UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE GERAÇÃO DE CÓDIGO A PARTIR DO DICIONÁRIO DE DADOS DO SQL SERVER PARA TECNOLOGIA ASP.NET

ANDRÉ CÉSAR HEIDEN

BLUMENAU
2005

2005/1-04

ANDRÉ CÉSAR HEIDEN

FERRAMENTA DE GERAÇÃO DE CÓDIGO A PARTIR DO DICIONÁRIO DE DADOS DO SQL SERVER PARA TECNOLOGIA ASP.NET

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Alexander Roberto Valdameri, Mestre - Orientador

BLUMENAU
2005

2005/1-04

FERRAMENTA DE GERAÇÃO DE CÓDIGO A PARTIR DO DICIONÁRIO DE DADOS DO SQL SERVER PARA TECNOLOGIA ASP.NET

Por

ANDRÉ CÉSAR HEIDEN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Jomi Fred Hubner, Mestre – FURB

Blumenau, junho de 2005.

AGRADECIMENTOS

Agradeço os meus pais, Vilmar e Mariana Heiden que sempre me incentivaram e nunca deixaram desistir nos momentos difíceis.

Agradeço a minha namorada Simoni, pela paciência em todos os momentos.

Agradeço também a todos os meus colegas e professores, por todos os momentos, cobranças e experiências repassados. Em especial ao meu orientador Alexander Roberto Valdameri que me apoiou na definição e no esclarecimento de dúvidas desse trabalho.

Não desista, continue, acredite naquilo que você faz, não desmereça seu trabalho, infelizmente em informática a palavra reconhecimento e mérito não existe nesse dicionário, trabalhamos em uma área ingrata, onde as pessoas somente dão valor quando perdem o recurso ou solução que provemos, e nessa hora é que temos nosso valor reconhecido. Busque sua satisfação pessoal, se você é feliz e gosta do que faz isso é o que importa.

(Autor Desconhecido)

RESUMO

O processo de desenvolvimento de *software* tem se especializado dia-a-dia. Neste contexto, o uso de ferramentas para auxiliar a automatização de rotinas tidas como triviais torna-se cada vez maior. O presente trabalho apresenta uma ferramenta *desktop* de geração de código que lê o dicionário de dados do Microsoft *SQL Server 2000* e gera páginas para tecnologia ASP.NET. A ferramenta foi desenvolvida utilizando a linguagem de programação C# e possui as opções de gerar páginas de inclusão, alteração e pesquisa além da opção de exclusão que está disponível na página de consulta. As páginas geradas também utilizam a linguagem de programação C# e controles do ASP.NET.

Palavras-chave: .NET *Framework*. ASP.NET. C#. Geração de código.

ABSTRACT

The process of software development has specialized day-by-day. In this context, the use of tools to assist the automatization of routines as trivial becomes each bigger time. The present work presents a tool desktop of code generation that reads the dictionary of data of Microsoft SQL Server 2000 and generates pages for technology ASP.NET. The tool was developed using the programming language C# and possessed the options to generate pages of inclusion, alteration and research beyond the exclusion option that is available in the consultation page. The pages also generated use the programming language C # and controls of the ASP.NET.

Key-Words: .NET *Framework*. ASP.NET. C#. Code generation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADO – Activex Data Objects

ASP – Active Server Pages

BLC – Base Class Library

C# – C-Sharp

CLR – Common Language Runtime

CLS – Common Language Specification

COM – Component Object Model

CSS – Cascading Style Sheets

CTS – Common Type System

DHTML – Dynamic Hypertext Markup Language

DLL – Dynamic Link Library

FTP – File Transfer Protocol

HTML – Hypertext Markup Language

HTTP – HyperText Transfer Protocol

IL – Intermediate Language

JIT – Just in Time

MSIL – Microsoft Intermediate Language

RAD – Rapid Application Development

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

UML – Unified Modeling Language

XML – Extensible Markup Language

W3C – World Wide Web Consortium

SUMÁRIO

1 INTRODUÇÃO

Devido as constantes e rápidas mudanças da tecnologia e principalmente das mídias eletrônicas, o mercado está cada vez mais dinâmico e competitivo. Como conseqüência, as empresas estão sendo obrigadas a se adaptar ao mundo *web*, que alterou o cotidiano, as relações e fundamentalmente o trabalho dos indivíduos. Hoje é difícil mencionar um endereço *web* que não apresente notícias, comunicados à imprensa, notas ou qualquer informação de caráter periódico. Com exceção de algumas páginas, a maioria dos *sites* tem sempre algo novo a informar. Portanto, é para essa infinidade de *sites* e também para *intranets* e *extranets* que aparece a necessidade de utilizar ferramentas de geração de código para agilizar processos que consomem tempo. Produtividade é de vital importância para se manter ativo no mercado, e com a plataforma .NET é possível ter grandes ganhos (MICROSOFT CORPORATION, 2004).

Programas de geração de código são programas que escrevem programas. Um gerador de código é uma ferramenta de auxílio ao processo de desenvolvimento de sistemas que atua na fase de implementação do projeto, gerando o código-fonte que seria criado pelo programador. De forma geral, esses geradores criam o código-fonte com base nas informações existentes no modelo de dados. Isso significa ganho de produtividade em interface de cadastro, listagens de registros e telas de baixa complexidade.

.NET é uma plataforma de *software* que conecta informações, sistemas, pessoas e dispositivos. A plataforma .NET conecta uma grande variedade de tecnologias de uso pessoal e de negócios, de dispositivos móveis a servidores corporativos, permitindo o acesso a informações, a qualquer hora, em qualquer lugar e em qualquer dispositivo (MICROSOFT CORPORATION, 2004).

Em se tratando de repositório de dados a tecnologia .NET apresenta forte

conectividade com o SGBD Microsoft SQL Server, objeto de estudo neste trabalho. Utilizando a ferramenta Microsoft *Visual Studio* .NET foi desenvolvida uma aplicação *desktop* que lê o dicionário de dados do *SQL Server* e cria páginas dinâmicas para *web* em tecnologia .NET para manutenção dos dados. Pessoas sem conhecimentos técnicos poderão manipular a ferramenta, pois é de fácil utilização e entendimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar uma ferramenta para geração de código que leia o dicionário de dados do *SQL Server* 2000 e crie páginas de manutenção de dados para tecnologia ASP.NET.

1.2 MOTIVAÇÃO

A motivação principal do desenvolvimento do trabalho foi à importância de ressaltar que com a dinâmica do mercado torna-se necessário utilizar uma ferramenta que agilize a criação de páginas dinâmicas para *web*. O profissional que for usá-la não precisa necessariamente possuir conhecimentos técnicos, o que a torna uma solução fácil de usar.

1.3 ESTRUTURA DO TRABALHO

No capítulo 2, são abordados os temas: plataforma Microsoft .NET, Microsoft .NET *Framework*, Microsoft *Intermediate Language* (MSIL), *Common Language Runtime* (CLR), ASP.NET, C#, Microsoft *Visual Studio* .NET 2003, ADO.NET, *Internet Information Server* (IIS), Microsoft *SQL Server* 2000 e geração de código.

O capítulo 3 trata de tópicos relacionados com a especificação e implementação da

aplicação, contendo os requisitos principais, diagramas, especificação e implementação da solução. Para ilustrar o funcionamento há a apresentação de um ambiente caracterizado para o estudo de caso.

Finalizando, no capítulo 4 tem-se a conclusão, dificuldades encontradas na elaboração do projeto e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados os temas: plataforma Microsoft .NET, Microsoft .NET *Framework*, Microsoft *Intermediate Language* (MSIL), *Common Language Runtime* (CLR), ASP.NET, C#, Microsoft *Visual Studio* .NET 2003, ADO.NET, *Internet Information Server* (IIS), Microsoft *SQL Server* 2000 e geração de código.

2.1 MODELO CLIENTE SERVIDOR

A *internet* é baseada no paradigma cliente servidor. Qualquer computador pode ser, simultaneamente, cliente e servidor mediante a circunstância. A análise deste modelo pode ser bipartida uma vez que tem-se por um lado, servidores de recursos *hardware* e, por outro, servidores de *software*.

Em nível de *hardware*, pode-se citar alguns servidores como: servidores de impressão (*Print Servers*), servidores para armazenamento de dados (*File Servers*), servidores de acesso à *internet* (*Proxy Servers*).

Este modelo pode ser estendido ao domínio das aplicações, isto é, ao *software* cliente servidor. Neste domínio, uma aplicação é dividida em duas partes, um *front end* (acesso frontal) e um *back end* (servidor de recursos).

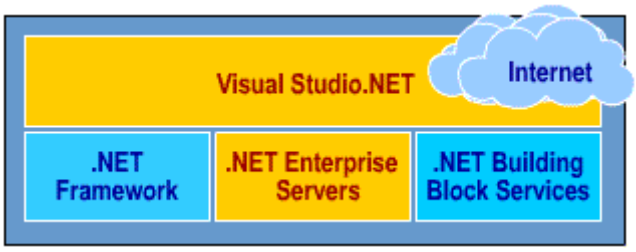
O modelo cliente servidor é muito usado em bases de dados, onde o servidor armazena e gera a informação e onde existem clientes que acedem concorrentemente aos recursos partilhados pelo servidor. O cliente é computador ou programa que acede a recursos fornecidos por outros computadores ou programas designados por servidores. O servidor é um computador remoto, normalmente muito bem equipado, de grande rendimento e desempenho, que tem a função de servir a diversos propósitos dos computadores que a ele requisitam

(clientes).

Para que um usuário possa navegar na *internet*, transferir arquivos, enviar ou receber *e-mails* ou utilizar qualquer outro tipo de serviço da *internet*, necessita antes de qualquer coisa estabelecer uma ligação com um computador remoto que lhe forneça estes serviços. Neste cenário, o computador do usuário é o cliente e o computador ao qual ele se conecta é o servidor ou prestador de serviços.

2.2 PLATAFORMA MICROSOFT .NET

A plataforma .NET (Figura 1) é um modelo de desenvolvimento no qual o aplicativo torna-se um dispositivo independente da plataforma, onde dados e funcionalidades do aplicativo podem ficar disponíveis na internet. O .NET *Framework* é a infra-estrutura do .NET (MICROSOFT CORPORATION, 2003).



Fonte: MSDN Library (2004)

Figura 1 – Estrutura do Microsoft .NET.

Criada como uma arquitetura aberta, .NET é uma plataforma que pode ser usada para construção e execução da nova geração de aplicativos Microsoft e aplicativos *web*.

A plataforma .NET é composta das tecnologias:

a) Microsoft .NET *Framework*;

b) Microsoft .NET *Building block services*;

c) Microsoft .NET *Enterprise servers*;

d) Microsoft *Visual Studio* .NET;

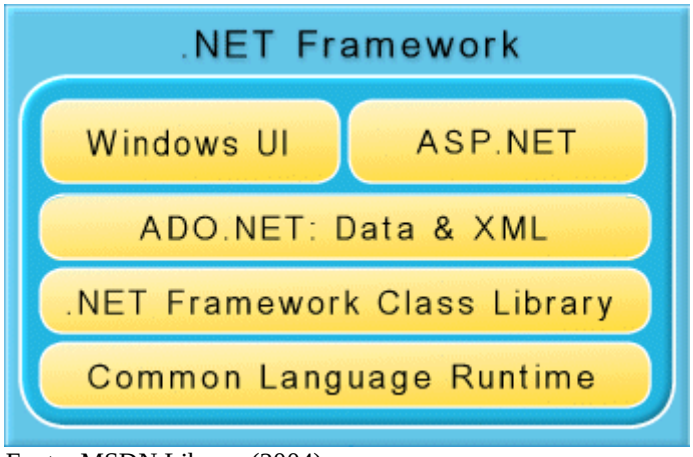
e) Microsoft *Windows* .NET.

Deitel et al. (2003, p. 13) afirmam que uma característica importante do .NET é a independência de linguagem e plataforma específica. Desenvolvedores têm a liberdade de criar aplicativos em qualquer linguagem compatível com essa tecnologia. Essa característica é válida para tecnologia *Active Server Pages* .NET (ASP.NET), páginas de servidor ativas da Microsoft, que permite criar aplicativos para *web*.

2.2.1 Microsoft .NET *Framework* / MSIL / CLR

Segundo Sant'anna (2001a), "do ponto de vista dos programadores, a .NET *Framework* é o sistema operacional. É através dela que são invocadas todas as funções necessárias ao funcionamento dos programas, sob qualquer sistema operacional" (Figura 2).

> O .NET *Framework* é a nova plataforma projetada para simplificar o desenvolvimento de aplicações, com um foco imediato no ambiente altamente distribuído da *internet* e um foco secundário no ambiente *desktop*. Ele fornece um conjunto de serviços para o sistema, as classes e os tipos de dados que convergem todas as camadas do desenvolvimento de *software* sobre o sistema operacional, proporcionando uma grande integração entre todas as camadas existentes e melhorando substancialmente a produtividade do desenvolvedor. Também pode ser visto como uma camada sobre o sistema operacional, que lhe endereça todas as necessidades de acesso. Muitos dos problemas acima citados são resolvidos por ele. Como a plataforma isola o sistema operacional, quando desenvolvemos uma aplicação, a fazemos olhando agora para o *framework*, e não mais para o SO. O primeiro resultado é a portabilidade para outras plataformas e *hardware*. (GALUPPO; MATHEUS; SANTOS, 2004, p. 16).

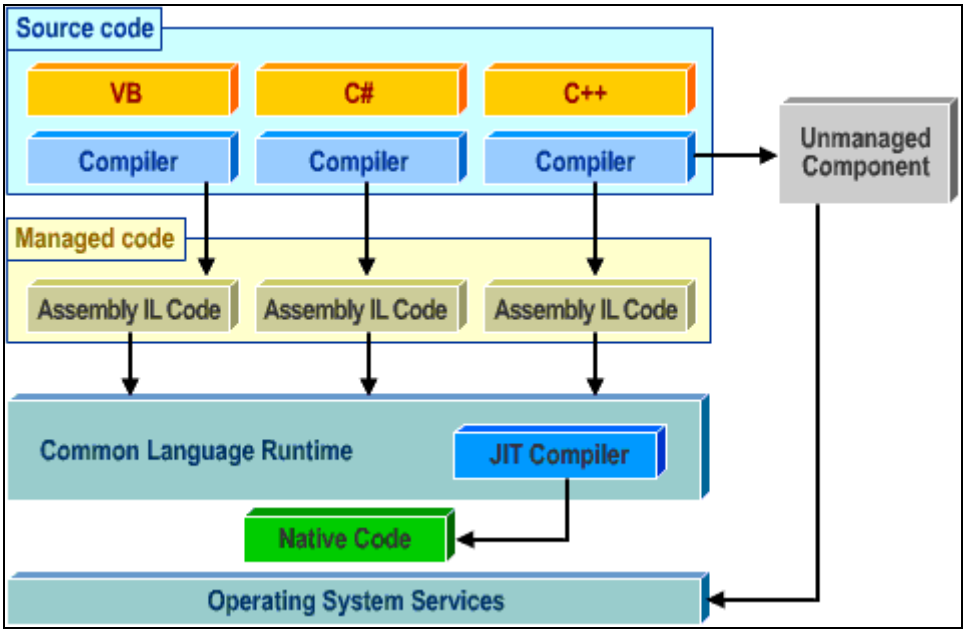


Fonte: MSDN Library (2004)

Figura 2 – Estrutura do .NET *Framework*.

Um projeto pode ser escrito utilizando mais de uma linguagem suportada pela tecnologia .NET, pois será compilado para uma linguagem intermediária, a MSIL. Os componentes utilizados não precisam ser registrados, basta copiar o programa de uma pasta para outra e continuará funcionamento normalmente.

CLR (Figura 3) fornece o ambiente de execução onde as aplicações .NET trabalham. O código que executa sob o CLR é chamado de *managed code* (código gerenciado). O CLR define regras de execução desse código, gerenciamento de segurança e de memória, segurança do tipo e interoperabilidade de interlinguagem.



Fonte: MSDN Library (2004)

Figura 3 – Estrutura do CLR.

Quando o código é compilado ele não é imediatamente convertido para linguagem de máquina. O compilador converte o fonte do programa para o MSIL, que são instruções generalizadas independentes de processador. Utilizando a tecnologia de compilação *Just-In-Time* (JIT) *Compiler*, o programa é convertido para código nativo no momento de sua execução.

2.2.2 ASP.NET / C#

O ASP.NET é a evolução do *Active Server Pages* (ASP) da Microsoft, sendo uma tecnologia que permite criar e controlar conteúdo dinâmico para *web*. O ASP.NET é um ambiente para construção, distribuição e execução de aplicações *web* baseado no .NET *Framework*. O ASP.NET é semelhante ao ASP no que se refere a ser um *plug-in Internet Services Application Programming Interface* (ISAPI) para o IIS, mas as semelhanças terminam aí. ASP.NET é uma versão superior do ASP. Embora ele ainda permita inserir o código diretamente em uma página e executar, agora está limitado a uma linguagem por página.

O ASP.NET aboliu as linguagens de *script* e agora fornece uma nova estrutura de página que lhe permite, entre outras inovações, separar o código do *design*, simplificar o modelo de distribuição para *xcopy*, melhorar a depuração, utilizar o armazenamento em *cache*, adicionar novas opções de estado de sessão e melhorar a disponibilidade.

Alguns benefícios da ASP.NET (MICROSOFT CORPORATION, 2003):

a) páginas ASP.NET são compiladas;
b) são construídas com controles de interface do lado do servidor. São controles de interface básicos (*textbox, label*), controles de validação, controles de dados (*datagrid, datalist*), controles mais complexos (calendários, *adrotator*);
c) ASP.NET faz parte da .NET *Framework*, logo todas as classes que compõem o *framework* podem ser usadas na criação de uma página *web*;
d) é totalmente orientada a objeto.

O processo de geração de página para o cliente (Figura 4) é diferente do ASP tradicional, os *webcontrols* (componentes do ASP.NET) que são inseridos na página não são "*tag*s" reconhecidas pelos *browsers*, sendo necessário executar uma conversão para ser

enviada e processada de forma compatível ao cliente (*browser*) (MICROSOFT CORPORATION, 2002). Nesta conversão é necessário atribuir nomes únicos para cada controle e criar rotinas que possam ser disparadas pelo lado do cliente, correspondendo às ações feitas pelo usuário, enviando ao servidor para que saiba quem disparou este evento.

Fonte: Microsoft Corporation (2002)

Figura 4 –Processo de geração de página para o cliente.

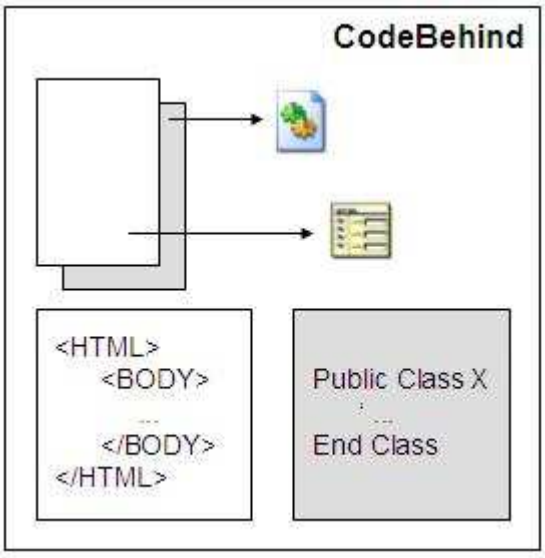
Na Figura 4 um *browser* solicita uma página a um servidor e envia junto com esta solicitação características (como versão, tipo, capacidade e sistema operacional). O servidor recebe estas informações, localiza a página e gera uma pseudopágina sobre o *layout* já definido.

O resultado deste processo é submetido a outro processo, que transforma esta pseudopágina no formato compatível do *browser* correspondente. Finalmente o resultado desta transformação é enviado ao cliente.

Os arquivos criados em ASP.NET possuem a extensão *.aspx*. É possível usar qualquer editor de texto para digitar o código de uma página ASP.NET. No ASP tinha-se o código que era executado dentro das *tags* <% ... %> que atualmente não é mais compilado e seu uso não aconselhado, pois evita a mescla de código de servidor juntamente com o código HTML.

Andrade (2004) afirma que uma das grandes inovações é o *CodeBehind*. Com ele é possível separar o código da programação do código HTML. O *CodeBehind* ao compilar o aplicativo esse código é "encapsulado" dentro da DLL, sendo necessário apenas o envio do

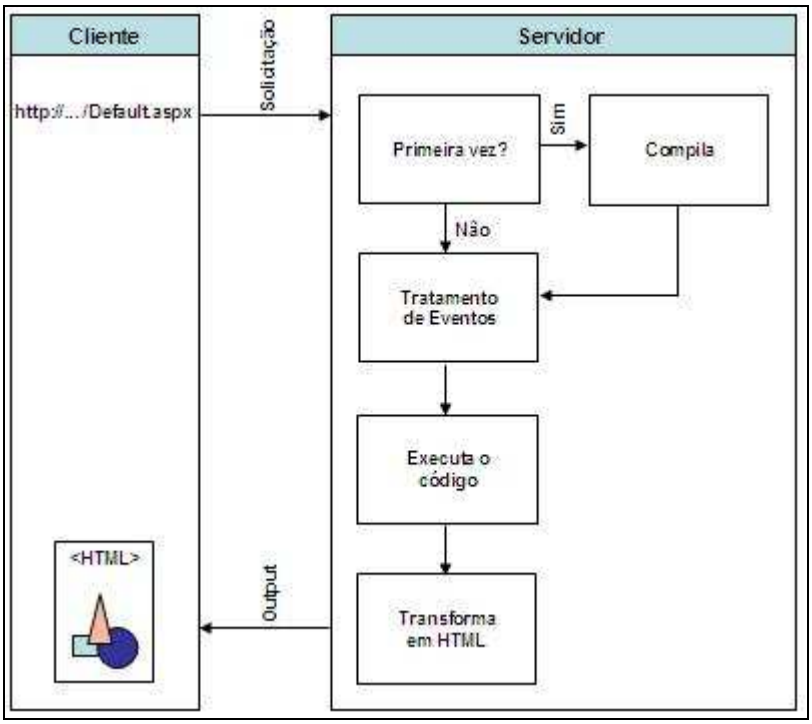
arquivo *ASPX* para o servidor de produção. Ao criar um novo *webform* no *Visual Studio* .NET, são criados dois arquivos (Figura 5): o arquivo *Arquivo.aspx* (arquivo que contém o código HTML) e o *Arquivo.aspx.cs* (arquivo de código propriamente dito). A extensão do arquivo de *CodeBehind* vai depender da linguagem escolhida pelo desenvolvedor. Para VB.NET tem-se *.aspx.vb* e C# tem-se *.aspx.cs*.



Fonte: Andrade (2004)

Figura 5 – *CodeBehind*.

Vale lembrar que é possível desenvolver no mesmo arquivo HTML e código. É comum que a primeira vez que uma página *ASPX* é executada ela sofre um pequeno retardo. Isto se deve à compilação do código. Nas vezes subsequentes, esta etapa não ocorre deixando a execução do aplicativo muito mais rápida. Isto ocorre (Figura 6) ao receber a solicitação de uma página, a mesma é compilada (caso seja a primeira vez), trata os eventos da página e de seus controles, executa o código referente a esses eventos e finalmente gera o código HTML e envia ao *browser*.



Fonte: Andrade (2004)

Figura 6 –Processo de execução da página ASP.NET.

Conforme já mencionado, com a tecnologia ASP.NET é possível criar aplicativos em qualquer linguagem compatível com *Commom Language Specification* (CLS). Algumas linguagens suportadas pela CLS são: *Perl*, *Python*, C++, *Visual Basic* .NET e C# (pronuncia-se C *Sharp*).

Segundo Sant'anna (2001b), C# é a mais nova linguagem que a Microsoft criou a partir do zero, unindo o que existe de melhor em todas as tecnologias no mercado e adicionou um pouco de facilidades e recursos. Deitel et al. (2003, p. 8) afirma que C# foi construída com base em linguagens amplamente usadas e bem desenvolvidas.

2.2.3 A linguagem C#

Segundo Galuppo (2001) C# é uma linguagem de programação da nova plataforma .NET, derivada de C/C++, simples, moderna, orientada a objetos e fortemente tipada (*type-safe*). C# possui o poder do C/C++ aliado a alta produtividade do *Visual Basic*. C# é a

linguagem nativa para .NET *Common Language Runtime* (CLR), mecanismo de execução da plataforma .NET. Isso possibilita a convivência com várias outras linguagens especifícadas pela CLS. Por exemplo, uma classe base pode ser escrita em C#, derivada em *Visual Basic* e novamente derivada em C#.

.NET *Common Language Runtime* é um ambiente baseado em componentes, e C# é desenhado para facilitar a criação de componentes. Os conceitos de componentes, como propriedades, métodos, eventos e atributos, são fortemente aplicados. Documentação pode ser escrita dentro dos componentes e exportadas para XML.

A linguagem C# não requer bibliotecas de tipo (*type libraries*), arquivos de cabeçalho (*header files*), arquivos IDL (*IDL files*). Os componentes criados em C#, são auto-descritivos e não necessitam de processo de registro. Tudo é objeto em C#, ao contrário de linguagens como Java ou C++, tipos de dados e objetos interagem. C# fornece um "sistema unificado de tipos", onde todos os tipos são tratados como objetos, sem perda de performance, ao contrário de linguagens como *Lisp* ou *Smalltalk*.

Coletor de Lixo (*Garbage Collection*) que fornece gerenciamento automático de memória, excessões (*Exceptions*) que disparam erros que podem ser tratados, segurança no tipo de dados (*Type-safety*) que assegura a manipulação de variáveis e *casts* e versão(*Versioning*), são recursos encontrados na linguagem para construção dessa categoria de *software*.

A linguagem C# é montada sobre a "herança" do C++, isso torna confortável a adaptação do programador C++. C# possibilita utilização de ponteiros, na execução de código não gerenciado.

A linguagem C# permite interoperabilidade com XML, *Simple Object Access Protocol* (SOAP), componentes COM, DLL e qualquer outra linguagem da plataforma .NET, mantendo integração total com os projetos existentes.

Escrevendo o tradicional programa *Hello World*, confira no Quadro 1.

```
using System;
class Hello
{
        public static void Main()
        {
                Console.WriteLine("Hello World!!!");
        }
 }
```

Quadro 1 – Programa exemplo.

O código do Quadro 1 pode ser digitado em qualquer editor de texto e salvo com o nome de *Hello.cs*. Para compilar o exemplo acima, no *prompt*, basta digitar *csc Hello.cs*, e para executar o programa é necessário digitar *Hello*. Figura 7 ilustra a compilação e execução da aplicação em C#.

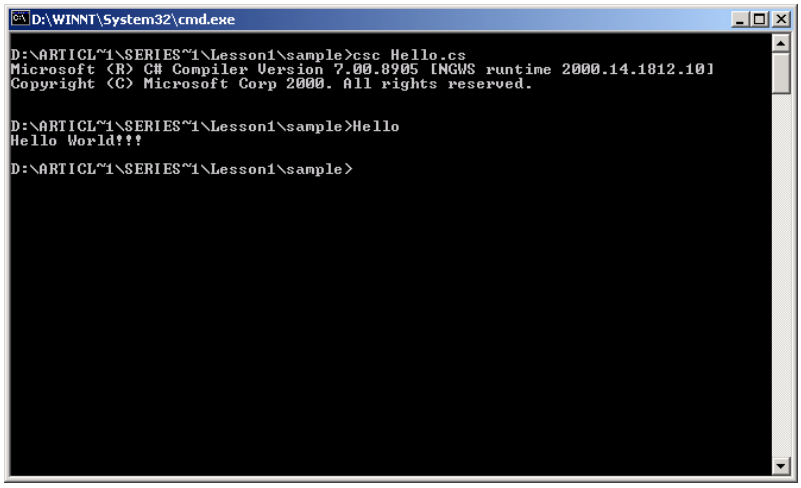


Figura 7 – Compilação e Execução do programa em C#.

Algumas considerações sobre o código do programa. A claúsula *using* referencia as classes a serem utilizadas, *System* atua como *namespace* das classes. O *namespace System* contém muitas classes, uma delas é a *Console*. O método *WriteLine*, simplesmente emite a *string* no *console*.

O método *Main* é o ponto de entrada de execução do programa. Na linguagem C# não existem funções globais, a classe que será executada inicialmente possui embutida a função estática *Main*. Uma ou mais classes podem conter a função *Main*, portanto apenas uma será o ponto de entrada, indicada na compilação pelo parâmetro /*main*:<tipo>, ou simplificando /m:<tipo>.

O método *Main* pode ser declarado de quatro formas:

a) retornando um vazio(*void*): *public static void Main();*

b) retornando um inteiro(*int*): *public static int Main();*

c) recebendo argumentos, através de um *array* de *string* e retornando um vazio: *public static void Main(string[] args);*

d) recebendo argumentos, através de um *array* de *string* e retornando um inteiro: *public static int Main(string[] args).*

Como toda linguagem de programação o C# apresenta seu grupo de tipos de dados básico. Esses tipos são conhecidos como tipos primitivos ou fundamentais por serem suportados diretamente pelo compilador, e serão utilizados durante a codificação na definição de variáveis, parâmetros, declarações e até mesmo em comparações. O Quadro 2 apresenta os tipos básicos da linguagem C# relacionados juntamente com os tipos de dados do .NET *Framework*. Em C#, todos eles possuem um correspondente na CLR, por exemplo *int*, em C#, refere-se a *System.Int32*.

Tipo C#	Tipo .NET	Descrição	Faixa de dados
bool	System.Boolean	Booleano	*true* ou *false*
byte	System.Byte	Inteiro de 8-bit com sinal	-127 a 128
char	System.Char	Caracter Unicode de 16-bit	U+0000 a U+ffff
decimal	System.Decimal	Inteiro de 96-bit com sinal com 28-29 dígitos significativos	$1,0 \times 10^{-28}$ a $7,9 \times 10^{28}$
double	System.Double	Flutuante IEEE 64-bit com 15-16 dígitos significativos	$\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$
float	System.Single	Flutuante IEEE 32-bit com 7 dígitos significativos	$\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$
Int	System.Int32	Inteiro de 32-bit com sinal	-2.147.483.648 a 2.147.483.647
long	System.Int64	Inteiro de 64-bit com sinal	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
Object	System.Object	Classe base	
Sbyte	System.Sbyte	Inteiro de 8-bit sem sinal	0 a 255
Short	System.Int16	Inteiro de 16-bit com sinal	-32,768 a 32,767
String	System.String	String de caracteres Unicode	
Uint	System.UInt32	Inteiro de 32-bit sem sinal	0 a 4,294,967,295
Ulong	System.UInt64	Inteiro de 64-bit sem sinal	0 a 18,446,744,073,709,551,615
Ushort	System.UInt16	Inteiro de 16-bit sem sinal	0 a 65,535

Quadro 2 – Tipos primitivos do C#

Conhecidos como laço ou *loop* (Quadro 3), os comandos de iteração executam repetidamente um comando ou bloco de comandos, a partir de uma determinada condição. Esta condição pode ser pré-definida ou com final em aberto. Em C#, fazem parte dos comandos de iteração: *while*, *do*, *for* e *foreach*.

while
`int i = 0;`
`while ( i < 5 )`
`{`
`  Console.WriteLine ( i );`
`  ++i;`
`}`
	Resultado do laço **while**:
	0
	1
	2
	3
	4

for
`int i = 0;`
`for ( int i = 0; i < 5; i++ )`
`{`
`  Console.WriteLine ( i );`
`}`
	Resultado do laço **For**:
	0
	1
	2
	3
	4

do ... while
`int i = 0;`
`do`
`{`
`  Console.WriteLine ( i );`
`  i++;`
`}`
`while ( i < 5 );`
	O laço **Do/While** é quase igual ao laço **While**.
	A única diferença é que o código dentro do laço será executado pelo menos uma vez pois a seguir é feita a verificação da condição.

foreach
`string[] nomes = new string[] { "Manuel", "José"};`
`foreach (string nome in nomes )`
`{`
`  Console.WriteLine ( nome );`
`}`
	O laço **foreach** pode ser usado para iterar através de uma coleção como um *array, ArrayList, etc.*
	A saída para o laço é:
	Manuel
	José

Quadro 3 – Estruturas de repetições (laços).

O comando *if* (Quadro 4) utiliza uma expressão, ou expressões, *booleana* para executar um comando ou um bloco de comandos. A cláusula *else* é opcional na utilização do *if*, no entanto, seu uso é comum em decisões com duas ou mais opções.

O comando *switch* (Quadro 4) utiliza o valor de uma determina expressão contra uma lista de valores constantes para execução de um ou mais comandos. O valor constante é tratado através da cláusula *case* e este pode ser numérico, caracter e *string*. A cláusula *default* é utilizada para qualquer caso não interceptado pelo *case*.

if ... else
`string nome = "Manuel";`
`if ( nome.Equals("Pedro") )`
`{`
`  Console.WriteLine("Bloco 'if'");`
`}`
`else`
`{`
`  Console.WriteLine("Bloco 'else'" );`
`}`
	Operador usado para testar condições lógicas e executar a porção de código baseado na condição.
	No exemplo se o nome definido for igual a 'Pedro' teremos: Bloco 'if'
	Se o nome definido não for igual a 'Pedro', teremos:
	Bloco 'else'

switch
`int i = 6;`
`switch (i)`
`{`
`  case 6:`
`  Console.WriteLine("6");`
`  break;`
`  case 4:`
`  Console.WriteLine("4");`
`  break;`
`  default:`
`  Console.WriteLine(i);`
`  break;`
`}`
	O comando Switch é uma boa opção se você tiver que escrever muitas condições if..else.
	No exemplo, dependendo do valor do item condicional o código do case apropriado será executado.
	A saída para o exemplo será: 6
	Note que usamos instruções break para sair do bloco, sem isto o próximo bloco Case seria executado.

Quadro 4 – Operadores Condicionais.

2.2.4 Microsoft *Visual Studio* .NET 2003

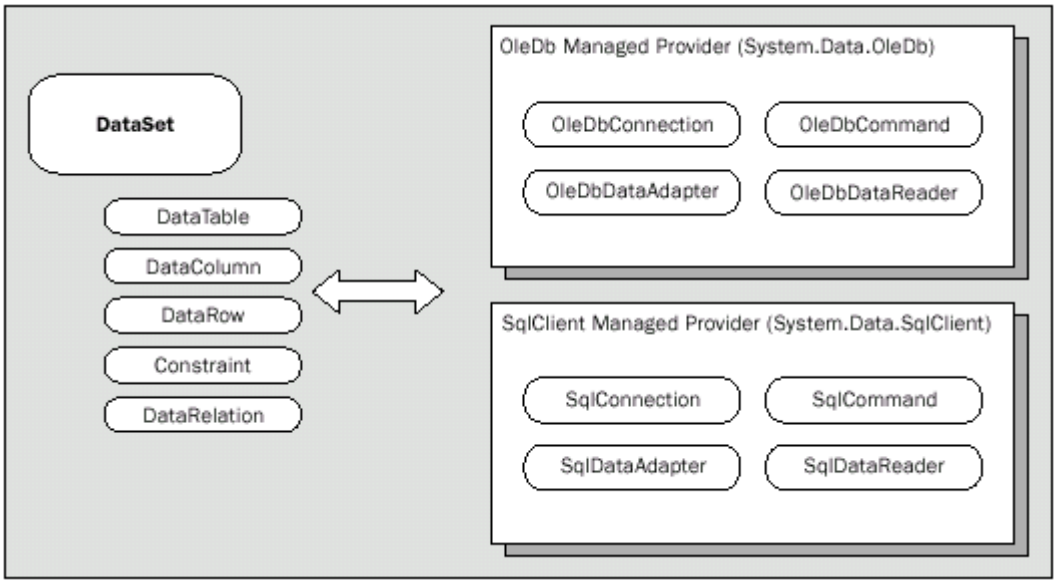
O *Visual Studio* .NET, conhecido como VS.NET é uma ferramenta com objetivo de criar aplicações *desktop*, *web* e para dispositivos móveis. O VS.NET tem uma interface de uma ferramenta *Rapid Application Development* (RAD) onde apenas arrastando, soltando e configurando os controles no formulário, é possível criar a interface da aplicação. Mas, é claro que será preciso inserir códigos para realizar determinadas tarefas. Dentro desse ambiente é possível desenvolver utilizando as linguagens C#, C++, J#, *JScript*, VB.NET e outras linguagens criadas por terceiros que são compatíveis com a CLS.

2.2.5 ADO.NET

ADO.NET é um conjunto de classes do .NET *Framework*, desenvolvidas para facilitar o acesso das aplicações a bases de dados de diversos tipos. ADO.NET não é uma nova versão do *Activex Data Objects* (ADO), é uma nova forma de manipular dados, que foi construído do

zero, sem reaproveitar a tecnologia ADO. Sua única herança é o nome, e mesmo assim somente sua abreviação, pois ADO.NET não significa *Activex Data Objects*.NET e sim, simplesmente, ADO.NET.

Para acessar as funcionalidades do ADO.NET é necessário utilizar componentes para realizar as tarefas pertinentes de acesso a dados. O principal componente da arquitetura ADO.NET é o *DataSet*. O *DataSet* proporciona o acesso a tabelas, linhas, colunas, relacionamentos, *constraints* e pode conter diversas tabelas e relacionamentos. A Figura 8 ilustra a arquitetura do ADO.NET.



Fonte: Macoratti (2002)

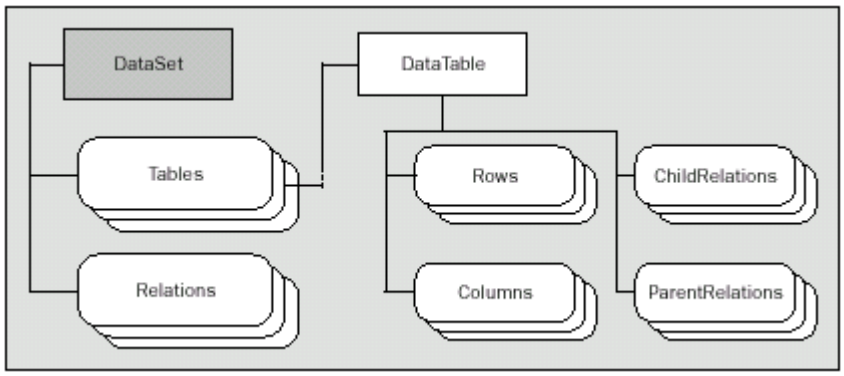
Figura 8 – Visão da arquitetura ADO.NET.

Na Figura 16 no lado direito tem-se os *Data Providers* os quais possuem um conjunto de funcionalidades para acessar os dados armazenados. Ao lado esquerda da Figura 16 é apresentado o *DataSet*, membro do *namespace System.Data* e seus componentes.

System.Data expõe os objetos usados para acessar e armazenar dados, através da criação de dados relacionados, armazenados na memória. Esses objetos são independentes da fonte de dados. O *namespace System.Data* contém: *DataSet*, *DataTable*, *DataRow* e *DataRelation*.

A classe *DataSet* contém toda funcionalidade para gerenciar dados armazenados na

memória através de um *chache* de dados desconectado. A Figura 9 mostra sua estrutura.



Fonte: Macoratti (2002)

Figura 9 – Estrutura do *DataSet*.

É possível notar que o *DataSet* contém, tabelas, relacionamentos entre tabelas e cada tabela contém um conjunto de linhas e colunas.

2.3 INTERNET INFORMATION SERVER

No desenvolvimento do trabalho foi utilizado um computador com o *Windows XP Professional.* Para torná-lo um servidor *web* foi preciso instalar o IIS, que é o serviço responsável pela disponibilização dos serviços *HyperText Transfer Protocol* (HTTP) e *File Transfer Protocol* (FTP). A versão do IIS disponível com o *Windows XP Professional* é a versão 5.1 que tem algumas limitações como:

a) limita-se a 10 conexões simultâneas;

b) não permite a delegação de direitos administrativos a outros usuários, tampouco permite a administração remota do servidor através do *Internet Services Manager* em HTML;

c) não permite definir limites de utilização dos recursos do servidor;

d) os usuários podem acessar o servidor somente pelo nome ou protocolo da Internet (IP);

e) com o *Windows XP Professional*, somente um *site web*, FTP e *Simple Mail Transfer Protocol* (SMTP) é suportado.

O IIS é um servidor *web* da Microsoft que acompanha o *Windows*. O IIS é responsável também pela execução dos aplicativos ASP.NET. É requisito fundamental que o IIS execute em um sistema operacional da Microsoft (NT/2K/XP), a versão do *Windows XP Home* não suporta o IIS.

O IIS é o servidor *web* que os programadores usam ao desenvolver aplicativos ASP.NET no *Visual Studio*. Possivelmente na instalação do *Windows*, o IIS tenha se instalado no seu computador sem sua percepção, portanto, verifique em: Iniciar -> Configurações -> Painel de Controle -> Adicionar ou Remover Programa. Aparecerá uma janela com os programas instalados no seu computador, selecione a opção de Adicionar/Remover componentes do *Windows*. Logo aparecerá uma janela com os componentes do *Windows* instalados no seu computador (os instalados estão marcados), se o componente *Internet Information Services* (IIS) está marcado ele já estará instalado. Caso não esteja marcado é necessário selecionar e o *Windows* pedirá o CD do *Windows XP* para instalação. Para verificar se a instalação ocorreu sem problemas, basta abrir uma janela do *Internet Explorer* e abrir a página *web* a seguir no seu computador: *http://localhost*.

A utilização do *Windows XP Professional* como servidor limita-se às pequenas empresas e desenvolvedores que precisam testar suas páginas *web* antes de colocá-las no servidor de produção.

2.4 MICROSOFT *SQL SERVER* 2000

O *SQL Server* é usado para gerenciar dois tipos de banco de dados, bancos de dados de processamento de transações *on-line* (OLTP) e bancos de dados de processamento analítico

on-line (OLAP).

O *SQL Server* é um sistema de gerenciamento de bancos de dados relacionais (*Relational Database Management System* - RDBMS), que (MICROSOFT CORPORATION, 2000):

a) gerencia o armazenamento de dados para transações e análises;

b) responde às solicitações dos aplicativos cliente;

c) usa *Transact-SQL*, linguagem de marcação extensível, expressões multidimensionais para enviar solicitações entre um cliente e o *SQL Server*.

2.5 GERAÇÃO DE CÓDIGO

Conforme Martin e Mcclure (1991, p. 424), os geradores de código são ferramentas que produzem código sem nenhum erro de sintaxe a partir de projetos, gráficos e especificações de alto nível. O código deve ser gerado a partir de tabelas de decisão, regras, diagramas de ação, diagramas de eventos, diagramas de transição de estado, representação de objetos, suas propriedades e relacionamentos, e assim por diante.

> Programas geradores de código são extremamente práticos quando se tem pouco tempo ou uma equipe pequena para desenvolver. Esta afirmação é verdadeira quando se imagina um aplicativo complexo quanto ao uso, que gere o código completo sem a necessidade de personalizações e fornece uma série de opções para a escolha da linguagem de programação ou do banco de dados que se deseja utilizar. (SANTOS, 2002, p. 1).

Benefícios da geração de código para o programador segundo Herrington (2003) são:

a) qualidade: evita grande quantidade de código escrito a mão, onde poderia ter códigos inconsistentes ou sem necessidade;

b) consistência: o código gerado pelo gerador é consistente conforme definição da sua *Application Program Interface* (API). Sem surpresa nos resultados, interface fácil de usar e entender;

c) único ponto de conhecimento: uma mudança do nome da tabela envolveria mudanças manuais no programa, na documentação, e na camada de teste. Uma arquitetura da geração do código permite que você mude um nome da tabela em uma única posição e regenere então o programa, a documentação, e a camada de teste;

d) projetos mais detalhados: a geração de código comprime o tempo de programação em determinados sistemas, assim mais tempo pode ser gasto fazendo o projeto do sistema evitando o retrabalho.

Muitas das vantagens para o programador podem ser importantes para o administrador, para o crescimento da produtividade e da qualidade. Há certos aspectos, portanto que são importantes unicamente em nível de negócios. Segundo Herrington (2003) as vantagens são:

a) consistência arquitetural: código gerado é bem documentado, mesmo que os membros saiam do projeto, o programa será de fácil entendimento;

b) abstração: a lógica do projeto é independente da linguagem, fica a critério dos programadores desenvolver novos *templates* que traduz a lógica do projeto em outras linguagens, analistas de negócios podem revisar e validar a lógica do projeto;

c) maior produtividade: a geração de código reduz a quantidade de horas gastas no desenvolvimento do projeto e mantém os programadores centrados em um único trabalho ao invés de grandes volumes de código.

d) ágil desenvolvimento: o *software* será fácil de modificar, caso ocorra alguma alteração basta gerar o código do projeto novamente.

2.6 FERRAMENTAS PARA GERAÇÃO DE CÓDIGO

O *CodeCharge* e o *DBDesigner* são exemplos de ferramentas de geração de código disponíveis no mercado. A seguir são apresentadas características sobre as mesmas.

2.6.1 *CodeCharge*

Conforme Santos (2002), o *CodeCharge* (Figura 10) é uma ferramenta *Rapid Application Development* (RAD) desenvolvida pela empresa norte-americana *Yes Software Corporation* que integra a função de geração de código para aplicativos de acesso a banco de dados via *web*, em um ambiente *Integrated Development Environment* (IDE) bastante simples. É possível criar *sites web* nas linguagens ASP, JSP, PHP, *Perl*, *ColdFusion*, entre outras.

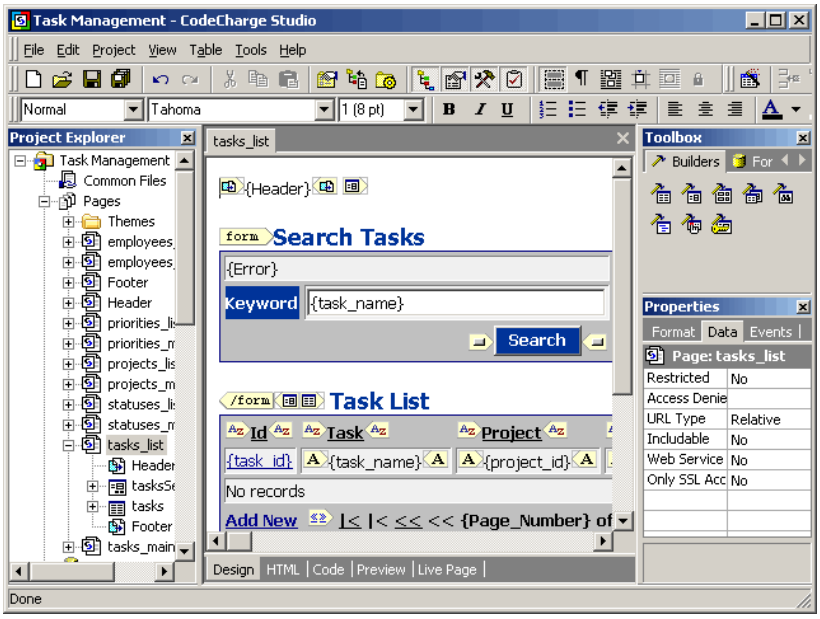


Figura 10 – IDE *CodeCharge*.

O *CodeCharge* suporta o uso de código *Structured Query Language* (SQL) customizado, além do próprio código gerado pela aplicação, através do uso de formulários de tabelas, de registros, *login*, pesquisas e menus. O desenvolvedor pode especificar através

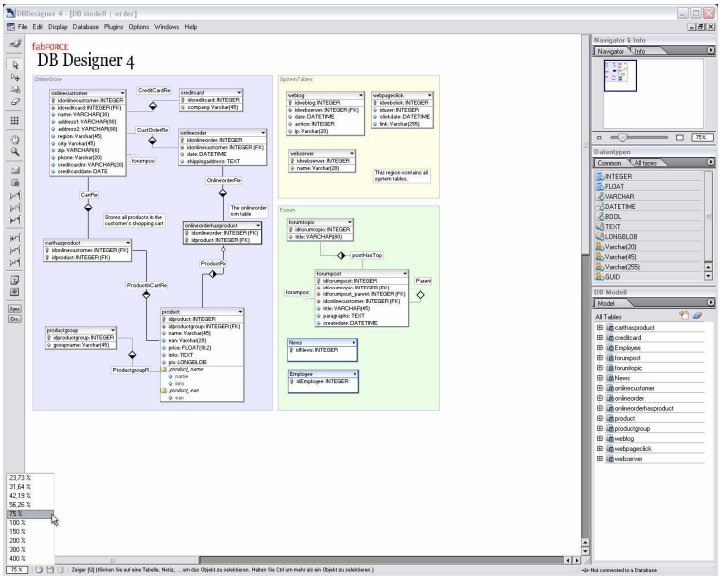
desses formulários, a ação baseada em funções SQL e o que mostrar quando o banco de dados for acessado, além de determinar os níveis de autenticação de usuário na aplicação.

Além disso, possui funcionalidades que auxiliam no trabalho de *design*, como uma interface para a geração de arquivos *Cascading Style Sheet* (CSS) e estilos HTML ou pode optar por *templates*.

O *CodeCharge* permite a conexão com qualquer banco de dados relacional, incluindo *Oracle*, *Sybase*, *MySQL*, Microsoft *SQL Server* e Microsoft *Access* que sejam passíveis de conexão via *Joint Engine Tecnology* (JET), *Open DataBase Connectivity* (ODBC), *Java DataBase Connectivity* (JDBC), *ActiveX Data objects* (ADO) e *PHPLib*. Os códigos gerados são portáveis em qualquer versão do *Windows*, *Unix* e *Linux*.

2.6.2 *DBDesigner*

É um editor visual para criação de projetos de banco de dados MySQL que integra criação, modelagem, desenvolvimento e manutenção dos bancos. O *DBDesigner* (Figura 11) é *open source* distribuído sobre a licença *General Public License* (GPL).



Fonte: Fabforce.Net (2003).

Figura 11 – IDE *DBDesigner*.

De acordo com Fabforce.Net (2003), no *DBDesigner* o foco é um modelo de dados. Um modelo de dados é uma visualização da meta-informação armazenada em uma base de dados (por exemplo, tabelas e índices, relacionamentos). Embora seja possível armazenar dados iniciais para cada tabela diretamente no modelo, é representada somente a meta-informação e não os próprios dados.

Um objeto pode ser uma tabela da base de dados com colunas e índices ou uma relação entre duas tabelas. Os novos modelos podem ser construídos colocando estes objetos no modelo ou podem ser recuperados das bases de dados existentes usando uma função da engenharia reversa. Pode ser criada uma base de dados exportando um modelo através de uma sentença SQL. O *DBDesigner* possui uma função para sincronizar as alterações na base de dados quando o modelo for alterado.

2.7 TRABALHOS CORRELATOS

O foco principal do trabalho desenvolvido e relatado em Dias (2002) foi fornecer uma ferramenta que auxiliasse na criação de páginas para atualização de banco de dados utilizando *Active Server Pages* (ASP), compatível com os banco de dados *Oracle*, *SQL Server* e conexões *Open Database Connectivity* (ODBC). A geração das páginas ASP é efetuada através de um *template* pré-definido, sendo que as principais páginas são: pesquisa, edição e criação de registros em um banco de dados.

Em Silveira (2003) é relatado o desenvolvimento de uma ferramenta de auxílio ao programador para criação automática de um sistema de cadastros e consultas na plataforma *web*, utilizando linguagem ASP. Baseado na estrutura de campos e relacionamentos da própria base de dados. As páginas geradas permitem a manipulação dos dados da base de dados, assim como a consulta.

Castilhos (2004) descreve o desenvolvimento de uma ferramenta CASE para criação, definição, documentação e geração de páginas ASP automaticamente. Fazendo a criação das tabelas, relacionamentos e a geração dos cadastros com inclusão, exclusão, alteração, consulta, paginação e barra de navegação, utilizando a linguagem ASP, baseado na estrutura dos campos relacionados na própria ferramenta CASE.

A ferramenta desenvolvida por DIAS (2002) cria um ambiente de atualização de dados na *web*. A ferramente analizada foi fonte de inspiração para desenvolvimento deste trabalho. Elas funcionam de formas distintas, mas foram utilizados os conceitos de configurar fonte de dados, seleção da tabela e configuração das colunas.

3 DESENVOLVIMENTO DO GERADOR DE CÓDIGO

Neste capítulo é apresentada a especificação e o desenvolvimento do aplicativo de geração de páginas para tecnologia ASP.NET.

3.1 REQUISITOS DO SISTEMA

O Quadro 5 apresenta os requisitos funcionais para o gerador de código, criados a partir dos trabalhos correlatos e das necessidades que o gerador proposto requeria.

Requisitos Funcionais
RF01: Configurar conexão com o banco de dados SQL Server 2000.
RF02: Listar databases, ler o dicionário de dados do SQL Server 2000 e identificar a estrutura de tabelas, campos e relacionamentos.
RF03: Cadastrar página, especificar o tipo de acesso que terá cada tabela (consulta, cadastro, alteração, exclusão).
RF04: Visualizar dados, seleção de colunas e validação de campos.
RF05: Configurar relacionamento entre colunas.
RF06: Criar código, gerar páginas web a partir das tabelas selecionadas, com campos dinâmicos conforme tipo de dados.

Quadro 5 – Requisitos funcionais.

Os requisitos não-funcionais declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades, o Quadro 6 lista os requisitos não funcionais previstos para o sistema.

Requisitos Não Funcionais
RNF01: Utilizar a linguagem de programação C# para criação da ferramenta e das páginas web.
RNF02: Visualizar as páginas no navegador Internet Explorer.

RNF03: As páginas geradas devem ser executadas em um computador com o IIS previamente instalado.

Quadro 6 – Requisitos não funcionais.

3.2 ESPECIFICAÇÃO

A especificação foi desenvolvida na ferramenta CASE *Enterprise Architect*. Utilizaram-se os diagramas da UML para demonstrar os casos de uso, classes e atividades. A ferramenta foi implementada utilizando a ferramenta RAD Microsoft *Visual Studio* .NET.

A Figura 12 mostra o diagrama de casos de uso do sistema, visto do ponto de vista do desenvolvedor, que irá gerar as páginas dinâmicas.

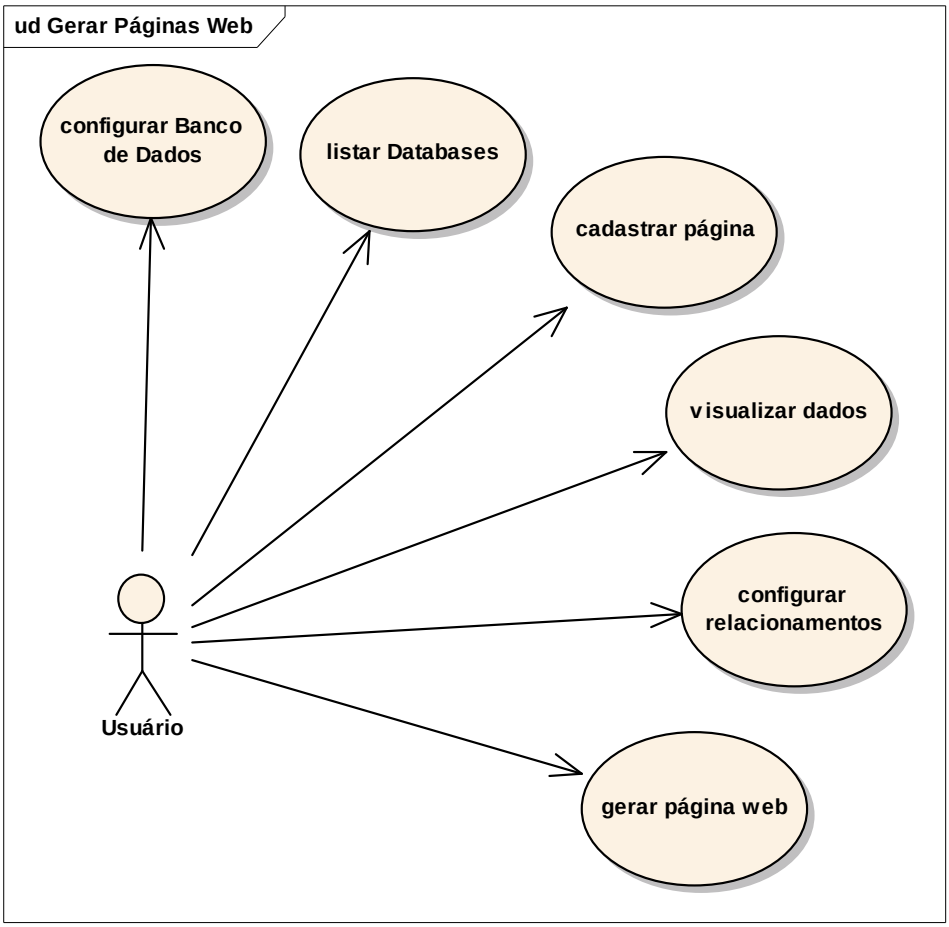


Figura 12 – Diagrama de Caso de Uso do sistema.

No Quadro 7 encontra-se a descrição do caso de uso configurar banco de dados.

Configurar banco de dados
Sumário: O usuário inicia abrindo um projeto salvo previamente ou criando um novo. Para dar início a um novo projeto é necessário informar os dados para conexão com o banco de dados. **Ator Principal:** Usuário **Precondições:** O usuário deve ter permissão para acesso ao banco de dados e deve existir uma *database* que não seje as utilizadas pelo SQL Server para seu controle. **Fluxo Principal:** **a)** o usuário informa Login, Senha e Host do banco de dados e clica em conectar; **b)** o sistema retorna lista de databases se conexão foi efetuada com sucesso.

Quadro 7 – Caso de Uso configurar banco de dados.

No Quadro 8 encontra-se a descrição do caso de uso listar *databases*.

Listar *databases*
Sumário: O usuário tem como obrigação selecionar uma *database* para continuar o processo. **Ator Principal:** Sistema **Precondições:** O sistema ter retornado lista de databases. **Fluxo Principal:** **a)** o usuário define *database*; **b)** o sistema encaminha usuário para guia de configurações.

Quadro 8 – Caso de Uso listar *databases*.

No Quadro 9 encontra-se a descrição do caso de uso cadastrar páginas.

Cadastrar página

Sumário: O usuário deve configurar nome, descrição, número de registros e as funcionalidades (incluir, alterar, excluir e pesquisar) das páginas *web*.

Ator Principal: Usuário

Precondições: Ter selecionado um *database*.

Fluxo Principal:

a) o usuário define tabela;

b) o usuário pode informar nome e descrição para página;

c) o usuário define a quantidade de registro na página de consulta;

d) o usuário define as funcionalidades que a página terá;

e) o usuário clica em Adicionar para manter os dados informados;

f) o sistema grava os dados informado pelo usuário na memória;

g) após usuário informar as tabelas que deseja gerar página *web* clica em Avançar;

h) o sistema busca no SGBD as colunas relacionadas com as tabelas informadas e encaminha usuário para guia de visualização.

Quadro 9 – Caso de Uso cadastrar página.

No Quadro 10 encontra-se a descrição do caso de uso visualizar dados.

Visualizar dados

Sumário: O usuário pode configurar o nome da coluna que irá aparecer na página de consulta, inclusão e alteração. Pode ocultar uma coluna da página de consulta, selecionar coluna como campo obrigatório nas páginas de inclusão e alteração e ainda pode informar uma validação para coluna desejada.

Ator Principal: Usuário

Precondições: Ter selecionado pelo menos uma tabela.

Fluxo Principal:

a) o sistema retorna lista de todas as colunas das tabelas selecionadas pelo usuário;

b) o usuário pode filtrar por uma tabela específica;

c) o usuário define nome da coluna;

d) o usuário define se a coluna estará ou não visível na página de consulta;

e) o usuário define se o campo (coluna) é obrigatório;

f) o usuário define validação do campo (coluna);

g) após configuração usuário clica no botão Avançar;

h) o sistema encaminha usuário para guia relacionamentos.

Quadro 10 – Caso de Uso visualizar dados.

No Quadro 11 encontra-se a descrição do caso de uso configurar relacionamentos.

Configurar relacionamentos

Sumário: O usuário tem a opção de relacionar uma coluna com uma lista de valores ou com uma outra coluna de outra tabela da mesma *database*.

Ator Principal: Usuário

Precondições: Ter selecionado pelo menos uma tabela e que essa tabela tenha alguma coluna.

Fluxo Principal:

a) o usuário seleciona uma tabela;

b) o usuário seleciona uma coluna;

c) o usuário seleciona relacionamento com outra tabela;

d) o sistema habilita *combobox* de tabela e colunas estrangeiras e desabilita campo de lista de valores.

e) o sistema retorna do SBGD tabelas e colunas para relacionamento;

f) o usuário define tabela, coluna para o atributo *value* do *combobox* e coluna para descrição do *combobox*;

g) o usuário clica no botão Salvar Coluna para gravar configuração;

h) o sistema armazena informações na memória;

i) o usuário clica em Avançar;

j) o sistema encaminha usuário para guia geração.

Fluxo Alternativo:

c) o usuário seleciona relacionamento com domínio;

d) o sistema habilita campo de lista de valores e desabilita *combobox* de tabela e colunas estrangeiras.

e) o usuário informa valores do domínio;

f) o usuário clica no botão Salvar Coluna para gravar configuração;

g) o sistema armazena informações na memória;

h) o usuário clica em Avançar;

i) o sistema encaminha usuário para guia geração.

Quadro 11 – Caso de Uso configurar relacionamentos.

No Quadro 12 encontra-se a descrição do caso de uso gerar página *web*.

Gerar página web

Sumário: O usuário deve selecionar um diretório virtual para geração das páginas *web*.

Ator Principal: Sistema

Precondições: Ter selecionado pelo menos uma tabela.

Fluxo Principal:

a) o usuário define diretório virtual para geração do código;

b) o usuário clica no botão Gerar página ASP.NET;

c) o sistema cria as páginas conforme configuração do usuário;

d) após terminar geração sistema retorna mensagem de sucesso ou erro.

Quadro 12 – Caso de Uso gerar página *web*.

A Figura 13 ilustra o diagrama de classes da ferramenta desenvolvida.

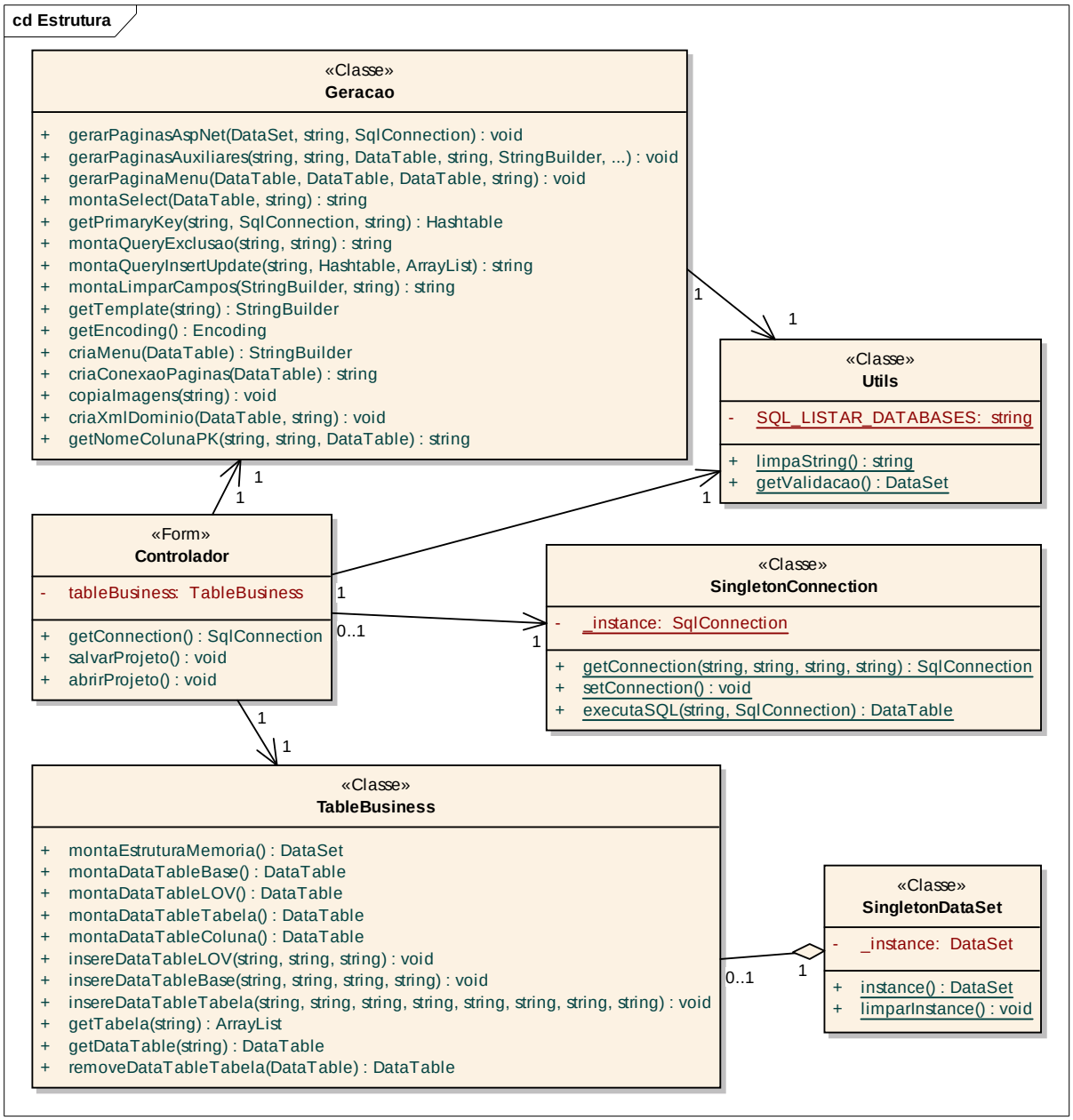


Figura 13 – Diagrama de Classes.

A classe *Controlador* é responsável por tratar os eventos do form e por chamar os métodos das demais classes.

A classe *Utils* é onde estão armazenados métodos diversos para manter o código limpo e organizado.

A classe *SingletonDataSet* é responsável por manter apenas uma instância do objeto *dataset* na memória.

A classe *SingletonConnection* é responsável por manter apenas uma instância do objeto *sqlconnection* na memória.

A classe *TableBusiness* cria a estrutura do objeto *dataset*. São definidas as colunas da *datatable base*, *datatable* tabela, *datatable* coluna e da *datatable lov*. Também é responsável por incluir os registros nas *datatables* que estão na memória.

A classe *Geracao* como o próprio nome já definiu é responsável por toda a geração das páginas ASP.NET.

A Figura 14 tem duas separações, tempo de projeto e tempo de execução. No tempo de projeto são definidos todos os detalhes das páginas que serão geradas. No tempo de execução está a aplicação final (páginas *web* ASP.NET). Entre essas duas partes fica o banco de dados. Todas as tabelas que o usuário selecionar para geração ficarão armazenadas na memória dentro do *DataSet*, dentro do *DataSet* estão também todos os detalhes sobre as tabelas selecionadas e as colunas das tabelas com suas cofigurações, ao gerar as páginas *web* não é mais necessário a utilização do banco de dados apenas os dados do *DataSet*.

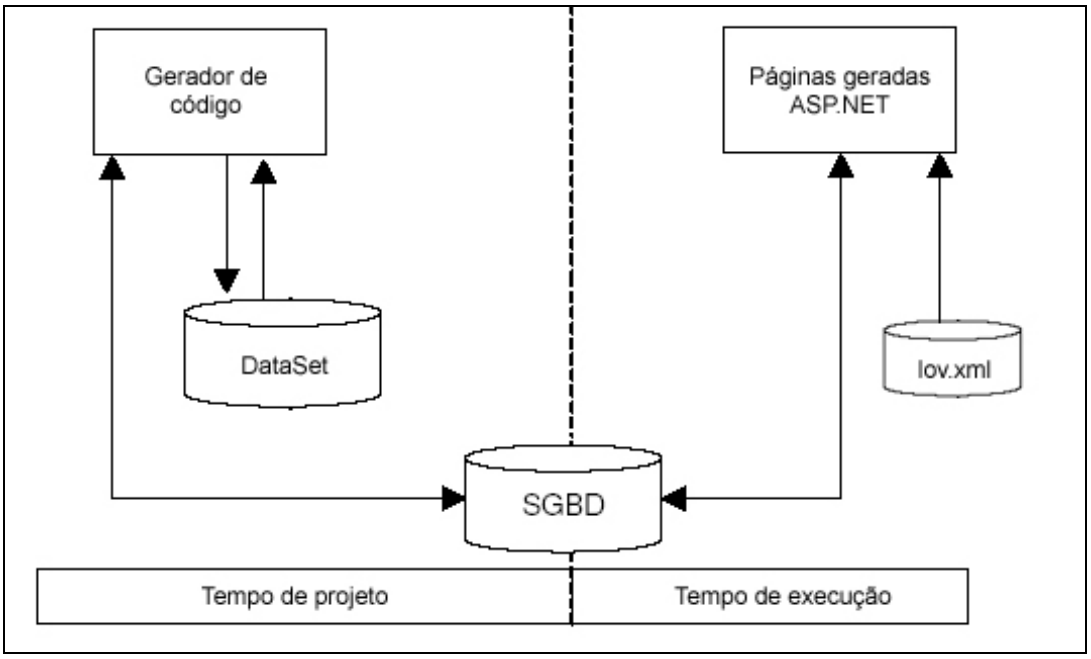


Figura 14 – Arquitetura do gerador.

A seguir são apresentados os diagramas de seqüência dos métodos principais da

ferramenta. A Figura 15 apresenta o diagrama de seqüência do método “inserirDataTableBase”, o qual é responsável por incluir dados no *DataTable dtBase*, que é o objeto responsável por armazenar dados sobre a conexão com o banco de dados, esse objeto está dentro do *DataSet* mostrado na Figura 14.

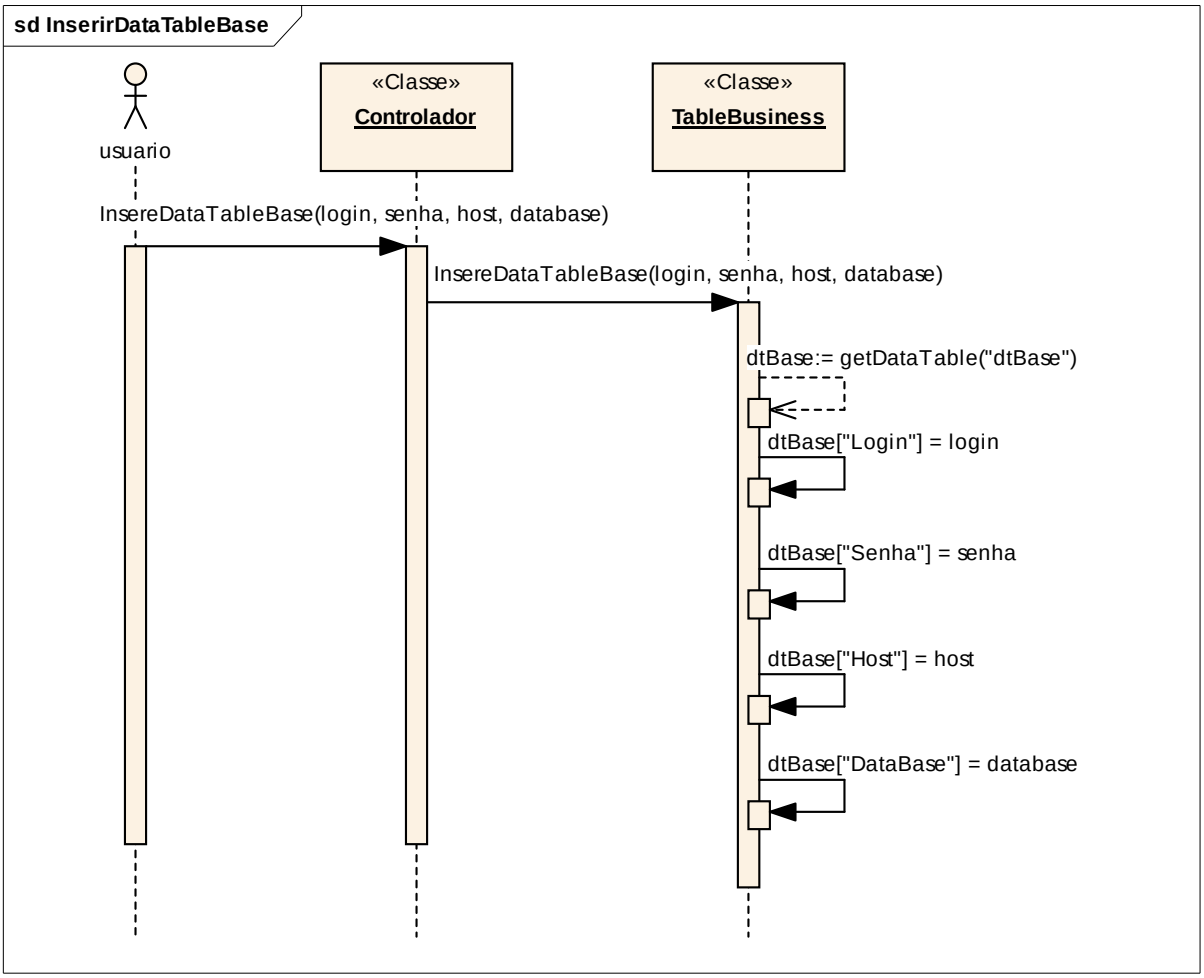


Figura 15 – Diagrama de Seqüência: inserirDataTableBase.

A Figura 16 apresenta o diagrama de seqüência do método “inserirDataTableTabela”, o qual é responsável por incluir dados no *DataTable dtTabela*.

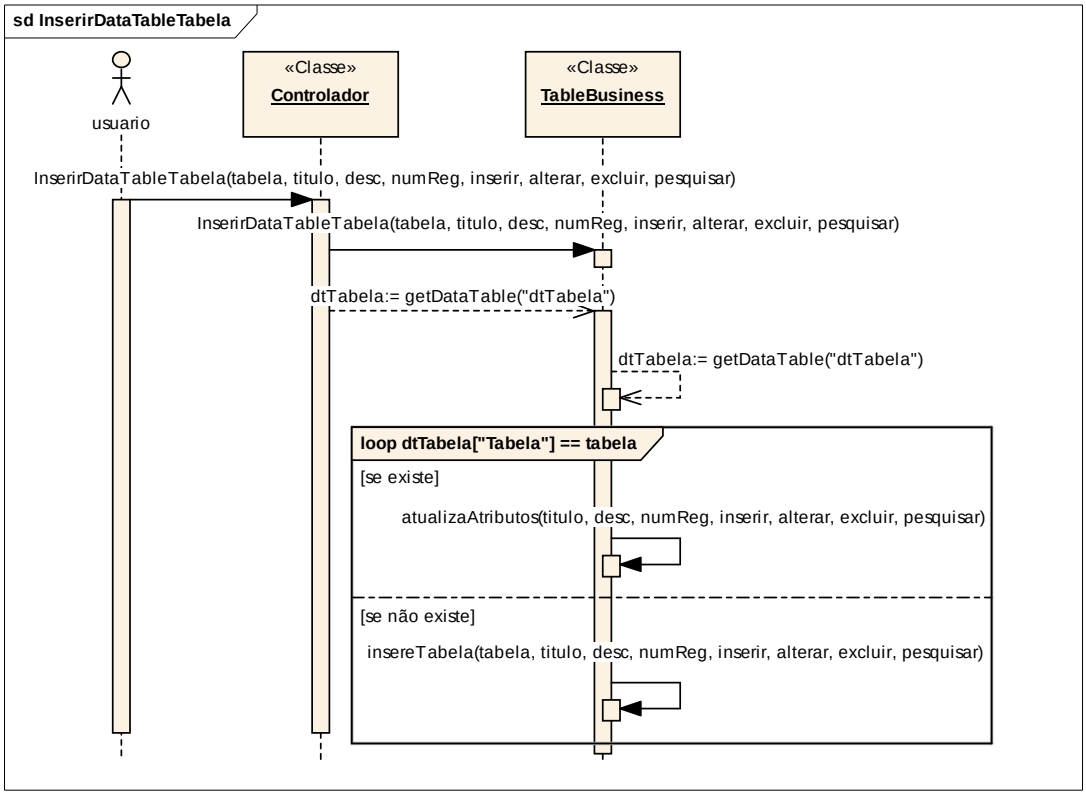


Figura 16 – Diagrama de Seqüência: insereDataTableTabela.

A Figura 17 apresenta o diagrama de seqüência do método “removeDataTableTabela”, o qual é responsável por remover dados no *DataTable dtTabela*.

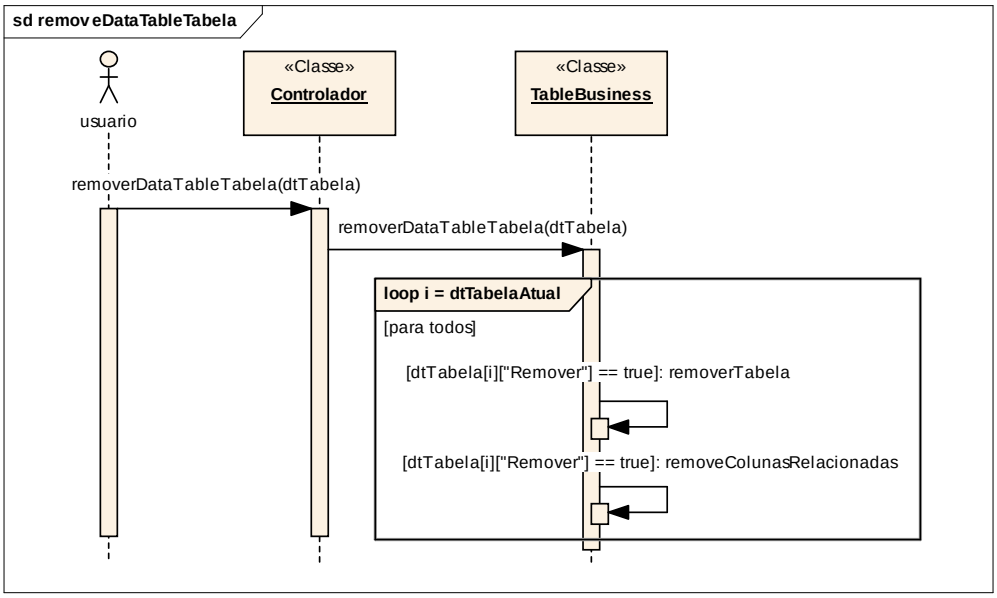


Figura 17 – Diagrama de Seqüência: removeDataTableTabela.

A Figura 18 apresenta o diagrama de seqüência do método “inserirDataTableColuna”, o qual é responsável por incluir dados no *DataTable dtColuna*.

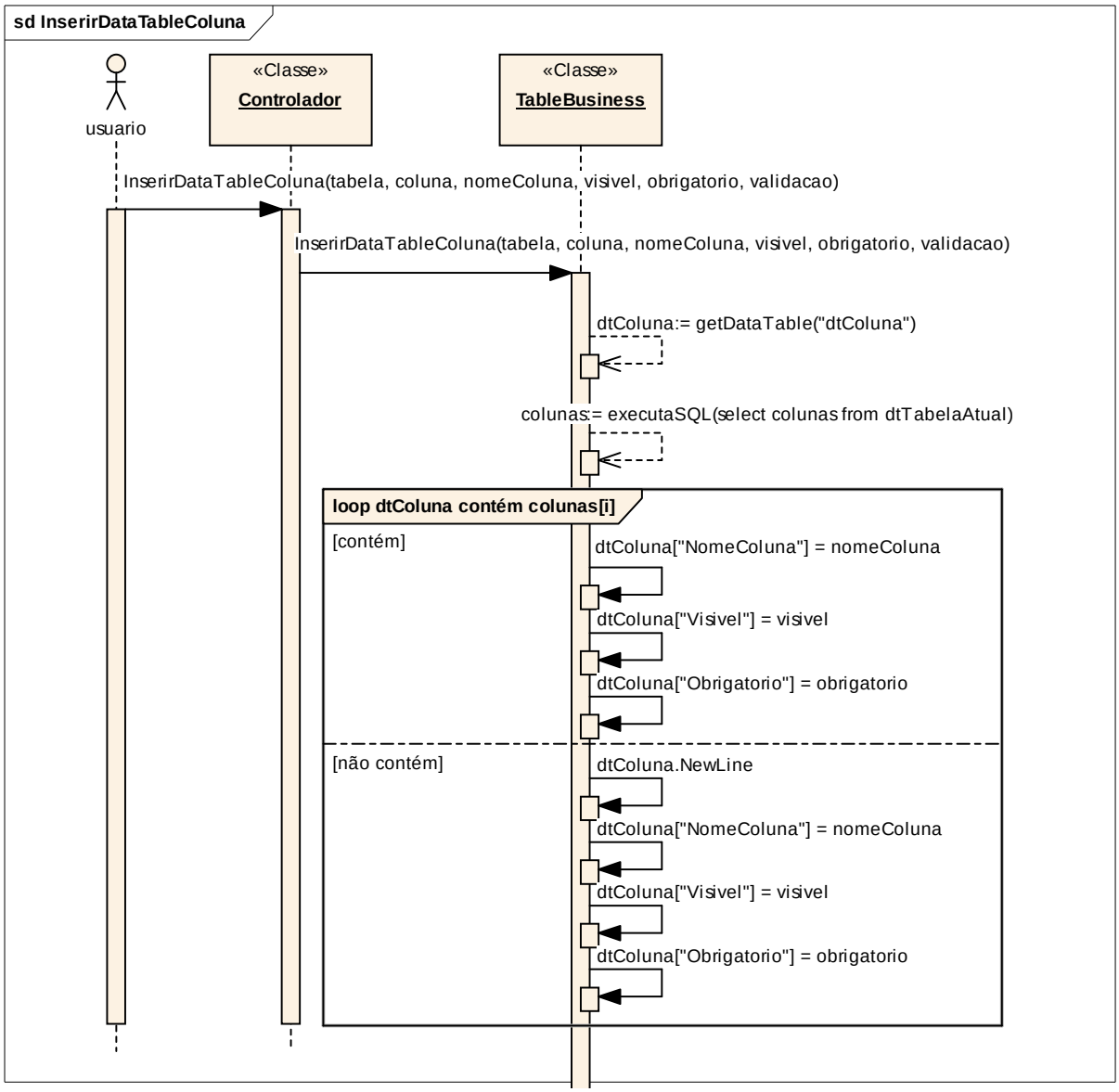


Figura 18 – Diagrama de Seqüência: inserirDataTableColuna.

A Figura 19 apresenta o diagrama de seqüência do processo de geração das páginas *web*. O qual é responsável por gerar a tela de menu, consulta, alteração, inclusão e pesquisa.

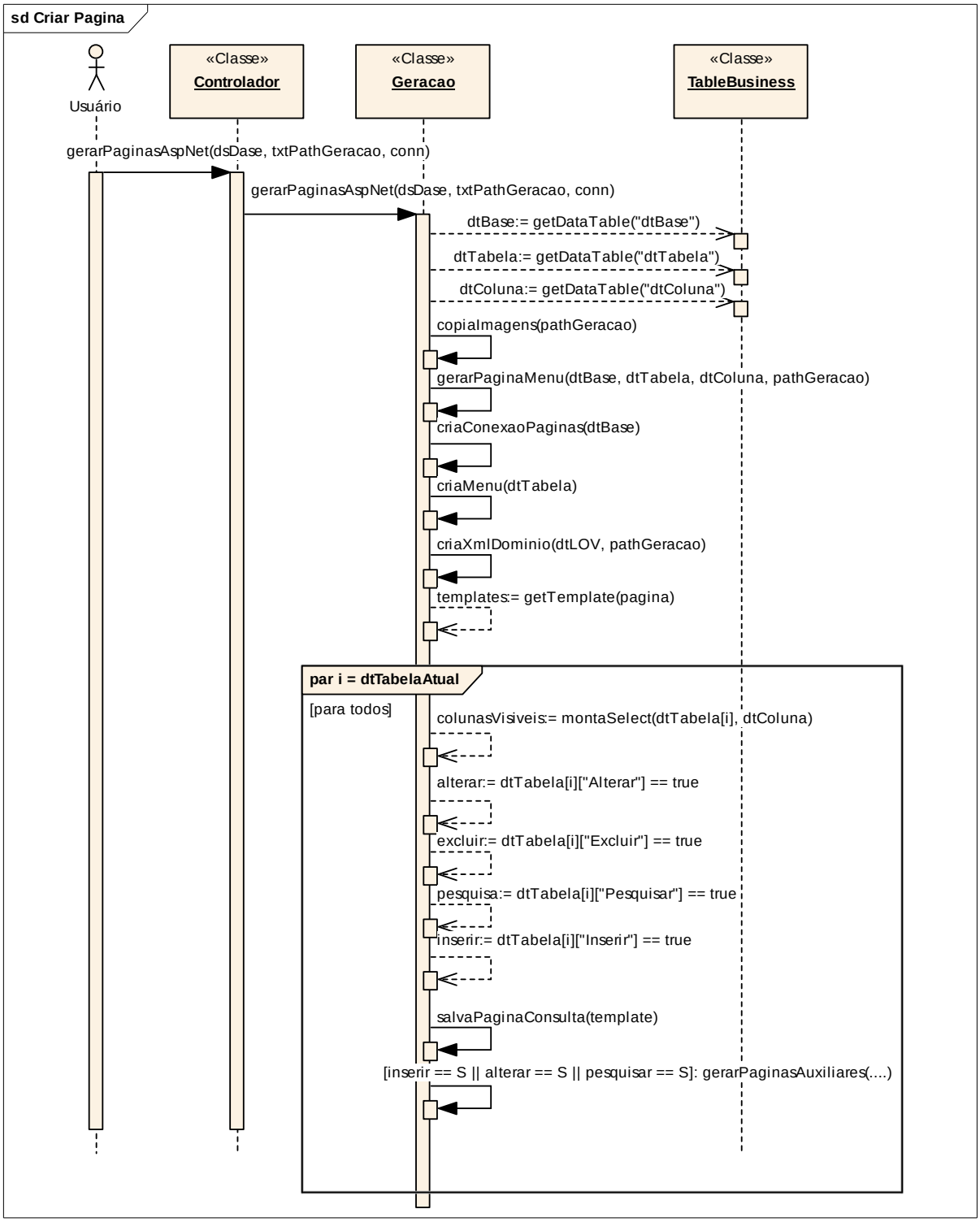


Figura 19 – Diagrama de Seqüência da geração das páginas.

A Figura 20 ilustra passo a passo as etapas que devem ser seguidas para a geração das páginas *web*.

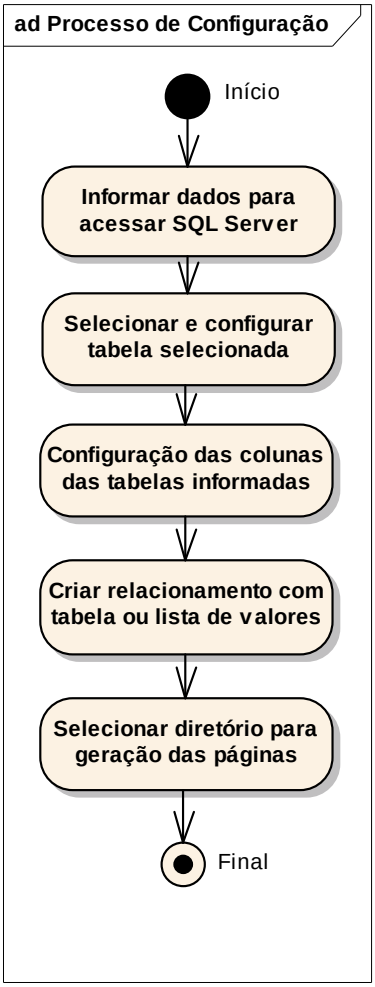


Figura 20 – Etapas da configuração.

3.3 IMPLEMENTAÇÃO

A seguir é apresentado como foi desenvolvida a ferramenta, como foi implementado o acesso ao dicionário de dados do Microsoft *SQL Server* 2000, como funciona a gravação do projeto, como foi desenvolvida a geração do código e recursos das páginas *web* geradas.

3.3.1 Acesso ao dicionário de dados do *SQL Server*

Quando é criada uma tabela no Microsoft *SQL Server*, com suas colunas, índices, tipos de dados, etc., estas informações são armazenadas no metadados. O metadados é um conjunto de tabelas que armazenam informações sobre os objetos que o usuário criou no banco de dados, como definição de tabela, código fonte de uma *Stored Procedure* ou definição de uma *view*.

Por exemplo, quando utilizá-se a instrução (Quadro 13) no Microsoft *SQL Server*, o *SQL Server* utiliza uma tabela de sistema chamada *syscolumns*, que contém todos os campos de todas as tabelas dentro de um banco de dados e verifica quais são os campos da TABELA1 que devem ser retornados para a instrução. Estes dados que estão nestas tabelas especiais são chamados metadados.

```
SELECT * FROM TABELA1
```

Quadro 13 – Instrução SQL.

Existem basicamente três maneiras de retornar metadados no Microsoft *SQL Server* que são:

a) através de tabelas de sistema;

b) através de funções do *SQL Server*;

c) através de *schema views*.

Os metadados são armazenados em tabelas de sistema. Todas as tabelas de sistema começam pelo prefixo *sys* e em hipótese alguma devem ser modificadas, pois caso alguma coisa errada ocorra com elas, o banco de dados inteiro pode deixar de funcionar. A Microsoft não recomeda o acesso direto (isto é, dar um *SELECT*) a estas tabelas, pois elas podem mudar de nome nas próximas versões do produto, tornando seu código inválido e sem nenhuma escalabilidade.

Segue (Quadro 14) algumas tabelas de sistema e um breve comentário sobre cada uma.

Sysobjects	Armazena informações sobre todos os objetos do banco de dados.
Sysindexes	Armazena informações específicas sobre índices do banco de dados.
Syscolumns	Armazena todas as informações sobre todas as tabelas do banco de dados.
Syscomments	Armazena o código fonte de *stored procedures* e funções do banco de dados.
Syslocks	Armazena informações sobre os *locks* dos objetos dos bancos de dados, existe somente no banco de dados *master*.
Sysdatabases	Armazena informações sobre os bancos de dados do servidor *SQL Server*, existe somente no banco de dados *master*.

Quadro 14 – Tabelas de sistema do Microsoft *SQL Server*.

Uma outra maneira e mais segura de se obter metadados é utilizando algumas funções já prontas do Microsoft *SQL Server* para acessar os dados. Estas funções só foram implementadas a partir do Microsoft *SQL Server* 7.0.

No Quadro 15, há duas funções para demonstração. Primeiro a função *OBJECT_ID()* que retorna um identificador interno do *SQL Server* para um objeto e depois a função *OBJECTPROPERTY()* para retornar se o objeto é uma tabela ou não.

```
SELECT OBJECTPROPERTY(OBJECT_ID('TABELA1'),'isTable')
```

Quadro 15 – Funções SQL.

O retorno da função depende de qual propriedade do objeto se está consultando. Neste caso, a propriedade chama-se '*isTable*' e a fução *OBJECTPROPERTY()* retorna o valor 1 se o objeto chamado TABELA1 for uma tabela, 0 se não for uma tabela e *NULL* se o objeto não existir no banco de dados atual.

O último método de acesso a metadados é o mais recomendado pela Microsoft e foi o método utilizado no desenvolvimento da ferramenta. O usuário irá consultar algumas *views* (instruções SQL pré-compiladas) que encapsulam o uso das tabelas de sistema. Estas *Schema*

Views só foram implementadas a partir do Microsoft *SQL Server* 7.0.

O Quadro 16 contem algumas instruções SQL utilizadas no desenvolvimento da ferramenta.

Instrução utilizada após o usuário informar Login, Senha e Host na ferramenta, retornar o nome das databases do servidor, menos as database de sistema, utilizadas pelo próprio SQL Server.

```
select catalog_name from INFORMATION_SCHEMA.SCHEMATA where catalog_name not in ('master', 'tempdb', 'model', 'msdb')
```

Instrução utilizada para listar todas as tabelas de uma database. É executada após o usuário selecionar uma database na ferramenta, retorna o nome da database e o nome da tabela.

```
select table_catalog, table_name from INFORMATION_SCHEMA.TABLES where table_type = 'BASE TABLE' and table_catalog = 'northwind'
```

Instrução utilizada para listar todas as colunas de uma tabela. É executada após o usuário selecionar as tabelas desejadas para geração, retorna o nome da database, nome da tabela e informações da coluna.

```
select table_catalog, table_name, column_name, ordinal_position, is_nullable, data_type from INFORMATION_SCHEMA.COLUMNS where table_catalog = 'northwind' and table_name = 'products'
```

Quadro 16 – Instruções SQL *Schema Views*.

3.3.2 Armazenando dados e persistência do projeto

Conforme esclarecido na especificação foram utilizadas classes do ADO.NET para manipulação dos dados, o objeto *DataSet* veio para substituir com vantagens o objeto *recordset* do ADO e guarda poucas similaridades. Enquanto o objeto *recordset* representa uma coleção de tabelas de dados o objeto *DataSet* representa uma cópia do banco de dados em memória. O *DataSet* está relacionado com tabelas (*Tables*) e relacionamentos (*Relations*) as tabelas contidas no *DataSet* pertence à classe *DataTable*. A Figura 21 representa os objetos criados na classe *TableBusiness* e *SingletonDataSet* conforme diagrama de classe apresentado

anteriormente, são nesses objetos que são armazenadas as informações das páginas que serão geradas com as características que o usuário escolheu na ferramenta.

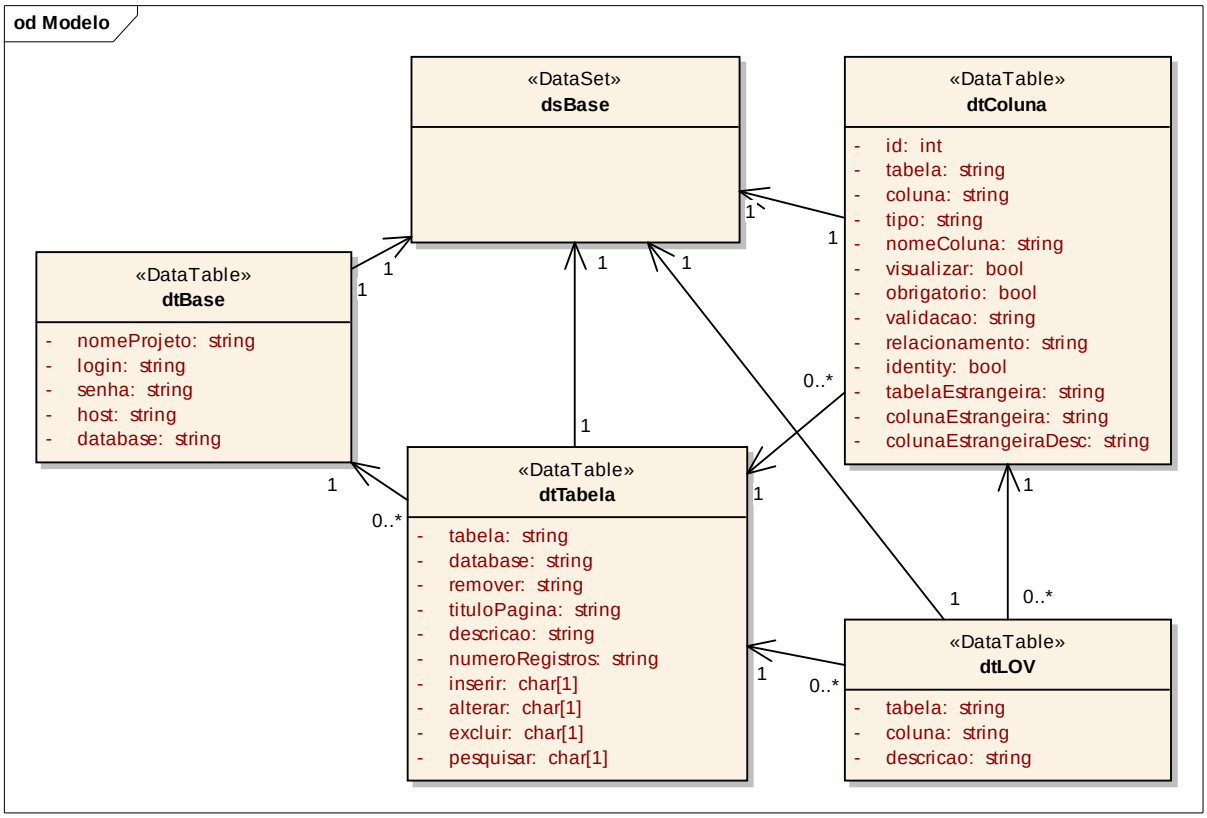


Figura 21 – Objetos criados na memória.

Os objetos da Figura 21 *dtBase*, *dtTabela*, *dtColuna* e *dtLOV*, são do tipo *DataTable* e estão agrupados dentro do objeto *dsBase* que é do tipo *DataSet*.

O objeto *dtBase* é responsável pelas informações de acesso ao banco de dados e pelas informações da *database* selecionada.

O objeto *dtTabela* é onde estão armazenadas as tabelas que serão geradas, cada tabela tem sua própria configuração.

O objeto *dtColuna* é onde estão armazenadas as colunas das tabelas que serão geradas, cada coluna também tem sua própria configuração.

O objeto *dtLOV* é onde estão armazenadas as listas de valores que serão utilizadas nas páginas geradas.

Todos os objetos apresentados na Figura 18 são mapeados implicitamente pelos

recursos avançados da tecnologia ADO.NET.

No Quadro 17 é possível visualizar na prática como está implementado, tem-se sempre apenas um *DataSet* na memória, foi utilizado o padrão de projeto *singleton* para criá-lo, em seguida é criado a estrutura de cada *DataTable* e adicionado ao *DataSet* que está na memória.

```
/// <summary>
/// Criar a estrutura do DataSet
/// </summary>
/// <returns>DataSet</returns>
public DataSet montaEstruturaMemoria()
{
        // É criado o DataSet caso ainda não exista
        DataSet ds = AspNetGenerate.SingletonDataSet.instance;
        // Criação das tabelas que fazem parte do DataSet
        // dtBase: informações de conexão com o banco e da database
        DataTable dtBase = this.montaDataTableBase();
        // dtTabela: informações das tabelas selecionadas
        DataTable dtTabela = this.montaDataTableTabela();
        // dtColuna: informações das colunas das tabelas selecionadas
        DataTable dtColuna = this.montaDataTableColuna();
        // dtLOV: informações das possíveis lista de valores
        DataTable dtLOV = this.montaDataTableLOV();
        // As tabelas são adicionadas ao DataSet
        ds.Tables.Add(dtBase);
        ds.Tables.Add(dtTabela);
        ds.Tables.Add(dtColuna);
        ds.Tables.Add(dtLOV);
        return ds;
}

/// <summary>
/// Definição do DataTable dtBase
/// </summary>
/// <returns>DataTable</returns>
private DataTable montaDataTableBase()
{
        // Criação do DataTable dtBase
        DataTable dtBase = new DataTable("dtBase");
        // Segue a definição das colunas do dtBase
        DataColumn dcNomeProjeto = new DataColumn();
        dcNomeProjeto.DataType = System.Type.GetType("System.String");
        dcNomeProjeto.ColumnName = "Nome Projeto";
        dtBase.Columns.Add(dcNomeProjeto);

        DataColumn dcLogin = new DataColumn();
        dcLogin.DataType = System.Type.GetType("System.String");
        dcLogin.ColumnName = "Login";
        dtBase.Columns.Add(dcLogin);

        DataColumn dcSenha = new DataColumn();
        dcSenha.DataType = System.Type.GetType("System.String");
        dcSenha.ColumnName = "Senha";
        dtBase.Columns.Add(dcSenha);

        DataColumn dcHost = new DataColumn();
        dcHost.DataType = System.Type.GetType("System.String");
        dcHost.ColumnName = "Host";
        dtBase.Columns.Add(dcHost);

        DataColumn dcDatabase = new DataColumn();
        dcDatabase.DataType = System.Type.GetType("System.String");
        dcDatabase.ColumnName = "Database";
        dtBase.Columns.Add(dcDatabase);

        return dtBase;
}
```

Quadro 17 – Criando *DataTable*.

Os dados internos contidos em um *DataSet* são mantidos no formato XML e a estrutura do *DataSet* é definida pelo *XML Schema Definition Language* (XSD), ou seja, XML e *DataSet* estão intimamente ligados.

A classe *DataSet* tem a habilidade de ser serializada, são utilizados os métodos *ReadXml* (carrega um esquema XML e dados para um *DataSet*) e *WriteXML* (escreve os dados e o esquema XML para um arquivo ou fluxo de dados) para abrir e salvar um projeto, o Quadro 18 apresenta as rotinas de salvar e abrir o projeto.

```
/// <summary>
/// Objetivo: Salvar o projeto atual em um arquivo XML
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mniSalvarProjeto_Click(object sender, System.EventArgs e)
{
	try
	{
		// Configuração do componente saveFileDialog
		saveFileDialog1.AddExtension = true;
		saveFileDialog1.CheckPathExists = true;
		saveFileDialog1.CreatePrompt = false;
		saveFileDialog1.OverwritePrompt = true;
		saveFileDialog1.ValidateNames = true;
		saveFileDialog1.ShowHelp = true;
		saveFileDialog1.DefaultExt = "xml";
		saveFileDialog1.Filter = "XML files (*.xml)|*.xml|" + "All files|*.*";
		saveFileDialog1.FilterIndex = 1;

		// Verifica se o usuário informou um local de destino
		if (saveFileDialog1.ShowDialog() == DialogResult.OK)
		{
			string fileName = saveFileDialog1.FileName;
			// Recupera o DataSet que está na memória
			DataSet ds = AspNetGenerate.SingletonDataSet.instance;
			// Salva DataSet em um arquivo XML no destino informado
			ds.WriteXml(fileName, XmlWriteMode.WriteSchema);
		}
	}
	catch (Exception exp)
	{
		MessageBox.Show("Erro: " + exp.Message);
	}
}

/// <summary>
/// Objetivo: Abrir um projeto existente
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mniAbrirProjeto_Click(object sender, System.EventArgs e)
{
	try
	{
		// Configuração do componente openFileDialog
		openFileDialog1.CheckFileExists = true;
		openFileDialog1.CheckPathExists = true;
		openFileDialog1.DefaultExt = "xml";
		openFileDialog1.DereferenceLinks = true;
		openFileDialog1.Filter = "XML files (*.xml)|*.xml|" + "All files|*.*";
		openFileDialog1.Multiselect = false;
		openFileDialog1.RestoreDirectory = true;
		openFileDialog1.ShowHelp = true;
		openFileDialog1.ShowReadOnly = false;
```

```
                openFileDialog1.Title = "Selecione projeto:";
                openFileDialog1.ValidateNames = true;

                // Verifica se o usuário selecionou arquivo
                if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {
                        string fileName = openFileDialog1.FileName;
                        // Chama rotina de inicialização que também
                        // é utilizada pela opção Aquivo > Novo
                        this.inicializaSistema(fileName);
                }
        }
        catch (Exception exp)
        {
                MessageBox.Show("Erro: " + exp.Message);
        }
}

/// <summary>
/// Objetivo: Abrir ou iniciar um projeto
/// </summary>
/// <param name="pathXML"></param>
private void inicializaSistema(string pathXML)
{
        // Remove todas as informações atuais do DataSet
        AspNetGenerate.SingletonDataSet.limparInstance();
        // Cria a estrutura de dados na memória. (Estrutura do DataSet)
        DataSet dsBase = this.tableBusiness.montaEstruturaMemoria();

        this.txtLogin.Text = "";
        this.txtSenha.Text = "";
        this.txtHost.Text = "";
        this.cmbBase.Items.Clear();
        this.cmbBase.SelectedIndex = -1;
        this.dgDatabase.DataSource = null;
        this.panel2.Visible = false;

        // Verifica se é para abrir um projeto
        if (pathXML != null)
        {
                // Carrega arquivo XML para o DataSet
                dsBase.ReadXml(pathXML);
                DataRow dr = dsBase.Tables["dtBase"].Rows[0];
                this.txtLogin.Text = dr["Login"].ToString();
                this.txtSenha.Text = dr["Senha"].ToString();
                this.txtHost.Text = dr["Host"].ToString();
        }

        this.tbValidaProjeto = false;
        this.tbValidaConfiguracao = false;
        this.tbValidaVisualizacao = false;
        this.tbValidaRelacionamento = false;

        this.tcProjeto.Visible = true;
        this.tcProjeto.SelectedTab = this.tb1;

        // seta datagrids de controle da aba memória
        this.dgMemoriaBase.DataSource = dsBase.Tables["dtBase"];
        this.dgMemoriaTabela.DataSource = dsBase.Tables["dtTabela"];
        this.dgMemoriaColuna.DataSource = dsBase.Tables["dtColuna"];
}
```

Quadro 18 – Rotinas para salvar e abrir o projeto.

3.3.3 *Templates* para geração de código

Existem quatro *templates* pré-definidos que são utilizados na hora da geração das

páginas. Os *templates* são utilizados na geração do menu, na geração da página de consulta, na página de inserção e alteração e também na página de pesquisa. Em cada *template* existe *tags* especiais que são substituídas por código C# na hora da geração das páginas *web*.

Para geração das páginas o *template* é lido e colocado na memória, para cada página que será gerada são substituídas as *tags* pelo respectivo código. No apêndice B é possível ver o código do *template* de consulta na íntegra.

No Quadro 19 tem-se a rotina do *template* de consulta que popula o *datagrid*. As *tags* especiais são *<%@@ conexao %>*, *<%@@ sql %>* e *<%@@ numRegs %>*. No Quadro 20 tem-se a mesma rotina do Quadro 19, mas após a substituição das *tags*. Elas foram substituídas por trechos de código.

No apêndice C é exibido o código fonte da página gerada na íntegra.

```
        private void PopulaGrid()
        {
                SqlConnection conn = new SqlConnection(<%@@ conexao %>);

                SqlDataAdapter da = new SqlDataAdapter();
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.Connection = conn;
                da.SelectCommand.CommandText = <%@@ sql %>;
                da.SelectCommand.CommandType = CommandType.Text;
                da.Fill(ds);

                DataGrid1.DataSource = ds;
                DataGrid1.PageSize = <%@@ numRegs %>;
                DataGrid1.DataBind();
        }
```

Quadro 19 – Rotina do *template* de consulta.

```
        private void PopulaGrid()
        {
                SqlConnection conn = new SqlConnection("workstation id=heiden;packet
size=4096;user id=sa;pwd=heiden;data source=heiden;persist security info=False;initial
catalog=Northwind");

                SqlDataAdapter da = new SqlDataAdapter();
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.Connection = conn;
                da.SelectCommand.CommandText = "select ProductID 'ProductID', ProductName
'ProductName', SupplierID 'SupplierID', CategoryID 'CategoryID', QuantityPerUnit
'QuantityPerUnit', UnitPrice 'UnitPrice', UnitsInStock 'UnitsInStock', UnitsOnOrder
'UnitsOnOrder', ReorderLevel 'ReorderLevel', Discontinued 'Discontinued' from [Products]";
                da.SelectCommand.CommandType = CommandType.Text;
                da.Fill(ds);

                DataGrid1.DataSource = ds;
                DataGrid1.PageSize = 4;
                DataGrid1.DataBind();
        }
```

Quadro 20 – Rotina do *template* de consulta após criação da página.

3.3.4 Recursos das páginas *web* geradas

Para estabelecer conexão e gerenciar dados nas páginas geradas também são usadas as funcionalidades do ADO.NET. No Quadro 21 é possível visualizar a rotina que carrega os dados na página de alteração gerada pela ferramenta.

```
private void Page_Load(object sender, System.EventArgs e)
{
        if (!Page.IsPostBack)
        {
                SqlConnection conn = new SqlConnection("workstation id=andre;packet
size=4096;user id=sa;pwd=heiden;data source=andre;persist security info=False;initial
catalog=Northwind");
                SqlDataAdapter da = new SqlDataAdapter();
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.Connection = conn;
                da.SelectCommand.CommandType = CommandType.Text;
                DataSet ds = new DataSet();

                string key = Request.QueryString["key"];
                this.key.Text = key;
                if (key != null && !key.Equals(""))
                {
                        da.SelectCommand.CommandText = "select * from cliente where cod_cli = "
+ key;
                        da.Fill(ds, "cliente");
                        this.cod_cli.Text = ds.Tables["cliente"].Rows[0]["cod_cli"].ToString();
                        this.des_nome.Text =
ds.Tables["cliente"].Rows[0]["des_nome"].ToString();
                        this.des_email.Text =
ds.Tables["cliente"].Rows[0]["des_email"].ToString();
                        this.obs.Text = ds.Tables["cliente"].Rows[0]["obs"].ToString();
                        this.preco.Text = ds.Tables["cliente"].Rows[0]["preco"].ToString();
                        this.idade.Text = ds.Tables["cliente"].Rows[0]["idade"].ToString();
                        this.medida.Text = ds.Tables["cliente"].Rows[0]["medida"].ToString();
                        this.data.Text = ds.Tables["cliente"].Rows[0]["data"].ToString();
                }
        }
}
```

Quadro 21 – Rotina que carrega dados para página de alteração.

Uma das muitas tarefas na qual o ASP.NET facilita a vida do desenvolvedor é a validação de dados de formulário. No ASP tradicional ou em qualquer outra linguagem de *script* para *web* sabe-se que validar um campo de *e-mail*, CEP ou CPF requer algum tempo para criar as rotinas de validação em *javascript*. Com ASP.NET a tarefa ficou muito simples, pois ela disponibiliza controles específicos para validação de dados que associados aos controles de formulários realizam a validação de forma simples.

Uma grande vantagem no modelo de validação do ASP.NET e que não se faz necessário saber onde ela será executada, se no servidor ou no *browser* pois ela se adapta ao tipo de *browser* que o usuário estiver usando. Se o *browser* for incompatível a validação será

feita apenas no servidor.

Os controles que estão disponíveis no Microsoft *Visual Studio* para validar dados são apresentados no Quadro 22.

RequiredFieldValidator	Torna o controle associado de preenchimento obrigatório e verifica se o usuário digitou ou selecionou algo. *Ex: Campos de preenchimento obrigatório.*
CompareValidator	Realiza a comparação do valor informando com um valor informado em outro controle ou com uma constante. *Ex: Validação de senhas.*
RangeValidator	Faz a validação do valor informado verificando se ele se encontra dentro de um intervalo de valores aceitos pela aplicação. *Permite-se a validação de um valor máximo, mínimo ou ambos.*
RegularExpressionValidator	Verifica se os dados de entrada coincidem com uma expressão regular. *Ex: validações de CEP, RG, CPF, etc.*
CustomValidator	Permite criar o próprio código de validação de dados.
ValidationSummary	Permite a exibição de um resumo de todas as validações feitas na página.

Quadro 22 – Controles de validação.

Nas páginas geradas são utilizados os controles *RequiredFieldValidator* e *RegularExpressionValidator* conforme as opções que foram configuradas pelo usuário antes da geração das páginas. A Figura 22 ilustra um controle *RequiredFieldValidator* em ação, quando a página é submetida e o campo não estiver preenchido a mensagem é exibida. No Quadro 23 é exibido o código gerado para validar o campo e o trecho de código da ferramenta que gera a validação.

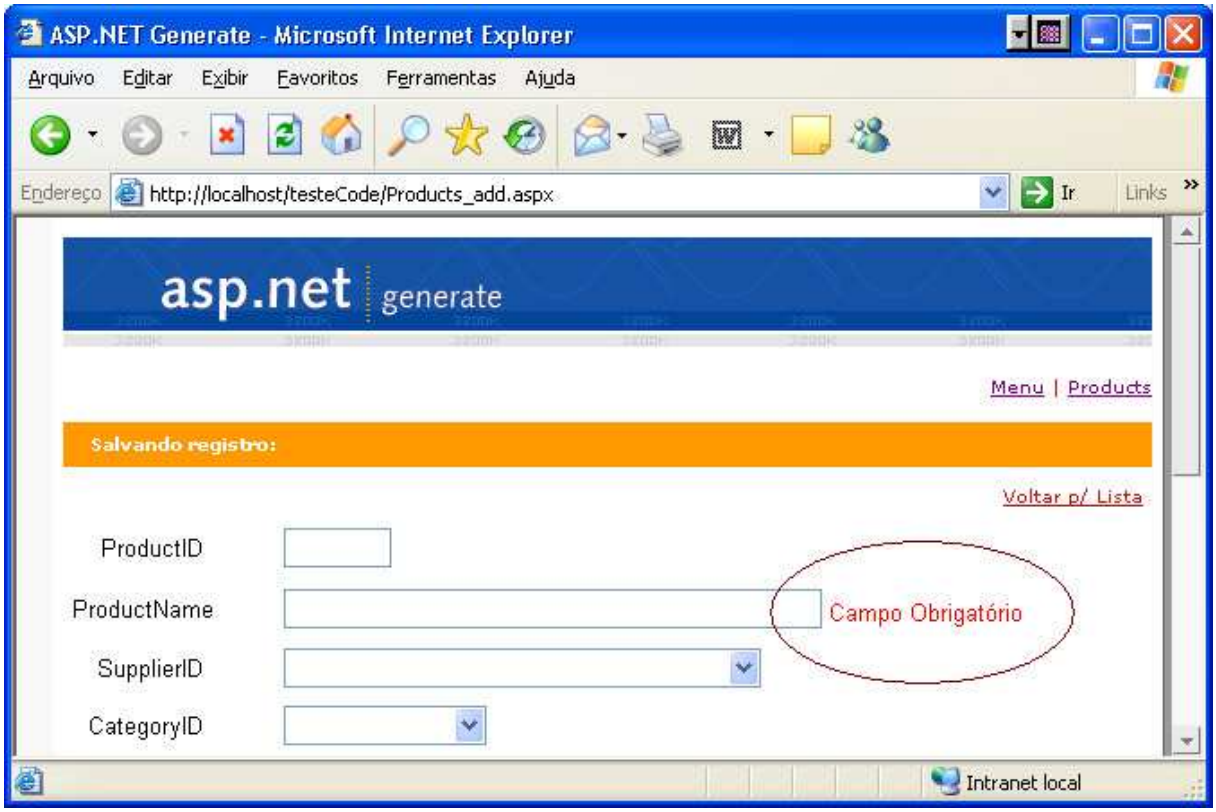


Figura 22 – Validação com *RequiredFieldValidator*.

```
<!-- Validação na página gerada. O WebControl é para entrada do nome do produto -->
<FONT face='Arial' size='2'>
<P align='left'>
       <!-- WebControl ProductName -->
        <asp:TextBox id='ProductName' runat='server' Width='300'></asp:TextBox>
        <!-- WebControl que valída o campo ProductName -->
        <asp:RequiredFieldValidator id='RequiredFieldValidator0' runat='server'
ErrorMessage='Campo Obrigatório' ControlToValidate='ProductName'>Campo
Obrigatório</asp:RequiredFieldValidator>
</P>
</FONT>


// Validação responsável por associar uma coluna a um webcontrol
RequiredFieldValidator.
// Se o campo Obrigatório está selecionado e também se o campo não é Identity
(autonumeração)
// e por último se o campo não é do tipo bit, cria validação
if ((bool)oRow["Obrigatório"] && ((bool)oRow["Identity"]) == false &&
!oRow["Tipo"].ToString().Equals("bit"))
{
       // Inclui o webcontrol de validação na página que será gerada
       sbHtml.Append("<asp:RequiredFieldValidator id='RequiredFieldValidator"+rf+"'
runat='server' ErrorMessage='Campo Obrigatório' ControlToValidate='"+ oRow["Coluna"]
+"'>Campo Obrigatório</asp:RequiredFieldValidator>\n");
       rf++;
}
```

Quadro 23 – Controles de validação.

3.3.5 Arquivo de validação e arquivo de domínios

Existe no gerador um arquivo XML que é responsável pela validação dos dados das

páginas geradas. Ele é composto basicamente de três atributos nome, expressão regular e mensagem de erro conforme Quadro 24 que ilustra um nodo do arquivo de validação. O arquivo de validação deve estar no diretório *AspNetGenerateFiles* que está no mesmo diretório do executável da ferramenta. É possível acrescentar outras validações, basta editar o arquivo em qualquer editor de texto e acrescentar a validação desejada.

Não é necessário a presença do arquivo de validação no diretório que as páginas são geradas, porque a validação já estará no controle *RegularExpressionValidator* que foi gerado na ferramenta e incluído na página gerada.

O apêndice A exibe o arquivo na integra.

```
<dtValidacao>
  <ValNome>E-mail</ValNome>
  <ValExpre>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*</ValExpre>
  <ValMensagem>Informe um e-mail válido.</ValMensagem>
</dtValidacao>
```

Quadro 24 – Nodo do arquivo de validação.

O arquivo de domínio é também um arquivo XML, o qual é responsável por todas as listas de valores das páginas ASP.NET geradas. Ao contrário do arquivo de validação o arquivo de domínio não é necessário estar no diretório da ferramenta, mas deve ser gerado no momento da geração das páginas no local especificado pelo usuário.

O arquivo de domínio gerado é chamado de *"lov.xml"* conforme Quadro 25 que ilustra uma lista de valores do mesmo.

```
<dtLOV>
  <Tabela>Categories</Tabela>
  <Coluna>CategoryName</Coluna>
  <Descricao>Valor 1</Descricao>
</dtLOV>
<dtLOV>
  <Tabela>Categories</Tabela>
  <Coluna>CategoryName</Coluna>
  <Descricao> Valor 2</Descricao>
</dtLOV>
<dtLOV>
  <Tabela>Categories</Tabela>
  <Coluna>CategoryName</Coluna>
  <Descricao> Valor 3</Descricao>
</dtLOV>
```

Quadro 25 – Lista de valores.

3.3.6 Operacionalidade da implementação

Para geração das páginas é necessário que exista um *database* criada no Microsoft *SQL Server* 2000. O usuário terá que informar os dados para conexão com o banco de dados.

A ferramenta irá armazenar apenas as tabelas que efetivamente serão escolhidas para geração, para que o programador possa configurá-las. Deve-se guardar também informações dos campos de cada tabela selecionada, tais como nome da coluna, tipo de dado e outras configurações.

Após a seleção do *database*, o programador poderá configurar as páginas que se referem às tabelas. Sobre as tabelas é possível registrar um título e indicar se é necessário a geração de página de inclusão, alteração, exclusão e pesquisa. Sobre os campos, pode-se alterar informações como descrição do campo, se é um campo requerido, se deve ser apresentado na consulta e se tem alguma validação. O tipo de controle gerado para cada campo depende do tipo de dado que está definido no banco de dados.

A Figura 23 ilusta os arquivos necessários para o funcionamento do gerador de código. No diretório em que o arquivo executável da ferramenta (item “A”) se encontra deverá existir o diretório *AspNetGenerateFiles*, com os arquivos listados na Figura 20 item “B”. A necessidade de cada arquivo já foi mencionda no decorrer do trabalho.

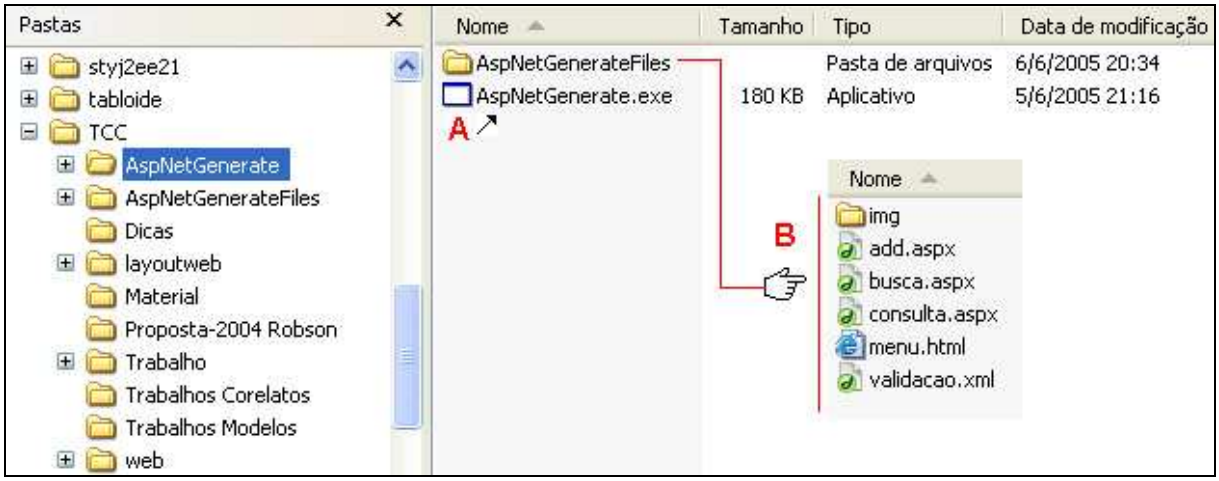


Figura 23 – Arquivos da ferramenta.

Para ilustrar a operacionalidade da ferramenta foi criada a *database* canil que contém as tabelas: cor, canil, cao, pessoa e cidade conforme Figura 24.

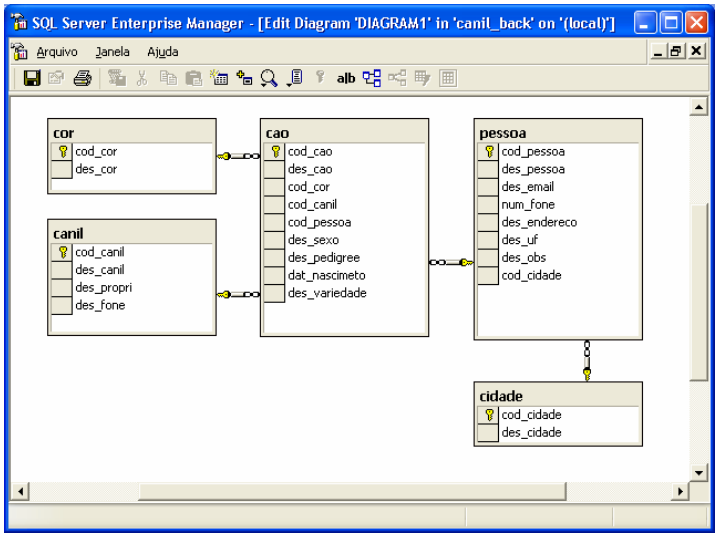


Figura 24 – Aba Projeto.

O processo de geração de páginas especificado neste trabalho deve seguir algumas etapas. Para iniciar o projeto (Figura 25) é necessário configurar a conexão com o banco de dados *SQL Server* 2000. Login, Senha e *Host* são necessárias para conectar no *SQL Server* e listar as *databases* que o usuário tem permissão. No *combobox* logo abaixo do botão Conectar são listadas as *databases* que o usuário tem permissão é obrigatória a seleção de uma delas. O *datagrid* a direita é apenas para efeito de visualização do usuário.

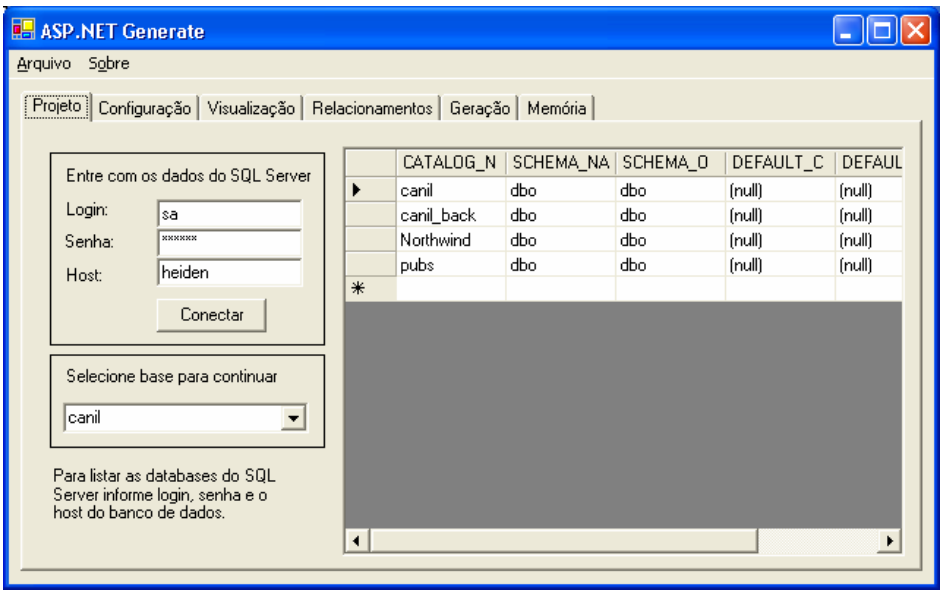


Figura 25 – Aba Projeto.

Na guia configuração das páginas *web* (Figura 26), primeiro deve-se escolher a tabela a ser manipulada, em seguida descriminar o título da página, que é usado como título da página *web* gerada e também na geração do menu. O campo descrição é usado no menu e na página de consulta, o número de registros é utilizado também na página de consulta para exibir a quantidade desejada de registros por página. Em seguida devem ser escolhidas as funcionalidades das páginas como: inclusão, alteração, exclusão e pesquisa de dados. Para finalizar essa etapa é necessário clicar sobre o botão Adicionar para incluir a tabela no *datagrid* da direita, são essas as páginas de serão geradas.

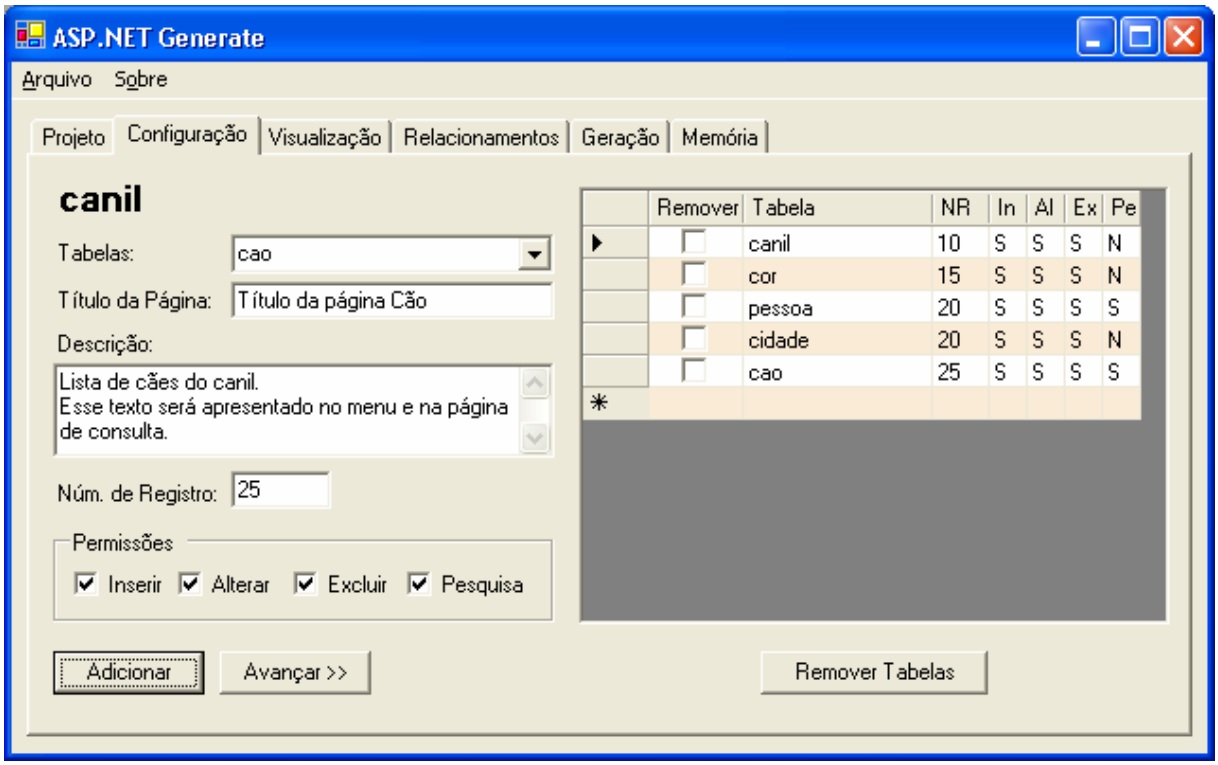


Figura 26 – Aba Configuração.

Na guia visualização (Figura 27), é possível alterar o nome das colunas que irão aparecer na tela de consulta, inserção e alteração. É possível também escolher se a coluna estará visível na página de consulta e se é obrigatório seu preenchimento nas páginas de inserção e alteração. Caso a coluna exija alguma validação específica é possível escolher entre algumas previamente cadastradas.

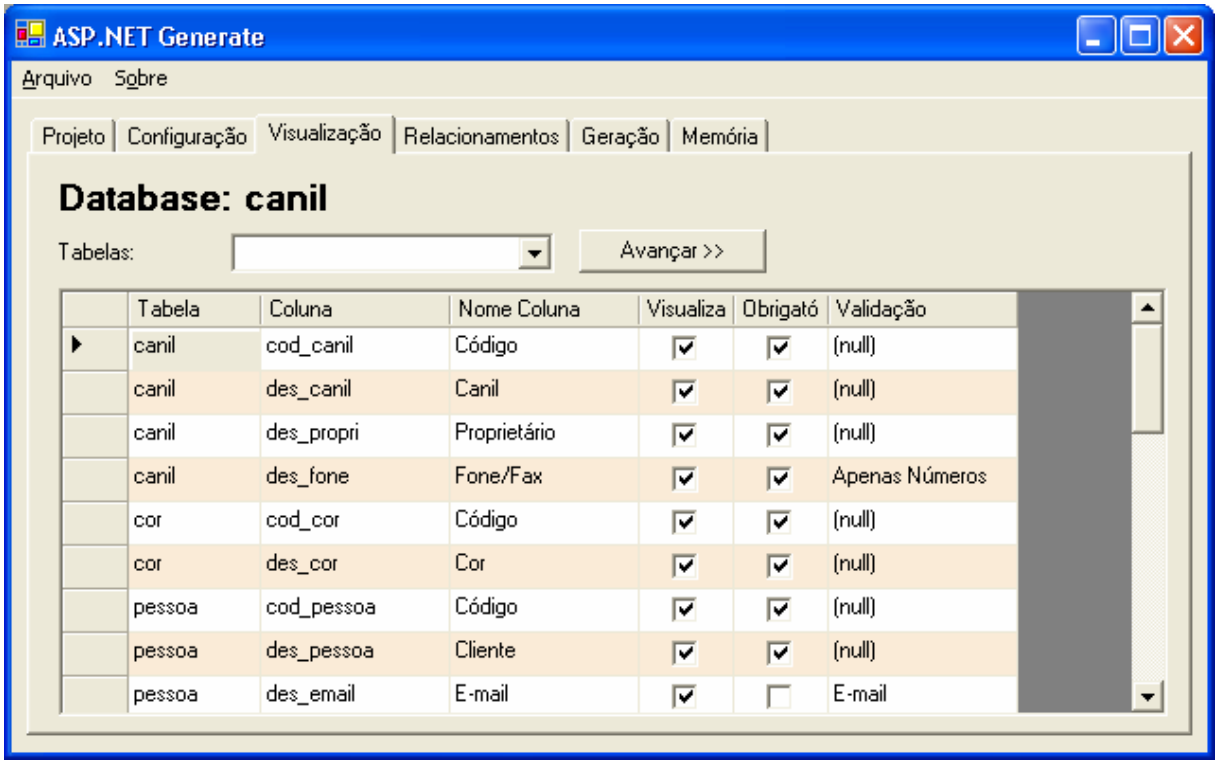


Figura 27 – Aba Visualização.

Em seguida são criados os relacionamentos (Figura 28). Os relacionamentos podem ser entre tabelas ou lista de valores. Essa opção é válida para todos os campos da tabela. Caso seja selecionado algum relacionamento, quando gerado o código da página será criado um campo *combobox* como os valores informados ou selecionados na tabela informada.

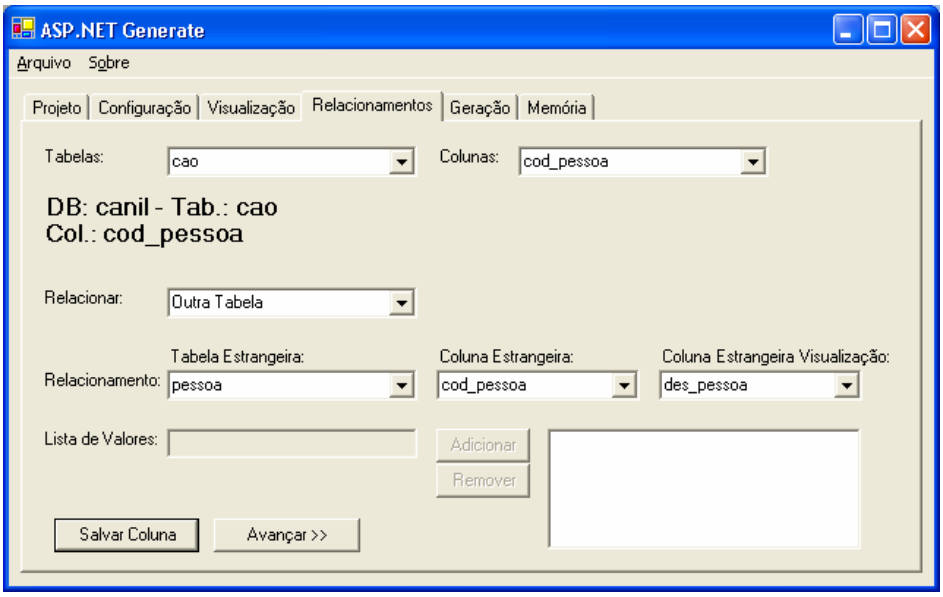


Figura 28 – Aba Relacionamento.

Agora é momento de gerar as páginas *web* (Figura 29) previamente configuradas. Elas são geradas no diretório informado pelo usuário. Caso a geração das páginas não for feita em um diretório virtual é necessária à configuração do *site* no IIS.

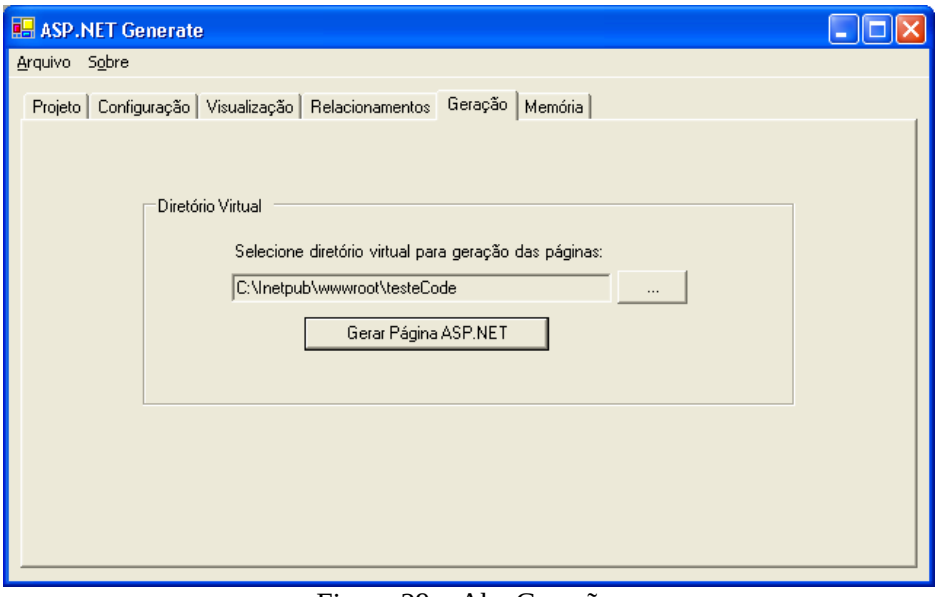


Figura 29 – Aba Geração.

A guia memória é apenas para visualização dos dados selecionados, é somente leitura. É possível apenas visualizar o estado dos *datatables* que estão na memória. A Figura 30 ilustra os arquivos gerados pela ferramenta. O item “A” é o diretório que foi selecionado para geração das páginas. O item “B” é o diretório onde estão armazenadas as imagens necessárias para perfeita visualização das páginas e o arquivo *Cascading Style Sheets* (CSS) que armazena os estilos de *links* e *tags* HTML. O item “*C lov.xml*” é onde estão as listas de valores que podem ser definidas na aba relacionamentos da ferramenta. O item “*D menu.aspx”* é o ponto de partida das páginas geradas onde estão os *links* para todas elas. O item “E” é uma página de consulta, o item “F” é uma página de inclusão e alteração conforme seleção na ferramenta e o item “G” é uma página de pesquisa gerada. O item “H” é composto do conjunto de três páginas possíveis de serem geradas conforme configuração da ferramenta, os arquivos com sufixo “_add” são as páginas de inclusão e alteração, os arquivos com sufixo “_busca” são as páginas de pesquisa o outro arquivo é a página de consulta onde são listados os registros.

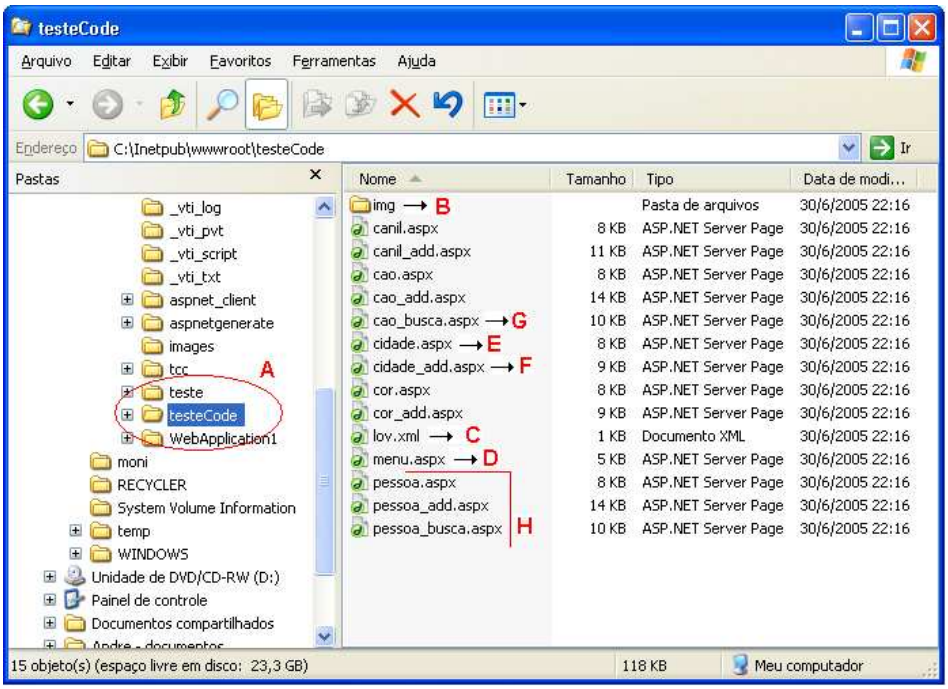


Figura 30 – Arquivos gerados.

A Figura 31 demonstra o menu gerado pela ferramenta.

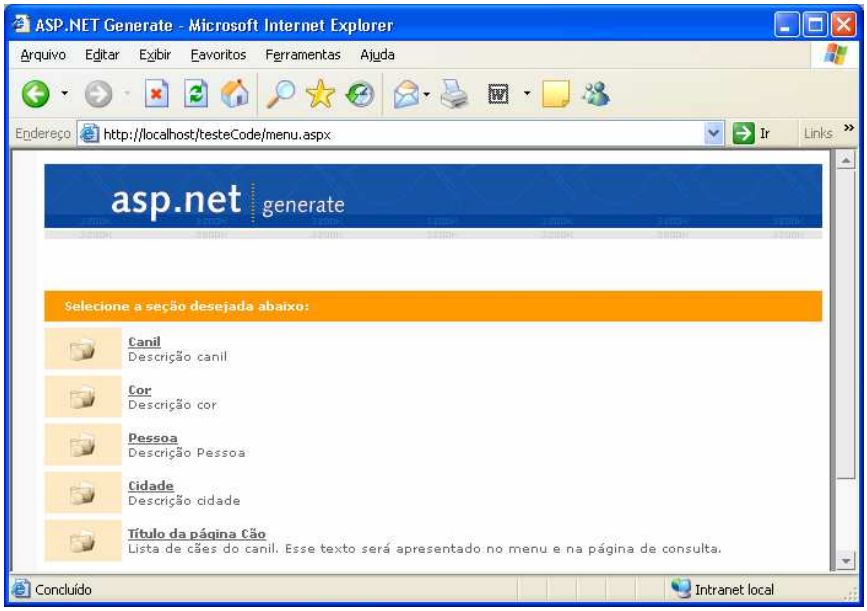


Figura 31 – Menu Gerado.

A Figura 32 ilustra a tela de consulta da tabela produtos que foi gerada. O número de registros por página é a quantidade escolhida na aba configuração da ferramenta. Também está disponível a opção de excluir, inserir e alterar que foram selecionadas também na aba configurações da ferramenta. É possível navegar entre as páginas geradas pelo menu que se encontra acima do *datagrid*.

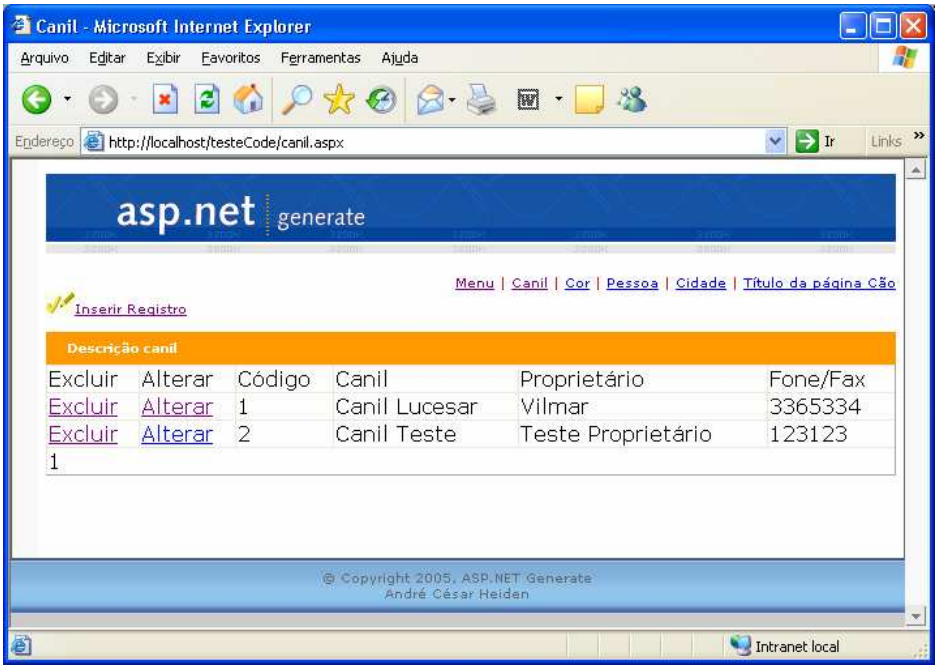


Figura 32 – Tela de consulta.

Antes do registro ser excluído é feito uma pergunta para o usuário se ele deseja

realmente excluir o registro. Inclusão e alteração, essas duas funcionalidades tem as memas regras e são implementadas na mesma página, conforme mostrado na Figura 33. Se o usuário tentar Salvar os dados com algum campo obrigatório não preenchido é mostrado uma mensagem do lado do campo informando que é obrigatório. O mesmo acontece com a validação. O item Fornecedor está relacionado com um controle *combobox* por que previamente na ferramenta foi preenchido um relacionamento para essa coluna.

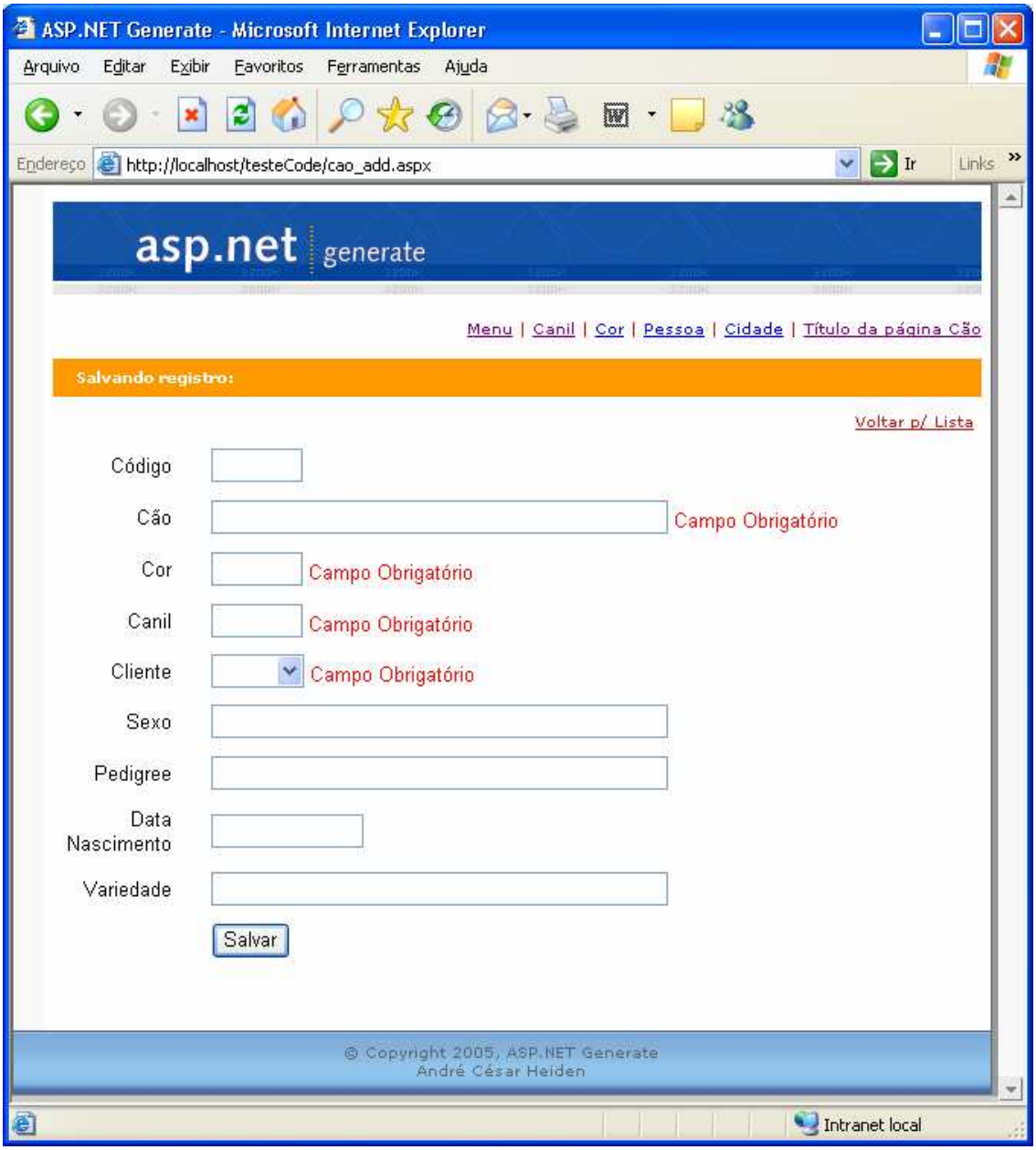


Figura 33 – Tela de Inclusão e alteração.

3.4 RESULTADOS E DISCUSSÃO

A ferramenta apresenta um diferencial importante que é a geração de código para uma tecnologia emergente ASP.NET utilizando a linguagem de programação C#.

O Quadro 26 relaciona os trabalhos correlatos com a ferramenta desenvolvida, evidenciando assim as características de cada uma delas.

Funcionalidades	**Este Projeto**	**Dias (2002)**	**Silveira (2003)**	**Castilhos (2004)**
As páginas são geradas a partir de uma aplicação desktop	x		x	
As páginas são geradas a partir de uma aplicação web		x		x
Compatível com banco de dados SQL Server 2000	x	x		x
Compatível com Microsoft Access		x	x	x
Conexão com outros bancos de dados		x		
Especificar o tipo de acesso que terá cada tabela (consulta, cadastro, alteração, exclusão)	x	x	x	x
Campos dinâmicos conforme tipo de dados	x	x	x	x
Validação de campos na inserção e alteração	x			
Definir número de registro por página	x			
Criar navegação entre páginas geradas	x		x	x
É possível definir o modelo entidade relacionamento				x
Gera páginas para ASP.NET utilizando a linguagem C#	x			

Quadro 26 – Funcionalidades específicas de cada trabalho.

Conforme Quadro 26, este trabalho tem como principal diferença a utilização da linguagem C# na ferramenta e nas páginas geradas, outras novidade são as validações através de expressões regulares e a definição do número de registros das páginas de consulta.

4 CONCLUSÕES

Este trabalho teve como objetivo o desenvolvimento de uma ferramenta para otimizar o serviço do desenvolvedor reduzindo o tempo destinado a criação de rotinas tidas como triviais de entrada e consulta de dados. Tal objetivo foi alcançado.

A ferramenta de desenvolvimento da plataforma .NET, o *Visual Studio* .NET e a linguagem escolhida para desenvolvimento, o C#, mostraram-se muito eficientes para o desenvolvimento de aplicações *winforms* que foi o caso da ferramenta de geração de código desenvolvida.

Da mesma forma que a ferramenta RAD da Microsoft e a linguagem se mostraram eficiente para aplicações *winforms*, cabe perfeitamente as mesmas mensões também para o desenvolvimento de páginas *web*, chamadas de *webforms*. A linguagem apresenta uma variedade de controles que aumentam a produtividade e agilizam a desenvolvimento.

A solução desenvolvida permite que usuários possam efetuar a configuração de tabelas do banco de dados do *SQL Server* 2000, e em seguida gerar páginas na tecnologia ASP.NET para manipulação de dados.

Este trabalho não cria controles de entrada de dados para tipos *binary*, *image*, *sql_variant* e *varbinary* do Microsoft *SQL Server* 2000, por que esses tipos de dados consitem de valores binários e não foi implementado na ferramenta o recurso de *upload* de arquivos para popular os campos dos tipos citados acima.

Com o estudo e execução desse trabalho foi agregado conhecimento e experiência no desenvolvimento de aplicações utilizando a tecnologia .NET.

4.1 EXTENSÕES

Como possíveis extensões para o trabalho, destacam-se:

a) gerar páginas a partir de outros bancos de dados;

b) opção de o usuário escolher o *layout* das páginas que serão geradas;

c) geração de página para outras linguagens de programação;

d) criar entrada de dados para tipos *binary*, *image*, *sql_variant* e *varbinary* do Microsoft *SQL Server* 2000.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, Daniel. **Migrando de ASP para ASP.NET**. [S.l.], 2004. Disponível em: <http://www.dotnetraptors.com.br/start/artigos/artigos_asp/2509.aspx>. Acesso em: 20 abr. 2005.

CASTILHOS, Cristiano. **Ferramenta case geradora de páginas em ASP**. 2004. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DEITEL, Harvey M. et al. **C#**: como programar. Tradução João Eduardo Nóbrega Tortello. São Paulo: Pearson Education, 2003.

DIAS, Adriano. **Aplicativo para atualização de banco de dados utilizando Active Server Pages (ASP)**. 2002. 42 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FABFORCE.NET. **General information - what is DBDesigner 4?**. [S.l.], 2003. Disponível em: <http://www.fabforce.net/dbdesigner4>. Acesso em: 05 mar. 2005.

GALUPPO, Fabio. **Visual C#**. São Paulo, 2001. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/visualc/Default.mspx>. Acesso em: 22 mar. 2005.

GALUPPO, Fabio; MATHEUS, Vanclei; SANTOS, Wallace. **Desenvolvendo com C#**. Porto Alegre: Bookman, 2004.

HADDAD, Renato. **Dez Razões para migrar para ASP.NET**. São Paulo, 2001. Disponível em: <http://www.microsoft.com/brasil/msdn/tecnologias/aspnet/aspnet_migrar.aspx>. Acesso em: 17 mar. 2005.

HERRINGTON, Jack. **Code generation in action**. Greenwich, CT: Manning, 2003.

MACORATTI, José Carlos. **ADO.NET - o acesso aos dados**. [S.l.], [2002?]. Disponível em: <http://www.macoratti.net/vbnet_5.htm>. Acesso em: 1 jun. 2005.

MARTIN, James; MCCLURE, Carma. **Técnicas estruturadas e case**. Tradução: Lúcia Faria Silva. São Paulo: Makron Books & McGraw-Hill, 1991.

MICROSOFT CORPORATION. **Administrando um banco de dados do Microsoft SQL Server 2000**. Argentina: Docuprint, 2000.

_____. **ASP.NET**. [S.l.],[2002?]. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/aspnet/PgAspNet.mspx >. Acesso em: 23 abr. 2005.

_____. **Introdução ao ASP.NET**. Argentina: Docuprint, 2003.

_____. **Introdução a .NET**: o que é .NET?. [S.l.],[2004?]. Disponível em: <http://www.microsoft.com/brasil/dotnet/introducao/oquee.asp>. Acesso em: 19 set. 2004.

MSDN LIBRARY. **Microsoft MSDN**. [S.l.], [2004]. Página de ajuda da Microsoft Corporation. Disponível em: <http://msdn.microsoft.com/library/default.asp>. Acesso em: 25 mar. 2005.

SANT'ANNA, Mauro. **.NET framework**. São Paulo, 2001a. Disponível em: <http://www.mas.com.br/Artigos/NET%20Framework.htm>. Acesso em: 19 set. 2004.

_____. C#: uma linguagem para o novo milênio. **Developers' Magazine**, [Rio de Janeiro],[maio 2001b]. Disponível em: <http://www.mas.com.br/Artigos/CSharp_Nova%20Ling.htm>. Acesso em: 21 set. 2004.

SANTOS, Edgar H. CodeCharge: gerador de códigos para aplicações web. **Comunicado técnico**, Campinas, SP, n. 45, dez. 2002. Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/content/2002/comuntec45.pdf>. Acesso em: 25 out. 2004.

SAUVÉ, Jacques Philippe. **Frameworks**. Paraíba, 2002. Disponível em: <http://jacques.dsc.ufcg.edu.br/cursos/map/html/frame/oque.htm>. Acesso em: 30 abr. 2004.

SILVEIRA, Claudionor. **Geração automática de cadastros e consultas para linguagem ASP baseado em banco de dados**. 2003. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Arquivo de validação

Segue no Quadro 27 o arquivo na integra com as validações. O arquivo deve estar no mesmo diretório do executável do gerador.

validacao.xml

```
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="pt-BR">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="dtValidacao">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ValNome" type="xs:string" minOccurs="0" />
                <xs:element name="ValExpre" type="xs:string" minOccurs="0" />
                <xs:element name="ValMensagem" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <dtValidacao>
    <ValNome>Apenas Números</ValNome>
    <ValExpre>\d*</ValExpre>
    <ValMensagem>Informe apenas números.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>Apenas 1 número</ValNome>
    <ValExpre>\d</ValExpre>
    <ValMensagem>Informe apenas um números.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>E-mail</ValNome>
    <ValExpre>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*</ValExpre>
    <ValMensagem>Informe um e-mail válido.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>CEP</ValNome>
    <ValExpre>\d{5}(-\d{3})?</ValExpre>
    <ValMensagem>CEP inválido. Ex. 00000-000 ou 00000</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>Select</ValNome>
    <ValExpre>SELECT\s[\w\*\)\(\,\s]+\sFROM\s[\w]+</ValExpre>
    <ValMensagem>Select inválido.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>Update</ValNome>
    <ValExpre>UPDATE\s[\w]+\sSET\s[\w\,\'\=]+</ValExpre>
    <ValMensagem>Update inválido.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>Insert</ValNome>
    <ValExpre>INSERT\sINTO\s[\d\w]+[\s\w\d\)\(\,]*\sVALUES\s\([\d\w\'\,\)]+</ValExpre>
    <ValMensagem>Insert inválido.</ValMensagem>
  </dtValidacao>
  <dtValidacao>
    <ValNome>Delete</ValNome>
    <ValExpre>DELETE\sFROM\s[\d\w\'\=]+</ValExpre>
    <ValMensagem>Delete inválido.</ValMensagem>
  </dtValidacao>
</NewDataSet>
```

Quadro 27 – Arquivo de validação.

APÊNDICE B – *Template* da página de consulta

No Quadro 28 é apresentado na íntegra o *template* da página de consulta.

consulta.aspx

```
<%@ Page Language="C#" EnableViewState="True" EnableSessionState="True"
SmartNavigation="False" %>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.SqlClient"%>
<%@ Import Namespace="System.Web.UI.WebControls" %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Collections" %>
<%@ Import Namespace="System.ComponentModel" %>
<%@ Import Namespace="System.Drawing" %>
<%@ Import Namespace="System.Web" %>
<%@ Import Namespace="System.Web.SessionState" %>
<%@ Import Namespace="System.Web.UI" %>
<%@ Import Namespace="System.Web.UI.HtmlControls" %>

<script runat="server">

        DataSet ds = new DataSet();

        private void Page_Load(object sender, System.EventArgs e) {
                if (!Page.IsPostBack)
                {
                        PopulaGrid();
                }
        }

        private void PopulaGrid()
        {
                SqlConnection conn = new SqlConnection(<%@@ conexao %>);

                SqlDataAdapter da = new SqlDataAdapter();
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.Connection = conn;
                string queryBusca = Request.QueryString["queryBusca"];
                da.SelectCommand.CommandText = <%@@ sql %> + queryBusca;
                da.SelectCommand.CommandType = CommandType.Text;
                da.Fill(ds);

                DataGrid1.DataSource = ds;
                DataGrid1.PageSize = <%@@ numRegs %>;
                DataGrid1.DataBind();
        }

        private void DataGrid1_PageIndexChanged(object source,
System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
        {
                DataGrid1.CurrentPageIndex = e.NewPageIndex;
                PopulaGrid();
        }

        private void DataGrid1_ItemDataBound(object sender,
System.Web.UI.WebControls.DataGridItemEventArgs e)
        {
                // Verificamos se não é uma linha de cabacelho ou rodapé
                if (e.Item.ItemType != ListItemType.Header && e.Item.ItemType !=
ListItemType.Footer)
                {
                        // Referencie o controle LinkButton
                        LinkButton deleteButton = (LinkButton)e.Item.Cells[0].Controls[0];

                        //incluimos o gerenciador de evento onclick
                        deleteButton.Attributes["onclick"] = "javascript:return " +
                         "confirm('Tem certeza que deseja excluir o registro?')";
                }
        }

        private void DataGrid1_DeleteCommand(object source,
```

```
System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
                SqlConnection conn = new SqlConnection(<%@@ conexao %>);
                SqlCommand sqlCom = new SqlCommand(<%@@ sqlRemove %>, conn);
                sqlCom.Parameters.Add(new SqlParameter("@Codigo",
DataGrid1.DataKeys[e.Item.ItemIndex]));
                conn.Open();
                sqlCom.ExecuteNonQuery();
                conn.Close();

                PopulaGrid();
        }

        private void DataGrid1_EditCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
                string cod = DataGrid1.DataKeys[e.Item.ItemIndex].ToString();
                Server.Transfer("<%@@ linkAlterar %>" + cod);
        }
</script>

<HTML>
<HEAD>
<TITLE><%@@ titulo %></TITLE>
<meta name="GENERATOR" Content="ASP.NET Generate - André César Heiden">
<meta name="CODE_LANGUAGE" Content="C#">

<link href="img/estilos.css" rel="stylesheet" type="text/css">
<BODY bgColor=#cccccc leftMargin="0" topMargin="0" marginheight="0" marginwidth="0">
<form id="Form1" method="post" runat="server">
<TABLE width="100%" valign="top" cellSpacing=0 cellPadding=1 align=center bgColor="#666666"
border="0">
  <TR>
    <TD>
                <TABLE cellSpacing=0 cellPadding=0 width="100%" vAlign="top" bgColor=#ffffff
border=0>
        <TR>
          <TD> <TABLE height="100%" cellSpacing=0 cellPadding=0 width="100%" border=0>
              <TR>
                <TD width=20 bgColor=#f8f8f8></TD>
                <TD vAlign=top>
                                        <TABLE cellSpacing=0 cellPadding=6 width="100%"
border=0>
                    <TR>
                      <TD class=v10> <TABLE cellSpacing=0 cellPadding=0 width="100%"
align=center border=0>
                        <TR>
                          <TD height=5></TD>
                        </TR>
                        <TR>
                          <TD height="61" background="img/bg_azul.gif" class="v10">
                            <div align="left"><a href="menu.aspx"><img src="img/aspnet.gif"
width="198" height="42" hspace="50" border="0" align="top"></a></div></TD>
                        </TR>
                        <TR>
                          <TD height="15" class="v10"> </TD>
                        </TR>
                        <TR>
                          <TD height="15" class="v10"><div align="right"><FONT
color=#bf1614>
                                                <%@@ menu %></FONT></div></TD>
                        </TR>
                        <TR>
                          <TD height="15" class="v10">
                                                        <%@@ linkInserirImg %><FONT
color=#bf1614><%@@ linkInserir %></FONT>
                                                </TD>
                        </TR>
                      </TABLE>
                        <BR>
                        <TABLE cellSpacing=0 cellPadding=0 width="100%" height="25"
border=0 bgcolor="#FF9900">
                          <TR>
                            <TD> <TABLE width="95%" border=0 align="center" cellPadding=0
cellSpacing=0>
                                <TR>
                                  <TD class=v10> <FONT color=#ffffff><STRONG><%@@ descricao
```

```
%></STRONG></FONT></TD>
                                    </TR>
                                  </TABLE></TD>
                              </TR>
                            </TABLE>

                                                  <asp:label id="lblMsgErro" Font-Bold="True"
runat="server" ForeColor="#aa0000" />
                                                  <asp:label id="lblMsgOK" Font-Bold="True"
runat="server" ForeColor="#3300CC" />

                                                <asp:DataGrid
                                                       id=DataGrid1
                                                       runat="server"
                                                       Width="100%"
                                                       AllowPaging="True"
                                                       DataSource="<%# ds %>"

        OnPageIndexChanged="DataGrid1_PageIndexChanged"

        OnDeleteCommand="DataGrid1_DeleteCommand"

        OnItemDataBound="DataGrid1_ItemDataBound"
                                                       OnEditCommand="DataGrid1_EditCommand"
                                                       DataKeyField="<%@@ dataKey %>" >
                                                       <FooterStyle HorizontalAlign="Center"
VerticalAlign="Middle"></FooterStyle>
                                                       <Columns>
                                                              <asp:ButtonColumn Text="<%@@
excluir %>" HeaderText="Excluir" CommandName="Delete"></asp:ButtonColumn>
                                                              <asp:EditCommandColumn
ButtonType="LinkButton" HeaderText="Alterar" EditText="<%@@ alterar
%>"></asp:EditCommandColumn>
                                                       </Columns>
                                                       <PagerStyle
Mode="NumericPages"></PagerStyle>
                                                </asp:DataGrid>

                                                </TD>
                                  </TR>
                                </TABLE>
                              <BR><BR><BR>
                                                </TD>
                          </TR>
                        </TABLE></TD>
                    </TR>
                  </TABLE></TD>
          </TR>
          <TR>
            <TD background=img/bottom.gif colSpan=2 height=41 > <TABLE cellSpacing=0
cellPadding=0 width="100%" border=0>
                <TR>
                  <TD class="v10" align="center">© Copyright 2005, ASP.NET Generate<BR>André
César Heiden</TD>
                </TR>
              </TABLE></TD>
          </TR>
        </TABLE></TD>
      <TD width=2 bgColor=#999999></TD>
    </TR>
    <TR>
      <TD bgColor=#999999 colSpan=2 height=1></TD>
    </TR>
</TABLE>
</form>
</BODY>
</HTML>
```

Quadro 28 – *Template* da página de consulta.

APÊNDICE C –Página de consulta gerada pelo gerador

No Quadro 29 é apresentado na íntegra a página de consulta gerada pelo gerador de código.

Products.aspx

```
<%@ Page Language="C#" EnableViewState="True" EnableSessionState="True"
SmartNavigation="False" %>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.SqlClient"%>
<%@ Import Namespace="System.Web.UI.WebControls" %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Collections" %>
<%@ Import Namespace="System.ComponentModel" %>
<%@ Import Namespace="System.Drawing" %>
<%@ Import Namespace="System.Web" %>
<%@ Import Namespace="System.Web.SessionState" %>
<%@ Import Namespace="System.Web.UI" %>
<%@ Import Namespace="System.Web.UI.HtmlControls" %>

<script runat="server">

        DataSet ds = new DataSet();

        private void Page_Load(object sender, System.EventArgs e) {
                if (!Page.IsPostBack)
                {
                        PopulaGrid();
                }
        }

        private void PopulaGrid()
        {
                SqlConnection conn = new SqlConnection("workstation id=heiden;packet
size=4096;user id=sa;pwd=heiden;data source=heiden;persist security info=False;initial
catalog=Northwind");

                SqlDataAdapter da = new SqlDataAdapter();
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.Connection = conn;
                string queryBusca = Request.QueryString["queryBusca"];
                da.SelectCommand.CommandText = "select ProductID 'ProductID', ProductName
'ProductName', SupplierID 'SupplierID', CategoryID 'CategoryID', QuantityPerUnit
'QuantityPerUnit', UnitPrice 'UnitPrice', UnitsInStock 'UnitsInStock', UnitsOnOrder
'UnitsOnOrder', ReorderLevel 'ReorderLevel', Discontinued 'Discontinued' from [Products]" +
queryBusca;
                da.SelectCommand.CommandType = CommandType.Text;
                da.Fill(ds);

                DataGrid1.DataSource = ds;
                DataGrid1.PageSize = 4;
                DataGrid1.DataBind();
        }

        private void DataGrid1_PageIndexChanged(object source,
System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
        {
                DataGrid1.CurrentPageIndex = e.NewPageIndex;
                PopulaGrid();
        }

        private void DataGrid1_ItemDataBound(object sender,
System.Web.UI.WebControls.DataGridItemEventArgs e)
        {
                // Verificamos se não é uma linha de cabacelho ou rodapé
                if (e.Item.ItemType != ListItemType.Header && e.Item.ItemType !=
ListItemType.Footer)
                {
                        // Referencie o controle LinkButton
```

```
                        LinkButton deleteButton = (LinkButton)e.Item.Cells[0].Controls[0];

                        //incluimos o gerenciador de evento onclick
                        deleteButton.Attributes["onclick"] = "javascript:return " +
                         "confirm('Tem certeza que deseja excluir o registro?')";
                }
        }

        private void DataGrid1_DeleteCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
                SqlConnection conn = new SqlConnection("workstation id=heiden;packet
size=4096;user id=sa;pwd=heiden;data source=heiden;persist security info=False;initial
catalog=Northwind");
                SqlCommand sqlCom = new SqlCommand("delete from [Products] where ProductID =
@Codigo", conn);
                sqlCom.Parameters.Add(new SqlParameter("@Codigo",
DataGrid1.DataKeys[e.Item.ItemIndex]));
                conn.Open();
                sqlCom.ExecuteNonQuery();
                conn.Close();

                PopulaGrid();
        }

        private void DataGrid1_EditCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
                string cod = DataGrid1.DataKeys[e.Item.ItemIndex].ToString();
                Server.Transfer("Products_add.aspx?key=" + cod);
        }
</script>

<HTML>
<HEAD>
<TITLE>Title Produtos</TITLE>
<meta name="GENERATOR" Content="ASP.NET Generate - André César Heiden">
<meta name="CODE_LANGUAGE" Content="C#">

<link href="img/estilos.css" rel="stylesheet" type="text/css">
<BODY bgColor=#cccccc leftMargin="0" topMargin="0" marginheight="0" marginwidth="0">
<form id="Form1" method="post" runat="server">
<TABLE width="100%" valign="top" cellSpacing=0 cellPadding=1 align=center bgColor="#666666"
border="0">
  <TR>
    <TD>
                <TABLE cellSpacing=0 cellPadding=0 width="100%" vAlign="top" bgColor=#ffffff
border=0>
         <TR>
           <TD> <TABLE height="100%" cellSpacing=0 cellPadding=0 width="100%" border=0>
               <TR>
                 <TD width=20 bgColor=#f8f8f8></TD>
                 <TD vAlign=top>
                                        <TABLE cellSpacing=0 cellPadding=6 width="100%"
border=0>
                     <TR>
                       <TD class=v10> <TABLE cellSpacing=0 cellPadding=0 width="100%"
align=center border=0>
                         <TR>
                           <TD height=5></TD>
                         </TR>
                         <TR>
                           <TD height="61" background="img/bg_azul.gif" class="v10">
                             <div align="left"><a href="menu.aspx"><img src="img/aspnet.gif"
width="198" height="42" hspace="50" border="0" align="top"></a></div></TD>
                         </TR>
                         <TR>
                           <TD height="15" class="v10"> </TD>
                         </TR>
                         <TR>
                           <TD height="15" class="v10"><div align="right"><FONT
color=#bf1614>
                                                <asp:HyperLink id='hlMenu0' runat='server'
NavigateUrl='menu.aspx'>Menu</asp:HyperLink> | <asp:HyperLink id='hlMenu1' runat='server'
NavigateUrl='Products.aspx'>Title Produtos</asp:HyperLink> | <asp:HyperLink id='hlMenu2'
runat='server' NavigateUrl='Territories.aspx'>Territories</asp:HyperLink></FONT></div></TD>
                         </TR>
```

```
                        <TR>
                          <TD height="15" class="v10">
                                                    <asp:hyperlink id='HyperLink0'
runat='server' NavigateUrl='Products_add.aspx' ImageUrl='img/ILS_icon_test.gif' width='22'
height='16' align='absmiddle'></asp:hyperlink><FONT color=#bf1614><asp:hyperlink
id='HyperLink1' runat='server' NavigateUrl='Products_add.aspx'>Inserir
Registro</asp:hyperlink></FONT>
                                                  </TD>
                        </TR>
                      </TABLE>
                        <BR>
                        <TABLE cellSpacing=0 cellPadding=0 width="100%" height="25"
border=0 bgcolor="#FF9900">
                          <TR>
                            <TD> <TABLE width="95%" border=0 align="center" cellPadding=0
cellSpacing=0>
                                <TR>
                                  <TD class=v10> <FONT color=#ffffff><STRONG>Descrição do
menu da tabela produtos.</STRONG></FONT></TD>
                                </TR>
                              </TABLE></TD>
                          </TR>
                        </TABLE>

                                                 <asp:label id="lblMsgErro" Font-Bold="True"
runat="server" ForeColor="#aa0000" />
                                                 <asp:label id="lblMsgOK" Font-Bold="True"
runat="server" ForeColor="#3300CC" />

                                               <asp:DataGrid
                                                      id=DataGrid1
                                                      runat="server"
                                                      Width="100%"
                                                      AllowPaging="True"
                                                      DataSource="<%# ds %>"

       OnPageIndexChanged="DataGrid1_PageIndexChanged"

       OnDeleteCommand="DataGrid1_DeleteCommand"

       OnItemDataBound="DataGrid1_ItemDataBound"
                                                      OnEditCommand="DataGrid1_EditCommand"
                                                      DataKeyField="ProductID" >
                                                      <FooterStyle HorizontalAlign="Center"
VerticalAlign="Middle"></FooterStyle>
                                                      <Columns>
                                                             <asp:ButtonColumn Text="Excluir"
HeaderText="Excluir" CommandName="Delete"></asp:ButtonColumn>
                                                      <asp:EditCommandColumn
ButtonType="LinkButton" HeaderText="Alterar" EditText="Alterar"></asp:EditCommandColumn>
                                                      </Columns>
                                                      <PagerStyle
Mode="NumericPages"></PagerStyle>
                                               </asp:DataGrid>

                                               </TD>
                          </TR>
                        </TABLE>
                      <BR><BR><BR>
                                          </TD>
                    </TR>
                  </TABLE></TD>
              </TR>
            </TABLE></TD>
        </TR>
        <TR>
          <TD background=img/bottom.gif colSpan=2 height=41 > <TABLE cellSpacing=0
cellPadding=0 width="100%" border=0>
              <TR>
                <TD class="v10" align="center">© Copyright 2005, ASP.NET Generate<BR>André
César Heiden</TD>
              </TR>
            </TABLE></TD>
        </TR>
      </TABLE></TD>
    <TD width=2 bgColor=#999999></TD>
  </TR>
```

```
  <TR>
    <TD bgColor=#999999 colSpan=2 height=1></TD>
  </TR>
</TABLE>
</form>
</BODY>
</HTML>
```

Quadro 29 – Página de consulta gerada pelo gerador.