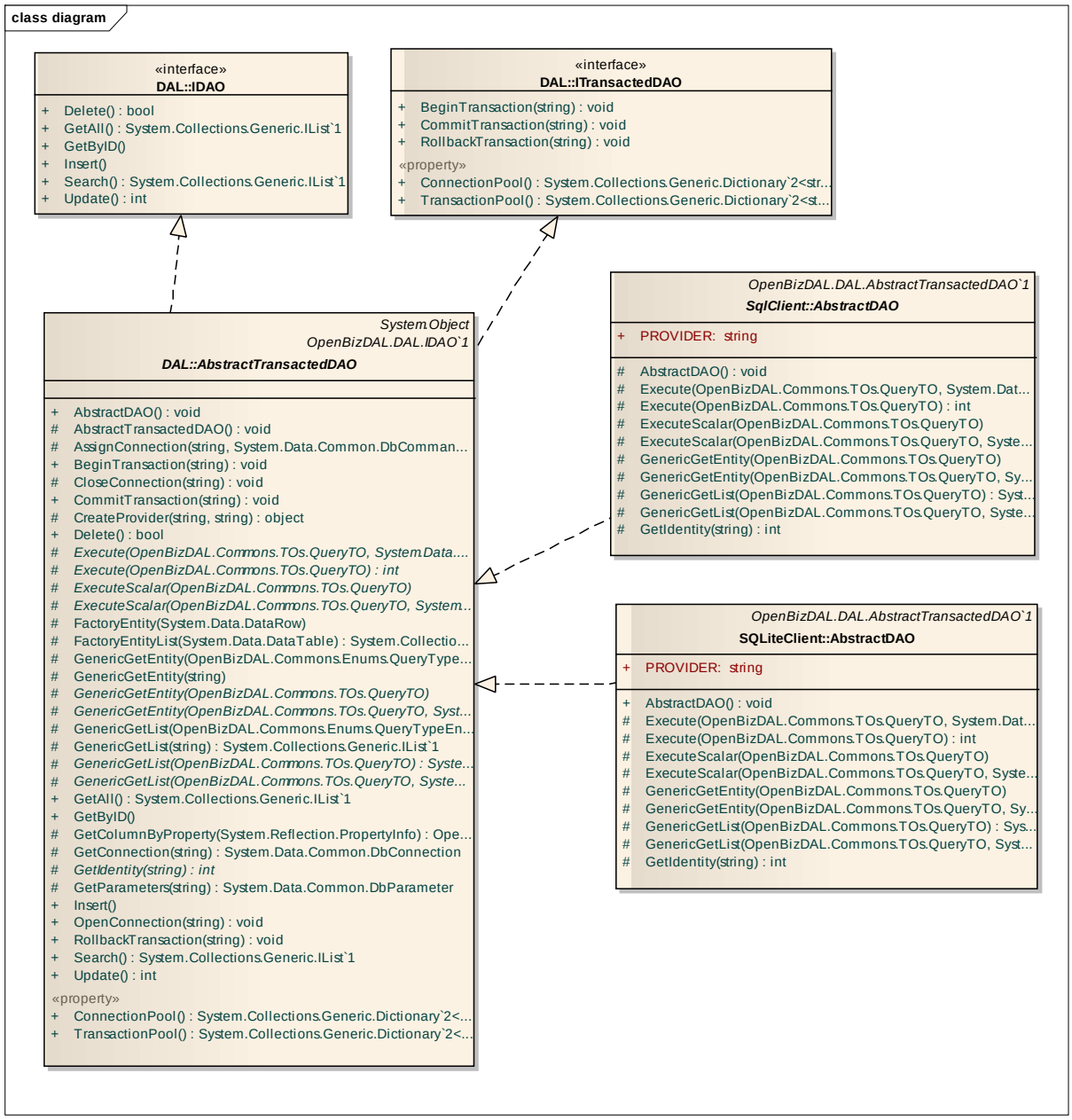

Figura 15 – Diagrama de classes da camada DAL

Toda a base de herança desse diagrama corresponde às interfaces *IDAO* e *ITransactedDAO*, que representam contratos que obrigam às classes concretas a tratar de inserção, remoção, atualização, pesquisa e gerenciamento de transações.

Essas interfaces são implementadas pela classe abstrata *AbstractTransactedDAO.* Ela faz a implementação básica do gerenciamento de acesso a base e transações. Mas, dependendo da particularidade de cada SGDB em acessar banco, o *AbstractTransactedDAO* mantém métodos abstratos para serem implementados por seus filhos.

Para cada provedor de acesso aos dados existe uma classe que herda do *AbstractTransactedDAO,* chamada de *AbstractDAO.* Como, a principio, o framework só suporta SQL Server e SQLite, há uma implementação do *AbstractTransactedDAO* para cada, porém para estender a outros SGDB basta criar outra classe implementando o *AbstractTransactedDAO.* Essas classes irão se comunicar com o provedor de dados utilizando o *framework* de mais baixo nível de acesso, o ADO.NET, que faz parte do *Microsoft .NET 3.5.*

Outra parte dessa camada corresponde às fábricas, ou *factories,* que são classes responsáveis por fabricar tipos de objetos, elas são implementações dos padrões de projeto *Factory Method* e *Abstract Factory*, como mostrado no diagrama abaixo:

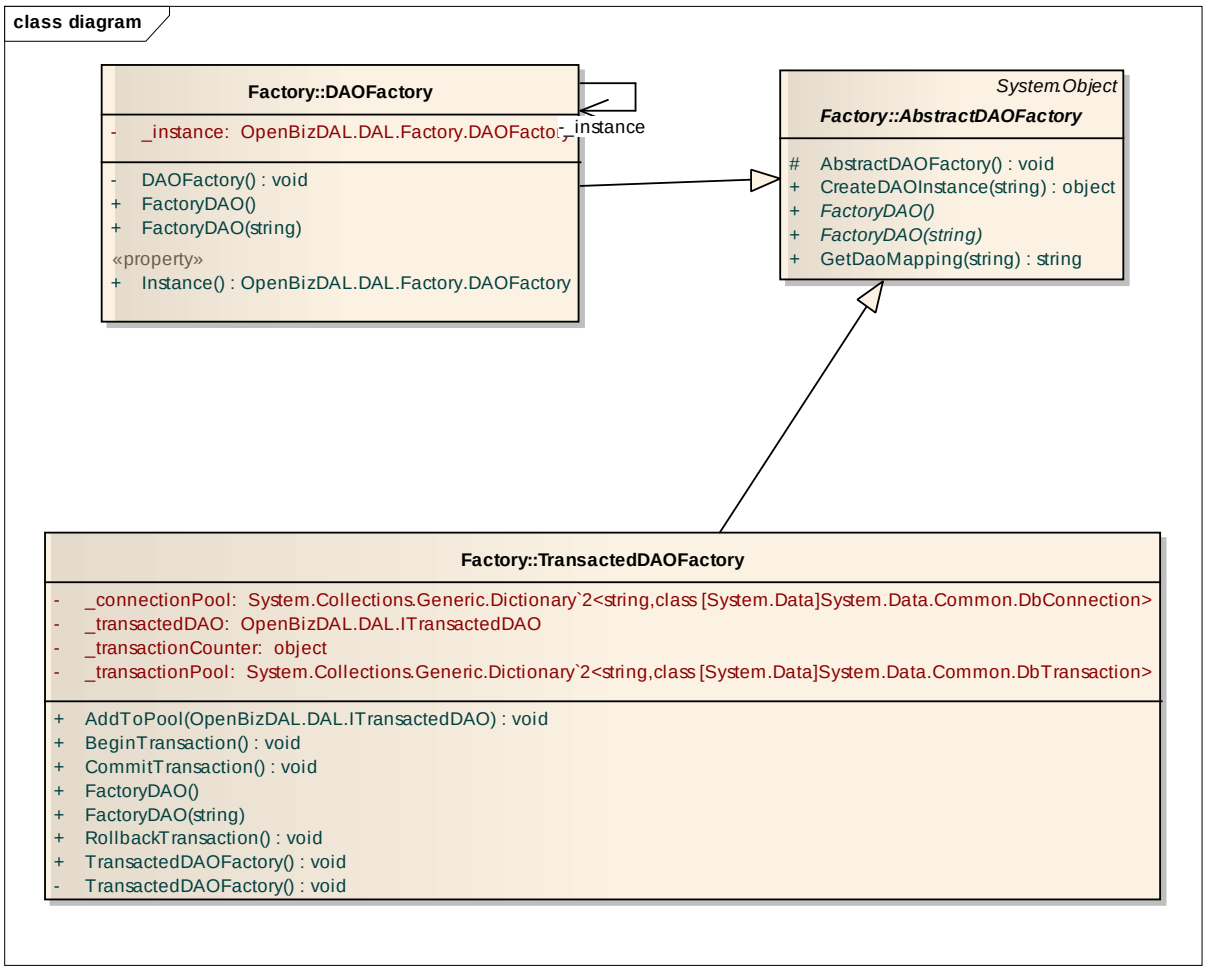

Figura 16 – Diagrama de classes das fábricas da camada *DAL*

Os conceitos por trás dessas fábricas são os mesmos usados pela camada *Business*, mesmos padrões de projeto e características de mapeamento de classes usando XML.

Diferentemente dos outros frameworks, o OpenBizDAL, em sua camada de acesso a dados, não gera consultas SQL. Todos os *scripts* SQL precisam ser escritos e adicionados ao projeto dentro de um arquivo XML para cada tabela do banco. Abaixo segue um exemplo para a classe *ShippersDAO*:

```
<?xml version="1.0" encoding="utf-8" ?>
<queries>
  <query queryAlias="Insert">
    INSERT INTO [Shippers]
    ([CompanyName]
    ,[Phone])
    VALUES
    (@CompanyName
    ,@Phone)
  </query>
  <query queryAlias="Update" connectionAlias=”sqlLite”>
    UPDATE [Shippers]
    SET
    [CompanyName] = @CompanyName
    ,[Phone] = @Phone
    WHERE ID = @ID
  </query>
  <query queryAlias="Delete">
    DELETE FROM Shippers WHERE ID = @ID
  </query>
  <query queryAlias="GetByID">
    SELECT * FROM Shippers WHERE ID = @ID
  </query>
  <query queryAlias="GetAll">
    SELECT * FROM [Shippers]
  </query>
  <query queryAlias="Search">
  </query>
</queries>
```

Outra importante funcionalidade mostrada no codigo XML é o campo “*connectionAlias*”, ele serve para indicar qual base de dados utilizar para aquela consulta, isso por que o OpenBizDAL suporta uma arquitetura com banco de dados distribuídos ou espelhados. Desta forma, o arquiteto pode, para cada tipo de consulta, empregar um banco de dados diferente, abordagem muito utilizada quando existem milhares de acesso simultâneos, e as consultas precisam ser separadas para melhorar o desempenho.

4.2.3 Commons

Commons é uma camada muito importante para o framework, pois, além de prover classes utilitárias para leitura de XML, esta camada inclui a classe abstrata usada como base para todas as entidades que o programador precisa usar. Abaixo segue alguns diagramas de classes desse pacote:

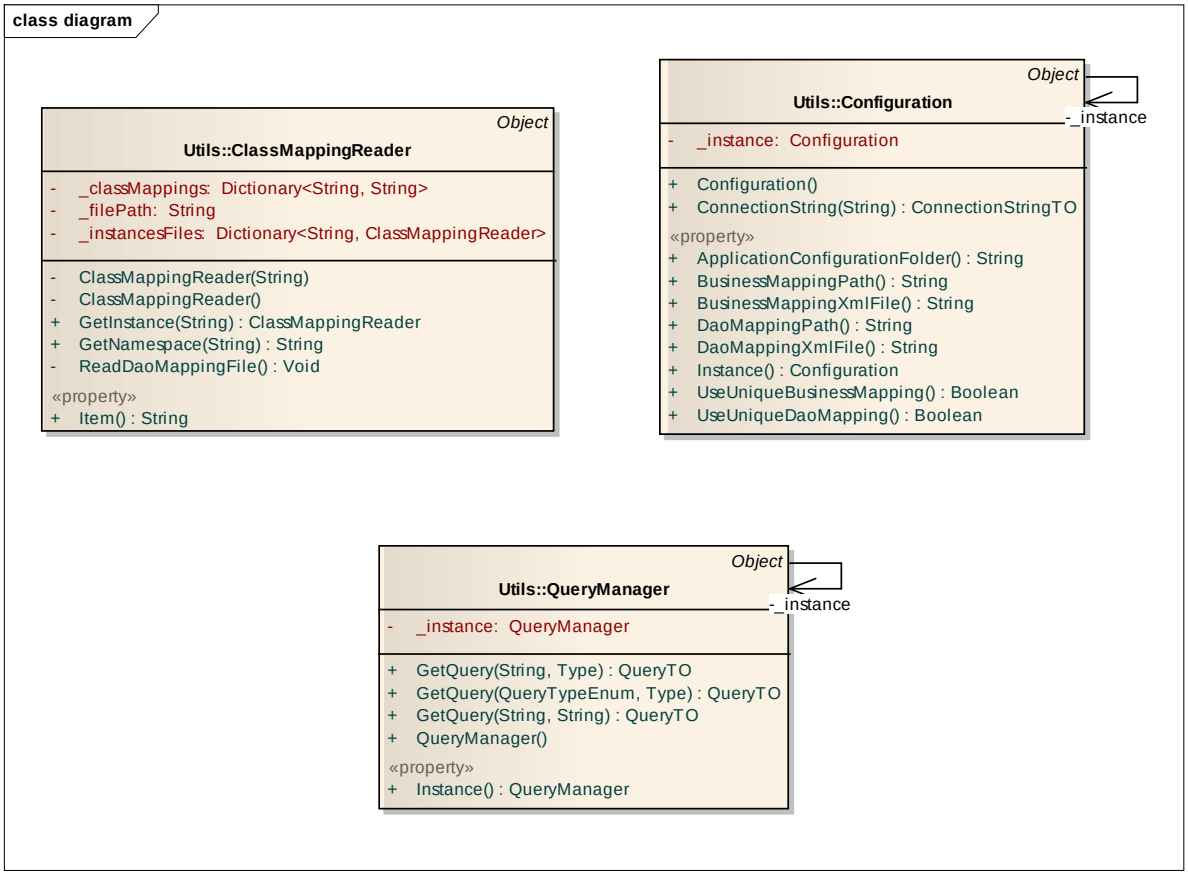

Figura 17 – Diagrama de classes do pacote utilitário de *Commons*

As classes mostradas na Figura 17 são classes utilitárias para leitura de XML:

- *ClassMappingReader* – Faz leitura do mapeamento utilizado pelas fábricas de *Business* e *DAL.* Mantém *cache* dos mapeamentos já utilizados para melhorar desempenho.

- *Configuration* – Cuida de todos os atributos que são escritas no arquivo de configuração da aplicação. Emprega o padrão de projeto *Singleton.*
- *QueryManager* – Faz leitura das consultas escritas pelo usuário.

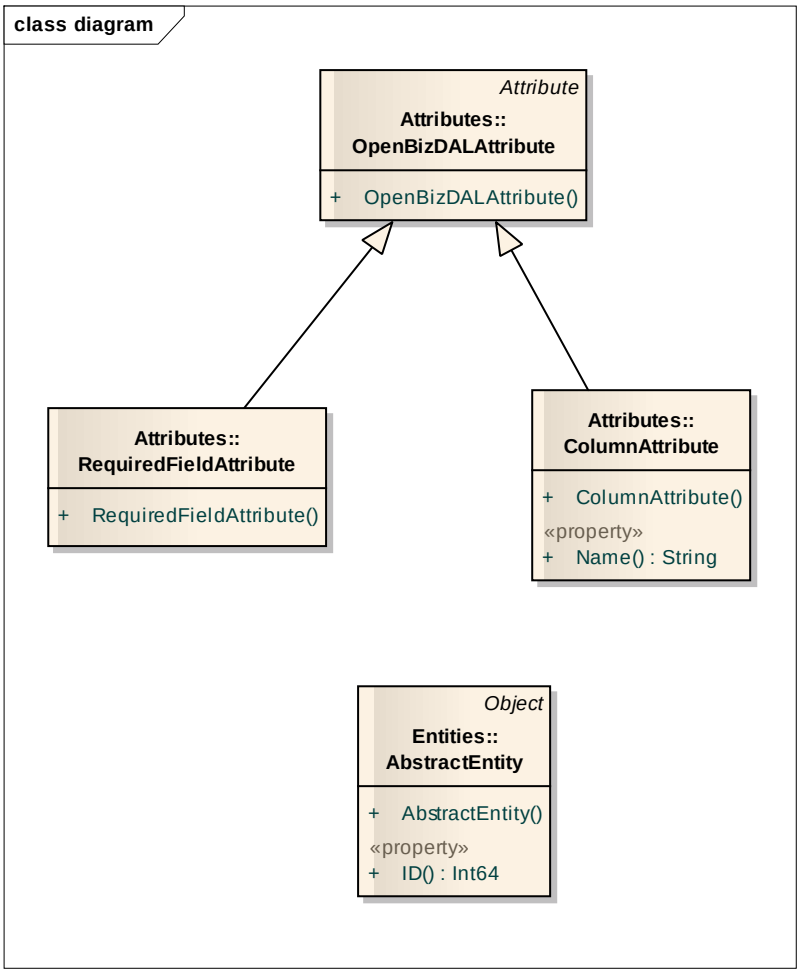

Figura 18 – Diagrama de classes relacionado às entidades em *Commons*

A Figura 18 mostra uma parte importante do *commons: AbstractEntity* que é a base para a criação de entidades pelo programador. O OpenBizDAL determina que entidades sejam como um espelho de uma tabela do banco, ou seja, se existir no banco uma tabela cliente com a coluna nome, deverá ter uma entidade cliente que herde de *AbstractEntity* com “Propriedade” nome. Propriedades em C# são membros nomeados de uma classe que oferece uma maneira de acessar campos através de chamadas *get* e *set* (MSDN, 2007).

Caso o desenvolvedor deseje não padronizar suas entidades com a base de dados, o framework possibilita que ele “marque” a propriedade com o nome da coluna do banco, utilizando o atributo “*ColumnAttribute*”. Atributos fornecem um método eficiente de associar informações declarativas em código C# (tipos, métodos, propriedades e assim por diante) (MSDN, 2007). Abaixo segue um trecho de código de uma entidade empregando as características acima:

```
public class CategoryEntity : AbstractEntity
{
        public string CategoryName { get; set; }
        public string Description { get; set; }

        [ColumnAttribute(Name="Picture")]
        public byte[] Image { get; set; }
}
```

4.3 Vantagens e desvantagens do OpenBizDal

Nessa seção iremos mostrar algumas características do framework demonstrando suas vantagens e desvantagens.

4.3.1 Vantagens

Por sua estrutura, o framework incentiva o desenvolvedor a criar uma arquitetura utilizando o padrão multicamadas e alguns dos padrões de projeto mais conhecidos, propiciando meios de criar uma camada de negócios que suporta multitransações entre objetos e bancos diferentes.

Prover uma maneira de fazer mapeamentos de classes para que as fábricas possam instanciar objetos, tanto de negócios quanto de acesso a dados, propiciando que mudanças possam ser feitas ainda em tempo de execução.

A camada DAL possibilita que os dados trafeguem de maneira eficiente, pois trata as conexões com o SGDB em baixo nível. Alem disso, suporta vários tipos de SGDB e é facilmente extensível para outros.

Manutenções na aplicação se tornam rápidas e para a maioria das correções o sistema não precisa ser reiniciado, basta apenas fazer troca de pacotes ou mudanças nos XMLs.

O *framework* é *open source* e utiliza uma licença LGPL que possibilita o uso em aplicações comerciais.

4.3.2 Desvantagens

Dente os aspectos desfavoráveis da ferramenta podemos citar:

- Não possui uma ferramenta que possibilite a geração automática de código a partir do banco de dados.
- A camada de acesso a dados não suporta mapeamento objeto relacional nativamente, para isso o próprio desenvolvedor precisa implementar sua solução.
- Consultas ao banco de dados não são feitas automaticamente. Todas elas devem ser escritas e colocadas em um arquivo XML. Quando o sistema é muito grande, implementar todas as classes e consultas pode ser trabalhoso.

5 Estudo de Caso

Durante o desenvolvimento deste trabalho foi implementada uma pequena aplicação para validar a modelagem e implementação do OpenBizDAL e ilustrar como o desenvolvedor pode utilizar as várias funcionalidades do framework.

O exemplo consiste em uma aplicação Windows desenvolvida utilizando o Visual Studio 2008 Express e banco de dados SQLite baseado em uma base chamada "*Northwind*" criada pela *Microsoft* para ser usado em testes para o SQL Server. Na Figura 19 é mostrado o diagrama do banco de dados contendo apenas cinco tabelas e associações simples:

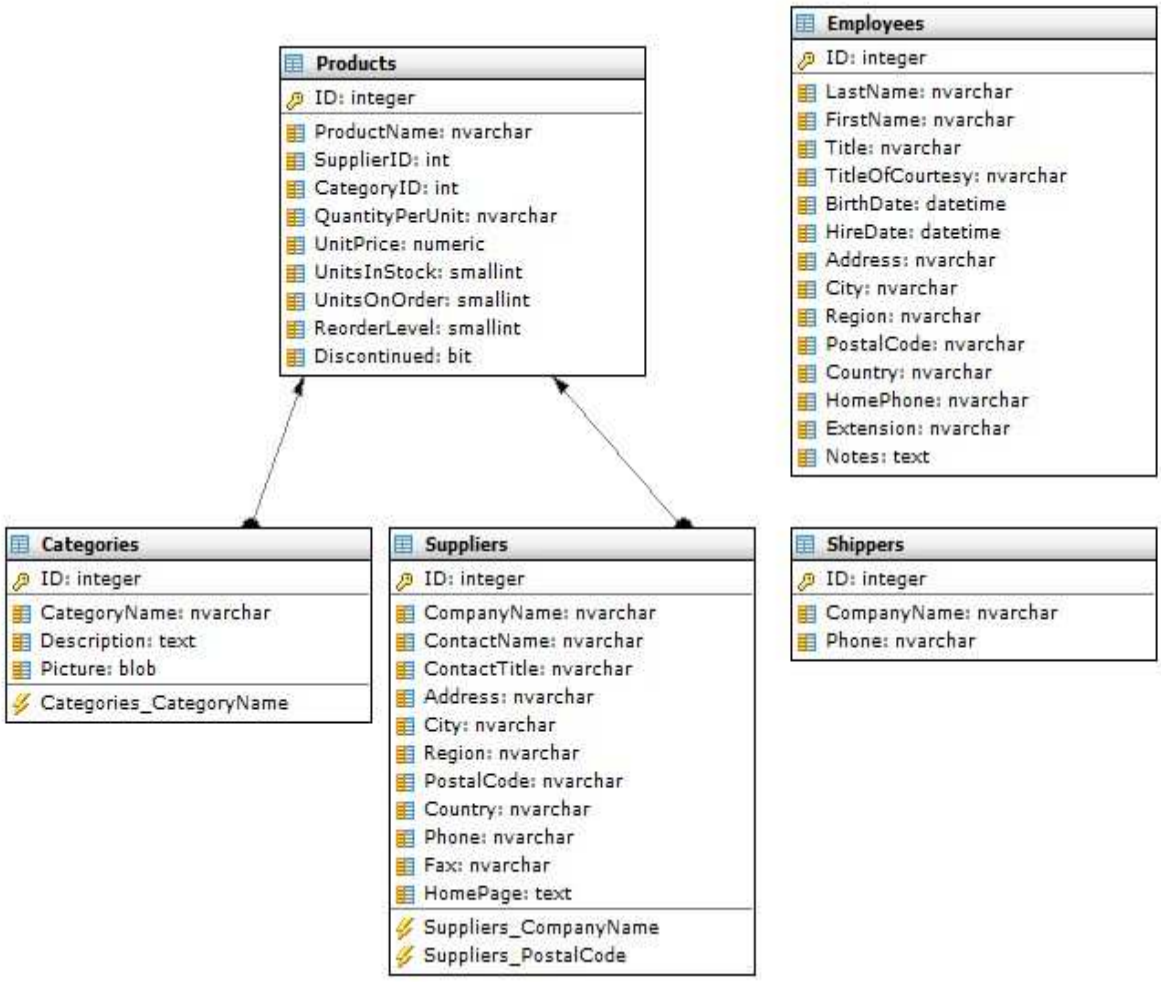

Figura 19 – Diagrama do banco de dados utilizado

A arquitetura foi dividida em três camadas: de negócios, acesso a dados e interface gráfica. O diagrama de pacotes abaixo mostra o modelo básico utilizado:

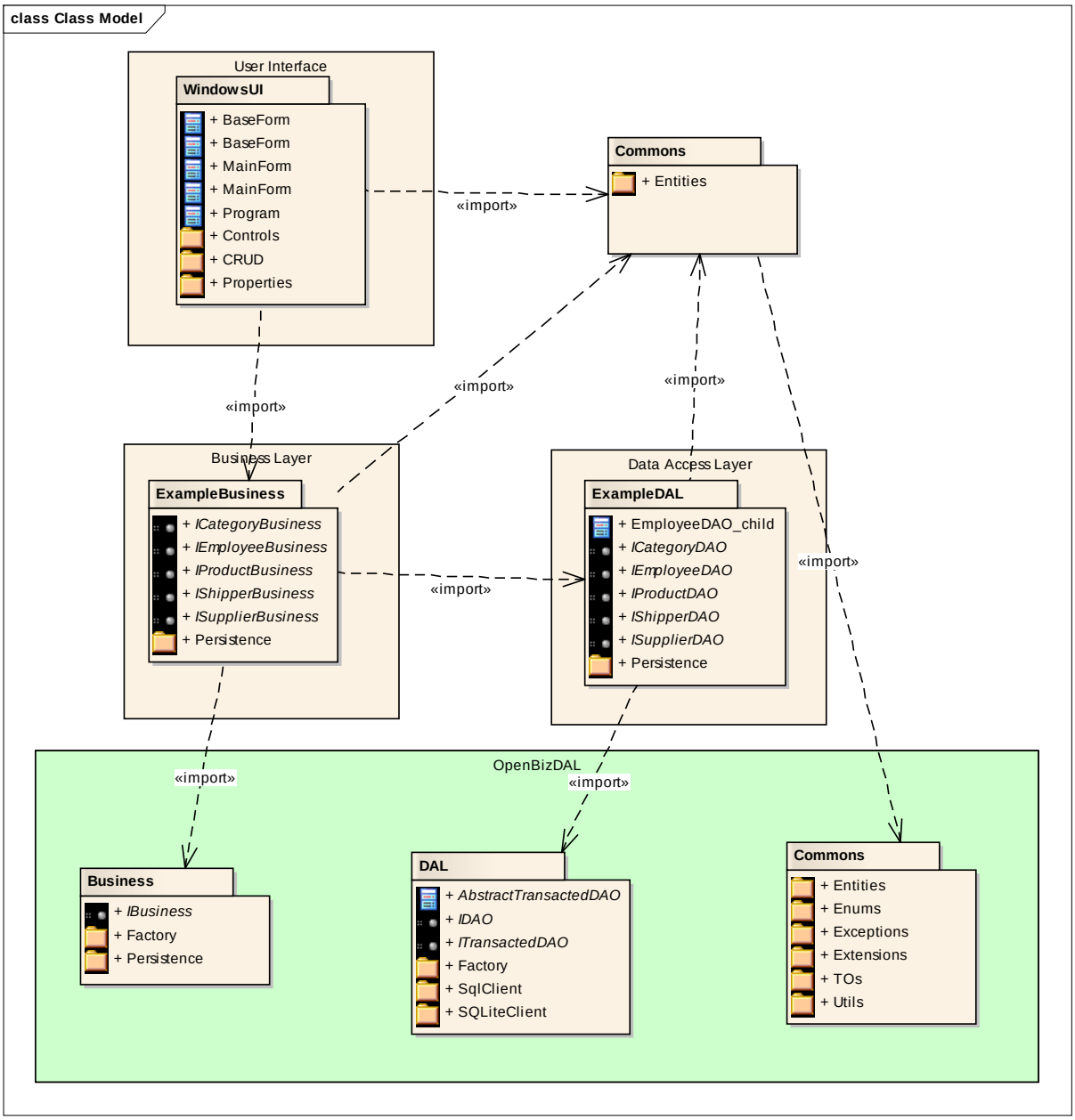

Figura 20 – Diagrama de classes da aplicação exemplo

A Figura 20 mostra as ligações existentes com cada camada, por exemplo, a interface gráfica apenas se comunica com a camada de negócios e essa apenas com a camada de acesso a dados. Verifica-se também a dependência com o framework OpenBizDAL: O Business da aplicação utiliza o Business do framework, e o DAL do software usa o DAL do framework.

Programar para interfaces e não para uma implementação é o princípio básico por trás desse exemplo. Cada camada somente terá interfaces expostas, e

para gerar as instancias de classes que as implementam é necessário utilizar as fábricas contidas no OpenBizDAL. As figuras abaixo mostram as interfaces expostas pelo ExampleDAL e ExampleBusiness:

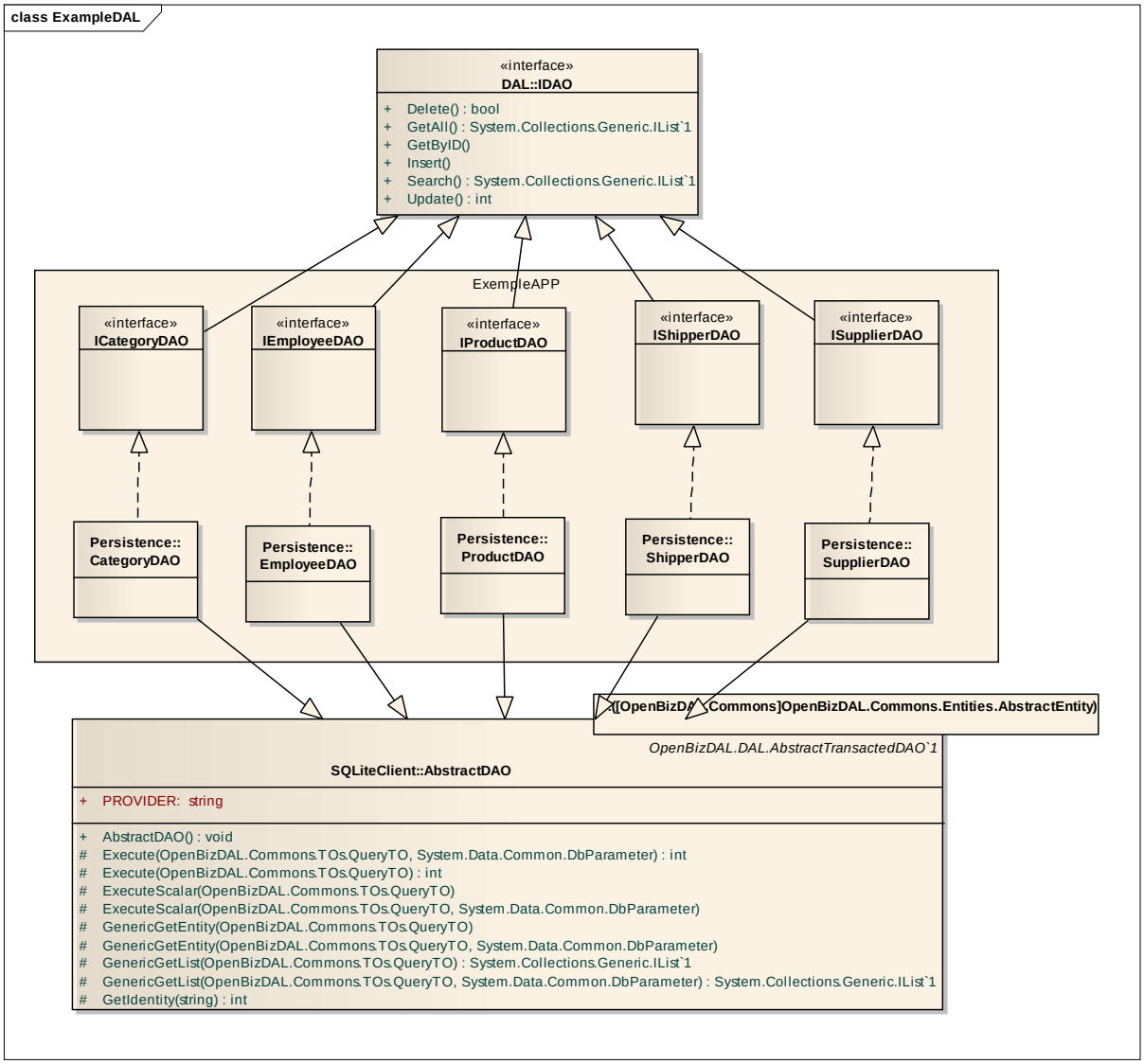

Figura 21 – Diagrama de classes da camada ExampleDAL

A Figura 21 mostra que cada interface exposta pela camada de dados do exemplo herda do IDAO contido no OpenBizDAL. E suas classes são implementadas e ocultas das camadas acima. O mesmo princípio aplica-se na Figura 22, mas na camada de negócios.

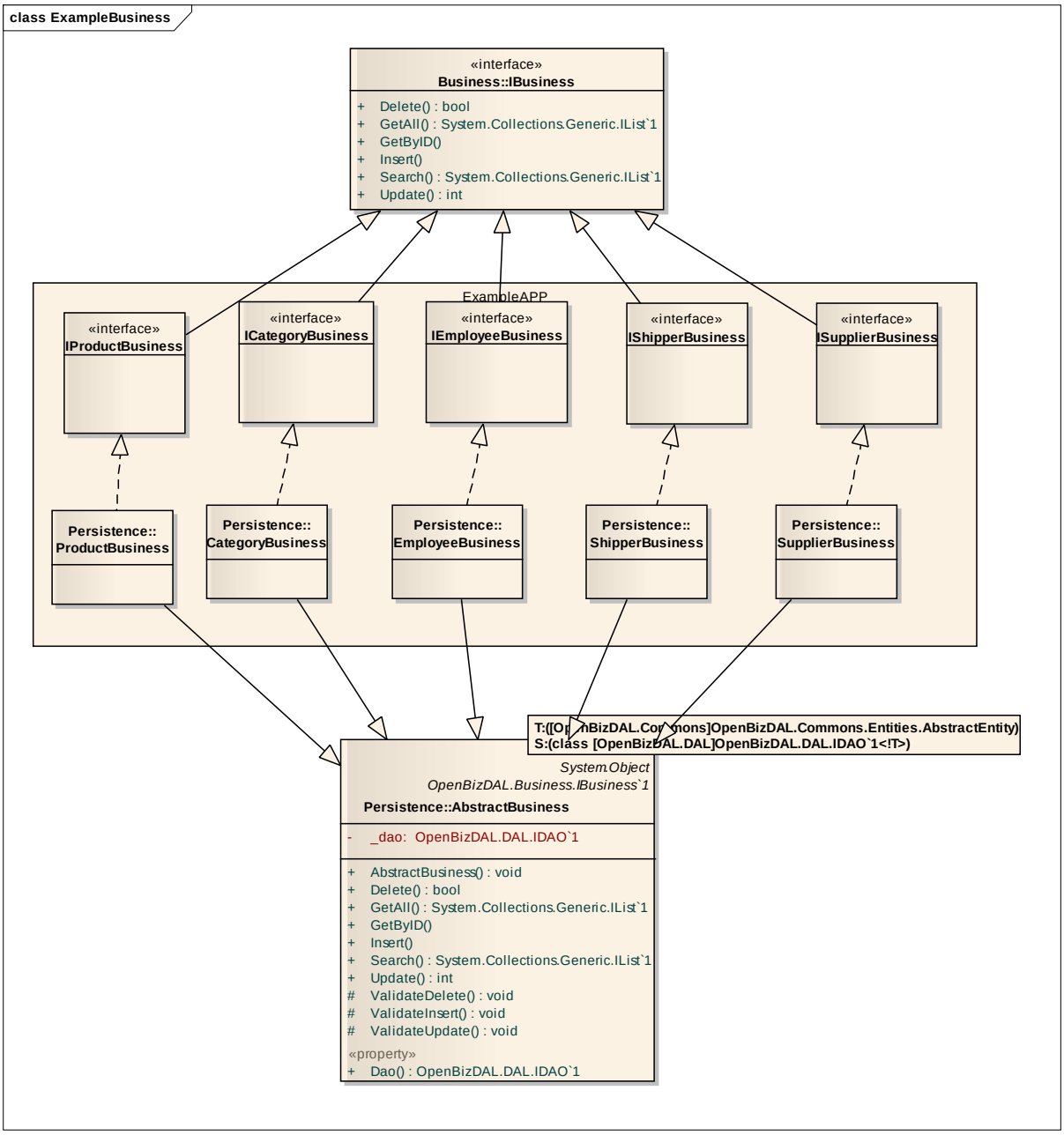

Figura 22 – Diagrama de classes da camada ExampleBusiness.

Para fabricar instâncias concretas dessas interfaces é utilizado o mapeamento criado nos arquivos de configuração do OpenBizDAL. Abaixo, o mapeamento utilizado pela camada *ExampleBusiness* que foi escrito dentro do arquivo *BusinessMapping.xml:*

```
<?xml version="1.0" encoding="utf-8" ?>
<mappings>
   <class name="ProductBusiness"
   fullClassName="ExampleBusiness.Persistence.ProductBusiness"/>
```

```
    <class name="EmployeeBusiness"
    fullClassName="ExampleBusiness.Persistence.EmployeeBusiness"/>
    <class name="CategoryBusiness"
    fullClassName="ExampleBusiness.Persistence.CategoryBusiness"/>
    <class name="SupplierBusiness"
    fullClassName="ExampleBusiness.Persistence.SupplierBusiness"/>
    <class name="ShipperBusiness"
    fullClassName="ExampleBusiness.Persistence.ShipperBusiness"/>
</mappings>
```

O atributo *name* corresponde ao nome da interface retirando o caractere "I", como exemplo, para interface *IProductBusiness* deve-se ter um registro no XML com o "*class name*" *ProductBuniss.* O valor correspondente ao "*fullClassName*" é utilizado pela fábrica para criar uma instancia que implementa aquela interface.

Já no caso do mapeamento do *ExampleDAL*, foi feito utilizando um mapeamento único, ou seja, não é necessário criar um XML como descrito acima. O trecho abaixo mostra como foi configurado para o *ExampleDAL.*

A diferença de configuração entre o *Business* e *DAL* demonstra apenas as duas distintas características, ambas as soluções podem ser aplicadas para qualquer camada:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="UseUniqueDaoMapping" value="true"/>
    <add key="DaoMappingPath" value="ExampleDAL.Persistence"/>
    <add key="ApplicationConfigurationFolder" value="Resources"/>
    <add key="UseUniqueBusinessMapping" value="false"/>
    <add key="BusinessMappingXmlFile"value="Mappings\BusinessMapping.xml"/>
  </appSettings>
  <connectionStrings>
    <add name="ApplicationConnectionString"
        providerName="System.Data.SQLite"
        connectionString="Data
Source=C:\Users\thiarley\Desktop\OpenBizDAL\Source\Example\ExampleApp\Database\SqlLite\North
WindSqlLite.db; Version=3; New=False;" />
  </connectionStrings>
</configuration>
```

Apenas aplicando o valor "*true*" para "*UseUniqueDaoMapping*" faz com que o framework utilize o caminho definido em "*DaoMappingPath*" para

automaticamente criar instancias das interfaces. O XML acima também define algumas configurações adicionais:

- *ApplicationConfigurationFolder* – Corresponde à pasta onde serão guardados todos os arquivos de configuração complementares, como consultas de banco e mapeamento de classes.
- *BusinessMappingXmlFile* – Quando o valor para "UseUniqueBusinessMapping" é definido como "false", o *framework* irá utilizar o arquivo de XML definido nessa chave.
- *ConnectionString* – Define todos os endereços de base de dados empregados pela aplicação.

A comunicação entre as camadas é feita através de entidades que, como já visto, são os espelhos das tabelas contidas na base de dados. A Figura 23 mostra que todas as entidades que representam os dados herdam da classe abstrata *AbstractEntity* do *OpenBizDAL*:

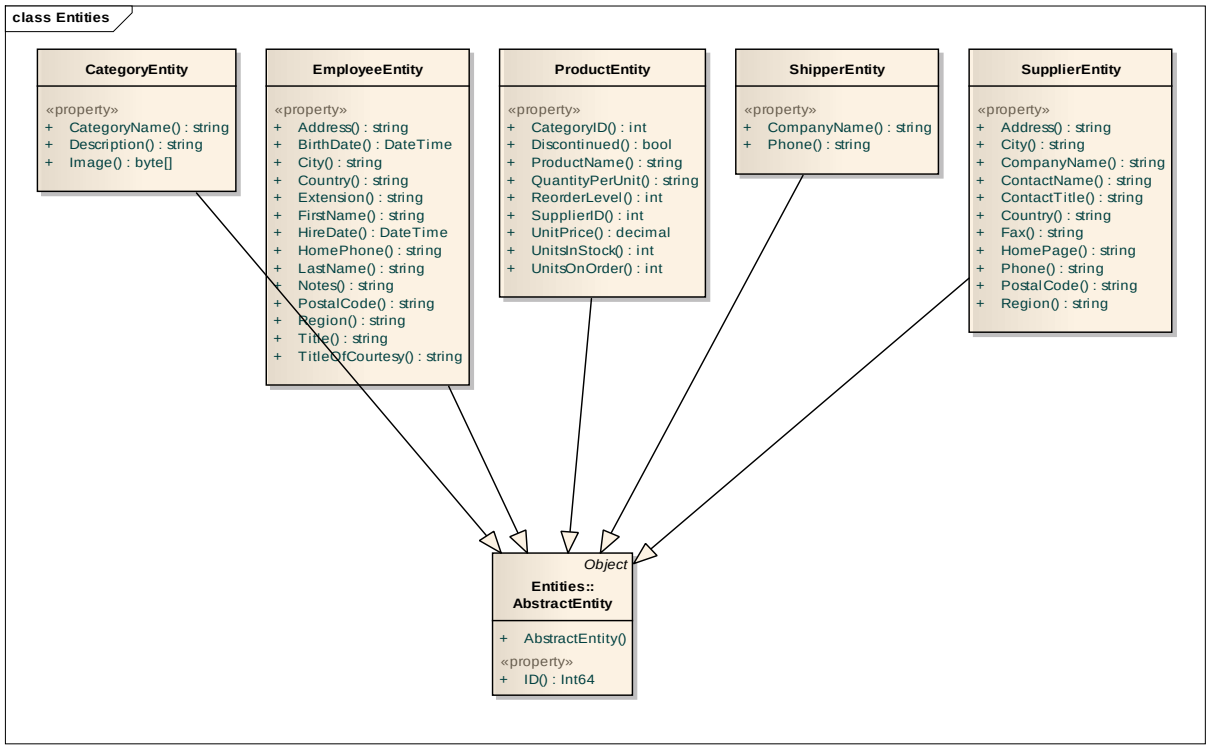

Figura 23 – Diagrama de classes das entidades

Para demonstrar a simplicidade da programação utilizando o *framework*, é apresentada a seguir a implementação dos objetos de *Business* e *DAL* para a tabela *Employee*:

```
public class EmployeeBusiness : AbstractBusiness<EmployeeEntity,IEmployeeDAO>,
IEmployeeBusiness
{
      protected override void ValidateInsert(EmployeeEntity entity)
      {
         if (entity == null)
            throw new ArgumentNullException();

         if (entity.FirstName == null || entity.LastName == null)
            throw new Exception("First Name or Last Name not informed");

         base.ValidateInsert(entity);
      }
 }

public class EmployeeDAO : AbstractDAO<EmployeeEntity> , IEmployeeDAO
{
}
```

O único requisito necessário para construir uma classe de negócio é herdar do *AbstractBusiness* do framework, caso deseje adicionar validações antes de uma operação de inserção, atualização ou remoção basta apenas fazer *override*(Re-implementa métodos da classe pai) dos métodos "*ValidateInsert*", "*ValidateUpdate*" e "*ValidateDelete*", o restante da operação é realizado automaticamente pela classe pai. Sem realizar validações, os objetos de acesso a dados funcionam seguindo o mesmo princípio.

Abaixo seguem algumas telas capturadas da aplicação em funcionamento. A Figura 24 mostra a tela principal do exemplo, composto apenas dos botões que irão direcionar o usuário para os cadastros de *employee* (empregado), product (produto), category (categoria), suplier (fornecedor) e shipper (serviço de entrega).

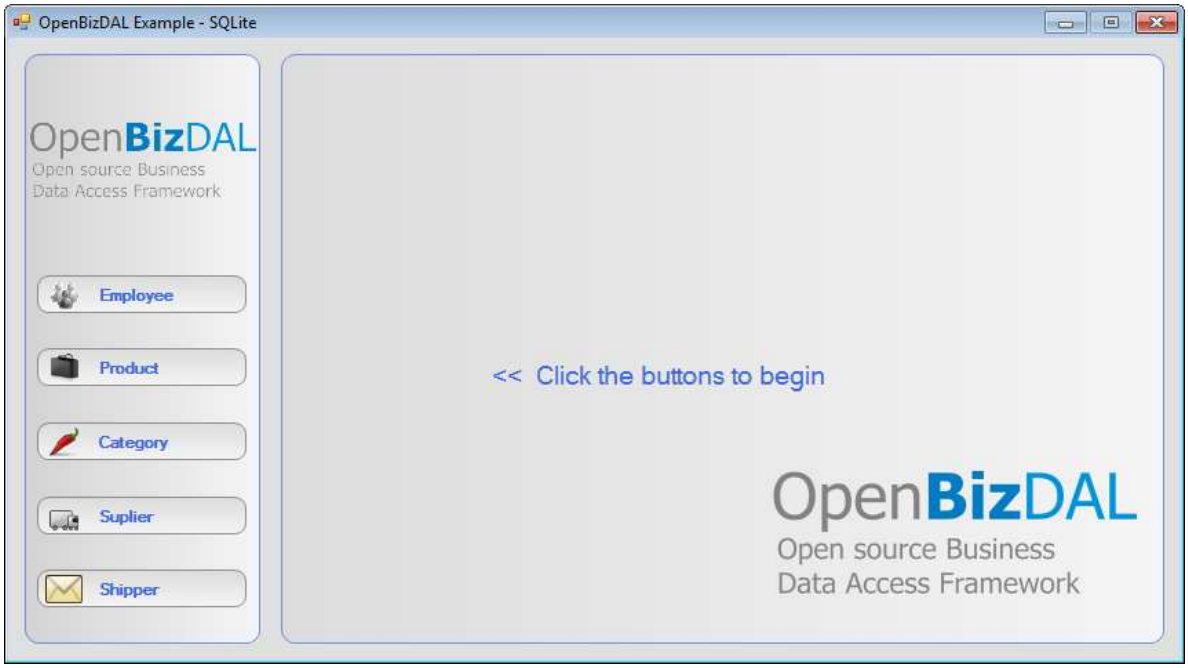

Figura 24 – Tela principal da aplicação exemplo.

A Figura 25 mostra a tela de cadastro de empregados, assim como a lista já cadastrada.

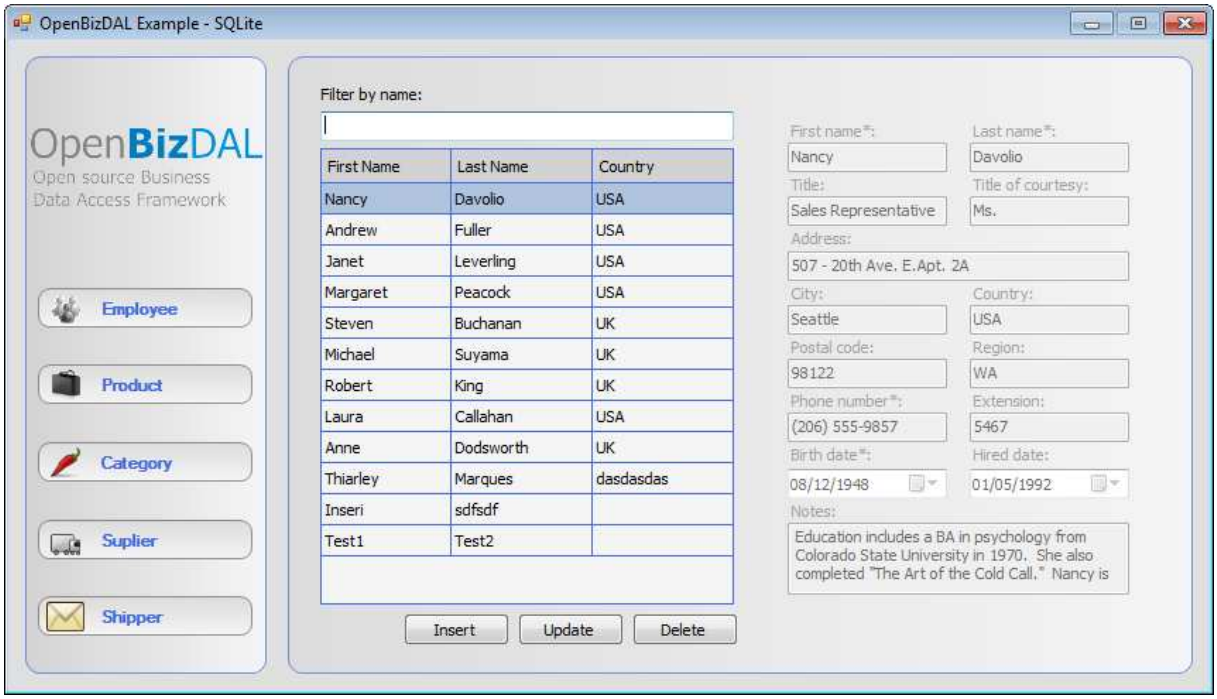

Figura 25 – Tela de cadastro de “*Employee”.*

A Figura 26 mostra os campos de inserção ativos quando o usuário deseja inserir algum novo cadastro.

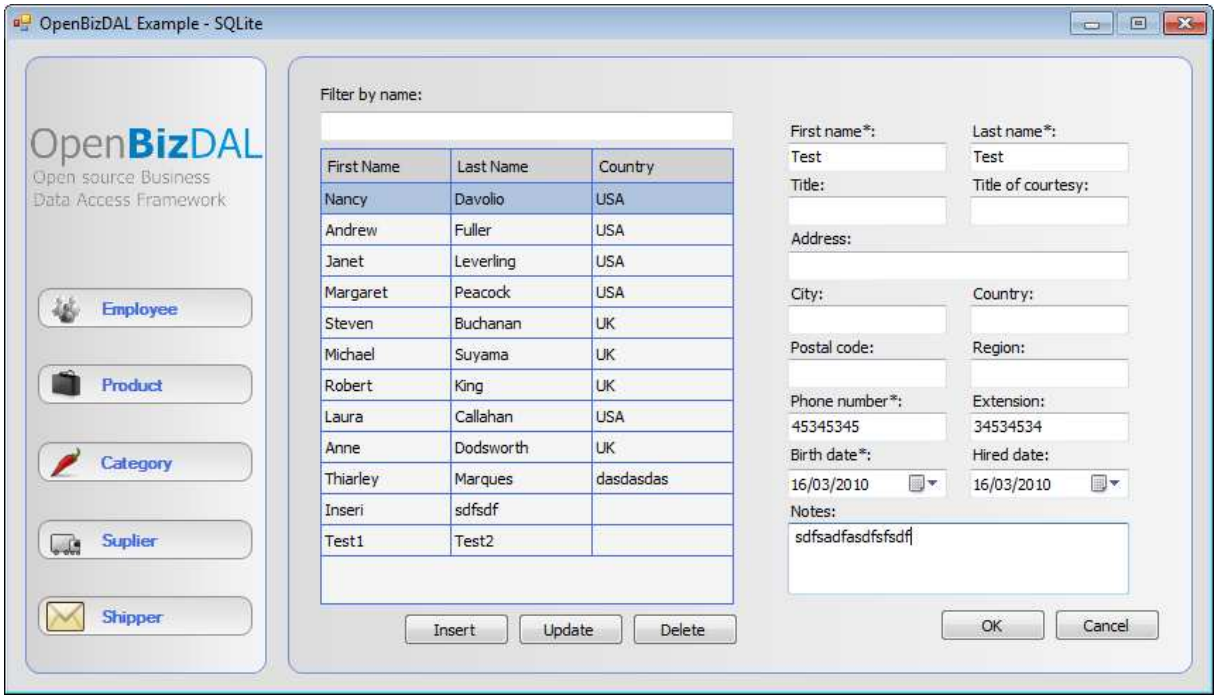

Figura 26 – Exemplo de criação de empregado.

A Figura 27 mostra a tela de cadastro de produtos, com opções de inserir, atualizar e remover.

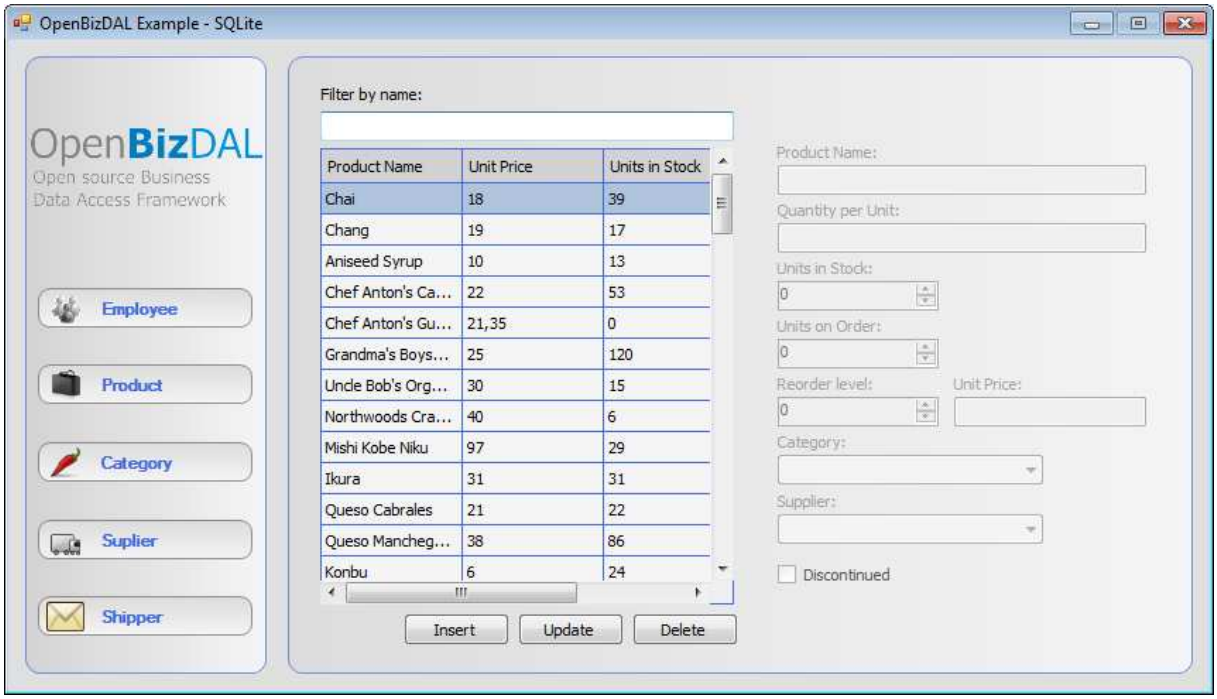

Figura 27 – Tela de cadastro de “*Product*”.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho é apresentada uma solução que visa tornar o desenvolvimento e projeto de um sistema menos oneroso e que incentiva o desenvolvedor a utilizar as melhores práticas de programação atraves da utilização de padrões de projeto.

A solução apresentada é o *framework* de código fonte aberto OpenBizDAL. Foi analisada sua arquitetura, características, principais padrões de projeto utilizados e suas vantagens e desvantagens. Também foi estudado um aplicativo implementado utilizando o produto, servindo de exemplo de forma a guiar a programação necessária para a utilização da ferramenta.

Foi visto outros frameworks com características semelhantes, *Linq to SQL, NHibernate e Entity framework*, enumerando suas vantagens e desvantagens. A Tabela 1 mostra um resumo comparativo entre as ferramentas:

Tabela 1 – Comparativo entre *frameworks*

Propriedades	LINQ to SQL	Entity Framework	Nhibernate	OpenBizDAL
Interface Gráfica para mapeamento	X	X	-	-
Multitrasação entre bancos diferentes (Suporte a sistemas distribuídos)	-	-	-	X
Open source	-	-	X	X
Consultas SQL fora do código em XML	-	-	X	X
Camada para regras de negócio	-	-	-	X
Flexibilidade e Produtividade	-	X	X	X
Uso de Herança e Polimorfismo	-	X	X	-
Lazy loading	X	X	X	-
Mapeamento Objeto Relacional	-	X	X	-
Mapeamento de classes concretas em XML	-	-	-	X

Incentiva o uso de padrões de projeto	**-**	**-**	**-**	**X**
Prover meios para desenvolvimento em multicamadas	**-**	**-**	**-**	**X**
Suporte a diferentes Sistemas de banco de dados	**-**	**X**	**X**	**X**

A Tabela 1 condensa as principais características do OpenBizDAL comparando com os demais *frameworks* apresentados. É possível notar que a grande desvantagem do OpenBizDAL está relacionado a falta de uma ferramenta utilitária gráfica para ajudar no desenvolvimento. Nota-se também que o mapeamento objeto relacional junto com herança e polimorfismo são pontos de deficiência na ferramenta. As principais vantagens estão relacionadas com o proposto para o desenvolvimento, como o suporte a vários bancos, multitransação, camada de regras de negócio, flexibilidade e principalmente o incentivo ao uso de padrões de projeto e arquitetura multicamadas.

6.1 Trabalhos futuros

A partir da contribuição apresentada pelo presente trabalho em relação ao desenvolvimento do framework, alguns trabalhos futuros podem ser indicados objetivando dar continuidade e aprimorar a ferramenta aqui proposta:

- Desenvolvimento de um *plugin* para *Visual Studio* com o objetivo de gerar código fonte e arquivos de configuração automaticamente, tendo como base tabelas do banco de dados.
- Dar suporte ao modelo objeto relacional, abstraindo a estrutura do banco e trabalhar apenas com o conceitual de dados.
- Criar um provedor LINQ, que seria chamado de *LINQ to OpenBizDAL*, facilitando mais ainda consultas realizadas.

Considerando que o *OpenBizDAL* é um projeto Open Source pode vir a beneficiar um grande numero de desenvolvedores e colaboradores que podem vir a contribuir na incorporação de experiências e na manutenção contínua em versões futuras.

REFERÊNCIAS

ALEXANDER, C. **A Pattern Language**, Oxford University Press, New York, 1977.

ALEXANDER, C. **The Timeless Way of Building**, Oxford University Press, New York, 1979.

APPLETON, B. **Patterns and Software: Essential Concepts and Terminology**, 2000, disponível em: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html >. Acesso em 13 de Mar 2010.

BAUER, Christian. KINQ, Gavin. **Hibernate in Action.** Manning Publications, 2004.

FAYAD, M.E., SCHMIDT, D., and JOHNSON, R. **Building Application Frameworks: Object-Oriented.** John Wiley & Sons, 1999.

FOWLER, M. **Patterns of Enterprise Application Architecture,** Addison Wesley, 2002.

FRANTZ, R. Z. **Introdução a Design Patterns para o desenvolvimento de software**, 2007, Disponível em: <http://www.jeebrasil.com.br/tags/designpatterns>. Acesso em: 12 Mar 2010.

FREEMAN, E. **Head First Design Patterns – Use a Cabeça Padrões de Projeto**. Alta Books, 2007.

GAMMA, E; HELM, R; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison Wesley, 1995.

GNU, Operation System, **GNU Lesser General Public License,** 2007. Disponível em <http://www.gnu.org/licenses/lgpl.html>. Acessado em 18 de Abril de 2010.

JENNINGS, Roger. **ADO.NET 3.5 with LINQ and the Entity Framework.** Wiley Publishing Inc, 2009.

JOHNSON, R. E. **How to Develop Frameworks.** Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2000.

KRIESER, Paulo. **Escolhendo a Linguagem: Java vs .NET.** 2009, disponível em <http://www.baguete.com.br/colunistas/colunas/51/paulo-krieser/29/05/2009/escolhendo-a-linguagem-java-vs-net>, acessado em 08 de Maio de 2010.

KUATÉ, Pierre H., HARRIS, Tobin, BAUER, Christian, KING, Gavin. **NHibernate in Action.** Manning Publications, 2008.

LUCENA, Carlos J. P., MARKIEWICZ, Marcus E. **Object Oriented Framework Development.** 2001, disponível em <http://www.acm.org/crossroads/xrds7-4/frameworks.html >, acessado em 17 de Março de 2010.

MACORATTI, José C. **Introdução ao ADO .NET Entity Framework II,** 2009. Disponível em <http://www.macoratti.net/09/06/vb_iaef.htm>. Acessado em 21 de Março de 2010.

MALLALIEU, Tim. **Update on LINQ to SQL and LINQ to Entities Roadmap,** 2008, Disponível em <http://blogs.msdn.com/adonet/archive/2008/10/29/update-on-linq-to-sql-and-linq-to-entities-roadmap.aspx>. Acessado em 20 de Março de 2010.

MSDN, Microsoft Developer Network. **LINQ to SQL: .NET Language-Integrated Query for Relational Data**, 2007, Disponível em < http://msdn.microsoft.com/en-us/library/bb425822.aspx> , Acessado em 20 de Março de 2010.

MSDN, Microsoft Developer Network. **Propriedades (Guia de programação do C#)**, 20??, Disponível em <http://msdn.microsoft.com/pt-br/library/x9fsa0sw.aspx> , Acessado em 20 de Março de 2010.

MSDN, Microsoft Developer Network. **Atributos (guia de programação C#)**, 20??, Disponível em < http://msdn.microsoft.com/pt-br/library/z0w1kczw.aspx> , Acessado em 20 de Março de 2010.

NHIBERNATE, **NHibernate Forge,** 2010. Disponível em <http://nhforge.org/doc/nh/en/index.html>, acessado em 18 de Abril de 2010.

SAMPAIO, Oscar A. K. **Aplicações Multicamadas,** 2008. Disponível em < http://www.scribd.com/doc/5055030/Aplicacoes-Multicamadas>, acessado em 08 de Maio e 2010.

SAUVÉ, Jacques Philippe, **Tipos de frameworks,** 2000, disponível em < http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/tipos.htm>, acessado em 08 de Maio de 2010.

SUN MICROSYSTEMS, I. **Core J2EE Patterns – Data Access Object**, 2002, disponível em <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>. Acessado em 15 de Março 2010.

WORTZEL, Avi. **Entity Framework vs. Linq to SQL**, 2008, Disponível em <http://blogs.microsoft.co.il/blogs/aviwortzel/archive/2008/01/04/entity-framework-vs-linq-to-sql.aspx >. Acessado em 21 de Março de 2010.