UNISC
UNIVERSIDADE DE SANTA CRUZ DO SUL

TRABALHO ELABORADO PARA A DISCIPLINA DE TÓPICOS ESPECIAIS EM COMPUTAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO EM NOVEMBRO DE 2004

ESTUDO DOS FRAMEWORKS BASEADOS NA COMMON LANGUAGE INFRASTRUCTURE (CLI): MICROSOFT .NET FRAMEWORK, MONO E DOTGNU PORTABLE.NET

Adilson Arthur Mohr
adi.mohr@via-rs.net

Ana Neri Kniphoff da Cruz
ananeri@cantustange.com.br

Everton Azeredo Carvalho
evertonac@yahoo.com.br

Vagner Pinto da Silva
vpsilvay@yahoo.com.br

Resumo: Este tutorial apresenta três projetos baseados na *Common Language Infrastruture* (CLI): *Microsoft .Net Framework*, *Mono* e *DotGNU Portable.Net*. Estes frameworks têm como principais objetivos facilitar a criação de aplicações para *desktop*, Web e dispositivos móveis, para plataformas Windows, GNU/Linux e MacOS X (é importante ressaltar que *Microsoft .Net Framework* foca-se em plataformas Windows, enquanto *Mono* e *DotGNU Portable.Net* procuram ser multiplataforma). Ele dá uma visão do andamento atual destes projetos, procurando explicar os conceitos e os recursos desses *frameworks*, tais como: sua arquitetura, as linguagens suportadas atualmente, comparações entre eles e a integração com a plataforma Java. No final são apresentados uma descrição dos projetos relacionados aos *frameworks*, testes realizados com algumas ferramentas, e alguns guias e exemplos de criação de aplicações com o uso destas ferramentas.

Palavras chave: Mono, Portable.Net, .Net Compact Framework, .Net, dotNet, framework, CLI, CLS, CLR, multilinguagem, multiplataforma, ADO.Net, ASP.Net, WebServices, XML, WinForms, WebForms, GTK, QT#, C#, P#, BOO, Java, IKVM, JILC, WebMatrix, MonoDevelop, Eclipse, Glade, #Develop, NUnit, NAnt, MonoDebugger, LOGO, ADP, Bamboo.Prevalence, Threading, Networking, NpgSQL

Índice Analítico

1 Introdução

A Microsoft há vários anos criou o projeto Windows DNA que tinha uma arquitetura para desenvolvimento e integração de aplicações web. No entanto, apresentava pouca flexibilidade, algo muito necessário atualmente. Sendo assim, logo se notou que havia a necessidade de elaboração de uma nova arquitetura, que visasse flexibilidade e facilidade para o desenvolvimento de aplicações, tanto gráficas (GUI) quanto web, surgindo então, o Microsoft .Net Framework.

Um framework, neste contexto, consiste basicamente de um conjunto de classes com interface bem definida, que proporcionam um modo padrão para criação de aplicações. Isto, aliado a uma arquitetura baseada na *Common Language Infrastructure* – CLI (Infra-estrutura de Linguagem Comum), torna possível o desenvolvimento de aplicações multilinguagem, tornando possível a integração e a comunicação entre diversas linguagens de programação (por exemplo C# e Delphi).

Inicialmente quem tinha controle total sobre a tecnologia .Net era a Microsoft. Ela é quem detinha para si o padrão do .Net. Porém, durante o ano de 2001 o controle da especificação da CLI e da linguagem padrão do .Net, o C# *(veja a seção 7 para mais informações)*, foi passado a ECMA (*European Computer Manufacturing Association* - A ECMA é um órgão europeu de padronização equivalente à ISO (*International Standards Organization*) , com sede nos Estados Unidos), fazendo com que o .Net começasse ganhar tons de arquitetura com padrão aberto. A Microsoft pretendia com isso facilitar a portabilidade de sua plataforma para outros sistemas operacionais diferentes do Windows, tornando o .Net algo melhor visto pelas companhias por tratar-se de algo baseado em padrões abertos.

A mais de dois anos surgiu o DotGNU Portable.Net, um framework baseado na CLI-ECMA, que tem por objetivo tornar o .Net multiplataforma. Tornando possível diminuir a dependência das companhias e suas aplicações do sistema operacional Windows. O Portable.Net segue a licença GPL o que garante uma implementação realmente livre.

Além da iniciativa do projeto DotGNU Portable.Net também surgiu o Projeto Mono, outra variação do .Net da Microsoft que segue a GPL. Este projeto é liderado pela companhia Novell, tendo como principal coordenador Miguel de Icaza – que foi o principal responsável pelo GNOME usado no Linux. O Mono, ao contrário do DotGNU, tem uma divulgação bem maior no Brasil devido ao Projeto Mono Brasil, que visa unir as contribuições brasileiras ao projeto internacional do Mono.

Tendo o projeto DotGNU quanto o projeto Mono, apesar de serem independentes e, em alguns casos, seguirem idéias diferentes de implementação, se complementam (maiores detalhes durante a documentação). Inicialmente poderíamos pensar que tais projetos seriam uma ameaça à implementação da Microsoft, porém isso não é verdade. A Microsoft passou o controle da especificação da CLI à ECMA justamente por perceber que muitos desenvolvedores e companhias estavam migrando para Java por esta ser multiplataforma. A Microsoft então pensou em também tornar multiplataforma o .Net. mas sem ter que arcar com os altos custos disso. O maior sinal da pró-atividade da Microsoft está no fato de ela estar trocando informações com a Novell, visando facilitar a portabilidade do seu framework .

1.1 Microsoft .Net Framework

A plataforma .Net foi criada pela Microsoft com o intuito de prover um framework integrado ao S.O. Microsoft Windows. Este framework permite a integração de aplicações web ou mesmo gráficas entre diferentes linguagens de programação e dispositivos de acesso.

O .Net é multilinguagem o que possibilita total integração entre linguagens como C#, Delphi Language (Delphi), Visual Basic (VB.Net), J#, Cobol, etc. Isto é obtido através de uma estrutura comum à todas as linguagens a *Common Language Runtime* (CLR). A CLR funciona como uma "máquina virtual": esta provê uma abstração para que aplicações sejam executadas numa camada própria no dispositivo (um PC, por exemplo) de forma a tornar essa tarefa independente de plataforma, pois todo o processo de execução está encapsulado nesta camada, ficando o S.O. sem a tarefa de gerenciar as aplicações e os recursos necessários por estas.

Este framework é uma evolução na forma como o Windows disponibilizava funções para sua manipulação (chamada de API), tradicionalmente apenas funções sem relação direta eram disponibilizadas. No .Net temos um conjunto enorme de classes para facilitar o desenvolvimento de aplicações, abstraindo o acesso de baixo nível ao sistema operacional, por isso o uso do termo framework. Esta forma de comunicação da aplicação com o SO permite dizer que um dos objetivos do .Net é tornar o S.O. orientado a objetos, já que todas as funções (métodos) disponibilizadas estão em classes.

Porém isso é apenas uma parte da plataforma Microsoft .Net. Pois, além disso, temos a CLR provendo interoperabilidade entre aplicações feitas em diferentes linguagens, a inclusão de uma nova linguagem denominada C# (baseada em linguagens como C++ e Java, com algumas melhorias), o ADO.Net e XML para manipular dados, e o ASP.Net para desenvolvimento de aplicações Web com mais facilidade.

1.1.1 Funções do Sistema Operacional

Todo sistema operacional define as maneiras pelas quais os programas devem chamar suas funções. O MS-DOS, por exemplo, define suas chamadas através de instruções *assembly* INT, passando e recebendo informações através de registradores da CPU. Estas instruções estão no código gerado pelo compilador ou, mais provavelmente, dentro da "biblioteca de runtime" do compilador "linkeditada" juntamente com seu programa. Isto vale para todas as linguagens como Clipper, C, Basic, Pascal e COBOL. Em 1985, a Microsoft introduziu uma grande novidade em seu novo "ambiente operacional" Windows: a maneira de chamar o sistema era através de funções em linguagem de alto nível, a princípio usando a linguagem C. Não era mais necessário se preocupar com registradores da CPU. O Uso de funções em C foi sem dúvida uma grande vantagem e permitiu que o próprio Windows fosse bem mais complexo e capaz, visto que os programas poderiam usar seus recursos com maior facilidade [SAN 2004].

O esquema de usar funções, no entanto, foi logo mostrando seus problemas. Um deles é que as funções não estavam formalmente agrupadas, nem mesmo para preservar o "*handle*" usado pelas mesmas funções. Por causa disto, logo surgiram "bibliotecas de classe" em C++, como a OWL da Borland e a MFC da própria Microsoft, para facilitar a programação. Lá por 1995, a MFC era de fato a "API do Windows", pelo menos para criar a estrutura principal dos programas como para suas janelas [SAN 2004].

Por outro lado, ferramentas como Visual Basic e Delphi introduziram um modelo completamente diferente: o usuário manipulava métodos, propriedades e eventos de

componentes, que por sua vez se responsabilizavam por chamar a API do Windows. Raramente o programador chamava a própria API diretamente. Ficamos desta forma com três maneiras diferentes de programar para Windows [SAN 2004]:

- Diretamente na API, uma forma pouco usada pela alta complexidade e baixa produtividade;
- Com bibliotecas de classes como a MFC, um recurso disponível apenas em C++;
- Com componentes específicos como em Delphi e Visual Basic, uma maneira de alta produtividade, mas incompatível com outras ferramentas.

1.1.2 Vantagens da .Net Framework

A principal idéia da .Net Framework é usar um modelo baseado em componentes como a única maneira de programar para o sistema operacional. Este é o modelo mais flexível e produtivo entre todos os disponíveis, bastante semelhante do Delphi ou Visual Basic. Isto, no entanto não é uma tarefa fácil [SAN 2004]:

- O modelo de objetos deve ser definido no próprio sistema operacional;
- Devem ser suportadas várias linguagens de programação;
- Diversos conceitos como propriedades e eventos devem ser suportados nativamente, não apenas os métodos e campos, como normalmente ocorre;
- Criar instâncias e até mesmo herdar uma classe da outra deve ser permitido, mesmo que só tenhamos o código binário disponível e não saibamos sequer a linguagem de desenvolvimento da classe original;
- Os objetos devem ser "auto-documentados"; ou seja, incluir informações detalhadas dos tipos que estão lá dentro. Este recurso não só facilita a chamada das classes, mas também é fundamental para permitir a validação do uso e manutenção da integridade do sistema de tipos em tempo de execução;
- O gerenciamento de memória deve ser feito pelo sistema operacional, para que um programa possa "passar um objeto" para outro programa sem dificuldades e não precisar se preocupar com alocação de memória;
- Deve existir uma preocupação com controle de versões; um programa pode precisar de uma versão mais antiga e outro de uma versão mais nova da mesma classe; ambos devem ser satisfeitos, mesmo que estejam rodando ao mesmo tempo.

O esquema acima é muito semelhante ao implementado pelo Delphi da Borland quando usamos "pacotes de runtime". Vale lembrar que os desafios acima foram enfrentados sem sucesso pela "*Next*", empresa fundada pelo Steve Jobs depois que ele saiu da Apple e também pela Taligent, esforço comum da Apple e IBM para criar justamente um "framework universal". Para encarar o desafio, a Microsoft colocou à frente do projeto a pessoa que já tinha liderado um projeto semelhante com sucesso: Anders Hejlsberg, o criador do Delphi. Pode-se dizer que os "pacotes de runtime" do Delphi são um "protótipo" do que foi feito no ".Net Framework". No entanto o ".Net Framework" foi bem além do que existia no Delphi [SAN 2004]:

- São suportadas diversas linguagens de alto nível: é possível criar um componente em uma linguagem, C# por exemplo, e uma classe derivada em outra, VB, por exemplo;
- Os componentes e sua descrição (Metadata) estão sempre no mesmo arquivo e não podem ser separados;
- O gerenciamento de memória é feito pelo sistema operacional e não pelos programas;

- O sistema de tipos não pode jamais ser violado; os "casts inseguros" não existem;
- Os tipos "simples" com o inteiro e "*double*" podem ser "derivados" de um ancestral comum ter propriedades e métodos. Isto é feito sem que haja um custo adicional para usos mais simples, como operações aritméticas;
- O suporte a "reflections" (informações de tipo em tempo de execução) foi bastante expandido;
- Todo "executável .Net" (na verdade chamados de “assembly:assemblies") possuem, além do nome, informações detalhadas de versões em três partes: número principal, secundário e de manutenção. Existe um mecanismo sofisticado de resolução de versões, baseado em regras (heurísticas) do sistema operacional, incluindo também sugestões do programa, como por exemplo "use a versão de manutenção mais nova a não ser que o programa exija uma versão específica";
- Os executáveis .Net são independentes do sistema operacional. Basta que haja um "sistema de runtime" que possa compilar os programas para a CPU real e que as classes necessárias estejam implementadas para o programa rodar. Nada nos programas compilados os amarra a alguma CPU ou sistema operacionais.

1.2 Mono

O Projeto Mono é uma iniciativa da comunidade do ‘Software Livre’ para desenvolver uma versão do *Microsoft .Net Framework*, baseada em GNU/Linux, como Software Livre. Ele incorpora componentes-chaves do .Net, incluindo um compilador para linguagem de programação C#, um compilador *just-in-time* (JIT) para o *Common Language Runtime* (CLR), e um *suite* completo de bibliotecas de classe [QEL & JAC 2003].

O projeto criado em 2001 tem o objetivo de permitir que os desenvolvedores criem aplicações .Net que rodem sobre o Windows ou qualquer plataforma suportada pelo *Mono*, incluindo MacOS X, GNU/Linux e Unix. Ele é liderado pela Ximian, a empresa de Software Livre co-fundada por Miguel de Icaza, que levou o desktop GNOME a um grande sucesso [QEL & JAC 2003].

Miguel de Icaza criou este projeto para facilitar o desenvolvimento, assim automaticamente todos os programas desenvolvidos para .Net passam a serem híbridos, podendo ser executados em qualquer ambiente que possua um *framework .Net* instalado [QEL & JAC 2003]. Como já mencionado o .Net baseia-se em padrões controlados pela ECMA o que facilita o desenvolvimento do *Mono* para interoperabilidade com a plataforma da Microsoft, mas nem tudo está portado para *Mono*, portanto, alguns recursos da .Net ainda não estão disponíveis. A fim de resolver esse problema, vários projetos *Mono* estão em andamento, sendo que uns para o desenvolvimento do framework e outros para desenvolver aplicações especificas. Um exemplo disso é o MonoBasic que a comunidade brasileira ajuda a desenvolver. Trata-se de um compilador .Net dentro do projeto Mono. Mais adiante será comentado sobre o estado atual do projeto Mono, que tem atualmente, como principal atrativo a portabilidade do desenvolvimento de aplicações Web (*ASP.Net*) .

1.2.1 Projeto Mono Brasil

Tem por objetivo libertar os desenvolvedores de plataformas proprietárias, disponibilizando a comunidade brasileira de software uma plataforma completa e atualizada

de desenvolvimento totalmente livre, que possa ser utilizada amplamente em qualquer plataforma seja de hardware ou de software, sendo assim com total interoperacionalidade com a plataforma Windows [QEL & JAC 2003].

O projeto pretende criar uma central de informações sobre a comunidade. Incentivar o uso da plataforma em projetos de pesquisa, facilitando o desenvolvimento de novas tecnologias. Promover projetos brasileiros inovadores e fornecer todas as informações necessárias para a formação de novos desenvolvedores. Busca de recursos para desenvolvimento de projetos de visibilidade nacional [QEL & JAC 2003].

O WebSite do projeto já possui diversos contribuintes, oferece sessões de documentação, artigos, palestras, how-to, exemplos e explicações detalhadas de como instalar o Mono [QEL & JAC 2003].

1.3 DotGNU Portable.Net

Assim como o Mono o projeto DotGNU Portable.Net foi iniciado a mais de dois anos com o intuito de tornar o .Net da Microsoft multiplataforma, diminuindo com isso a dependência de desenvolvedores e companhias do S.O. Windows.

O Portable.Net tem pouca divulgação no Brasil, no entanto é um projeto que em tratando-se da portabilidade de aplicações gráficas usando Windows Forms *(veja a seção 2.2.1 para informações)* está mais avançado que o Mono. Já quanto ao desenvolvimento de aplicações Web está bem atrasado em relação ao Mono.

O Portable.Net não é um concorrente do Mono. Ambos os projetos Mono e Portable.Net complementam-se. No caso do Portable.Net isso é bem evidenciado, pois ao buscarmos os binários ou mesmo o código-fonte no *site* do projeto (www.dotgnu.org) encontraremos bibliotecas do Mono incluídas. Este também possibilita alternativas as ferramentas disponibilizadas pelo Mono, principalmente compiladores C# e VB.Net.

2 Estrutura dos Frameworks: Microsoft .Net Framework, Mono e DotGNU Portable.Net

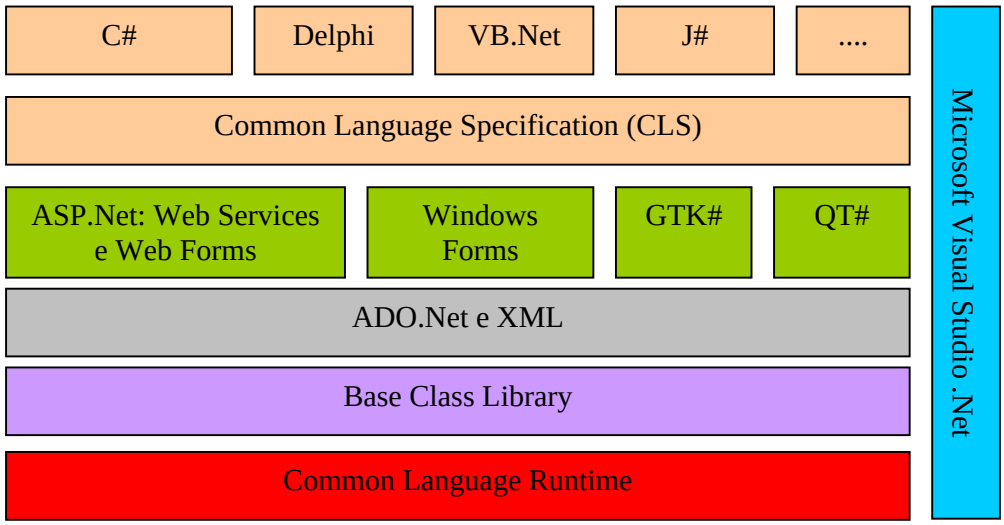

Figura 1 – Estrutura dos Frameworks

Tanto *Microsoft .Net Framework*, *Mono* e *DotGNU Portable.Net* estão divididos em camadas, conforme mostra a **figura 1** abaixo:

Na primeira e segunda camada estão as linguagens de programação e a *Common Language Specification (CLS)*. Como mencionado anteriormente o framework é 'multilinguagem', ou seja, posso desenvolver uma mesma aplicação composta por várias linguagens, desde que sejam compatíveis com a *CLS* do framework .Net. Sendo assim, é possível a integração entre códigos de aplicações criadas em diferentes linguagens .Net, permitindo o reaproveitamento de código e ganhos de produtividade.

Na terceira camada aparecem as opções disponíveis para desenvolvimento de aplicações web e aplicações gráficas (GUI). As opções disponíveis variam de acordo com o framework. Em se tratando de Microsoft .Net o que foi disponibilizado é *Windows Forms* (WinForms) para aplicações gráficas e *ASP.Net* para Web Forms e Web Services. O Mono também incluiu o *GTK#* e *QT#*, além de estar implementando a portabilidade de WinForms. Já o Portable.Net, por sua vez, está mais centrado na portabilidade de WinForms e não suporta até o momento ASP.Net. Mais adiante este assunto será mais bem abordado.

Na quarta camada temos *ADO.Net*. Trata-se da camada responsável pela manipulação de dados tanto referentes ao acesso a sistemas gerenciadores de banco de dados (SGBD's) quanto a XML. Ela provê funcionalidades básicas para manipulação de dados de forma a permitir sua extensibilidade através de providers (provedores). Este conceito permite que sejam criados novos providers, que utilizam as funções do *ADO.Net*, para permitir a comunicação com diversos SGDB's.

Na quinta camada está a *Base Class Library* (Biblioteca de Classe Base) que consiste numa série de classes para facilitar o desenvolvimento de aplicações. Podemos comparar esta camada às funções da API do Windows, com a grande vantagem de ser totalmente orientada a objetos e não apenas um conjunto de funções desconexas como ocorre em Win32.

Na última camada está a *Common Language Runtime – CLR* (Execução de Linguagem Comum). Esta camada é equivalente a *Java Virtual Machine* – JVM (Máquina Virtual de Java). Ela possui um compilador em tempo de execução (JIT – *Just in time*) para permitir que uma mesma aplicação possa rodar em diversos sistemas operacionais. Foi introduzido o conceito de *assembly*, que trata-se de executáveis e DLL's compiladas como binários intermediários, por não incluírem na compilação elementos específicos de sistema operacional. Na execução do assemblies então é feita a compilação final através da CLR.

Ao lado da **figura 1** aparece o *Microsoft Visual Studio.Net*, atingindo todas as camadas. Mas este não é o único, outros exemplos de ambientes de desenvolvimento para .Net, como o Borland Delphi 8, também possuem a mesma função. Ou seja, o que estamos tentando mostrar é que um ambiente de desenvolvimento, como Visual Studio, usufrui todas as camadas, tornando os desenvolvimentos padronizados, permitindo a integração que mencionamos logo no início dessa seção, pois aplicações desenvolvidas em diferentes compiladores (ou ambientes de desenvolvimento) podem ser compartilhadas.

2.1 Common Language Infrastructure (CLI)

O Common Language Infrastructure (CLI) é a especificação dos principais serviços do .Net Framework e que implementa a tecnologia que permite que um aplicativo seja desenvolvido em diversas linguagens de programação e executados num mesmo ambiente de execução. O CLI foi enviado pela Microsoft a ECMA (*European Computer Manufacturing Association*, Associação dos Fabricantes de Computadores da Europa), um órgão

padronizador europeu equivalente à ISO (*International Standards Organization*, Organização de Padronização Internacional) com cede nos Estados Unidos [TOL 2004].

No dia 13 de dezembro de 2001 a ECMA aprovou por unanimidade a especificação 1.0 do CLI e da linguagem C#, tornando-os padrões abertos de mercado controlados agora pelo ECMA e não mais pela Microsoft. Qualquer empresa pode visitar o site da organização (http://www.ecma.ch) e baixar o arquivo com a especificação, implementando a sua própria versão do CLI, em qualquer sistema operacional [TOL 2004].

Para acelerar a adoção destes padrões pelo mercado, a Microsoft, que já possui o código utilizado no.Net Framework comercial, realizou duas implementações do CLI, uma delas para Windows XP e outra para FreeBSD, versão de Unix de código aberto muito utilizada pelo mundo acadêmico [TOL 2004].

Pelo anúncio realizado em Cambridge, a Microsoft tornou pública estas implementações através de seu web site MSDN (http://msdn.microsoft.com/, pesquise por SSCLI) para ser utilizada em projetos de pesquisas de universidades ou dentro do currículo programático dos cursos de tecnologia das mesmas. Trata-se aproximadamente de 1,9 milhão de linhas de código disponíveis para *download* [TOL 2004].

Mesmo que você não esteja no mundo acadêmico pode se beneficiar desta iniciativa. É uma ótima oportunidade para entender como funciona o ambiente de execução do .Net Framework e seus recursos técnicos como o JIT (*Just In Time*) Compiler ou o Garbage Colector. Está presente no código também o código do compilador C# (escrito em linguagem C) e outro de JScript (este já escrito em C#) [TOL 2004].

A licença da Microsoft permite a modificação e expansão do código disponível, mas não permite a comercialização das novas versões [TOL 2004].

Esta iniciativa da Microsoft originou os projetos Mono e DotGNU Portable.Net, que, como já mencionados, visam portar o .Net Framework para outras plataformas, como GNU/Linux e MacOS X, pois todos estes projetos também seguem a CLI. No site do Projeto Mono Brasil (http://monobrasil.softwarelivre.org) é destacado que o objetivo é criar um clone do .Net que rode em diversas plataformas, essa idéia também é compartilhada pelo projeto DotGNU Portable.Net.

2.2 Desenvolvimento de Aplicações Gráficas (GUI)

2.2.1 Windows Forms

Windows Forms é um pacote criado para a plataforma .Net, para desenvolvimento de aplicações gráficas que tirem vantagem das funcionalidades do ambiente Windows. Compartilha os princípios de design do Web Forms, apesar de ter classes e implementações distintas. Entretanto, por motivos de consistência e facilidade de desenvolvimento, ambos os *namespaces* têm classes parecidas, como um *button*, que tem um texto, um evento *onClick* e propriedades como *Font*, *ForeColor* e *BackColor*. O Windows Forms termina com o paradigma dos aplicativos Win32.

2.2.1.1 Introdução

Para a elaboração desta seção foi utilizado como base o texto de [BUR 2004].

Todos os controles são baseados na classe **System.Windows.Forms.Control**, que trabalha diretamente com a API do Windows assim como com layout e desenho. A figura abaixo apresenta a hierarquia do controle do Windows Forms.

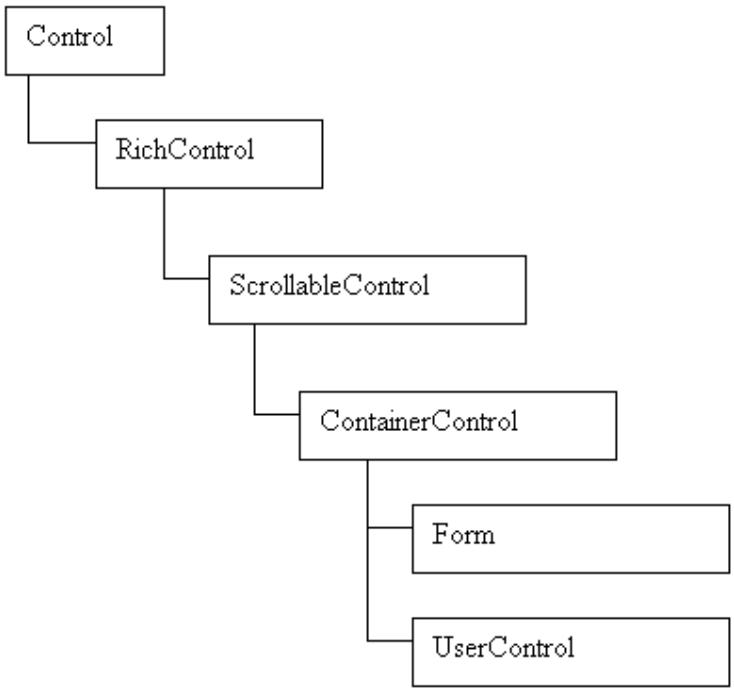

Figura 2 - hierarquia de controle do Windows Forms (fonte: [BUR 2004])

O Windows Forms tenta transformar o desenvolvimento para que este seja feito o mais rápido possível. Por exemplo, em Win32, janelas do tipo WS_BORDER e WS_CAPTION só podem ser modificadas no momento da sua criação, enquanto que WS_VISIBLE e WS_CHILD podem ser modificados após serem criadas. O Windows Forms tenta eliminar estes entraves, fazendo com que as operações ocorram de qualquer maneira, em qualquer tempo ou ordem, para produzir o efeito desejado.

Obter notificações e eventos também é mais fácil. Todos os eventos do Windows Forms são baseados na funcionalidade da *Common Language Runtime* chamada *Delegates*, que são ponteiros de função seguros. Qualquer evento em qualquer função pode ter criado um *handler*. Por exemplo, um clique em um botão:

```
public class ButtonClickForm: System.Windows.Forms.Form {
        private System.Windows.Forms.Button button1;
        public ButtonClickForm() {
// criar o botão
        button1 = new System.Windows.Forms.Button();
// adicionar o handler
        button1.Click += new System.EventHandler(button1_Click);
// colocar o botão no formulário (form)
        this.Controls.Add(button1);
}
```

```
private void button1_Click(object sender, EventArgs e) {
        MessageBox.Show("button1 clicked!");
}
```

Criamos um botão e adicionamos um *handler* a ele, que será acionado usando poucas linhas de código quando o botão for clicado. Note que, apesar do *handler* ser privado, o código que o criou tem acesso ao método, e o botão poderá disparar o método quando for clicado.

Criar um projeto Windows Forms também ficou mais simples. Com o VS.Net, é criado somente um arquivo .cs, não existindo arquivos de cabeçalho (*header*), de definição de interface, *bootstrap, resources* e biblioteca. Toda a informação necessária está contida no código do *form*. Com o uso de um arquivo único, entre outras coisas, fica mais fácil criar processos fora do ambiente inicial, no caso o VS.Net, para outras linguagens .Net.

2.2.1.2 Basic Layout

Para a elaboração desta seção foi utilizado como base o texto [BUR 2004].

Criar janelas que respondam corretamente a um redimensionamento, por exemplo, é uma tarefa complexa. Windows Forms trata isso usando o Basic Layout, que é composto basicamente por âncoras e docas. O *control* possui a propriedade *Anchor* que define a distância mínima que um componente terá de determinada borda. Por exemplo, um botão com o *Anchor* setado como AnchorStyles.BottomRight manterá sempre a mesma distância da borda inferior direita se a janela for dimensionada.

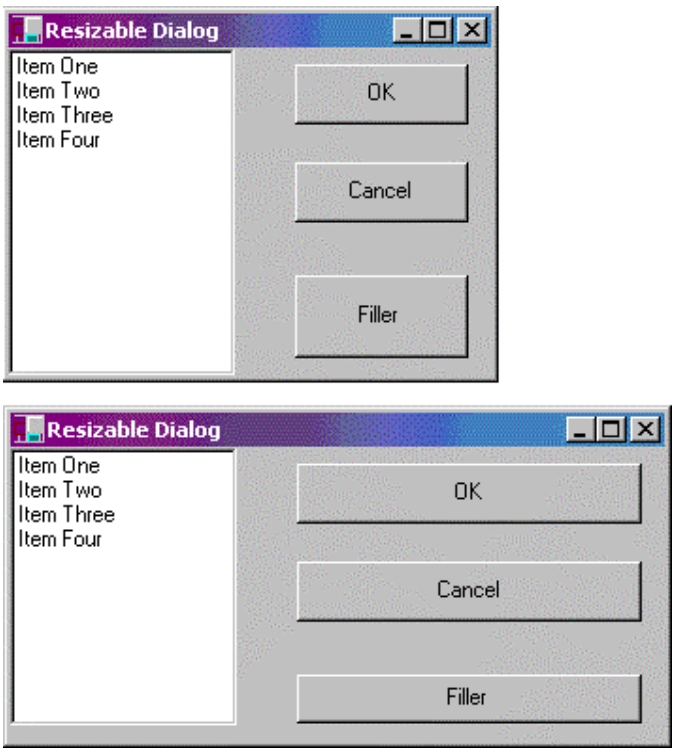

Figura 3 – Uma dialog box redimensionável (fonte: [BUR 2004])

Já a propriedade *Dock* descreve a qual borda um componente deve estar amarrado. Ele deve manter-se o mais próximo possível de tal borda, preenchendo-a. No exemplo abaixo, o *ListBox* tem o *Dock* setado à esquerda, e os botões ancorados em cima, à esquerda e a direita da janela, para que mantenham suas posições e tamanho relativo. Não foi necessário escrever nenhum código para isto.

2.2.1.3 GDI+

Para a elaboração desta seção foi utilizado como base o texto de [BUR 2004].

O Windows Forms ainda tira vantagem do GDI+, que é a próxima geração de desenvolvimento 2D da Microsoft. É totalmente orientado a objeto, com diversos objetos gráficos criados para serem simples de usar. Desenvolvedores podem utilizar diversas novas características, como *blending*, gradiente, textura, *anti-aliasing*, e usar formatos fora do padrão bitmap. As novas características devem proporcionar melhores aplicações com menos esforço de desenvolvimento.

As operações de objetos gráficos, disponíveis pelo System.Drawing.Graphics, são executadas através do GDI+. A figura 4 mostra um botão com gradiente:

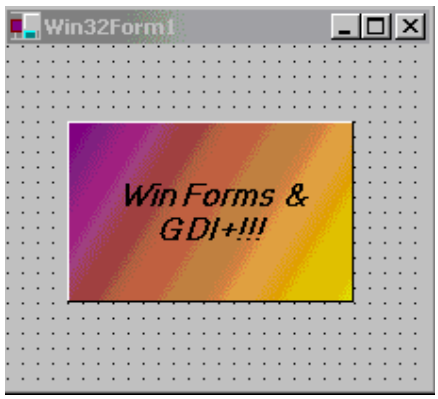

Figura 4 – Botão criado usando o GDI+ (fonte: [BUR 2004])

O código para gerar isso pode ser visto abaixo.

```
public class GradientButton : Button {
// para guardar as definições de cores
private Color startColor;
private Color endColor;

// para desenhar o texto
private static StringFormat format = new StringFormat();
public GradientButton() : base() {
// inicializar as cores
startColor = SystemColors.InactiveCaption;
endColor = SystemColors.ActiveCaption;
format.Alignment = StringAlignment.Center;
format.LineAlignment = StringAlignment.Center;
}

/// <summary>
```

```
/// cor final do gradiente
// </summary>
 public Color EndColor {
get {
return this.endColor;
}
set {
this.endColor = value;
// repaint se necessario
if (this.IsHandleCreated && this.Visible) {
Invalidate();
}
}
}

/// <summary>
/// cor inicial do gradiente
// </summary>
public Color StartColor {
get {
return this.startColor;
}
set {
this.startColor = value;
// repaint se necessario
if (this.IsHandleCreated && this.Visible) {
Invalidate();
}
}
}

protected override void OnPaint(PaintEventArgs pe) {
// pintar o fundo normal p/ obter as bordas, etc.
base.OnPaint(pe);
Graphics g = pe.Graphics;
Rectangle clientRect = this.ClientRectangle;
// p/ nao pintar em cima da borda
clientRect.Inflate(-1,-1);
// cria o gradiente
Brush backgroundBrush = new LinearGradientBrush(
new Point(clientRect.X,clientRect.Y),
new Point(clientRect.Width, clientRect.Height),
startColor,
endColor);
// preenche o fundo com o gradiente
g.FillRectangle(backgroundBrush, clientRect);
// desenha o texto no centro
g.DrawString(this.Text, this.Font, new SolidBrush(this.ForeColor), clientRect, format);
}
}
```

2.2.1.4 Acesso profundo ao sistema

Para a elaboração desta seção foi utilizado como base o texto de [BUR 2004].

O Windows Forms possibilita que o desenvolvedor tenha acesso ao sistema, caso ele encontre dificuldades em criar alguns casos especiais ao desenhar algo mais criativo. Todos os controles têm um *handle*, que dá acesso ao *handle* da janela, assim como os objetos GDI. Mas o *Control* ainda possui um método chamado *WndProc*, que pode ser sobrescrito (*override*) para adicionar-se *handles* para mensagens não suportadas pelo Windows Forms.

Por exemplo, uma aplicação pode ter que responder ao WM_COMPACTING, que não passa de um broadcast para todas as janelas quando o sistema detecta baixa memória. Tal função não está ainda implementada no Windows Forms, de modo que adicionar um *handle* como segue:

```
/// <summary>
/// Summary description for Win32Form1.
/// </summary>
public class CompactableForm : System.Windows.Forms.Form {
public event EventHandler Compacting;
protected override void OnCompacting(EventArgs e) {
// ver se o runtime pode liberar algo
System.GC.Collect();
// chamar qualquer handler.
if (Compacting != null) Compacting(this, e);
}
protected override void WndProc(ref Message m) {
case (m.msg) {
case win.WM_COMPACTING: OnCompacting(EventArgs.Empty);
break;
}
base.WndProc(m);
}
}
```

Como se pode observar, com poucas linhas, qualquer classe pode utilizar o novo método *CompactableForm* para ser notificado e responder quando o sistema estiver buscando por recursos não usados.

2.2.1.5 WinForms em Mono

O *namespace* Windows.Forms não está totalmente portado para Mono. A implementação está em andamento, porém ainda há muito a fazer para que aplicações WinForms rodem em outras plataformas que não o Microsoft Windows.

Inicialmente o Mono estava usando como base a biblioteca *Wine* que já existia para permitir que aplicações Win32 fossem executadas no GNU/Linux. Mas devido a dificuldade de portabilidade desta biblioteca para também rodar aplicações .Net com WinForms o projeto Mono decidiu começar algo novo. No site do Mono (www.go-mono.com) há algumas telas de aplicações WinForms rodando no GNU/Linux.

2.2.1.6 WinForms em dotGNU Portable.Net

O dotGNU Portable.Net está bem mais avançado que o Mono na portabilidade de WinForms para outras plataformas que não o Microsoft Windows.

Vários métodos e classes do namespace Windows.Forms ainda não estão portados, o que deve ser feito em breve. Uma aplicação simples com WinForms já pode executada no GNU/Linux, como mostra a **figura 5**.

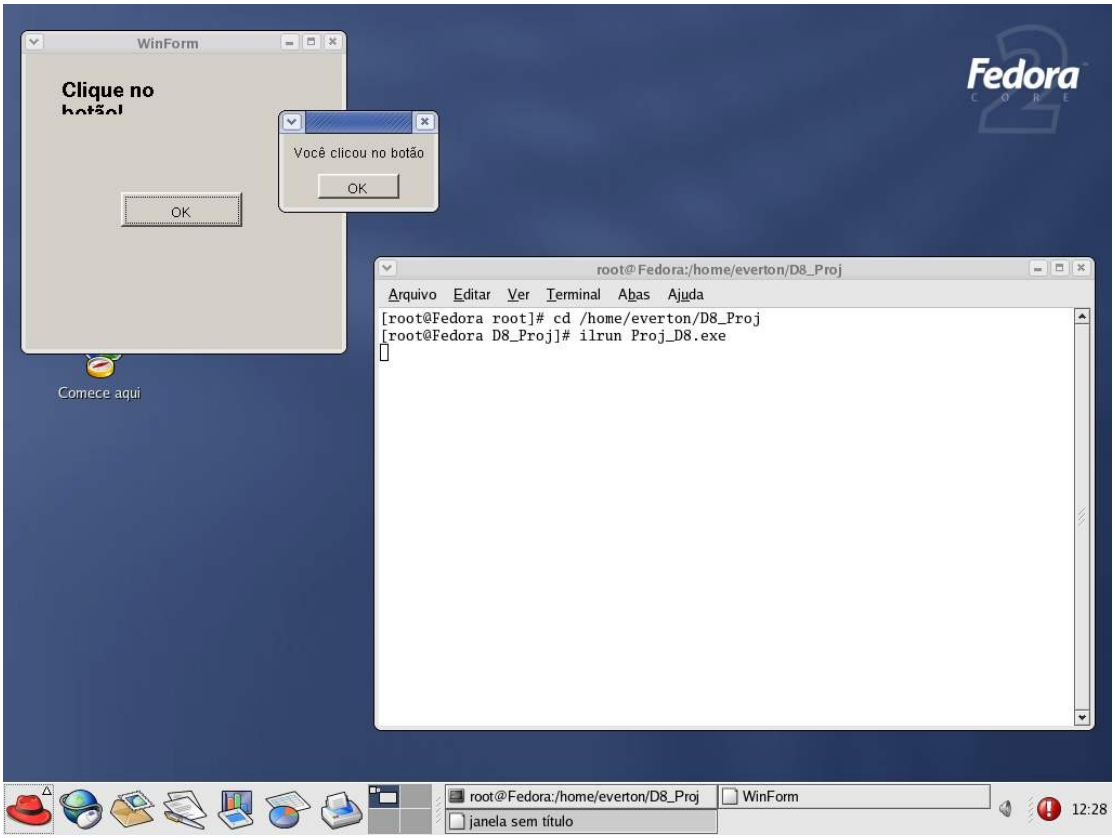

Figura 5 – Aplicação com WinForms rodando na distribuição GNU/Linux Fedora Core 2 pelo DotGNUPortable.Net

O exemplo da **figura 5** foi desenvolvido usando o ambiente Delphi 8 da Borland. Vale ressaltar que o mesmo executável gerado pelo Delphi 8, em Windows, foi rodado no GNU/Linux pelo Portable.Net, usando inicialmente o comando *ilrun* para especificar que a aplicação deveria rodar pelo Portable.Net, seguido do nome do executável. Nota-se que houve corte no texto "Clique no botão" o que não acontece ao rodar a aplicação na plataforma Windows.

Para saber o status atual da portabilidade de WinForms para Portable.Net, o *site* www.dotgnu.org disponibiliza informações sobre o andamento de cada implementação, como quais classes e respectivos métodos estão portados.

2.2.2 GTK#

Para determinados usuários, uma interface gráfica pode ser indispensável. Para outros no mínimo, desejável. Por isso, os programadores precisam conhecer uma ferramenta de programação capaz de gerar interfaces amigáveis, como o GTK [QEL & JAC 2003].

O GTK+ é um conjunto de ferramentas para desenvolver interfaces GUI, bastante completo, que possui uma grande quantidade de extensões. É prática comum que os projetos de software livre baseados em GTK+ liberem seus componentes mais genéricos através de bibliotecas. É chamado de GIMP *toolkit* porque se originou do programa da manipulação da imagem do GNU (GIMP), mas GTK tem sido usado em um grande número projetos de software, incluindo o projeto GNU *Network ObjectModel Environment* (GNOME), que serviu para populariza-lo. O GTK+ é implementado na linguagem C, mas possui interfaces (*wrappers*) para diversas outras linguagens, como *C++*, *Python*, *Perl*, *Eiffel*, *Ada*, *Ruby*, etc. oferecendo grande liberdade de escolha [QEL & JAC 2003].

O GTK+ é desenvolvido especialmente para a plataforma UNIX, mas possui uma versão Windows. O conjunto de ferramentas GTK+ é baseado no padrão de projeto *Model-View-Controller*, que é a base de muitas bibliotecas modernas de interface gráfica. Este padrão visa separar o objeto de aplicação (*model*) de forma como ele é apresentado (*view*) e da forma como o usuário o controla (*controller*). Esta divisão é vantajosa pois código de interface é freqüentemente alterado e pode ser baseado em um *framework*. Trata-se de um modelo baseado em eventos [QEL & JAC 2003].

2.2.2.1 Arquitetura do GTK+

Foi projetado de forma modular, a biblioteca GTK é baseada em outras quatro bibliotecas [QEL & JAC 2003]:

- Glib: constitui uma biblioteca de utilitários gerais, que pode ser utiliza independentemente do GTK+. Inclui funções como um alocador da memória com suporte estendido à depuração, funções para uso de listas ligadas, funções para aumentar a probabilidade do programa, etc.
- Atk: biblioteca que suportada acessibilidade. Pango: biblioteca que suporta internacionalização do GTK+.
- Gdk: biblioteca responsável pelas funcionalidades visuais. Constitui um conjunto de funções gráficas úteis para que o programador não precise lidar diretamente com a Xlib (biblioteca gráfica do XFree86).

Estas bibliotecas são básicas e muitos programas GTK+ raramente precisam acessá-las diretamente. A única exceção é a Glib, que define os tipos de dados que devem ser usados nos programas GTK para facilitar o porte para diferentes arquiteturas [QEL & JAC 2003].

O GTK contém os *widgets* que podem ser utilizados pelo programador, além de toda a estrutura para a criação de *widgets* novos. Todo e qualquer componente gráfico do GTK+ é definido por um tipo de variável especial, o *GtkWidget*, que contém informações como nome, cor, posição na tela, etc [QEL & JAC 2003].

Um aspecto importante de um *framework* GUI é a forma como a apresentação e organização dos componentes gráficos em uma janela são feitos. Vários *frameworks* solicitam que o usuário especifique a posição exata de cada componente em relação à janela, em coordenadas cartesianas. Um exemplo deste tipo simples de organização é a suíte de

desenvolvimento do Delphi. Outros *frameworks* requerem que o usuário informe apenas a organização lógica dos componentes e a posição relativa a outros componentes, e o posicionamento final é calculado pelos algoritmos do *framework* [QEL & JAC 2003].

Tanto o GTK+ como as GUI's Java mais populares (AWT e Swing) utilizam este método. A área da janela é dividida em caixas (*boxes*), e os componentes (*widgets*) são colocados nas caixas. Cada caixa define sua forma de posicionar os componentes, e assim é definida a posição de cada componente. As caixas também são consideradas componentes e repassam as chamadas aos seus componentes, conforme o padrão de projeto Composite [QEL & JAC 2003].

Para definir a organização e o posicionamento de uma janela, basta definir as caixas que a compõem. Componentes como janela e painéis são capazes de armazenarem componentes. Eles podem armazenar diretamente um e apenas um componente. Este único componente pode ser uma caixa ou uma combinação de várias caixas, de forma que o número de componentes de uma janela não é limitado. Há componentes que não podem armazenar outros, como rótulos e botões [QEL & JAC 2003].

Pode-se dizer que em GTK+, há basicamente seis organizações de componentes que uma janela pode ter: Nenhuma Caixa (apenas um componente), Caixa Horizontal, Vertical, Tabela, Posicionamento Fixo e Combinações de Caixas. A grande vantagem de utilizar *Packing* é a flexibilidade no dimensionamento das janelas. Como os componentes não têm tamanho fixo, caso o idioma da aplicação seja alterado, o tamanho dos componentes ainda será consistente. Esta capacidade facilita a internacionalização das aplicações desenvolvidas com GTK+. Também facilita a ampliação ou redução de janelas, uma vez que a escala aplicada a janela também será aplicada a todos os seus componentes. O navegador Mozilla é um exemplo de utilização deste recurso [QEL & JAC 2003].

O GTK# (GTK-sharp) é uma ligação da linguagem C# ao *toolkit* do Gtk+ e outras bibliotecas que fazem parte da plataforma do GNOME. Criação da Ximian como um background do Gnome, é a primeira ligação entre as classes Mono e o Gnome. GTK+, e também o Glade, gnome-db e GStreamer são suportados. A grande maioria de suas bibliotecas é suportada por sistemas Unix e derivados, MacOS X e Windows e estão integradas no desktop Gnome. Há também um esforço para que se consiga exportar um widgets GTK+ para Windows e MacOS X [QEL & JAC 2003].

Várias ferramentas livres utilizam dessa linguagem para criação de interfaces, agregando ao GTK# outras linguagens como C# por exemplo. O Glade é uma das ferramentas para construção de interfaces que utiliza o GTK+ e Gnome. Suporta GTK#, pode produzir código fonte em C. Mas C++, Ada95, Python e Perl também estão disponíveis, através de ferramentas externas que manipulam o arquivo XML gerado por ele, um exemplo é a interface C++ do GTK+, conhecida como Gtkmm [QEL & JAC 2003].

Outra importante IDE de desenvolvimento é o C# Studio, que possibilita o uso C#/MONO/GTK#. O MonoDevelop também esta ficando cada vez mais interessante e funcional [QEL & JAC 2003].

À medida que cresce o número de usuários do *framework* Mono, as ferramentas se tornam mais funcionais, e consequentemente gera mais flexibilidade de desenvolvimento. É por essa razão que o GTK# vem sendo cada vez mais utilizado, e preferido pelos desenvolvedores do *framework* MONO [QEL & JAC 2003].

2.2.3 QT#

O QT# foi criado pela comunidade KDE, como o GTK# para o GNOME, e fornece uma ligação às bibliotecas de desenvolvimento do KDE. A base das bibliotecas está disponível também para Windows e MacOS X, mas são livres somente no Unix/Linux.

2.3 Desenvolvimento de Aplicações Web

2.3.1 ASP.Net

2.3.1.1 Introdução ao ASP.Net

ASP.Net é um *framework* construído sobre o .Net, utilizado para construções de aplicações Web. Um dos principais recursos do ASP.Net são os Web Forms, que tornam o desenvolvimento de *sites* dinâmicos uma tarefa bastante simples e atrativa. ASP.Net também oferece suporte à construção de XML Web Services [MOU2 2004].

Por muitos anos, os desenvolvedores têm utilizado a tecnologia ASP para construir páginas dinâmicas. Similar ao ASP o ASP.Net roda um servidor Web e permite o desenvolvimento de *sites* personalizados e com conteúdo rico, adicionando diversas melhorias e novos recursos em relação ao ASP tradicional [MOU2 2004].

O desenvolvimento de aplicações ASP.Net é similar ao desenvolvimento de aplicações Windows. O componente fundamental do ASP.Net é o Web Form. Um Web Form é uma página dinâmica que pode acessar recursos do servidor, gerando conteúdo a ser visualizado no *browser* [MOU2 2004].

Uma página Web tradicional geralmente usa script no lado cliente (*browser*) para realizar tarefas básicas. Um Web Form ASP.Net, em contrapartida, também pode rodar código no servidor como, por exemplo, acessar um banco de dados, gerar Web Forms adicionais etc [MOU2 2004].

Como um Web Form ASP.Net não se baseia em scripts do lado cliente, ele não depende do tipo de *browser* ou sistema operacional do cliente. Essa independência permite a você desenvolver um Web Form que pode ser visualizado em diversos *browsers*.

As **figuras 6** e **7** mostram a arquitetura de uma aplicação ASP.Net.

A seguir veremos algumas das principais vantagens do desenvolvimento com ASP.Net.

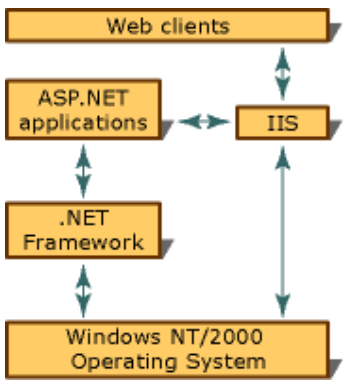

Figura 6 – Arquitetura do ASP.Net utilizando o servidor IIS da Microsoft

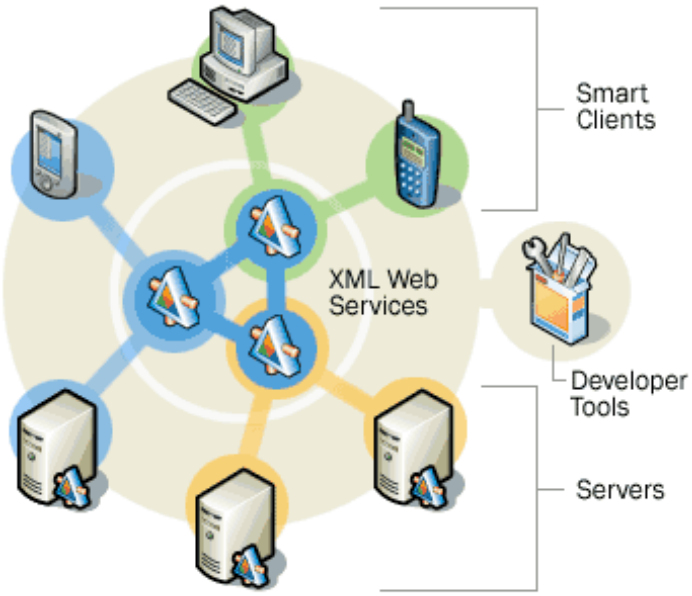

Figura 7 – Exemplo de arquitetura do ASP.Net utilizando Web Services

2.3.1.2 Um Framework, várias ferramentas e linguagens.

Como o ASP.Net faz parte do .Net Framework, você pode desenvolver aplicações Web em qualquer linguagem baseada em .Net [MOU2 2004].

Novos recursos como Web Forms e Web Controls, permitem que ferramentas de desenvolvimento, como o Visual Studio .Net, Web Matrix, C# Builder e Delphi 8, possam criar aplicações para a Web de forma rápida e prática, assim como os desenvolvedores já estavam acostumados a programar aplicações para Windows [MOU2 2004].

Não é mais necessário um profundo conhecimento em linguagens como HTML, *DHTML*, *JavaScript* ou *VBScript* para desenvolver aplicações WEB, pois a maioria do código necessário já é gerado automaticamente pelo ASP.Net [MOU2 2004].

2.3.1.3 Páginas compiladas no servidor: mais performance

No ASP.Net as páginas da aplicação são compiladas em um *assembly* que roda no servidor (nesse caso uma .DLL). Isso é diferente do ASP, que apenas interpretava o código, montava o HTML e enviava ao browser que o solicitou. Com isso, aplicações desenvolvidas em ASP.Net são no mínimo 25% mais rápidas que aplicações desenvolvidas em ASP [MOU2 2004].

Em aplicações com um grande número de acesso, esse ganho de performance torna-se ainda mais elevado, fazendo com que o ASP.Net seja realmente uma ferramenta excepcional para a criação de aplicações distribuídas na Web [MOU2 2004].

2.3.1.4 Separação entre lógica e interface

O uso de Web Forms no ASP.Net permite fazer separação entre o código de interface com o usuário e o código da lógica de programação (chamada de métodos no servidor, acesso a dados, etc.).

Você pode escolher qualquer linguagem .Net para programar a lógica da aplicação. Esse novo recurso é conhecido como *code Behind*, permitindo assim que o design das páginas seja realizado por um Web Designer, enquanto o desenvolvedor programa a parte lógica. Apesar disso, ainda é possível programar da forma convencional, utilizando código e *design* em um único arquivo modelo suportado pelo WebMatrix *(para mais informações leia a seção 4.4)* [MOU2 2004].

2.3.1.5 Controles no servidor

Um dos maiores benefícios do ASP.Net são os chamados *Server control*, que podem ser de dois tipos: *HTML Server Controls* ou *Web Server Control*. Eles encapsulam diversas funcionalidades, podem gerar automaticamente conteúdo HTML e manter seu estado entre sessões [MOU2 2004].

Controles como *DataList*, *DataGrid* e *Repeater* facilitam a publicação de conteúdo de banco de dados. No ASP tradicional você teria que montar tabelas, mesclando HTML com script e programar um loop para ler todo um *RecordSet* [MOU2 2004].

Com os novos controles, basta definir as propriedades *DataSource* e *DataMember* e utilizar o método *DataBind*, programando apenas duas ou três linhas de código. Validar entradas de usuários está bem mais simples, através da utilização dos chamados *Validation controls*. Em artigos posteriores estaremos explorando bastante a utilização desse recurso [MOU2 2004].

2.3.1.5.1 ASP.Net Server Controls

ASP.Net Server Controls são componentes que rodam no servidor e encapsulam a interface de usuário e outras funcionalidades relacionadas. Além disso, eles detectam automaticamente as características dos navegadores utilizados, gerando e enviando somente conteúdo compatível com os mesmos [MOU3 2004].

Server Controls são declarados dentro do arquivo aspx, usando “tags” customizadas que contenham o valor de atributo *runat=“server”*. Desse modo, você estará habilitando eventos no lado servidor e tratamento automáticos do estado para esses controles [MOU3 2004].

Os Server Controls possuem um modelo de programação orientado a objetos, disponibilizando propriedades, métodos e eventos que podem ser utilizados em suas aplicações, no código que roda no lado servidor. Dessa forma, você utiliza o mesmo modelo de programação que está acostumado a escrever(em aplicações Windows), para construção de aplicações Web. Isso unifica os dois mundos, antes isolados, onde programadores Windows poderão facilmente programar para Web: você não precisa mais conhecer frameworks diferentes de programação [MOU3 2004].

Outro recurso disponibilizado por diversos Servers Controls é o que chamamos de *data binding*, que permite “Ligar” um controle a um determinado atributo de uma fonte de dados. Você pode implementar esse recurso utilizando formatos especiais de expressão, que possui informações sobre qual fonte de dados você deseja ligar (imagine algo semelhante à

configuração do *DataSource/DataField* de um controle data-aware da VCL). Falaremos mais sobre o assunto nas próximas edições [MOU3 2004].

2.3.1.5.2 Tipos de Servers Controls

Podemos dividir os Server Controls em quatro categorias [MOU3 2004]:

1. *HTML Server Controls* – modelo de objetos que possuem estrutura bastante semelhante aos elementos HTML (*tags*) que eles geram;
2. *Web Server Controls* – são objetos que possuem uma implementação bem mais robusta que os *HTML Server Controls*. Seu modelo de objetos não reflete necessariamente a sintaxe HTML. Permitindo a construção de controles bem mais sofisticados (calendários, por exemplo), do que simples botões e caixas de texto;
3. *Validation Controls* – são um tipo especial de *Web Server Controls*, utilizados para validar entradas de usuários nos outros controles. Por exemplo, eles permitem a verificação de um campo requerido, se um valor está em uma determinada faixa de valores específicos ou se é de um determinado tipo;
4. *User Controls* - utilizados para adicionar elementos que se repetem em diversas páginas aspx. São geralmente usados na construção de menus, barras de ferramentas e outros elementos reutilizáveis. É algo semelhante ao modelo de *Component Template* utilizado em algumas versões do Delphi.

2.3.1.6 Depuração simplificada

Se você é um programador ASP e quiser migrar para ASP.Net esqueça os comandos *Response.Write* e *Response.End*, muito utilizados para encontrar erros em páginas ASP tradicionais (por incrível que pareça, essa é a melhor maneira de depurar uma página ASP) [MOU2 2004].

Utilizando uma ferramenta RAD (*Rapid Application Development)* para desenvolvimento de aplicações em ASP.Net (como o Borland Delphi 8), você pode usufruir de todos os recursos do depurador integrado, da mesma forma que já utilizava para o desenvolvimento de suas aplicações tradicionais [MOU2 2004].

Se você estiver usando *breakpoints* no código, por exemplo, o controle da aplicação passará para dentro da ferramenta de desenvolvimento quando o código no servidor dor disparado pelo *browser*. Você poderá executar o código passo a passo, inspecionar o valor de variáveis, adicionar *watches* e utilizar quaisquer outros recursos de depuração que já esteja acostumado [MOU2 2004].

2.3.1.7 Manipulação de erros

O .Net permite o tratamento de erros através de blocos *Try-Catch-Finally ou Try-Except / Try-Finally* no caso específico do Delphi, assim você ganha um poderoso recurso para tratamento de erros nas suas páginas Web [MOU2 2004].

No ASP tradicional, a única forma de tratar um erro é através da instrução *On Error Resume Next*, sendo muito difícil identificar o tipo de erro ocorrido. No ASP.Net é possível identificar qualquer erro e tratá-lo no lugar correto do código [MOU2 2004].

2.3.1.8 Distribuição simplificada e baseada em componentes

O conceito de componentização, muito utilizado em aplicações distribuídas, é amplamente utilizado no ASP.Net. Além dos benefícios, muitas não presentes no desenvolvimento ASP tradicional: facilidade de manutenção, reaproveitamento de código, portabilidade, escalabilidade, segurança etc [MOU2 2004].

Se você é um programador Delphi e desenvolve aplicações ASP distribuídas, deve estar acostumado a escrever objetos COM e acessá-los dentro das páginas ASP. Para fazer isso é necessário realizar o registro do componente, o que implica em etapas adicionais na hora de distribuí-los [MOU2 2004].

Já no ASP.Net, o uso de componentes é bem mais simples: basta referenciá-los no projeto. Um componente desenvolvido para o .Net dispensa a necessidade de registro, sua distribuição é feita através de uma simples cópia do *assembly* (DLL) para o servidor [MOU2 2004].

2.3.1.9 ASP.Net acessando banco de dados com ADO.Net

Ao construir aplicações ASP.Net, você utilizará o ADO.Net para acesso a dados. O ADO.Net é um framework que permite acesso à diversas fontes de dados, como Oracle, Microsoft SQL Server e quaisquer outras que possam ser disponibilizadas através de OLE DB, ODBC ou XML [MOU2 2004].

Nota: Consulte a seção 2.4 para obter mais informações sobre ADO.Net.

Apesar dos nomes semelhantes, ADO e ADO.Net não se parecem em quase nada. No ADO, por exemplo, você utiliza um *RecordSet* para disponibilizar dados na memória e acessá-los no browser. No ADO.Net não existe o *RecordSet*. Você faz uso de um *DataSet* ou *DataReader*. Essas duas classes possuem diversas propriedades e métodos para acesso a dados, tanto em modelos conectados quando desconectados [MOU2 2004].

Um *DataSet* mantém os dados em *cache* na memória, podendo ser utilizado para fazer relacionamentos entre tabelas, além de atualizar os dados no banco e otimizar as chamadas repetidas de uma mesma página. Já o *DataReader* é ideal para ser utilizado em páginas onde a manipulação de dados não é necessária, pois ele busca os dados no banco e os envia para a página, não guardando o resultado em cache [MOU2 2004].

Conhecidos os principais recursos do ASP.Net veremos agora como funciona o seu modelo de execução e o tratamento de requisições enviadas pelo browser, e como o processamento é feito no servidor [MOU2 2004].

2.3.1.10 Modelo de execução do ASP.Net

Quando o cliente requisita a página pela primeira vez, a seguinte série de eventos ocorre [MOU2 2004]:

1. O browser emite uma requisição GET HTTP servidor;
2. Um dispositivo do framework ASP.Net executa um *parser* que interpreta o código fonte e identifica todas as suas dependências;
3. A página é compilada em MSIL (*Microsoft intermediate Language* – código intermediário);

4. O *run-time* do .Net (CLR- *Common Language Runtime*) carrega e executa o código MSIL.

Quando algum usuário requisita a mesma página Web pela segunda vez, a seguinte série de eventos ocorre [MOU2 2004]:

1. O browser emite uma requisição GET HTTP do servidor;
2. O CLR carrega e executa imediatamente o código MSIL que já havia sido compilado durante a primeira requisição do usuário.

2.3.1.11 Como funcionam os Web Forms

O Web Form gera automaticamente o código HTML que é enviado para o browser, enquanto que o código de tratamento dos controles permanece no servidor Web [MOU2 2004].

Esse misto, entre interface cliente e código no servidor, é uma diferença crucial entre Web Forms e páginas Web tradicionais. Enquanto uma página Web tradicional requer que todo o código seja enviado e processado no browser, um Web Form necessita enviar apenas os controles de interface, pois o processamento da página é mantido no servidor. Esse mecanismo aumenta o limite de browsers suportados, melhorando a segurança e funcionalidade da página Web [MOU2 2004].

Web Form são comumente referenciados como páginas ASP.Net ou simplesmente "aspx". Eles funcionam como containers para o texto e os controles que você deseja mostrar no browser [MOU2 2004].

Páginas ASP.Net (.aspx) e *Active Server Pages* (.asp) podem coexistir no mesmo servidor, onde a extensão do arquivo determina quem deverá processá-la [MOU2 2004].

Com o novo recurso do ASP.Net, conhecido como *Code-Behind*, o código de execução das páginas .aspx fica em um arquivo separado (.pas, no caso do Delphi). Esse arquivo é compilado em MSIL, gerando *assemblies* (.DLL), o que protege de certa maneira a propriedade intelectual do código, uma vez que não é necessário enviar os fontes para o servidor no momento da instalação da aplicação, apenas os arquivos .aspx e os *assemblies* [MOU2 2004].

2.3.1.12 Examinando um exemplo de código ASP.Net

Vamos analisar o código e a estrutura de uma página ASP.Net (aspx) de exemplo, como mostrado a serguir:

```
    <%@    Page    language="c#"    Debug="true"    Codebehind="WebForm1.pas"
AutoEventWireup="false" Inherits="WebForm1.TWebForm1" %>
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

    <html>
      <head>
        <title></title>
        <meta name="GENERATOR" content="Borland Package Library 7.1">
      </head>

      <body ms_positioning="GridLayout">
      <form runat="server">
        <asp:button id=Button1
```

```
                    style="Z-INDEX: 1; LEFT: 62px; POSITION: absolute; TOP:
54px"
                    runat="server" text="Bot">
        </asp:button>
      </form>
    </body>
    </html>
```

Analisando o código, podemos decompor as funcionalidades de um Web Form em três níveis de atributos [MOU2 2004]:

- da página (*page attributes*) – define funções globais;
- do corpo da página (*body attributes*) – definem como uma página será mostrada;
- do formulário (*form attributes*) – definem como grupos de controles serão processados.

A tag *@Page* define atributos específicos da página que serão usados pelo *page parser* do ASP.Net e pelo compilador; você pode incluir apenas uma tag *@Page* por arquivo *.aspx.*

Os atributos da tag *body* definem a aparência dos objetos que serão mostrados no browser. Por exemplo, o atributo *ms positioning* diz respeito ao layout da página, determinando como os controles e o texto serão posicionados. Estão disponíveis duas opções:

- *FlowLayout* – controles ajustam-se automaticamente na tela, dependendo do tamanho da janela do browser;
- *Gridlayout* – controles são fixados na página através de coordenadas absolutas. Esse é o valor padrão para as páginas ASP.Net desenvolvidas no Delphi.

A *tag form* define como grupos de controles serão processados. Apesar de você pode ter diversos formulários HTML em uma página, apenas um formulário *server-side* poderá estar presente em uma página aspx [MOU2 2004].

Repare ainda o atributo *runat* nos controles, ajustado para o valor *server*, indicando que os mesmos irão rodar no servidor Web [MOU2 2004].

2.3.1.13 Web Services em ASP.Net

O Web Services veio preencher uma lacuna no mercado que era a necessidade de integração entre os mais diversos ambientes existentes que não se falavam entre si por falta de um "Padrão" e fortalecer o desenvolvimento distribuído de forma a trabalhar em conjunto com os novos recursos de comunicação da Internet [DUR 2004].

Os Web Services funcionam de forma semelhante a um componente que você instala na maquina local. Porém esse componente pode ficar residente em qualquer maquina, seja ela na sua empresa, na empresa de terceiros ou até na "China". É isso mesmo, até na "China". Então usando a *Internet* e o XML (*Extensible Markup Language*) para troca de informações você executa um método desse componente que dispara o processamento no servidor remoto e você recebe a resposta na sua aplicação. A principal diferença em relação aos componentes atuais é a padronização do XML para troca de informações, fazendo com que a comunicação do mesmo ultrapasse as barreiras dos *Firewall* impostos para segurança de nossas redes. Pois a única coisa que trafega é o XML e usando a porta 80 (Padrão) em conjunto com o já conhecido HTTP e o SOAP(Sabão) que cuida do transporte dos dados. Para troca de informação entre a aplicação e o Web Services é necessário que essa aplicação conheça o funcionamento do mesmo, isso é feito por meio da leitura do WSDL. Nada mais é que um documento ou "contrato" em XML que contém todas as regras do Web Services. Ainda temos

mais uma sigla a falar que é o UDDI, uma espécie de paginas amarelas de Web Services, cujo objetivo é encontrar facilmente os *links* para os componentes [DUR 2004].

Ao contrário do que muitos pensam, o padrão Web Services não é proprietário da Microsoft, pelo contrário é definido por um consórcio de diversas empresas WS-I de forma a manter a integração entre todas as plataformas que venham a utilizar esse padrão, permitindo assim a perfeita comunicação e troca de informações. O que aconteceu é que a Microsoft acreditou desde cedo no XML e investiu muito em todos os seus produtos, principalmente na plataforma .NET, permitindo assim a fácil e rápida integração com seus produtos [DUR 2004].

No ambiente .NET os Web Services se encaixam dentro do grupo de aplicações ASP.NET conforme visto nas **figuras 1** e **7**. A **figura 8** ilustra a arquitetura de Web Services e a **figura 9** mostra o ciclo de vida de uma aplicação Web Service.

O XML foi adotado por já ser um padrão da indústria e ser independente da plataforma adotada, sendo ainda controlado pelo World Wide Web Consortium (W3C). Facilmente você consegue transformar o mesmo em outro formato independente da aplicação entender o XML [DUR 2004].

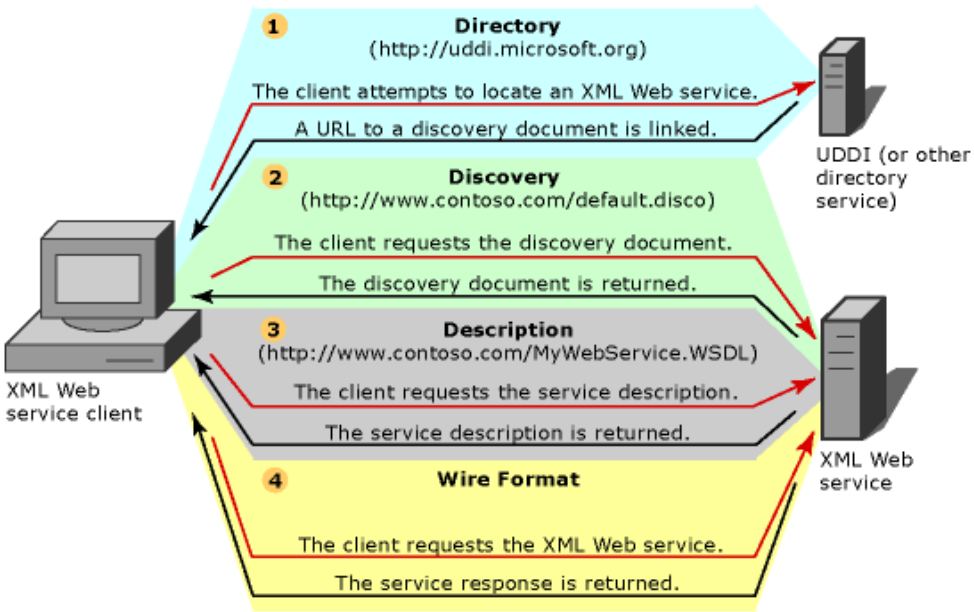

Figura 8 – Arquitetura de Web Services (fonte: msdn.microsoft.com)

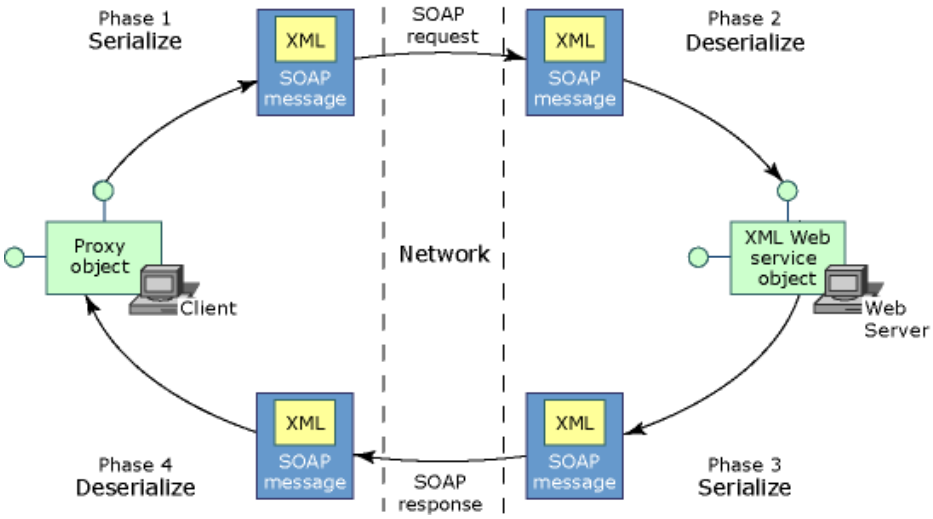

Figura 9 – Ciclo de vida de aplicações Web Services (fonte: msdn.microsoft.com)

2.3.1.14 Portabilidade de ASP.Net entre os frameworks e Servidores Web

Atualmente o Mono possui grande portabilidade de aplicações ASP.Net. Este seria um dos principais atrativos de Mono. Já dotGNU Portable.Net ainda não possui implementação equivalente ao ASP.Net da Microsoft.

Aplicações ASP.Net podem ser publicadas pelo Mono utilizando-se servidores Web como:

- **XSP**: é um servidor Web integrado ao Mono, de fácil utilização e ideal para testes. Foi implementado em C#. Roda tanto em MacOS X, GNU/Linux e Windows.
- **Apache**: seria uma alternativa mais robusta para hospedar páginas ASP.Net no GNU/Linux. Pode-se tanto utilizar o Apache nas versões 1.3 ou 2.0, no entanto é necessária a instalação de um módulo, chamado "mod_mono", para que o Apache tenha suporte a ASP.Net. Vale ressaltar que em Windows este módulo não funciona atualmente.

Para hospedar páginas ASP.Net no Windows a principal opção seria o **Microsoft Internet Information Service** (**IIS**).

2.4 Acesso a dados com ADO.Net

2.4.1 Introdução ao ADO.Net

O .Net Framework inclui uma nova tecnologia de acesso a dados, o ADO.Net, que é uma evolução do antigo ADO (*ActiveX Data Objects*), sendo projetado principalmente para ser utilizado em aplicações distribuídas na Web [MOU1 2004].

O ADO.Net é uma forma de manipulação de dados baseado em *DataSets* desconectados, disponibilizando um completo e poderoso modelo de programação para acesso a dados. Além disso, o ADO.Net usa o formato XML para transmissão de dados [MOU1 2004].

Numa aplicação cliente/servidor tradicional os componentes de acesso geralmente abrem conexões como o banco de dados, que permanecem ativas enquanto as aplicações estiverem rodando. Isso aumenta o consumo de recursos no servidor e aumenta o tráfego na rede, reduzindo a performance e escalabilidade das aplicações. No ADO.Net, as aplicações são conectadas com as fontes de dados apenas no momento de recuperar e atualizar os dados [MOU1 2004].

O ADO.Net também garante a escalabilidade requerida por aplicações Web que compartilham dados. Essas aplicações devem servir a dezenas, centenas e até milhares de usuários ao mesmo tempo. O ADO.Net não retém *locks* e conexões ativas com os bancos de dados, as quais podem consumir recursos que são limitados. Isso possibilita aumentar o número de usuários com pequeno aumento na demanda de recursos do sistema [MOU1 2004].

2.4.2 Arquitetura do ADO.Net e DataSets

Uma das principais vantagens do ADO.Net é possibilitar aos desenvolvedores manipularem dados independentemente de sua localização. O ponto central de qualquer solução de software que utiliza ADO.Net é o *DataSet.* Um *DataSet* é uma cópia em memória dos dados. A **figura 10** ilustra a arquitetura do ADO.Net [MOU1 2004].

O personagem principal nesse cenário é o *DataSet*, que constitui uma visão desconectada do banco, sem a necessidade de uma conexão ativa, possibilitando assim grande escalabilidade. Uma das maiores vantagens de se utilizar um *DataSet* é que ele disponibiliza uma visão de múltiplas tabelas / *views* e seus relacionamentos. Você pode pensar no *DataSet* como uma "miniatura" relacional do banco de dados em memória [MOU1 2004].

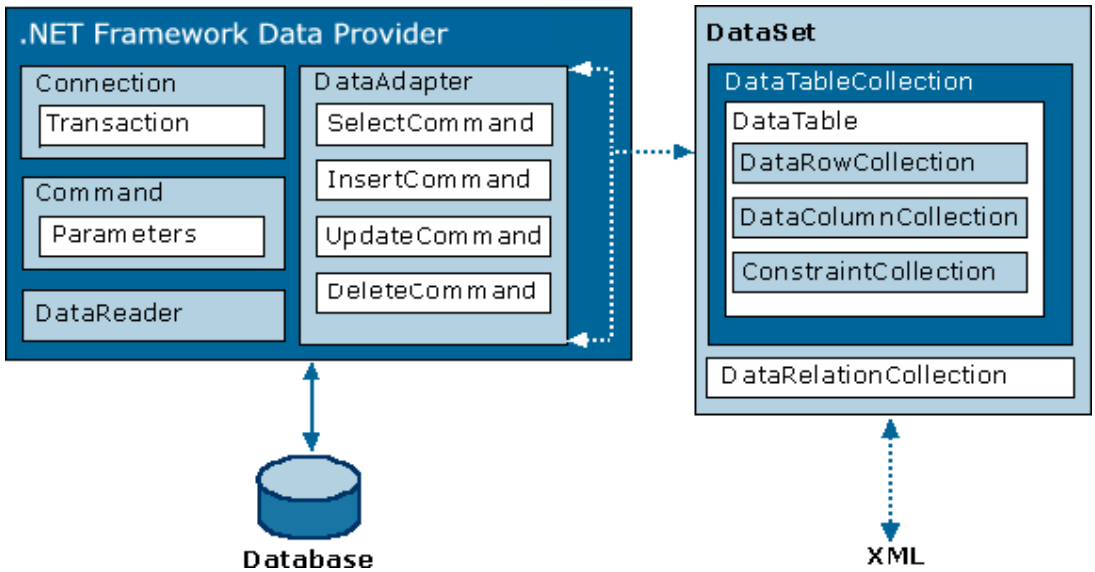

Figura 10 – Arquitetura do ADO.Net (fonte: msdn.microsoft.com)

O ADO.Net também pode ser utilizado em aplicações multicamadas que podem ser divididas da seguinte forma [MOU1 2004]:

- A camada de apresentação, que provê a interface de usuário;

- A camada de negócio, que provê a lógica de negócio;
- A camada de dados, que provê a estrutura para armazenar os dados.

Em aplicações multicamadas, os dados são geralmente passados do banco de dados para objetos presentes na camada de negócio, e destes para a camada de apresentação. Para transmitir dados de uma camada para outra, o ADO.Net permite a representação de um *DataSet* em formato XML [MOU1 2004].

2.5 Common Language Runtime (CLR)

O *Common Language Runtime* (Execução de Linguagem Comum) é o mecanismo de execução para aplicativos .Net Framework [SAN 2001]. Este mecanismo é o mesmo também para Mono e dotGNU Portable.Net. A **figura 11** ilustra a arquitetura da CLR.

O CLR oferece vários serviços, entre os quais [SAN 2001]:

- Gerenciamento de código (Carga e execução);
- Identificação de memória do aplicativo;
- Verificação de segurança de tipo;
- Conversão de II para código nativo;
- Acesso a metadados (informações de tipo avançadas);
- Gerenciamento de memória para objetos gerenciados;
- Execução de segurança de acesso ao código;
- Tratamento de exceções, incluindo exceções entre várias linguagens;
- Operação entre código gerenciado, objeto COM e DLL's preexistentes (dados e código não gerenciados);
- Automação de layout de objeto;
- Suporte para serviços de desenvolvedores (uso de perfis, depuração etc.).

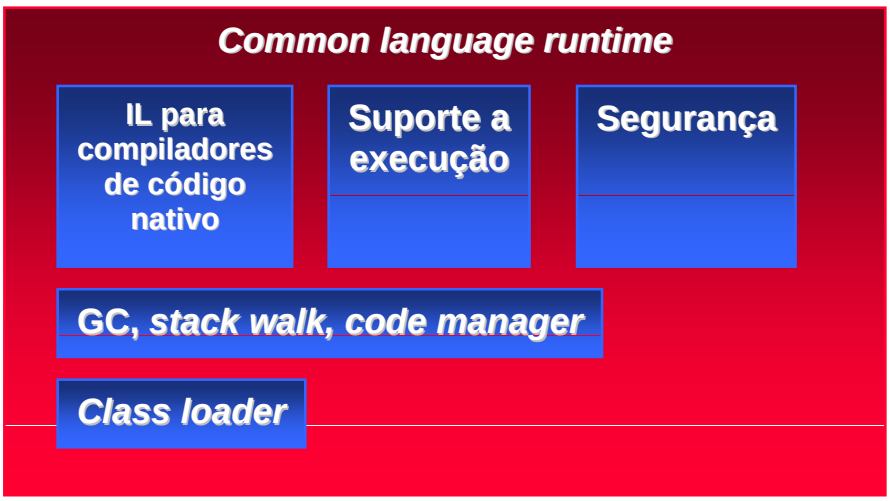

Figura 11 – Arquitetura da Common Language Runtime (CLR)

A **figura 12** mostra o modelo de execução da CLR.

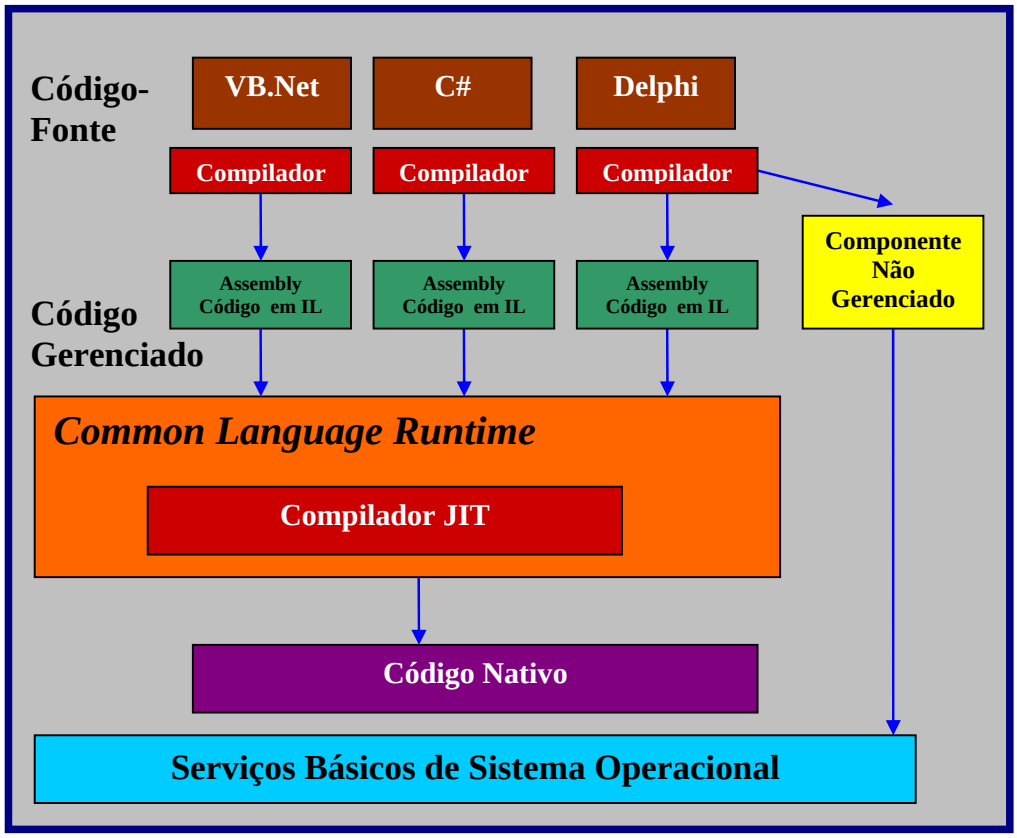

Figura 12 – Modelo de Execução da Common Language Runtime (CLR)

2.5.1 Questões Técnicas de *Runtime*

Esta seção apresenta as principais questões referentes a *runtime* (execução). São questões que na maioria das vezes servem para todos os *frameworks* – Microsoft .Net Framework, Mono e dotGNU Portable.Net –, com exceção daquelas referentes ao Microsoft Windows que estão relacionadas principalmente ao Microsoft .Net Framework.

2.5.1.1 O que é CTS – *Common Type System*?

O *Common Type System* é um sistema de tipos complexos, incorporado ao *Common Language Runtime*, que aceita os tipos e operações encontradas na maioria das linguagens de programação. O CTS aceita a implementação completa de uma ampla gama de linguagens de programação [SAN 2001].

2.5.1.2 O que é CLS – *Common Language Specification*?

A *Common Language Specification* é um conjunto de construções e restrições que serve como guia para escritores de biblioteca e escritores de compiladores. Permite que bibliotecas se integrem umas com as outras. A *Common Language Specification* é um

subconjunto de *Common Type System*. A *Common Language Specification* também é importante para desenvolvedores de aplicativos que estejam escrevendo código a ser usado por outros desenvolvedores. Quando os desenvolvedores projetam API's acessíveis publicamente seguindo as regras da CLS, essas API's são facilmente usadas de outras linguagens de programação voltadas para o *Common Language Runtime* [SAN 2001].

2.5.1.3 A *Microsoft Intermediate Language* e os JITters

Para facilitar aos escritores de linguagens porta suas linguagens para .Net, a Microsoft desenvolveu uma linguagem que apresenta afinidade com a linguagem *assembly* chamada de linguagem intermediária da Microsoft (MSIL - *Microsoft Intermediate Language*). Para compilar aplicativos para .Net, os compiladores fornecem o código-fonte como entrada e produzem MSIL como saída. A MSIL em si é uma linguagem completa com a qual você pode escrever aplicativos. Entretanto, como com a linguagem assembly, você provavelmente nunca faria isso exceto em circunstâncias incomuns. Como a MSIL é a própria linguagem, cada equipe de compiladores toma sua própria decisão cobre o quanto da MSIL ele suportará. Entretanto, se você for escritor de compilador e quiser criar uma linguagem que interopera com outras linguagens, deve se restringir aos recursos especificados pelo CLS [ARC 2001] .

Ao compilar um aplicativo C# ou qualquer aplicativo escrito em uma linguagem compatível com CLS, o aplicativo é compilado em MSIL. Essa MSIL é então compilada mais uma vez em instruções nativas da CPU quando o aplicativo é executado pela primeira vez pelo CLR. (Na realidade, somente as funções chamadas são compiladas na primeira vez em que são invocadas). Vejamos o que realmente está acontecendo sob o capô [ARC 2001]:

1. Você escreve o código-fonte em C#.
2. Você então o compila utilizando o compilador dor C# (csc.exe) em um EXE.
3. O compilador do C# gera a saída do código da MSIL e um manifesto em uma parte de leitura do EXE que tem um cabeçalho PE (executável portável para Win32) padrão.
4. Até aqui, tudo bem. Mas eis a parte importante: quando o compilador gera a saída, ele também importa uma função chamada _CorExeMain a partir o tempo de execução da .Net.
5. Quando o aplicativo é executado, o sistema operacional carrega o PE, bem como qualquer biblioteca de vínculo dinâmico (Dlls) dependente, como aquela que exporta a função CorExeMain (mscoree.dll), da mesma maneira como faz com qualquer PE Válido.
6. O utilitário de carga do sistema operacional então pula para o ponto de entrada dentro do PE, que é colocado aí pelo compilador do C#. Mais uma vez, essa é exatamente a maneira como qualquer outro PE é executado no Windows.
7. Mas como o sistema operacional obviamente não pode executar o código da MSIL, o ponto de entrada é somente um pequeno stub que pula para a função CorExeMain em mscoree.dll.
8. A função CorExeMain inicia a execução do código da MSIL que foi colocado no PE.
9. Como o código da MSIL não pode ser executado diretamente – porque ela não está em um formato de máquina executável – o CRL compila a MSIL utilizando um JITter (compilador just-in-time[JIT]) em instruções de CPU nativas ao processar a MSIL. A compilação JIT ocorre apenas quando os métodos no programa são

chamados. O código compilado do executável é armazenado em cache na máquina e é recompilado somente se houver alguma alteração no código-fonte.

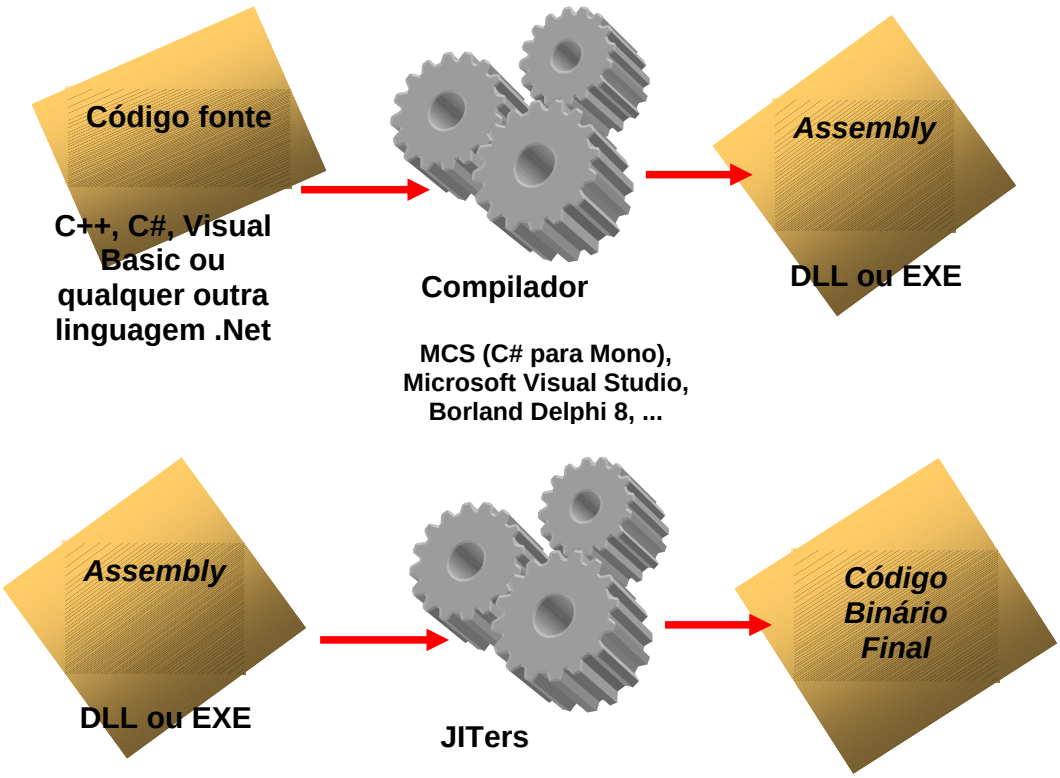

Figura 13 – Processo de Compilação

A **figura 13** ilustra o processo de compilação. Um JITter que pode ser utilizado para converter a MSIL em código nativo, dependendo das circunstâncias é este:

- **Geração de código em tempo de instalação.** A geração de código em tempo de instalação compila um assembly interno no código binário específico da CPU, da mesma maneira que um compilador C++ o faz. Um *assembly* é o pacote de código que é enviado para o compilador. Essa compilação é feita em tempo de instalação, quando o usuário final muito provavelmente deve notar que o assembly está sendo compilado *Just-in-time*. A vantagem de geração de código em tempo de instalação é que ele permite compilar o *assembly* inteiro só uma vez antes de você executá-lo. Como o *assembly* inteiro é compilado, você não tem de se preocupar com questões de desempenho intermitente toda vez que um método em seu código for executado pela primeira vez. É como pagar à vista por um plano de férias do tipo *time-share* (de compartilhamento de tempo). Embora que pagar o plano de férias de uma vez seja doloroso, a vantagem é que você nunca tem de se preocupar em pagar por acomodações novamente. Quando e se você utiliza esse utilitário depende do tamanho de seu sistema específico a seu ambiente de instalação/distribuição. Em geral, se for criar um aplicativo de instalação para sistema, você deve ir em frente

e utilizar esse JITer de modo que o usuário tenha uma versão completamente otimizada do sistema da maneira como ele recebe o produto [ARC 2001].

2.5.1.4 O que é código gerenciado e dados gerenciados?

Código gerenciado é escrito voltado para serviços do **Common Language Runtime**. Para atender a esses serviços, o código deve fornecer um nível mínimo de informações (metadados) para o *run-time*. Todo o código C#, Visual Basic.Net e JScript.Net é gerenciado por padrão. O código Visual Studio.NET C++ não é gerenciado por padrão, mas o compilador pode produzir código gerenciado especificando uma chave de linha de comando (/CLR) [SAN 2001].

Intimamente relacionados ao código gerenciado estão os dados gerenciados – os dados que são alocados e desalocados pelo *garbage collector* do *Common Language Runtime*. Todos os dados C#, Visual Basic.NET e JScript.NET são gerenciados por padrão. Os dados C# podem, no entanto, ser marcados como não-gerenciados pelo uso de palavras-chave especiais. Os dados do Visual Studio.NET C++ são não-gerenciados por padrão (mesmo quando se usa a chave /CLR), mas ao serem usadas às extensões gerenciadas para C++, uma classe pode ser marcada como gerenciada pelo uso da palavra-chave gc. Como o nome sugere, isso significa que a memória para instância da classe é gerenciada pelo *garbage collector*. Além disso, a classe se torna um membro totalmente participante da comunidade .NET Framework, com os benéficos e restrições inerentes. Um exemplo de benefício é a interoperabilidade adequada com classes escritas em outras linguagens (por exemplo, uma classe de C++ gerenciada pode herdar de uma classe do Visual Basic). Um exemplo de restrição é que uma classe gerenciada só pode herdar de uma classe base [SAN 2001].

2.5.1.5 *Assemblies*

2.5.1.5.1 O que é um *assembly*?

Um *assembly* é um *building block* principal de um aplicativo .NET Framework. É um conjunto de funcionalidades incorporado, com versão e instalado como uma unidade de implementação (como um ou mais arquivos). Todos os tipos e recursos gerenciados são marcados como acessíveis somente dentro de usa unidade de implementação, ou como acessíveis pelo código fora dessa unidade [SAN 2001].

Os *assemblies* são autodescritivos por meio de seus manifestos, que é parte integral de cada *assembly*. O manifesto [SAN 2001]:

- Estabelece a identidade do *assembly* (na forma de um nome de texto), versão, cultura e assinatura digital (se o *assembly* for compartilhado entre aplicativos).
- Define quaisquer arquivos (por nome e *hash* de arquivo) compõem a implementação do *assembly*.
- Especifica os tipos e recursos que compõem o *assembly*, incluindo quis são exportados do *assembly*.
- Dispõem em itens as dependências de tempo de compilação em relação a outros assemblies.
- Especifica o conjunto de permissões necessárias para que o *assembly* seja executado corretamente.

Essas informações são usadas no *run-time* para resolver referências, realizar políticas de vinculação de versões e validar a integridade de assemblies carregados. O *run-time* pode determinar e localizar o *assembly* de qualquer objeto em execução, desde que todos os tipos estejam carregados no contexto de um *assembly*. Os assemblies também são a unidade em que as permissões de segurança de acesso ao código são aplicadas. A evidência da identidade de cada *assembly* é considerada separadamente ao determinar quis permissões conceder ao código que ele contém [SAN 2001].

A **figura 14** ilustra a arquitetura de um *assembly*. Já a **figura 15** mostra a estrutura de metadados contida num *assembly*, que inclui o seu manifesto.

A natureza autodescritiva dos *assemblies* também ajuda a tornar viáveis a instalação com impacto zero e a instalação de *XCOPY* [SAN 2001].

Figura 14 – Arquitetura de um Assembly

Figura 15 – Estrutura de Metadados de um Assembly

2.5.1.5.2 O que são *assemblies* privados e *assemblies* compartilhados?

Um *assembly* privado é usado somente por aplicativos e é armazenado no diretório de instalação desse aplicativo (ou em um subdiretório dele). Um *assembly* compartilhado é o que pode servir de referência a mais de um aplicativo. Para um *assembly* ser compartilhado, ele deve ser criado explicitamente com essa finalidade, sendo dado a ele um nome criptograficamente forte (o que se chama de nome compartilhado). Por contraste, um nome de assembly privado precisa ser exclusivo somente dentro do aplicativo que o utiliza [SAN 2001].

Fazendo uma distinção entre assemblies privados e compartilhados, introduzimos a noção de compartilhamento como uma decisão explicita. Simplesmente instalando-se *assemblies* privados em um diretório de aplicativos, é possível garantir que esse aplicativo será executado somente com os bits com que foi criado e instalado. As referências a *assemblies* privados somente serão resolvidas localmente ao diretório provado de aplicativos [SAN 2001].

Existem diversas razões para decidir criar e usar assemblies compartilhados, como a capacidade de expressar políticas de versão. O fato de que *assemblies* compartilhados possuem um nome criptograficamente forte significa que somente o autor do *assembly* possui a chave para produzir uma nova versão desse *assembly*. Assim, se você fizer uma declaração da política de que deseja aceitar uma nova versão de um *assembly*, poderá ter uma certa confiança de que atualizações de versão serão controladas e verificadas pelo autor. Caso contrário, não precisará aceitá-las [SAN 2001].

No caso de aplicativos instalados localmente, um *assembly* compartilhado normalmente é instalado explicitamente no cache do *assembly* global de *assemblies* mantidos pela .NET Framework. O segredo para os recursos de gerenciamento de versão da .NET Framework é que o código que foi transferido por *download* não afeta a execução dos aplicativos instalados localmente. O código transferido por *download* é colocado em um cache de *download* especial e não se encontra globalmente disponível na máquina, mesmo que alguns dos componentes transferidos por *download* sejam criados como *assemblies* compartilhados [SAN 2001].

As classe que acompanham o .NET Framework são todas criadas como *assemblies* compartilhados [SAN 2001].

2.5.1.5.3 Se eu quiser criar um *assembly* compartilhado, será necessário o *overhead* de gerenciamento e assinatura de pares de chaves?

Criar um *assembly* compartilhado não envolve trabalhar com chaves criptográficas. Somente a chave pública é estritamente necessária quando a *assembly* está sendo criado. Os compiladores voltados para a .NET Framework oferecem opções de linhas de comando (ou usam atributos personalizados) para fornecimento da chave pública ao criar o *assembly*, ele deve ser totalmente assinado com a chave privada correspondente. Isso é feito usando-se uma ferramenta SDK chamada SN.exe (*Strong Name* – nome forte) [SAN 2001].

A assinatura de nome forte não envolve certificados como o *Authenticode* faz. Não há organizações de terceiros envolvidos, não há taxas a pagar e não há cadeias de certificados. Além disso, o *overhead* de verificar um nome forte é muito menos do que para o *Authenticode*. No entanto, nomes fortes não fazem declarações sobre confiança em um publicador específico. Os nomes fortes permitem garantir que o conteúdo de um dado

assembly não seja mexido e que o *assembly* carregado em seu favor no *run-time* venha do mesmo publicador contra o qual você desenvolveu. Mas não há declaração sobre se você pode confiar na identidade desse publicador [SAN 2001].

2.5.1.5.4 Qual a diferença entre um espaço de nome (*namespace*) e um nome de *assembly*?

Um espaço de nome é um esquema de nomeação lógica para tipos em que um nome de tipo simples, como *MyType*, é precedido de um nome hierárquico separado por ponto. Esse esquema de nomeação está totalmente sob controle do desenvolvedor. Por exemplo, espera-se que os tipos *MyCompany.FileAccess.A* e *MyCompany.FileAcess.B* tenham funcionalidades em relação a acesso a arquivo. A .NET Framework sua um esquema de nomeação hierárquica para agrupar tipos em categorias lógicas de funcionalidade relacionada, como o *framework* do aplicativo ASP.NET, ou de funcionalidade de *remoting* (localização remota). As ferramentas de desenho podem utilizar *namespaces* para facilitar aos desenvolvedores navegar a fazer referência a tipo em seu código. O conceito de namespace não está relacionado ao de um *assembly*. Um único *assembly* pode conter tipos cujos nomes hierárquicos tenham diferentes raízes de *namespace* e uma raiz de *namespace* lógica pode se estender por vários *assemblies*. Na .NET Framework, um namespace pe uma conveniência lógica de nomeação no momento do projeto, no passo que um *assembly* estabelece escopo de nome para tipos no *run-time* [SAN 2001].

2.5.1.6 Instalação e identificação de aplicativos

2.5.1.6.1 Quais opções estão disponíveis para a instalação de meus aplicativos .NET?

A .NET Framework simplifica a instalação tornando viável a instalação de impacto zero e a instalação de *XCOPY* de aplicativos. Como todas as solicitações são resolvidas primeiras no diretório privado de aplicativo, simplesmente copiar os arquivos de diretórios de um aplicativo para o disco é tudo o que é necessário para executar esse aplicativo. Na há necessidade de registro [SAN 2001].

Este cenário é especialmente convincente para aplicativos da Web, Web Services e aplicativos *desktop* independentes. No entanto, há situações em que *XCOPY* não é suficiente como um mecanismo de distribuição. Um exemplo é quando o aplicativo possui pouco código privado e conta com a disponibilidade de assemblies compartilhados, ou quando o aplicativo não está instalado localmente (mas sim transferido por *download* a pedido). Nesses casos, a .NET Framework fornece serviços extensivos de *download* de código e integrações com o Windows Installer. O suporte de download de código pela .NET Framework oferece muitas vantagens em relação às plataformas atuais, incluindo *download* incremental, segurança de acesso ao código (não mais diálogos de Authenticode) e identificação de aplicativos (código transferido por download em favor de um aplicativo que não afete os demais). O Windows Installer é outro mecanismo eficiente de instalação que se encontra disponível nos aplicativos .NET. Todos os recursos do Windows Installer, incluindo publicação, anúncio e reparo de aplicativos, estarão disponíveis nos aplicativos .NET no Windows Installer 1.5.

2.5.1.6.2 Eu escrevi um *assembly* que gostaria de usar em mais de um aplicativo. Onde instalá-lo?

Os *assemblies* a serem usados por vários aplicativos (por exemplo, assemblies compartilhados) são instalados no cache global de assemblies [SAN 2001].

2.5.1.6.3 Como posso ver quais *assemblies* estão instalados no *cache global* de *assemblies* no Windows?

A .Net Framework vem com uma extensão *shell* do Windows para exibição do cache de assemblies [SAN 2001]. Pode-se usar Windows Explorer visualizá-los. No Windows 9x os *assemblies* globais podem ser encontrados por padrão na pasta "\Windows\assembly", já na versão 2000 em "\WinNT\assembly" por *default*.

2.5.1.6.4 O que é um domínio de aplicativo?

Um domínio de aplicativo (geralmente *AppDomain*) é um processo visual que serve para identificar (isolar) um aplicativo. Todos os objetos criados dentro do mesmo escopo de aplicativo (em outras palavras, em algum lugar na seqüência de ativações de objetos que comecem com o ponto de entrada do aplicativo) são criados dentro do mesmo domínio de aplicativo. Podem existir vários domínios de aplicativos em um único sistema operacional, tornando-se um meio leve de identificação dos aplicativos [SAN 2001].

Um processo OS fornece identificação com um espaço de endereço de memória distinto. É eficiente, mas caro também, e não atinge os números necessários para grandes servidores da Web. O *Common Language Runtime*, por outro lado, executa a identificação do aplicativo gerenciando o uso de memória do código que esteja executando no domínio do aplicativo. Isso garante que a memória não será acessada fora dos limites do domínio. É importante observar que somente o código de segurança de tipo pode ser gerenciado dessa maneira (o *run-time* não pode garantir identificação quando é carregado um código sem segurança em um domínio de aplicativo) [SAN 2001].

2.5.1.7 Coleta de lixo

2.5.1.7.1 O que é coleta de lixo?

Coleta de lixo é um mecanismo que permite ao computador detectar quando um objeto não pode mais ser acessado. A memória usada por esse objeto é liberada automaticamente (assim como é chamada uma rotina de limpeza, chamada "finalizadora", que é escrita pelo usuário). Alguns *garbage collectors*, como o usado pela .NET, compactam a memória e, portanto, diminuem o conjunto de trabalho de seu programa [SAN 2001].

2.5.1.7.2 Como a coleta de lixo não-determinista afeta o meu código?

Para a maioria dos programadores, ter um *garbage collector* (e usar objetos de coleta de lixo) significa que não há necessidade de se preocupar com a desalocação de memória nem

com a referência de objetos de contabilidade, mesmo que você use estruturas de dados sofisticadas. No entanto, serão necessárias algumas alterações no estilo de codificação, se você desalocar recursos do sistema (*handles* de arquivo, bloqueios, etc.) no mesmo bloco de código que libera a memória para um objeto. Com um objeto de coleta de lixo, você deve fornecer um método que libere os recursos do sistema de forma determinista (ou seja, sob controle do seu programa) e deixar o *garbage collector* liberar a memória quando ele compactar o conjunto de trabalho [SAN 2001].

2.5.1.7.3 Posso evitar o uso do *heap* de coleta de lixo?

Todas as linguagens voltadas para o *run-time* permitem a alocação de objetos de classe no *heap* coletado no lixo. Isso traz benefícios em termos da última alocação e evita a necessidade de solução por parte dos programadores quando eles devem explicitamente "liberar" cada objeto [SAN 2001].

O CLR fornece também o que se chama de *ValueTypes* — são com classes, só que os objetos *ValueType* são alocados na pilha de *run-time* (e não no *heap*) e, portanto, são reclamados automaticamente quando o código sai do procedimento em que são definidos. É assim que as "estruturas" da C# operam [SAN 2001].

As extensões gerenciadas da C++ permitem que você escolha onde os objetos de classe serão alocados. Se declaradas como classes gerenciadas, com palavra-chave *gc*, elas serão alocadas a partir do *heap* de coleta de lixo. Se não incluírem a palavra-chave *gc*, elas se comportarão como objetos C++ regulares alocados do *heap* da C++, e liberados explicitamente com o método de liberação ("Free") [SAN 2001].

2.5.1.8 *Remoting*

2.5.1.8.1 Como a comunicação interna ao processo e a comunicação cruzando o processo trabalham no *Common Language Runtime*?

Existem dois aspectos na comunicação interna ao processo: entre contextos dentro de um único domínio de aplicativo ou por domínios de aplicativos. Entre contextos no mesmo domínio de aplicativo, os *proxies* são usados como mecanismos de interpretação. Nenhuma manobra/serialização (*marshaling*/*serialization*) está envolvida. Ao cruzar domínios de aplicativos, fazemos manobras/serialização usando o protocolo binário de *run-time* [SAN 2001].

A comunicação cruzando os processos usa um canal conectável e um protocolo formatador, cada qual para finalidade específica [SAN 2001]:

- Se o desenvolvedor especificar um ponto final usando a ferramenta soapsuds.exe para gerar um *proxy* de metadados, o canal http com o formatador SOAP será o padrão [SAN 2001].
- Se um desenvolvedor estiver fazendo *remoting* explícito no mundo gerenciado, será necessário ser explícito sobre qual canal o formatador utilizar. Isso pode ser expresso administrativamente, através de arquivos de configuração, ou com chamadas APl para carregar canais específicos. As opções são [SAN 2001]:
  - Canal HTTP com formatador SOAP (o HTTP trabalha bem na lnternet, ou a qualquer momento o tráfego deve percorrer os *firewallls*);

- Canal TCP com formatador binário (o TCP é uma opção de alto desempenho para redes locais (LANs));
- Canal SMTP com formatador SOAP (só faz sentido em máquina cruzada).

Quando são feitas transições entre código gerenciado e não-gerenciado, a infra-estrutura COM (especificamente, DCOM) é usada para *remoting*. Em versões intermediárias do CLR, isso se aplica também a componentes em serviço (componentes que usam serviços COM +). Na versão final, deve ser possível configurar um componente que possa ser remoto [SAN 2001].

A coleta de lixo distribuído é gerenciada por um sistema chamado "tempo de vida baseado na concessão". Cada objeto possui um tempo de concessão e, quando esse tempo expira, o objeto é desconectado da infra-estrutura de *remoting* de CLR. Os objetos possuem um tempo de renovação padrão – a concessão é renovada quando uma chamada bem-sucedida é feita do cliente para o objeto. O cliente também pode renovar explicitamente a concessão [SAN 2001].

2.5.1.9 Interoperabilidade

2.5.1.9.1 Posso usar objetos COM de um programa .NET Framework:

Sim. Todo componente COM instalado hoje pode ser usado a partir de código gerenciado, em casos comuns, a adaptação é totalmente automática [SAN 2001].

Especificamente, os componentes COM são acessado, da .NET Framework pelo uso de um RCW — runtime *callble wrapper* (decodificador chamável no *run-time*). Esse decodificador expões as interfaces COM pelo componente COM as interfaces compatíveis com a .NET Framework. No caso de interfaces de automação OLE, o RCW pode ser gerado automaticamente a partir de uma biblioteca de tipos. No caso de interfaces de automação não-OLE, um desenvolvedor pode escrever um RCW personalizado e mapear manualmente os tipos expostos pela interface COM para os tipos compatíveis com a .NET Framework [SAN 2001].

2.5.1.9.2 Os componentes da .NET Framework podem ser usados a partir de um programa COM?

Sim. Os tipos gerenciados que você cria hoje podem ser acessados a parir de COM e, em geral, a configuração é totalmente automática. Há alguns novos recursos do ambiente de desenvolvimento gerenciado que não são acessíveis a partir do COM. Por exemplo, métodos estáticos e construtores parametrizados não podem ser usados a partir do COM. Em geral, é uma boa idéia decidir de antemão quem será o usuário para um determinado tipo. Se o tipo for ser usando a partir do COM, você pode se limitar a usar os recursos acessíveis a partir do COM [SAN 2001].

Dependendo da linguagem usada para escrever o tipo gerenciado pode ou não se visível por padrão [SAN 2001].

Especificamente, os componentes da .NET Framework são acessados a partir do COM usando um CCW - COM *callable wrapper* (decodificador chamável no COM). É semelhante a um RCW (veja a pergunta anterior), mas trabalha na direção oposta. Novamente, se as ferramentas de desenvolvimento da .NET Framework não puderem gerar automaticamente o

decodificador, ou se o comportamento automático não for o desejado, um CCW personalizado poderá ser desenvolvido [SAN 2001].

2.5.1.9.3 Posso usar a API Win32 de um programa .NET Framework?

Sim. Usando *p/Invoke*, os programas .NET Framework poderão acessar bibliotecas de códigos nativos por meio de pontos de entrada DLL estáticos [SAN 2001]. O Mono pode no futuro também permitir que aplicações Win32 rodem no GNU/Linux utilizando este mesmo princípio. No entanto, atualmente não é uma prioridade da comunidade responsável pelo desenvolvimento do Mono.

2.5.1.9.4 Como administrar a segurança da minha máquina? De uma empresa?

No momento, a ferramenta da linha de comando CASPol é a única maneira de administrar a segurança. A política de segurança consiste em dois níveis: de máquina e pelo usuário. Há planos de fornecer uma ferramenta de administração plena de recursos, assim como suporte para administração de política de empresa, como parte da primeira versão da .NET Framework [SAN 2001].

2.5.1.9.5 Como a segurança baseada na evidência trabalha com segurança do Windows?

A segurança baseada na evidência (que autoriza o código) trabalha junto coma à segurança do Windows 2000 (que é baseado na identidade de *logon*). Por exemplo, para acessar um arquivo, o código e também deve estar sendo executado sob uma identidade de *logon* que tenha direitos de acesso a arquivos NFTS. As bibliotecas gerenciadas incluídas da .NET Framework também fornecem classes para a segurança baseada no papel desempenhado. Elas permitem que o aplicativo trabalhe com identidade de *logon* e grupos de usuários do Windows [SAN 2001].

2.6 Uma Visão Geral do .Net Compact Framework

2.6.1 Introdução

O .Net Compact Framework é a plataforma de desenvolvimento para *Smart Device* da iniciativa Microsoft .Net e a chave para atingir clientes com grandes experiências – a qualquer hora, qualquer lugar e em qualquer dispositivo. O .Net Compact Framework trás ao mundo do código gerenciado o XML Web Services para *Smart Devices*, e habilita a execução com segurança, *download* de aplicações em devices como *Personal Digital Assistants* (PDAs), telefones celulares e outros afins.

Em função do .Net Compact Framework ser um sub-conjunto do .Net Framework, os desenvolvedores podem facilmente usar os conhecimentos de programação e os códigos existentes através de dispositivos, desktop e servidores. A Microsoft liberou no Visual Studio

.Net 2003 o *Smart Device Application*, que contém o .Net Compact Framework. Isso significa que qualquer um dos 4 milhões de desenvolvedores Visual Studio que tem experiência com o Net Framework será capaz de desenvolver aplicações para qualquer dispositivo que rode .Net Compact Framework. Devido a isso, muitos desenvolvedores Visual Studio .Net serão desenvolvedores para *Smart Device*, existirão exponencialmente mais desenvolvedores para .Net Compact Framework que qualquer outro dispositivo ou plataforma de programação móvel [HAD 2004]. A **figura 16** ilustra a arquitetura do Microsoft .Net Compact Framework.

Além disso, pelo fato de compartilhar as ferramentas e modelos de programação do .Net Framework, o .Net Compact Framework terá dramaticamente redução do custo, e acréscimo de eficiência para o desenvolvimento de aplicações para *Smart Devices*. Companhias que acreditavam ter um alto custo no desenvolvimento de aplicações móveis, agora vão experimentar a produtividade no processo de desenvolvimento sem precisar retreinar os desenvolvedores. Isto irá ajudar as companhias futuramente a reduzir os custos para realizar negócios para desenvolver novas aplicações móveis.

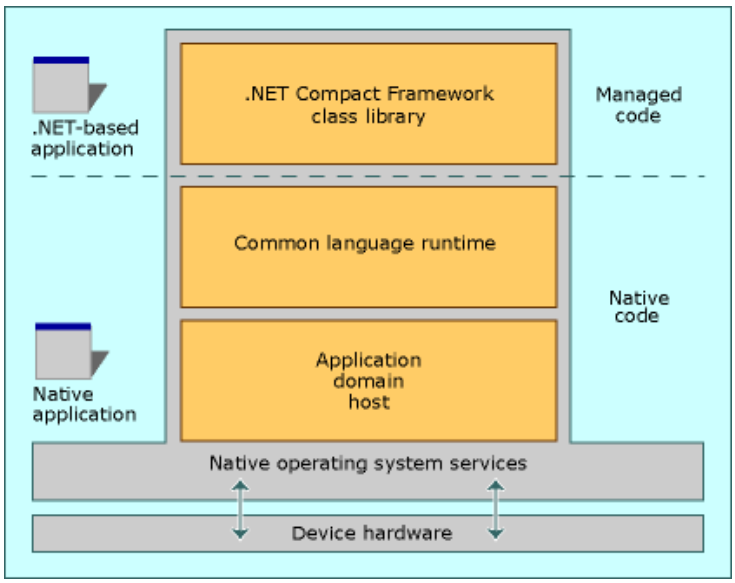

Figura 16 – Arquitetura do Microsoft .Net Compact Framework (fonte: msdn.microsoft.com)

2.6.2 Vantagens

Em função do .Net Compact Framework empregar o mesmo modelo de programação para vários tipos de aparelhos simplifica o processo de desenvolvimento da aplicação. Muitos dos códigos para esta aplicação, como uma lógica de negócio, camada de acesso a dados, e camada de XML Web Service, podem ser compartilhados através de múltiplos aparelhos e computadores. Isto aumenta consideravelmente a eficiência no desenvolvimento de aplicações.

3 Instalação

Nesta seção será explicado brevemente como instalar cada *framework*. Maiores detalhes podem ser verificados nos sites indicados.

3.1 Mono

3.1.1 Instalação de pacotes no Redhat & sistemas baseados em RPM

Para começar a instalação do Mono você necessitará fazer o *download* dos rpms para a distribuição Linux [QEL & JAC 2003]:

http://www.go-mono.org/download.html
libgc-6.1-1.i386.rpm
libgc-devel-6.1-1.i386.rpm
mono-0.23-1.i386.rpm
mono-devel-0.23-1.i386.rpm
Se quiser usar Gtk#, acesse ftp://ftp.gnome-db.org/pub/gnome-db/
libgda-0.10.0-1.i386.rpm
libgnomedb-0.10.0-1.i386.rpm

e http://gtk-sharp.sourceforge.net/
gtk-sharp-0.7-1.i386.rpm

3.1.2 Instalando o Mono no Slackware

Pacotes [QEL & JAC 2003]:

http://www.go-mono.org/archive/mono-0.26.tar.gz - Runtime
http://www.go-mono.org/archive/mcs-0.26.tar.gz - Compilador C#

Instalando os pacotes [QEL & JAC 2003]:

Utilizando os pacotes .gz, para fazer a instalação da maneira tradicional, "*configure, make* e *make install*" [QEL & JAC 2003].

Primeiro o *runtime*, descompacte o pacote mono-026.tar.gz em uma pasta qualquer [QEL & JAC 2003].

Feito isso será criado um diretório com o nome de mono-0.26 por exemplo, dentro deste diretório digite o comando *./configure --prefix=/usr/local* , feito isso é só seguir com *make* depois *make instlall* [QEL & JAC 2003].

O *runtime* está instalado, agora é preciso descompactar o compilador, dentro do diretório digite *./configure* em seguida *make*, depois *make install.* Assim já tem o *runtime* e o compilador mcs instalado, pronto [QEL & JAC 2003].

3.1.3 Instalação no Windows

Para Windows faça o *download* do instalador na página de *download* http://www.go-mono.com/download.html, o arquivo instalador da versão desejada. Pode salvá-lo, ou mandar executar diretamente [QEL & JAC 2003].

Executar o instalador [QEL & JAC 2003]:

Na tela de "*Welcome to the Mono Setup Wizard*", clique "*Next*".

A tela seguinte é para escolha do diretório de instalação, que por *default* virá com "C:\Program Files\Mono-0.26". Altere o diretório apenas se efetivamente necessário, e clique "*Install*".

Ao completar o processo de instalação, clique "*Close*".

Testar a instalação [QEL & JAC 2003]:

Abrir uma janela de console (Command Prompt).

Digite: *mcs --about{enter}*. Deverá surgir:

-- The Mono C# compiler is (C) 2001, 2002, 2003 Ximian, Inc.
-- The compiler source code is released under the terms of the GNU GPL
-- For more information on Mono, visit the project Web site
-- http://www.go-mono.com
-- The compiler was written by Miguel de Icaza, Ravi Pratap and Martin Baulig

Problemas [QEL & JAC 2003]:

Mono precisa de pelo menos Windows95, mas tem problemas com caracteres Unicode nessa versão, melhor instalar em Windows NT/2000/XP.

Instalando o GTK# [QEL & JAC 2003]

Buscar o arquivo do instalador http://taschenorakel.de/mathias/archive/gtk-sharp/gtk-sharp-0.10-win32-4.exe exemplo da versão 0.10. Pode salvá-lo, ou mandar executar diretamente

Executar o instalador do GTK#.

Na tela de "Welcome to the Gtk# *Runtime and Development Environment Setup Wizard*", clique "*Next*".

A tela seguinte é da licença LGPL Clique "*I Agree*".

Se você tiver mais de uma versão do .Net *framework* aparecerá uma tela para selecionar em quais será integrado o Gtk#. Escolha e clique "*Next*"z.

Depois virá a tela para escolha do diretório de instalação, que por *default* virá com o diretório onde o Mono estiver instalado. Altere o diretório apenas se efetivamente necessário, e clique "*Install*".

Ao completar o processo de instalação, clique "*Next*" e na tela seguinte "*Finish*".

Fazer o download do instalador do GTK+ para Windows

Buscar o arquivo do instalador http://www.dropline.net/gtk/download.php da versão mais recente. Use a versão *Runtime Environment*. Pode salvá-lo, ou mandar executar diretamente.

Executar o instalador do GTK+ para Windows [QEL & JAC 2003]:

Na tela de "*Welcome to the Gtk+ Runtime Environment Setup Wizard*", clique "*Next*". A tela seguinte é da licença GPL. Selecione "*I accept the agreement*" e clique "*Next*". Depois virá a tela para escolha do diretório de instalação, que por *default* virá "C:\Program Files\Common Files\GTK\2.0". Altere o diretório apenas se efetivamente necessário, e clique "*Next*". E na tela seguinte clique "*Install*".

Ao completar o processo de instalação, clique "*Finish*".

3.1.4 Mono/Utilizando CVS

CVS (*Control Version System*) que é um sistema de controle de versões de documentos. Este programa é um software padrão das distribuições Linux. Ele gerencia todas as atualizações de software. Não vem em questão entrar em detalhes sobre a instalação de um servidor de CVS mas ele pode ser bem útil na utilização do Mono, por isso para maiores informações a conectiva dispõem de um guia tratando desse assunto http://bazar.conectiva.com.br/~godoy/cvs/admin/ [QEL & JAC 2003].

Para utilizar o HandBook via CVS basta realizar o *download* do *HandBook* do Mono você precisa configurar uma variável de ambiente no seu sistema. Para isso basta digitar a seguinte linha no seu terminal ou colocá-la dentro do arquivo *.bashrc* (este arquivo encontra-se no seu diretório do seu usuário ou do /etc/.bashrc) [QEL & JAC 2003]

```
export CVSROOT=:pserver:anonymous@anoncvs.go-mono.com:/mono
```

Para efetuar o *download* do manual você deve realizar o *login* no servidor digitando [QEL & JAC 2003]:

```
cvs login
```

O servidor do CVS irá solicitar uma senha, mas como estamos realizando o *login* como *anonymous* basta informar o seu endereço de e-mail quando ele solicitar a senha. Agora a seguinte linha faz do *check out* (*co* é *check out* abreviado) e será realizado o *download* do Guia do mono [QEL & JAC 2003].

```
cvs -z3 co monkeyguide
```

3.1.5 Instalação do MonoDoc

É necessário o Gtk# para o funcionamento do Monodoc. Faça o download do Gtk# via CVS e instale utilizando os comandos abaixo [QEL & JAC 2003]:

```
root@thor:/home/thor/src/# export CVSROOT=:pserver:anonymous@anoncvs.go-mono.com:/mono

root@thor:/home/thor/src/# cvs login
root@thor:/home/thor/src/# cvs -z3 co gtk-sharp
root@thor:/home/thor/src/# cd gtk-sharp
root@thor:/home/thor/src/# ./autogen.sh --prefix=/usr/local
root@thor:/home/thor/src/# make
root@thor:/home/thor/src/# make install
```

Faça o *download* do "*GtkHtml*" e do "*Gal*", estes pacotes podem ser obtidos no linuxpackages.net http://www.linuxpackages.net/ no formato TGZ e com isso você poderá instala-los utilizando o *installpkg*, abaixo seguem os *links* para *download* dos 2 arquivos [QEL & JAC 2003]:

```
GtkHTML http://www3.linuxpackages.net/packages/Slackware-9/robert/gnome/gtkhtml3-3.0.8-i686-1rob.tgz
GAL http://www3.linuxpackages.net/packages/Slackware-9/robert/gnome/gal2-1.99.9-i686-1rob.tgz
```

Feito o download é só instalar utilizando os comandos abaixo [QEL & JAC 2003]:

```
root@thor:/home/thor/src/# installpkg gtkhtml3-3.0.8-i686-1rob.tgz
root@thor:/home/thor/src/# installpkg gal2-1.99.9-i686-1rob.tgz
```

Faça o download do MonoDoc <http://www.go-mono.org/archive/monodoc-0.6.tar.gz>. Descompacte e instale utilizando os comandos abaixo [QEL & JAC 2003]:

```
root@thor:/home/thor/src/# tar -xvzf monodoc-0.6.tar.gz
root@thor:/home/thor/src/# cd monodoc-0.6
root@thor:/home/thor/src/monodoc-0.6# ./configure --prefix=/usr/local
root@thor:/home/thor/src/monodoc-0.6# make
root@thor:/home/thor/src/monodoc-0.6# make install
```

3.2 dotGNU Portable.Net

Para instalar o dotGNU Portable.Net utilizando os binários, deve-se baixar os binários disponíveis para seu sistema operacional e distribuição, disponíveis em http://www.dotgnu.org/pnet-packages.html.

Estão disponíveis neste site pacotes para RedHat, Fedora (estes devem funcionar na maioria dos sistemas baseados em RPM), Debian, Mandrake, PLD, BeOS, MacOS X (através do sistema Darwinports) e Microsoft Windows.

Para instalação usando os códigos-fonte, pode-se seguir conforme demonstrado em http://www.dotgnu.org/pnet-install.html. As versões mais recentes estão disponíveis em http://www.southern-storm.com.au/portable_net.html, e também disponibilizadas em CVS. Para usar via CVS, com usuário anônimo, utilize:

```
export CVS_RSH="ssh"
cvs -d :ext:anoncvs@savannah.gnu.org:/cvsroot/dotgnu-pnet co .
```

Nota: antes de rodar a versão do CVS deve-se rodar o auto_gen.sh para criar o Makefile.in e os arquivos de configuração.

Os três componentes principais a serem instalados, em ordem, devem ser:

- treecc
- pnet
- pnetlib

Opcionalmente pode-se instalar ainda o pnetC (para usar o compilador C), o ml-pnet (para bibliotecas Mono, tais como System.Data) e o cscctest (disponível somente via CVS pois está em constante mudança).

Será necessário o uso de *flex* e *bison* para construir (*build*) o Portable.Net, já o yacc pode funcionar em alguns BSD, mas não é recomendado.

Descompacte e instale os pacotes como segue:

```
gunzip -d <nome-versao.tar.gz | tar xvf -
cd nome-versao
./configure
make
make install
```

Caso esteja fazendo isto via CVS, rode

```
./auto_gen.sh antes de ./configure.
```

3.2.1 Instalando no Windows

Baixe o Cygwin do site http://sources.redhat.com/cygwin/. Siga as instruções anteriores, isto irá construir a versão para Cygwin, que deverá ser instalada e rodar através do Cygwin. Outra maneira, para rodar como uma aplicação normal para Windows, fora do Cygwin, deve-se então construir a versão "mingw32", conforme segue :

```
gunzip -d <nome-versao.tar.gz | tar xvf -
cd nome-versao
./configure --disable-cygwin
make
make install
```

3.2.2 Instalação no MacOS X

Para MacOS X acesse o site www.dotgnu.org.

3.2.3 Rodando os exemplos

Os exemplos se encontram no diretório "samples". Para testá-los, use :

```
$ cd samples
$ ../engine/ilrun hello.exe
$ ../engine/ilrun fib.exe
```

3.3 Microsoft .Net Framework

A instalação do Microsoft .Net Framework é simples. Deve-se fazer o download dos arquivos de instalação no site da Microsoft (http://www.microsoft.com/net/products/).

Para rodar aplicações .Net apenas é necessária a instalação do Microsoft .Net Framework (pouco mais de 20 Mb) e para o desenvolvimento deve-se instalar o Microsoft .Net Framework SDK (aproximadamente 100 Mb). Também pode-se instalar o Microsoft .Net Compact Framework para o caso de desenvolvimento de aplicações para dispositivos móveis. Já há algumas atualizações e service *packs* disponíveis.

4 IDEs

4.1 Eclipse

Eclipse é uma plataforma de desenvolvimento *open source*, extensível e *Java-based*, que integra ferramentas de desenvolvimento, através de uma arquitetura aberta, extensível e baseada em *plug-ins*. A plataforma do eclipse é projetada para ambientes integrados de desenvolvimento (IDEs) que podem ser usadas para criar diversas aplicações como WebSites, programas utilizando Java, programas em C++, e *Enterprise JavaBeans*. Que significa? Isso parte inicialmente em ter apenas um editor com alguns atalhos para fazer diversas tarefas de programação comuns mais fáceis, você pode estender o IDE a alguma linguagem criar *plug-ins* que devem funcionar para OS suportados pelo Eclipse. Com a *multi-language* natural do Mono, este parece ser um projeto muito importante para o futuro desenvolvimento das aplicações neste *framework*. Portanto esta é uma IDE muito útil no desenvolvimento de aplicações utilizando C#. Já existe um *plug-in* C# para o Eclipse, este embora faltando algumas características interessantes, implementa a estrutura e sintaxe da linguagem. Um outro ponto extra é que o eclipse pode usar o *toolkit* Gtk2 no GNU/Linux. Isto é ótimo para usuários de Gnome, porque se adapta melhor ao ambiente [QEL & JAC 2003].

4.2 #Develop

#Develop (SharpDevelop) é uma IDE livre para C# e projetos do VB.Net da plataforma do Net da Microsoft. É *open-source* (GPL). Possui características melhores para se trabalhar com SharpDevelop que o Eclipse. A única desvantagem é, não funciona em

Linux/Mono, porque ele implementa em *System.Windows.Forms*, que não são suportados, ainda [QEL & JAC 2003].

Sobre SharpDevelop:

- Escrito em C#
- Compilador C# e VB.Net
- Editor de código C#, ASP.Net, ADO.Net, XML, HTML
- Destaque na sintaxe usada em C #, HTML, ASP, ASP.Net, VBScript, VB.Net,
- XML
- Facilidade de extensão com ferramentas externas
- Facilidade de extensão com Plug-Ins

Para mais informações sobre SharpDevelop e download consulte http://www.icsharpcode.net/OpenSource/SD/default.asp

Para instalação o #Develop vem com um instalador e requer a estrutura do .Net *framework*.

4.3 MonoDevelop

O MonoDevelop (www.monodevelop.com) é uma IDE para implementação de código-fonte utilizando a linguagem C#.

Esta ferramenta possui integração com o namespace GTK# para desenvolver aplicações gráficas, porém não possui editores visuais para modelar a interface com o usuário, para isto deve-se utilizar o Glade *(veja seção 6.4 pra mais informações)*. Possui também integração com o Mono Debugger *(veja seção 6.3 pra mais informações)* para prover maior facilidade de depuração de aplicações.

4.4 Microsoft ASP.Net WebMatrix

O Microsoft ASP.Net WebMatrix (www.asp.net) é uma ferramenta para desenvolvimento de aplicações ASP.Net. As **figuras 17** e **18** mostram o ambiente do WebMatrix.

Algumas características desta ferramenta:

- Possui componentes visuais o que permite desenvolver usando o conceito de WebForms;
- Suporte a linguagem C#;
- WebServer integrado para facilitar o teste das aplicações ASP.Net.

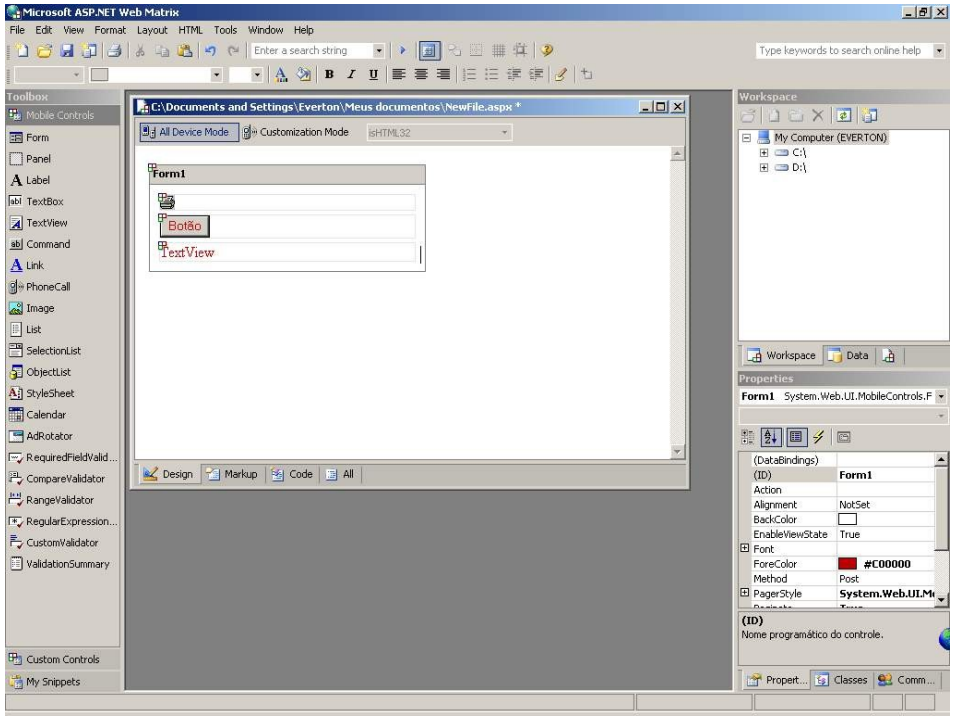

Figura 17 – WebMatrix: WebForm

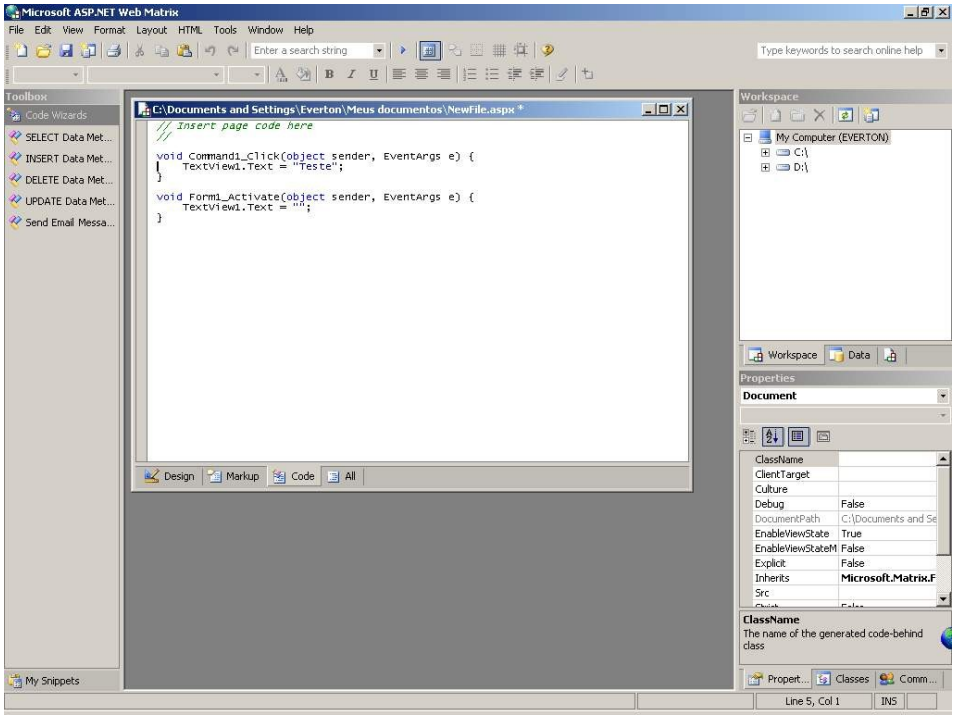

Figura 18 – WebMatrix: Editor de código-fonte que pode estar relacionado a WebForm

É uma ferramenta gratuita, disponibilizada pela Microsoft com o intuito de promover o ASP.Net, permitindo uma alternativa a ferramenta comercial Microsoft Visual Studio .Net – que certamente possui bem mais recursos e, por isso, elevado custo de aquisição.

5 Ferramentas de Apoio

5.1 NUnit

NUnit é uma ferramenta *unit-test* (ambiente de testes) para todas as linguagens do *framework* Net. Inicialmente portado de JUnit, a versão atual 2.0, é a segunda liberação desta ferramenta xUnit baseada na ferramenta de unidade de testes do .Net da Microsoft. Escrita inteiramente em C# foi planejada para superar muitas das características da ferramenta usada pelo .Net, um exemplo é o uso de atributos e outras capacidades relacionadas. NUnit traz o xUnit a todas as linguagens do .Net [QEL & JAC 2003].

NUnit é distribuído com mono [QEL & JAC 2003].

NUnit-GTK é a versão GUI do ambiente de teste NUnit [QEL & JAC 2003].

5.2 NAnt

O NAnt é uma *build tool*, do gênero do famoso make do Linux. Contudo supera o *make* em várias coisas: para começar é multiplataforma; depois organizamos os nossos *buildfiles* num ficheiro XML, que fornece intrinsecamente uma boa estrutura aos ficheiros. As IDE's são feitas para organizarmos os nossos projetos, e compilá-los com relativa facilidade. E pronto, é só isto. Uma *build tool* permite fazer montes de outras coisas como gerar documentação, interagir com o CVS, gerar um *zip* com uma versão e copiá-lo remotamente para outro lado qualquer, enfim, dá para fazer o que quisermos [QEL & JAC 2003].

Na verdade, o NAnt é inspirado no Ant programa de *bulid* para Java. Basicamente é um *port* para o .Net, ambas as ferramentas são quase idênticas [QEL & JAC 2003].

5.3 Mono Debugger

O Mono Debugger é uma ferramenta importante para o desenvolvimento. Martin Baulig, o autor do debugger mono escreveu provavelmente o melhor debugger do planeta. Faz exame não somente de aplicações mono, mas pode também eliminar erros do código nativo de C [QEL & JAC 2003].

5.4 Glade

O Glade é uma ferramenta para modelagem visual de aplicações, ou seja, modelar a interface.

O Glade já era utilizado antes do surgimento do Mono, usando GTK. Com a criação do Mono e por conseqüência do GTK#, esta ferramenta foi adaptada a partir da sua versão 2 pra permitir a modelagem visual usando GTK#. Não possui uma IDE para implementação de código-fonte, é necessário modelar a interface e depois relacionar os eventos dos

componentes visuais com implementações feitas em IDE's como o MonoDevelop *(veja seção 4.3 pra mais informações).*

O Glade e GTK# estão integrados ao Mono, com isso para utilizá-los basta instalar este framework. A **figura 19** mostra o ambiente do Glade.

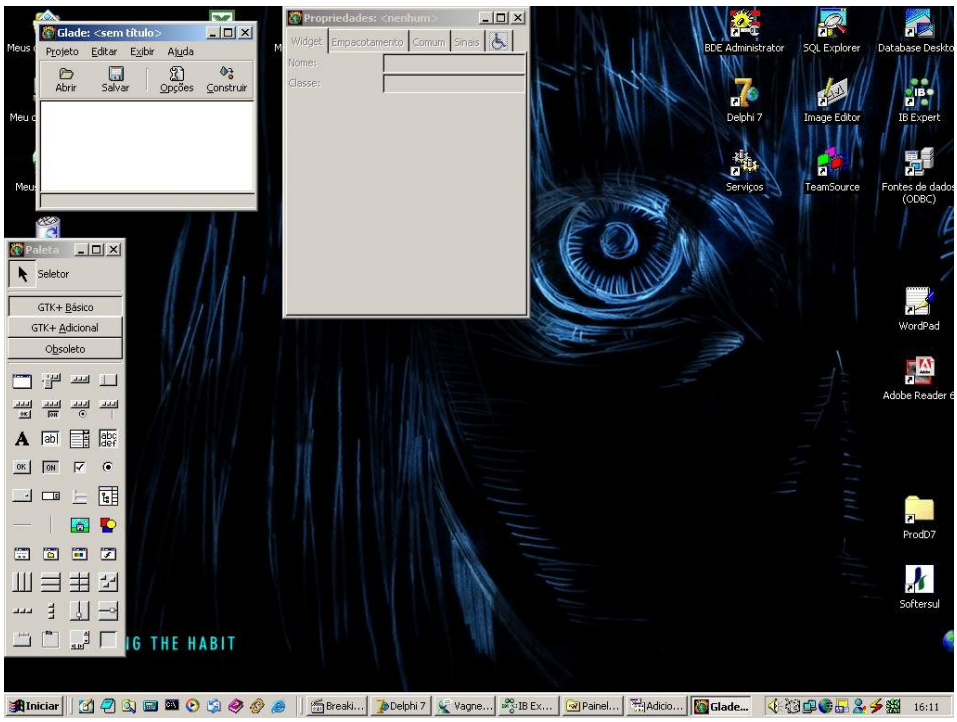

Figura 19 – Glade

6 Outras Tecnologias Implementadas

6.1 LOGO

Mono traz a linguagem *LOGO* ao *desktop* do *Gnome* (mas também Windows, MacOS X, KDE). Como todas as linguagens mono pode empregar as bibliotecas de classe, suportando assim a base de dados e as aplicações XML [QEL & JAC 2003].

6.2 Threading

Multithread-Aplications são aplicações, que não seguem somente um caminho, mas sim diversos, que são executadas lado a lado. Dessa maneira uma aplicação pode processar uma tarefa, sem que o usuário precise parar de executar outras ações com a ferramenta [QEL & JAC 2003].

Um bom exemplo seria um aplicativo com funcionalidade da busca, que deixa o usuário trabalhando, enquanto procura [QEL & JAC 2003].

6.3 Networking

O System.Net namespace contém diversas classes para trabalhar com conexões de TCP/IP ou de HTTP. Nós começaremos de baixo nível com o TCP-Client [QEL & JAC 2003].

Protocolo: Quando os dados são emitidos sobre uma rede são na maioria de casos emitidos sobre um protocolo específico. TCP/IP é o protocolo, usado na maioria das vezes. Tem somente algumas exceções, como os jogos, que funcionam com protocolos diferentes como o UDP. TCP/IP é um protocolo orientado a conexão [QEL & JAC 2003].

Sockets: Por ser orientado à conexão, o TCP/IP força ambas as partes da conexão a fazer um "*handshake*" antes que a conexão esteja estabelecida. Além disso, os dados são emitidos em pacotes, e pode acontecer que pacotes, que são emitidos primeiro cheguem mais tarde, assim eles devem ser requisitados novamente. Felizmente o programador não tem que se importar com tais coisas, porque a rede é segurada pelo sistema da operacional. O nível mais baixo, onde você pode fazer a rede é através dos Sockets [QEL & JAC 2003].

Em cada aplicação existem dois tipos diferentes [QEL & JAC 2003]:

- Server: está escutando em uma porta especial e espera um ou mais Client para conectar.
- Client: pode conectar ao Server e emitirá alguns dados (pedido) e o Server pode responder ao pedido emitindo alguns dados para ele (resposta).

6.4 XML

XML (*EXtensible Markup Language*), trata-se de uma linguagem que é considerada uma grande evolução na internet. Porém, para quem não é programador ou não trabalhe com o uso de linguagens e ferramentas para a Web, é quase imperceptível às vantagens do XML. O XML é uma especificação técnica desenvolvida pela W3C (*World Wide Web Consortium* – entidade responsável pela definição da área gráfica da Internet), para superar as limitações do HTML, que é o padrão das páginas da Web [QEL & JAC 2003].

A linguagem XML é definida como o formato universal para dados estruturados na Web. Esses dados consistem em tabelas, desenhos, parâmetros de configuração, etc. A linguagem então, trata de definir regras que permitem escrever esses documentos de forma que sejam adequadamente visíveis ao computador [QEL & JAC 2003].

É possível [QEL & JAC 2003]:

- Ler e escrever documentos XML com o XML DOM.
- Ler documentos XML com XPath, uma linguagem Query muito poderosa.
- Usar transformações Xslt.
- Trabalhar com XML e ADO.Net *(consulte a seção 2.4 para obter mais informações)*.

Classes para manipulação de arquivos XML estão presentes tanto no Microsoft .Net Framework quanto no Mono.

6.5 P#

P# é um compilador que permite converter códigos-fonte em Prolog para implementações em C#.

O objetivo do P# é permitir que implementações em Prolog possam ser compartilhadas e executadas nos *frameworks* .Net. A conversão gera classes em C# equivalentes ao código escrito em Prolog, com isto, é possível que outras linguagens .Net possam utilizá-las. Após a conversão de Prolog para C#, deve-se então compilar os códigos-fonte gerados para assemblies .Net, utilizando qualquer compilador C# disponível.

No teste que realizamos, a conversão de um simples código em Prolog, com aproximadamente 10 linhas, gerou em torno de 15 arquivos C# (com extensão cs). Com cada arquivo C# contendo classes e métodos com a implementação equivalente do código Prolog. Em termos práticos, não verificamos a praticidade e usabilidade deste tipo de conversão, mas de qualquer forma é uma alternativa para integração com Prolog.

Mais informações podem ser obtidas em http://www.dcs.ed.ac.uk/home/stg/Psharp/.

6.6 Bamboo.Prevalence

6.6.1 Objetivo

O Bamboo.Prevalence é um porte do Prevayler (www.prevayler.org) originalmente escrito em Java para a linguagem C# com melhorias (Prevalência Transparente é uma delas), mantido por Rodrigo Bamboo, agora com compatibilidade para Mono [TEI 2004].

6.6.2 Implementação em Java

O Prevayler é um sistema de persistência de dados que usa uma forma diferenciada de armazenamento de dados baseado em objetos chamado prevalência, derivado do fato que os objetos têm que prevalecer (permanecer) em memória. O ganho de performance é impressionante: 9.000 vezes mais rápido que o banco de dados Oracle, para consultas. Outro atrativo é o suporte total a OO (Orientação a Objetos), sem a necessidade de camadas de mapeamento objeto para relacional e vice-versa [TEI 2004].

O Prevayler foi idealizado e desenvolvido em Java por Klaus Wuestefeld, um brasileiro, como software livre e hoje existem implementações para várias linguagens [TEI 2004].

6.6.3 Implementação em .Net

A implementação para .Net é o Bamboo.Prevalence (GPL), e seu criador é outro brasileiro o Rodrigo B. de Oliveira, que hoje colabora com o Mono Brasil [TEI 2004].

Leia o autor explicando sobre o que tem de diferente no Bamboo.Prevalence em relação ao Prevayler [TEI 2004]:

> *"O bp foi baseado no conceito de prevalência mas não muito no código do prevayler. Adotei (ou tentei adotar) as mesmas convenções de nome e semânticas como existiam na versão 1.0 do prevayler: .snapshot, .commandlog, Command, etc, mas as coisas lá mudaram desde então. A versão 2.0 do bp deve acompanhar o prevayler 2.0.*
> *Reescrevi o código do zero para acomodar melhor as mudanças de framework (java x .net) e de minhas próprias idéias sobre como as coisas deviam funcionar.*
> ***O que existe de diferente?***

Em um nível mais alto, há uma diferença básica de filosofia nas distribuições: o prevayler é bem econômico, no bp eu coloco tudo que foi necessário ou útil nos vários projetos em que o utilizei, por isso vc encontra coisas como:

- *Bamboo.Prevalence.Indexing: indexação e busca para objetos (nesse momento apenas texto livre, pois foi tudo que precisei ;-));*
- *Bamboo.Prevalence.VersionMigration: migração de esquemas de dados (objetos);*
- *Bamboo.Prevalence.Collections: algumas implementações úteis de listas (List, ReaderWriterList), enumeradores, entre outras coisas;*
- *Bamboo.Prevalence.Util: utilitários para limpeza automática de diretórios, etc;*

Descendo o nível um pouco, o bp introduz também o conceito de objeto Query além do comum objeto Command, isso facilita alguns cenários de programação (o programador não precisa se preocupar com sincronização em ambientes multi-threaded) - embora eu não utilize esse suporte com tanta freqüência.
Outra novidade do bp: prevalência transparente através de proxies (meu artigo explica o assunto).
Existem vários outros pequenos detalhes e decisões de projeto diferentes aqui e lá:

- *o bp utiliza um ReaderWriterLock ao invés de um lock normal para sincronizar comandos, queries (incluindo snapshots);*
- *no bp você deve adquirir um PrevalenceEngine através de uma fábrica;*
- *no bp você passa o tipo (System.Type) do sistema prevalente, no prevayler você passa o próprio sistema (ainda que ele possa ser descartado mais tarde);*
- *no bp você pode adquirir uma referência ao PrevalenceEngine, PrevalentSystem e clock do sistema de forma muito simples no escopo de um comando ou query:*

```
void AddArticle(Article article)
        {
            DateTime now = PrevalenceEngine.Now;
            MySystem theSystem =
(MySystem)PrevalenceEngine.CurrentPrevalentSystem;
            ...
        }
```

Isso torna a programação um pouco mais conveniente em diversos pontos.

- *o bp pode ser pausado (isso é extremamente conveniente para atualização quente de aplicações);*
- *o bp suporta um modelo de decoração de comandos baseado em atributos, isso permite coisas como lembrar qual o contexto de segurança no momento da execução de um comando (eu utilizo isso para integração com o framework de segurança de código do .net);*
- *o bp integra-se com o modelo de configuração do .net (app.config ou web.config) - você pode configurar todos os detalhes de um engine no web.config e apenas utilizar o objeto dentro do código;*

Tenho certeza de que existem outras diferenças, mas as principais estão aí.
No geral, a superfície do bp é bem maior que a do prevayler, isso é bom e ruim.
Rodrigo"

Para mais informações acesse os sites http://bbooprevalence.sourceforge.net/ e http://monobrasil.softwarelivre.org .

6.7 NpgSQL

O NpgSQL é um provedor ADO.Net, 100% em managed C#, para acessar bases de dados PostgreSQL**.** Projeto criado pelo Francisco Figueiredo Jr. [BIN 2004].

O NpgSQL é um .Net Data Provider para o servidor PostgreSQL. Ele é feito 100% em C#. O projeto foi iniciado em maio de 2002 e posteriormente, em fevereiro de 2003 foi adotado na biblioteca de classes do Projeto Mono. O NpgSQL pode ser compilado e executado no Mono, Portable.Net e na implementação da MS, permitindo uma grande portabilidade das aplicações que utilizam o PostgreSQL. Atualmente o NpgSQL permite que o desenvolvedor tenha acesso a quase todas as funcionalidades que o servidor disponibiliza com relação ao armazenamento e recuperação de dados, chamadas de funções e manipulação de informações de definição de dados. Além disso, está sendo implementado também suporte a internacionalização para as mensagens retornadas [BIN 2004].

O NpgSQL implementa o protocolo 2.0 e o 3.0, como especificado no site http://developer.postgresql.org/ [BIN 2004].

Mais informações podem ser obtidas em http://gborg.postgresql.org/project/npgsql .

6.8 Linguagem BOO

BOO é uma nova linguagem de programação orientada a objetos, estaticamente tipada, com sintaxe de alto nível e extensível para a CLI que traz quatro inovações principais à realidade das linguagens estaticamente tipadas: atributos sintáticos, *mixins*, macros e uma *pipeline* de compilação aberta e extensível. É baseada na sintaxe da linguagem Python.

Para saber mais acesse http://boo.codehaus.org/ .

6.9 Advanced Data Provider - ADP

O Advanced Data Provider - ADP é um provider transparente para .Net, que tem a vantagem de fazer a carga dinâmica dos providers utilizados. Executa em .Net Framework, Mono e Portable.Net, em Linux e Windows. O site oficial do projeto (http://advanced-ado.sourceforge.net/) informa que foram feitos testes com os seguintes bancos de dados: Microsoft SQLServer, Oracle, Firebird, SQLite, PostgresSQL, MySQL. Mas os desenvolvedores acreditam que todos os providers disponíveis devem funcionar.

7 C# & VB Compiler e Assembler tools

O MCS: *The Ximian C# Compiler*, é atualmente capaz de compilar-se e compilar aplicações C# (ele inclui uma *suite* de testes para você usar). É usado também para compilar o Mono. Há ainda algumas áreas que não foram abrangidas pelo compilador Mono, mas são muito poucas (atributos de segurança). A *suite* de testes é mantida para seguir o progresso do compilador e os vários programas rotineiramente compilados e executados [QEL & JAC 2003].

O MCS não compila o código original, mas um tipo do *bytecode*, que possa ser processado pelo *runtime* do Mono [QEL & JAC 2003].

MBAS: Mono's Basic.Net Compiler é um compilador CLI para a linguagem Visual Basic, uma extensão da versão do VisualBasic.Net. É baseado no compilador do MCS e ainda em

forte desenvolvimento, embora muitas características de linguagem já são suportadas [QEL & JAC 2003].

Os compiladores .Net como o mcs ou o mbas não compilam o código original, mas à linguagem intermediária comum (CLI). O disassembler Mono é escrito em C. O programa Monodis é usado para armazenar os índices de uma imagem CLI [QEL & JAC 2003].

O Assembler Mono é, o contrário do disassembler Mono, mas também é escrito em C# [QEL & JAC 2003].

7.1 C#

Esta seção tem por objetivo apresentar C# que é a linguagem padrão de todos os *frameworks* – Microsoft .Net Framework, Mono e dotGNU Portable.Net.

Por tratar-se da linguagem nativa dos *frameworks* abordados por este tutorial, consideramos importante incluir uma visão geral de como programar usando C#, analisando suas características e estruturas de comandos. **As informações desta seção foram retiradas de [QEL & JAC 2003]**. Em [ARC 2001] pode-se obter mais informações de como desenvolver usando C#.

7.1.1 Estrutura, Interpretador / Compilador?

Esta seção foi retirada de [QEL & JAC 2003].

C# é uma linguagem criada por Anders Heljsberg da Microsoft. C# foi projetado para ser uma linguagem fácil de ser utilizada. Muitos elementos dessa linguagem são similares a Java e a C++. A linguagem desenvolveu-se sobre as queixas das linguagens anteriores. Microsoft apresentou C# para *Common Language Infrastructure to* ECMA (*European Computer Manufacturers Association*) em 2000. O ECMA ratificou C# como um padrão em 2001 a ECMA-334. E a ISO (*International Standards Organization*) ratificou o padrão da ECMA em 2002.

O que é C#. Um interpretador ou compilador?

A resposta é: ambos.

Para compreender como isto acontece e porque isso é bom, é necessário explicar como trabalha o Mono. Mono C# compilador em linha de comando compatível com a estrutura do Net do compilador C# da Microsoft. Todas as linguagens mono, incluindo C# são compiladas com o próprio compilador. Porém a saída é um arquivo binário original, mais um bytecode especial, chamado CLI (IL ou MSIL).

Este *bytecode* não pode ser executado pelo computador. É importante compreender que, todas as linguagens mono usam CLI, C# assim como MonoBasic.

O arquivo binário será interpretado mais tarde no runtime pelo Mono *Executing Engine* no computador dos usuários.

As principais características do C# são as seguintes:

Todas as variáveis e código são declarados no escopo de classes. É possível, contudo, declarar tipos ("*structs*" e enumerações) fora do escopo de classes. Nem tudo é uma classe...

Tipagem forte:

As enumerações são tipos próprios e incompatíveis com outras enumerações. Existe um tipo lógico (*bool*) incompatível com inteiros. Os tipos intrínsecos são: lógico, inteiros de vários tamanhos pré-definidos (8, 16, 32 e 64 bits, com e sem sinal), ponto flutuante IEEE de 4 e 8 *bytes*, *string* e decimal. Só existe um único tipo "*char*", também incompatível com inteiros. Tanto o "*char*" como a "*string*" armazenam apenas caracteres Unicode (16 bits por caractere). O tipo "decimal" é armazenado como uma mantissa binária de 96 bits e um expoente na base 10, para um total de 128 bits. A precisão do decimal é de pelo menos 28 dígitos decimais, o que evita a maioria dos erros de arredondamento comuns aos formatos de ponto flutuante em binário. Existe também "*structs*", boas para serem usadas em situações "*leves*", como por exemplo, uma coordenada (X, Y), quando o custo em memória e tempo de execução de uma classe seria grande e desnecessário.

Os objetos e "*arrays*" são necessariamente alocados dinamicamente no "*heap*" com o uso do operador "*new*".

O índice dos "*arrays*" começa com zero e sua faixa é sempre verificada em tempo de execução.

O C# inicializa a maioria das variáveis com zero e efetua diversas verificações de lógica, como se uma variável foi atribuída antes de ser usada, se um parâmetro de saída foi atribuído e se um inteiro teve sua faixa violada.

Todas as conversões de tipo ("*cast*") são validadas em função do tipo real da variável em tempo de execução, sem exceções.

O operador "." é usado em diversos lugares, quando em C++ seriam usados ".", "::" e "->".

Existe um outro tipo de *loop* além dos oriundos do C (*for, while, do..while*), o "*foreach*", usado para varrer todos os elementos de um *array* ou "coleção".

O "*switch*" elenca opções mutuamente exclusivas, por definição, e pode ser usado com *strings*. O "*break*" depois de cada opção é obrigatório.

O único mecanismo de tratamento de erros do C# é a *exception*.

Não existem macros, mas existe compilação condicional (#*ifdef*, etc).

Os templates não são suportados, pelo menos por enquanto. Talvez seja possível criar um mecanismo semelhante aos templates no futuro. De qualquer forma, o C# tem um suporte bastante abrangente a "*reflections*", o que pode substituir templates em várias situações.

O C# suporta sobrecarga de funções e de operadores, como o C++, mas não tem argumentos "*default*".

O C# possui operadores de conversão, mas existe uma sintaxe para indicar se a conversão deve ser implícita ou explícita. O construtor não é usado como operador de conversão.

O modelo C# de orientação a objeto tem as seguintes características básicas:

Herança simples, com um ancestral comum a todos os objetos chamado "*System.Object*" O ancestral comum concentra funções de criação, comparação, conversão para *string* e informações de tipo em tempo de execução.

Embora a herança seja simples, as classes podem implementar várias "interfaces". Isto traz as vantagens da herança múltipla sem muitos de seus problemas. Uma interface funciona como se fosse uma "classe abstrata", que possui apenas protótipos de métodos, sem nenhuma implementação.

Podemos declarar "*properties*", que funcionam sintaticamente como campos, mas na verdade chamam um par de métodos para atribuir ou receber o valor da "*property*". As propriedades podem ser também "indexadas" com um inteiro, funcionando como se fossem

"*arrays*" ou indexadas com uma "*string*", quando passam a funcionar como um dicionário. O ambiente de desenvolvimento sabe criar "editores de propriedades" para alterar seus valores em tempo de desenvolvimento.

Os métodos não são a princípio virtuais e devem ser explicitamente declarados como tais com a palavra reservada "virtual". Existe um protocolo específico para indicar se um método de classe derivada reimplementa um método virtual (*override*) ou o torna não-virtual (*new*).

Podemos declarar um tipo que é um "ponteiro para método", chamado "*delegate*". Um "*delegate*" contém, a princípio, o endereço da função e também do método que a implementa. Todos os eventos, tão importantes para o funcionamento do ambiente de desenvolvimento, são "*delegates*". Os *delegates* permitem que uma classe chame métodos em outras sem exigir que esta outra classe seja derivada de um ancestral conhecido.

As informações de tipos em tempo de execução ("*reflections*") permitem coisas que normalmente as linguagens compiladas não são capazes como: criar um objeto de uma classe dado seu nome, atualizar propriedades dados seu nome e valor e chamar métodos dados seu nome e argumentos. Tanto o ambiente de desenvolvimento como de execução confiam pesadamente neste mecanismo para funcionarem.

Podemos atribuir "atributos" a classes e métodos. Os atributos funcionam mais ou menos como uma diretiva de compilação, mas são resolvidos em tempo de execução. Podemos criar novos atributos.

Existe um mecanismo para herança de formulários.

Ponteiros:

Não existem ponteiros na nova plataforma. Isto não quer dizer que não temos a eficiência dos ponteiros: muitos objetos são tratados por referência. As referências são "ponteiros domesticados": embora internamente elas sejam ponteiros, elas não podem apontar para locais arbitrários de memória.

A memória não precisa ser liberada pelo programador. Um "g*arbage collector*" faz o serviço. Isto evita uma série de erros como "vazamentos de memória" e uso de uma variável cuja memória já foi liberada.

Boxing:

Os objetos oferecem um modelo muito conveniente para lidar com elementos em nossos programas através da abstração proporcionada por propriedades, métodos, eventos e do mecanismo de herança. O problema é que os objetos têm o custo adicional ao serem sempre acessados através de ponteiros ("*this*", "*self*") e terem que ser criados e destruídos.

Este custo é irrelevante quando estamos lidando com um objeto complexo e pesado como um formulário na tela ou um arquivo em disco. Mas é um custo muito caro para tipos simples como um inteiro, especialmente visto que a CPU consegue lidar com inteiros de maneira muito eficiente.

A plataforma resolve este problema de uma maneira brilhante: existem duas categorias de tipos: por valor e por referência. Os tipos por valor podem ser automaticamente convertidos para referências através de um processo chamado "*boxing*". Isto permite tratar os tipos intrínsecos como se eles tivessem propriedades e métodos, como por exemplo:

```
int x = 10;
string s = x.ToString();
```

O C# é um C++ "limpo", com várias boas idéias comuns em outras linguagens e algumas novas, como "*boxing*", "*delegates*", "*garbage collection*" e "*attributes*".

7.1.2 Aplicação Exemplo

Esta seção foi retirada de [QEL & JAC 2003].

Depois de uma visão geral da plataforma, uma primeira noção da linguagem C# começa com o simples programa "hello, mundo!". Para depois mostrar alguns conceitos de C# tais como usar *namespaces*, declarar classes, e declarar métodos.

```
// Hello World! - Primeiro Programa
using System;
class Test {
        public static void Main() {
                Console.WriteLine("Hello, World!");
        }
}
```

7.1.3 Namespace / Definição de Classes / Main()

Esta seção foi retirada de [QEL & JAC 2003].

Mono possui mais de 3000 classes. Há bibliotecas adicionais como GTK# (650 classes), isso faz este número aumentar. Pode acontecer, que duas classes em Mono tenham o mesmo nome, ou você pode precisar criar uma classe nova que tenha um nome já usado. Isso é possível porque o conceito de *namespaces* existe. Cada classe na biblioteca de classes Mono é posta em um *namespace*. Por exemplo, a classe do console que nós usamos está no *namespace System*. A classe da janela está no *namespace* do GTK. As classes nos *namespaces* devem ser acessadas colocando o *namespace* na frente do nome da classe.

```
System.Console.WriteLine("Hello, World!");
// em vez de Console.WriteLine("Hello, World!");
```

Isto pode resultar em muito trabalho na hora de escrever o código, quando for usar *namespaces*. Isto porque o compilador testa todos os *namespaces* usados pela classe, e se encontrá-lo duas vezes mostrará o erro. Assim em vez de escrever *System.Console*.. nós podemos escrever:

```
using System;
...
Console.WriteLine("Hello, World!");
```

É comum usar o *System.* Ninguém escreveria *System.Console.WriteLine()*. Você também pode pôr seu próprio código nos *namespaces*. Tudo que você tem que fazer é:

```
namespace mynamespace {
// ...
}
```

Você pode pôr um *namespace* dentro de um outro *namespace*:

```
namespace HigherNamspace {
namespace mynamespace {
// ...
}
}
```

Isto é igual a:

```
namespace HigherNamspace.mynamespace {
// …
}
```

Definição de uma classe:

As classes são a base da programação orientada do objeto. Têm um comportamento similar ao de uma *structs* utilizada em linguagens como C/C++, as classes C# são muito similares as de outras linguagens orienta a objetos como o Java.

Uma classe é um tipo especial definido pelo usuário, que encapsula as variáveis (chamadas de Campos) e as funções (chamadas de Métodos). Em C# o código deve ser organizado em métodos, que necessitam sempre estar contidos nas classes.

Criação de uma classe:

```
class Hello{
}
```

Método Main()

Em C # você pode ter uma classe com diversos métodos:

```
class Hello {
        void Method1() {
        }
        void Method2() {
        }
}
```

O que será executado primeiro? Seria errado supor o Method1, porque está acima de Method2, você poderia também escrever Method2 acima de Method1. Na verdade o programa não compilará. Há sempre um método chamado Main(), que é executado quando a aplicação é iniciada. É responsável pela chamada dos outros métodos.

```
static void Main() {
}
```

Normalmente os métodos são sempre parte de um objeto. A instrução static diz ao compilador que este método não é parte de um objeto. Isso porque o *Main()* deve ser o primeiro método a ser executado, ele não retorna valor nenhum.

7.1.4 Utilizando o Compilador C#

Esta seção foi retirada de [QEL & JAC 2003].

Os fontes em C# tem extensão "*cs*". Todos os fontes em um "projeto" são compilados diretamente para um único "assembly:assemblies" (.EXE ou .DLL). Não existe um arquivo intermediário (.OBJ ou .DCU) como no Delphi.

Qualquer fonte pode referenciar outro fonte no projeto com a sintaxe "*namespace.classe.membro*". Note que podemos incluir "*namespaces*" dentro de "*namespaces*". Não é necessário declarar previamente os identificadores ou usar "*using*".

Se o *namespace* vai gerar um .EXE, uma de suas classes deve ter um método *public static int Main(string[] args)* que será o ponto de entrada do executável.

Compilar uma classe usando a linha de comando:

```
csc file.cs
```

Quando as classes requeridas são ligadas por mais de um arquivo, estes devem ser listados, separando-os por espaços, como em:

```
csc file1.cs file2.cs
```

Um compilador C# classes são arquivos executáveis ou uma biblioteca de ligação dinâmica - arquivo DLL. Um arquivo executável contem um programa executável (nas classes compiladas de um arquivo executável, deve haver somente um método ' principal '). ma biblioteca de ligação dinâmica contém conjunto das classes, que podem então ser instanciadas e utilizadas pelos programas em execução.

Se as classes fazem referência a classes externas, o compilador C# deverá encontrá-las. Por *default*, o compilador olha o conjunto ' *mscorlib.dll* ', que suporta as funções básicas do *runtime*. Para apontar ao compilador o sentido de outra classes, é preciso usar o interruptor de /r descrito como descrito na (Tabela 1).

Interrupções do Compilador	**Descrição**
/r:dll or /reference:dll por exemplo: /r:System.xml.dll, System.Net.dll	Diz ao compilador C# quais DLLs deve incluir. Porque os *namespaces* da biblioteca padrão não são referenciados automaticamente pelo compilador. Este interruptor permite que você inclua suas próprias DLLs.
/out: file por exemplo: /out:fred.dll	Especifica o nome de arquivo onde o código compilado deverá ser escrito.
/doc: file por exemplo: /doc:doc.xml	Criação da documentação XML do arquivo especificado.
/t:type or /target:type por exemplo: /t:exe – produz um arquivo executável de console (default) /t:library – produz um arquivo dll /t:module –cria para cada arquivo sua própria dll, exemplo, fred.cs será convertido para fred.dll /t:winexe – produz um arquivo executável, uma janela.	Isso é usado para especificar o tipo de arquivo de saída produzido.

Tabela 1 - Principais interruptores do compilador (fonte: [QEL & JAC 2003])

7.1.5 Tipos de Dados

Esta seção foi retirada de [QEL & JAC 2003].

Tipos Variáveis: Tipo de referência e valor;

C# é um tipo de linguagem segura. As variáveis são declaradas como sendo tipos particulares, e compreendem somente valores do tipo declarado.

As variáveis podem conter valores ou então referências, ou podem ser ponteiros.

Diferença entre tipos valor e referência:

- onde uma variável v variável um tipo valor, contém diretamente um objeto com algum valor. Nenhum outro v' pode conter diretamente o objeto contido por v (embora v' pôde conter um objeto com o mesmo valor).
- onde a variável v contém um tipo de referência, que contém algo que consulta um objeto. Uma outra v' variável pode conter uma referência ao mesmo objeto referenciado por v.

7.1.6 Tipo Valor

Esta seção foi retirada de [QEL & JAC 2003].

Em C# é possível definir seus próprios tipos declarando *enumerators* ou *structs*. Estes tipos definidos pelo usuário são tratados da mesma maneira que tipos predefinidos do C#, embora estes sejam otimizados pelo compilador. A (Tabela 2) lista os tipos predefinidos. Porque em C# todos os tipos de valores aparentemente fundamentais são definidos sobre um tipo de objeto. Então temos que os tipos .*System* definidos no *framework* .Net correspondem a tipos pré-definidos.

C # Tipo	**Estrutura do Net (sistema)**	**Assinado?**	**Bytes Ocupados**	**Valores Possíveis**
Sbyte	System.Sbyte	Sim	1	-128 a 127
Short	System.Int16	Sim	2	-32768 a 32767
Interno	System.Int32	Sim	4	-2147483648 a 2147483647
Longo	System.Int64	Sim	8	-9223372036854775808 a 9223372036854775807
Byte	System.Byte	Não	1	0 a 255
Ushort	System.Uint16	Não	2	0 a 65535
Uint	System.Uint32	Não	4	0 a 4294967295
Ulong	System.Uint64	Não	8	0 a 18446744073709551615
Flutuador	System.Single	Sim	4	Aproximadamente ±1.5 x 10-45 a ±3.4 x 1038 com 7 figuras significativas
Dobro	System.Double	Sim	8	Aproximadamente ±5.0 x 10-324 a ±1.7 x 10308 com 15 ou 16 figuras significativas Sim 12
Decimal	System.Decimal	Sim	12	Aproximadamente ±1.0 x

				10-28 a ±7.9 x 1028 com 28 ou 29 figuras significativas
Char	System.Char	N/A	2	Algum caracter de *Unicode* (bocado 16)
Bool	System.Boolean	N/A	1/2	Verdadeiro ou falso

Tabela 2 – Tipos pré-definidos em C# fonte: [QEL & JAC 2003])

Exemplo de declaração de duas variáveis e definidas como inteiro:

```
interno x = 10;
interno y = x;
```

7.1.7 Tipo de Referência

Esta seção foi retirada de [QEL & JAC 2003].

Os tipos pré-definidos de referência são *object* e *string*. Novos tipos de referência podem ser definidos usando as declarações '*class*', '*interface*' e, '*delegate*'.

Um tipo referência guarda o valor de um endereço de memória ocupado pelo objeto que referência. Considere a seguinte parte de código, em que duas variáveis fazem referência ao mesmo objeto (no exemplo, o objeto tem a propriedade numérica ' myValue ').

```
object x = new object();
x.myValue = 10;
object y = x;
y.myValue = 20; //após esta atribuição x.myValue e y.myValue tem valor 20.
```

Este código ilustra como mudar a propriedade de um objeto que usa uma referência particular, esta é refletida em todas as referências restantes. Embora as strings sejam tipos de referência, elas funcionam como tipos do valor. Quando uma string for atribuída ao valor de outra, por exemplo:

```
string s1 = "hello";
string s2 = s1;
```

Então s2 referência neste momento o mesmo objeto que a string s1. Entretanto, quando o valor de s1 for mudado, por exemplo, com

```
s1 = "adeus";
```

o que acontece é que um novo objeto é criado para string que s1 aponta. Então, s1 é igual a "adeus", visto que se ainda é igual a "hello".

A razão para este comportamento é que os objetos da string são imutáveis. Isto é, as propriedades destes objetos não podem sofrer mudanças. Portanto para mudar as referências de uma string, um novo objeto deve ser criado para a mesma.

Boxing:

C # permite que você converta qualquer tipo do valor a um tipo correspondente o da referência, e converta o tipo resultante ' *boxed* ' de volta outra vez. A seguinte parte de código demonstra o *boxing*. Quando a segunda linha executa, um objeto está iniciado como o valor ' *box* ', e o valor guardado por i é copiado para o objeto.

```
int i = 123;
object box = i;
if (box is int)
{Console.Write("Box contains an int");}
```

7.1.8 Operadores

Esta seção foi retirada de [QEL & JAC 2003].

C# tem um número padrão de operadores, como em C, C++ e Java. A maioria é bem familiar aos programadores. A (Tabela 3) lista os operadores padrão, indicando também a quais operadores é possível realizar sobrecarga.

Type operators	Type equality / compatibility	a is String	Não
	Type retrieval	typeof (int)	Não
Arithmetic	Multiplication	c*d	Sim
	Division	c/d	Sim
	Remainder	c%d	Sim
	Addition	c+d	Sim
	Subtraction	c-d	Sim
	Shift bits right	c>>3	Sim
	Shift bits left	c<<3	Sim
Relational and Logical	Less than	c<d	Sim
	Greater than	c>d	Sim
	Less than or equal to	c<=d	Sim
	Greater than or equal to	c>=d	Sim
	Equality	c==d	Sim
	Inequality	c!=d	Sim
	Bitwise and	c&d	Sim
	Bitwise or	c|d	Sim
	Logical and	c&&d	Não
	Logical or	c||d	Não
	Conditional	int c=(d<10) ? 5:15	Não

Tabela 3 – Operadores Padrão/Sobrecarga (fonte: [QEL & JAC 2003])

Para sobrecarregar um operador em uma classe, se define um método usando a palavra chave “operador”. Por exemplo, a sobrecarrega do operador de igualdade:

```
public static bool operator == (Value a, Value b)
{return a.Int == b.Int}
```

7.1.9 Ponteiros

Esta seção foi retirada de [QEL & JAC 2003].

Um ponteiro é uma variável que guarda o endereço de memória de uma outra de tipo diferente. Em C#, ponteiros são declarados somente para guardar endereços de memória de tipos de valor. A não ser que os ponteiros sejam declarados implicitamente, usando o símbolo *, exemplo:

```
int *p;
```

Esta declaração usa um ponteiro ' p ', que aponta ao endereço de memória inicial de um inteiro (armazenado em quatro *bytes*).

O elemento * p (' p ' prefixado pelo símbolo *deferencer* ' * ') é usado para consultar a posição de memória que p armazena. Então dado sua declaração, * p pode aparecer em atribuições de inteiros:

```
* p = 5;
```

Este código dá o valor 5 ao inteiro que foi inicializado pela declaração.

O efeito desta atribuição deve mudar a posição de memória armazenada por p. Não muda o valor do inteiro inicializado na declaração original. Agora, p apontará ao começo dos quatro *bytes* atuais na posição de memória 5.

Um outro símbolo importante para usar ponteiros é o operador &, que neste contexto retorna o endereço de memória da variável. Exemplo:

```
interno i = 5;
interno * p;
p = &i;
```

Dado o código acima,

```
* p = 10;
```

muda o valor de i para 10, desde que ' * p ' seja lido como um inteiro situado no valor de memória armazenado por p.

O principal problema em usar ponteiros em C# é que C# opera com um processo de *garbage collection* em *background*. Para liberar mais memória, este coletor do lixo pode mudar a posição de memória de um objeto atual sem aviso. Assim o ponteiro que aponta previamente a esse objeto se torna invalido por não ter o endereço real. Tal cenário conduz a dois problemas. Primeiro poderia comprometer a execução do programa. Segundo poderia afetar a integridade de outros programas.

Por causa destes problemas, o uso dos ponteiros é restringido.

Ponteiros, métodos e disposições.

Embora nós indiquemos acima que ponteiros podem ser usados somente com tipos de valor, uma exceção envolve os *arrays*.

Um ponteiro pode ser declarado com relação a um *array*, como:

```
int[ ] a = {4, 5};
interno * b = a;
```

O que acontece neste caso é que a posição de memória armazenada por b é a posição do primeiro tipo armazenada por a. Este primeiro tipo deve é como antes um valor.

O código abaixo mostra que é possível passar os valores de um *array* usando um ponteiro.

```
using System;
public class Teste {
        public static void Main() {
                int[] a = {4, 5};
                changeVal(a);
                Console.WriteLine(a[0]);
                Console.WriteLine(a[1]);
        }
        public unsafe static void changeVal(int[] a) {
                fixed (int *b = a) {
                        *b = 5;
                        *(b + 1) = 7;
                }
        }
}
```

7.1.10 Estruturas de Controle (*If/Else, For, Do/While...*)

Esta seção foi retirada de [QEL & JAC 2003].

Estruturas de controle são indicações que uma linguagem oferece sobre o controle de uma aplicação. São usadas para manter o comportamento lógico dentro do programa.

Pode ser: condicional (decide qual ação deve ser executada); repetitiva (determinado grupo de ações pode executar por um tempo n determinado).

Sem estas estruturas de controle, toda a aplicação seria inútil.

7.1.11 Controle De Fluxo: Indicações de Laço

Esta seção foi retirada de [QEL & JAC 2003].

7.1.11.1 while

Esta seção foi retirada de [QEL & JAC 2003].

Sintaxe: while (expression) statement[s]

O laço *while* executa uma indicação, ou bloco de indicações que estejam entre aspas, repetidamente até que a circunstância especificada pela expressão booleana retorne falso. Por exemplo:

```
int a = 0;
while (a < 3) {
        System.Console.WriteLine(a);
        a++;
}
```

7.1.11.2 do-while

Esta seção foi retirada de [QEL & JAC 2003].

sintaxe: do statement[s] while (expression)

É idêntico ao *while*, o que difere é a condição "*do*" que deixa a verificação da expressão para o fim do bloco. Isso faz com que mesmo a circunstância sendo inicialmente falsa, o bloco execute uma vez. Por exemplo:

```
int a = 4;
do {
        System.Console.WriteLine(a);
        a++;
} while (a < 3);
```

7.1.11.3 for

Esta seção foi retirada de [QEL & JAC 2003].

sintaxe: for (statement1; expression; statement2) statement[s]3

A cláusula "*for*" contém três partes. Statement1 é executado antes que o laço esteja incorporado. O laço que é executado corresponde ao seguinte laço ' *while* ':

```
for (int a =0; a<5; a++) {
        System.Console.WriteLine(a);
}
```

O laço "*for*" tende a ser usado quando a necessidade de manter o valor do *iterator*. Geralmente, a primeira indicação inicializa o *iterator*, a circunstância avaliá-lo de encontro a um valor final, e a segunda indicação muda o valor do *iterator*.

```
for (int a =0; a<5; a++) {
        System.Console.WriteLine(a);
}
```

7.1.11.4 foreach

Esta seção foi retirada de [QEL & JAC 2003].

sintaxe: foreach (variable1 in variable2) statement[s]

O laço ' *foreach*' é usado para relacionar os valores contidos em um objeto que executar a relação de *IEnumerable*. Quando um laço ' *foreach* ' executa, o dado variable1 está ajustado a cada valor do objeto nomeado por variable2. Esses laços são usados para acessar

valores de *arrays*. Assim, nós podemos ter laços com os valores de um *array* da seguinte maneira:

```
int[] a = new int[]{1,2,3};
foreach (int b in a)
        System.Console.WriteLine(b);
```

O principal inconveniente desses laços é cada valor extraído ser apenas de leitura.

7.1.12 Controle de Fluxo: Jump

Esta seção foi retirada de [QEL & JAC 2003].

7.1.12.1 break

Esta seção foi retirada de [QEL & JAC 2003].

A indicação ' *break*' quebra execução de laços ' *while* ', ' *for* ' e ' *switch* '.Exemplo:

```
int a = 0;
while (true) {
        System.Console.WriteLine(a);
        a++;
        if (a == 5)
        break;
}
```

7.1.12.2 continue

Esta seção foi retirada de [QEL & JAC 2003].

A indicação ' continue ' pode ser colocada em qualquer parte da estrutura do laço. Quando executa, move o contador de programa imediatamente para a iteração seguinte no laço. Exemplo:

```
int y = 0;
for (int x=1; x<101; x++) {
        if ((x % 7) == 0)
        continue;
        y++;
}
```

7.1.12.3 goto

Esta seção foi retirada de [QEL & JAC 2003].

A indicação ' *goto* ' é usada para saltar a uma parte particular de código de programa. É usada também na indicação do ' *switch* '. Pode-se usar uma indicação ' *goto* ' em um laço, como no exemplo (uma vez que, isso não é recomendado):

```
int a = 0;
start:
System.Console.WriteLine(a);
```

```
a++;
if (a < 5)
        goto start;
```

7.1.13 Controle de Fluxo: Estruturas de Seleção

Esta seção foi retirada de [QEL & JAC 2003].

7.1.13.1 if – else

Esta seção foi retirada de [QEL & JAC 2003].

Instruções usada na execução de blocos de código, avalia uma expressão booleana para executar uma instrução ou bloco de instruções. A cláusula ' *else* ', é opcional. Exemplo:

```
if (a == 5)
        System.Console.WriteLine("A is 5");
else
        System.Console.WriteLine("A is not 5");
```

Se as indicações puderem ser emuladas usando um operador condicional. O operador condicional retorna um de dois valores, dependendo do valor de uma expressão booleana.

```
int i = (myBoolean) ? 1 : 0 ;
```

Ajusta i =1 se *myBoolean* é verdadeiro, e i = 0 se *myBoolean* for falso. A instrução ' if ' prevista no exemplo anterior poderia ser escrita da seguinte maneira:

```
System.Console.WriteLine( a==5 ? "A is 5" : "A is not 5");
```

7.1.13.2 switch – default

Esta seção foi retirada de [QEL & JAC 2003].

A instrução ' *Switch* ' prove uma maneira mais limpa de usar múltiplas indicações *if-else*. No exemplo, a variável cujo valor está em questão é ' a '. Se for igual a 1, então a saída será ' a>0 '; se for igual a 2, então a saída será ' a>1 e a>0 '. Se não, diz-se que a variável não está ajustada.

```
switch(a) {
        case 2:
                Console.WriteLine("a>1 and ");
                goto case 1;
        case 1:
                Console.WriteLine("a>0");
                break;
        default:
                Console.WriteLine("a is not set");
                break;
}
```

Cada case pode especificar uma ação condicional diferente no código, ou não especificar nada:

```
break;
goto case k; (onde k e um caso especifico)
goto default;
```

7.1.14 Arrays

Esta seção foi retirada de [QEL & JAC 2003].

O tipo de cada *array* declarado é dado primeiro pelo tipo básico de elementos que pode guardar, e segundo pela dimensão do *array*. Unidimensional, uma única dimensão. São declarados usando parênteses, por exemplo:

```
int[] i = new int[100];
```

Esta linha do código declara a variável i para ser um array inteiro do tamanho 100. Contém o espaço para 100 elementos do tipo inteiro, variando de i[0] a i[99].

Para fazer atribuição a um *array* basta especificar valores para cada elemento, como no seguinte código:

```
int[ ] i = int[2 novo ];
i[0 ] = 1;
i[1 ] = 2;
```

Ou junto a declaração com a atribuição dos valores nos elementos:

```
int[ ] i = int[ novo ] {1.2};
```

ou assim:

```
int[ ] i = {1,2};
```

Por definição, como nós vimos, todo o começo dos arrays tem seu limite mais baixo como. Entretanto, usando a classe System.Array é possível criar e manipular *arrays* com um limite alternativo inicial mais baixo inicial.

A propriedade (de leitura apenas) do comprimento de um array guarda o número total de seus elementos através de todas suas dimensões. Porque os *arrays* unidimensionais, esta propriedade limitara o comprimento a única dimensão. Por exemplo, dado a definição do array i acima, i tem tamanho 2.

Um *array* multidimensional retangular tem uma única disposição com mais de uma dimensão, com a dimensão definida na declaração do array. O código mostra um *array* 2 por 3 multidimensional:

```
int[,] squareArray = new int[2,3];
```

Como nos unidimensionais, os arrays retangulares podem ser definidos na declaração. Por exemplo,

```
int[, ] squareArray = {{1, 2, 3}, {4, 5, 6}};
```

A classe de *System.Array* inclui um número de métodos para determinar o tamanho e os limites de um *array*. Estes incluem os métodos *GetUpperBound*(int i) e *GetLowerBound*(int i), que retornam, as dimensões superior e inferior.

Por exemplo, se o comprimento da segunda dimensão de *squareArray* é 3, a expressão

```
squareArray.GetLowerBound(1)
```

retorna 0, e a expressão

```
squareArray.GetUpperBound(1)
```

retorna 2.

É possível criar *arrays* multidimensionais com dimensões irregulares. Essa flexibilidade é pelo fato de que os arrays multidimensionais são executados como *arrays* de *arrays*. A seguinte parte de código demonstra como se pôde declarar um *array* composto de um grupo de 4 e de um grupo de 6 elementos:

```
int[][] jag = new int[2][];
jag[0] = new int [4];
jag[1] = new int [6];
```

O código revela que cada um jag[0] e jag[1] tem uma referência a um *array* unidimensional do tipo inteiro. Para inicializar um *array* desse tipo, atribuindo valores a seus elementos, veja o exemplo:

```
int[][] jag = new int[][] {new int[] {1, 2, 3, 4}, new int[] {5, 6, 7, 8, 9, 10}};
```

Tenha cuidado usando métodos como *GetLowerBound*, *GetUpperBound*, *GetLength*, etc. com este tipo arrays, com são criados através de arrays unidimensionais, não devem ser tratados como tendo dimensões múltiplas da mesma maneira que *arrays* retangulares.

Exemplo, com utilização de um *loop*:

```
for (int i = 0; i < jag.GetLength(0); i++)
        for (int j = 0; j < jag[i].GetLength(0); j++)
Console.WriteLine(jag[i][j]);
```

8 Comparações entre Java e os Frameworks .Net

8.1 Sobre o princípio de funcionamento

As duas plataformas de desenvolvimento trabalham com princípios semelhantes. Tanto Java quanto *frameworks* .Net são orientados a objetos, ou seja, implementam classes na IDE de suas SDKs. Além disso, Java e os *frameworks* .Net compilam o programa desenvolvido pelo programador para uma linguagem intermediária, *bytecode* em Java e *assemblies* em *frameworks* .Net, logo, as duas necessitam de uma máquina virtual, VM em Java e CLR em *frameworks* .Net.

8.2 Sobre os ambientes para desenvolvimento

A maioria dos desenvolvedores diz que - “A melhor linguagem para programar é aquela que você está adaptado”. Isto é verdade até certo ponto, pois a melhor linguagem é

aquela que nos provê os melhores recursos de engenharia de software, nos garantindo uma boa produtividade. É neste ponto que os *frameworks* .Net levam alguma vantagem em relação a Java, pois desenvolvedores que estão acostumados com outras linguagens, que não Java, podem continuar a utilizá-las. Poucas mudanças são necessárias e a maioria das IDEs possibilitam migrar automaticamente os códigos anteriores a .Net.

8.3 Sobre as licenças

O Java da SUN Microsystems é considerado por muitos como 'Software Livre'. Porém isto não é verdadeiro. Java é dependente de softwares proprietários por duas principais razões. Primeiro: o código fonte da VM de Java não é disponibilizado - Conforme a SUN Microsystems isso é necessário para que não haja despadronização. Segundo: muitas classes implementadas nas bibliotecas padrão de Java não tem seu código fonte disponibilizado, algumas são até segredo de industria. Além disso, a licença da SUN não permite modificações no código, conforme verificado no trecho retirado da própria licença do SDK:

> *" 2.LICENSE TO USE. Subject to the terms and conditions of this Agreement, including, but not limited to the Java Technology Restrictions of the Supplemental License Terms, Sun grants you a non-exclusive, non-transferable, limited license without license fees to reproduce and use internally Software complete and unmodified for the sole purpose of running Programs. Additional licenses for developers and/or publishers are granted in the Supplemental License Terms."*
>
> *"3.RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Unless enforcement* ***is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software****. You acknowledge that Licensed Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun Microsystems, Inc. disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement. Additional restrictions for developers and/or publishers licenses are set forth in the Supplemental License Terms."*

Sendo assim, como pode ser observado no trecho em negrito, é proibidas a modificação, a descompilação e a engenharia reversa. O que contraria a filosofia de software livre:

> *"Um programa é software livre se seus usuários tem certas liberdades cruciais. Grosseiramente falando, elas são: a liberdade de executá-lo, a liberdade de estudá-lo e modificar o código-fonte, a liberdade de redistribuir o código-fonte e os binários, e a liberdade de publicar versões melhoradas." (boa explicação sobre o debate de Java como software livre pode ser obtido em http://www.portaljava.com.br/home/modules.php?name=News&file=article&sid=809, com o título - A Armadilha Java).*

Nos *frameworks* .Net, conforme comentado em outras seções, a arquitetura é padronizada e aberta, sendo controlada pela ECMA. Isso possibilita que comunidade do software livre possa desenvolver para .Net, com licença GPL (*General Public Licence*), sem que haja qualquer incompatibilidade.

8.4 Os projetos software livre da tecnologia Java

Para combater tais problemas encontra-se em desenvolvimento, pela comunidade software livre, projetos para a tecnologia Java, *GNU Java Compiler* e o *GNU Classpath*. Maiores detalhes podem ser obtidos em http://gcc.gnu.org/java/ e http://www.gnu.org/software/classpath/classpath.html .

9 Integração entre Java e .Net

Esta seção apresenta opções de integração entre Java e os *frameworks* discutidos – Microsoft .Net Framework, Mono e dotGNU Portable.Net.

9.1 IKVM.Net

IKVM.Net (www.ikvm.net) é um projeto que permite integração entre os *frameworks* – Microsoft .Net, Mono e Portable.Net – com Java.

Ele disponibiliza:

- **Uma máquina virtual Java**: possibilita que *bytecode* Java possa rodar dentro de qualquer um dos *frameworks* discutidos. Classes Java em formato *bytecode* podem ser acessadas por *assemblies* .Net, permitindo a integração em tempo de execução. No entanto, deve-se ressaltar a queda na performance, pois as aplicações em Java devem ser convertidas para *assemblies* .Net antes de serem executadas.
- **Uma implementação para .Net de bibliotecas de classes Java**: através destas classes que a conversão de aplicações Java para .Net é possibilitada.
- **Ferramentas para ativar a interoperabilidade entre .Net e Java**: conversão de *bytecode* Java para assemblies .Net ou vice-versa. Com isto, um desenvolvedor em tempo de projeto pode converter facilmente classes existentes de uma plataforma para outra.

O IKVM deve ser integrado ao Mono em breve, o que permitirá rodar normalmente aplicações Java como se fossem aplicações Mono. Mas há perda de desempenho em relação a rodar a mesma aplicação usando uma máquina virtual Java, já que, como mencionado antes, é necessário em tempo de execução a conversão de *bytecode* Java para *assemblies* .Net e, por fim, utilizar a CLR para compilar os *assemblies* para código binário final. O mais indicado é a conversão em tempo de projeto para melhora na performance.

9.2 JILC - Java Bytecode to IL Compiler

JILC (*Java Bytecode to IL Compiler*) é um compilador de *bytecode* Java para *assemblies* .Net. Maiores informações podem ser obtidas em http://jilcsourceforge.net .

10 Conclusões

O mundo todo vive a era da informação. Hoje em dia tudo é dinâmico e veloz. Ter a informação certa no momento certo é crucial para a sobrevivência competitiva das corporações, das nações, e por que não, das pessoas. Sendo assim, faz-se necessário que haja mudanças na forma de desenvolver os programas que manipulam as informações, pois muito esforço e tempo são desperdiçados na migração de sistemas quando esses começam se tornar obsoletos. É hora dos desenvolvedores de softwares acompanharem a dinâmica do mundo de hoje.

Sendo assim, nosso trabalho centralizou-se na investigação das novas soluções para a criação de programas – os projetos baseados na *Common Language Infrastructure (CLI)*.

A *CLI* representa uma nova forma dos programas interagirem com o sistema operacional. Duas de suas camadas são as principais responsáveis por essa mudança – a *Common Language Specification (CLS)* e a *Common Language Runtime (CLR)*. A *CLS* é a camada que tem a função de tornar os programas *Multilinguagens*. Isto é, agora um mesmo programa pode ser desenvolvido composto por diversas linguagens, desde que todas elas sejam compatíveis com a *CLS* (no popular, compatíveis com .Net), tornando possível um grande ganho de produtividade durante a fase de desenvolvimento, pois tudo o que um programador precisa saber é desenvolver programas para .Net, independentemente de qual linguagem ele está adaptado. Já a *CLR* é a camada responsável por tornar os programas *Multiplataformas*, ou seja, ela faz com que o programa seja independente de tipo de sistema operacional (GNU/Linux, Microsoft Windows ou MacOS), tornando possível que fusões e migrações entre sistemas sejam mais amigáveis, diminuindo o tempo desperdiçado com isso.

A *CLI* é um padrão aberto, por isso vários projetos estão sendo desenvolvidos com base nela – o Microsoft .Net Framework (Microsoft Windows), o dotGNU Portable.Net (GNU/Linux e MacOS) e o Mono (GNU/Linux e MacOS). Testamos as principais características listadas no parágrafo anterior (*multilinguagem e multiplataforma)* nas plataformas Windows e GNU/Linux. A Microsoft .Net hoje é a que está mais evoluída. Ela é a única que oferece solução completa, pois permite o desenvolvimento para *desktop* e web, para diferentes tipos de dispositivos, através de ambientes como o Microsoft Visual Studio e o Borland Delphi,. Já o dotGNU Portable.Net e o Mono ainda não estão tão completos quando o Microsoft .Net Framework, pois durante nossos testes, soluções criadas em linguagens do ambiente Visual Studio e Borland Delphi, não rodaram perfeitamente nessas plataformas. Mas, a expectativa que pudemos verificar nas documentações dos projetos dotGNU Portable.Net e Mono é que em breve isso esteja portado, tornando possível a tão desejada característica de *Multiplataforma*.

O padrão *CLI* não pára por aí. Vários outros projetos estão sendo desenvolvidos para tornar Java também portável para este padrão. Alguns testes que realizamos com o IKVM, por exemplo, mostrou que essa realidade será possível. Dessa forma, teremos uma solução completa para desenvolvimento, fazendo com que a linguagem na qual a aplicação foi desenvolvida seja um mero item de ganho de produtividade.

O mundo está mudado. Ele está dinâmico, veloz e a informação é peça chave na competitividade. Nosso papel de desenvolvedores (e pesquisadores) é estar sincronizado com essa realidade, sempre buscando novas soluções que nos permitam essa condição. Portanto, nesse trabalho, dentro de nossas possibilidades, procuramos trazer o maior número de informações sobre essa nova solução para desenvolvimento de softwares, dando um ponto de partida. Sugerimos que você arregace as mangas e comece já a desenvolver, também, nessa tecnologia.

11 Referências

[ARC 2001] Archer, Tom. Descobrindo C# - tradução de DocWare Traduções Técnicas, Edson Furmankiewicz e Sandra Figueiredo. Editora Campus. Rio de Janeiro, 2001.

[BIN 2004] Binhara, Alessandro. NpgSQL http://monobrasil.softwarelivre.org/twiki/bin/view/Main/NpgSQL, acessado em novembro de 2004.

[BUR 2004] Burke, Shawn. Using the Microsoft .Net Framework to Create Windows-based Applications, publicado em http://msdn.microsoft.com/netframework/programming/winforms/?pull=/library/en-us/dndotnet/html/netwinforms.asp#netwinforms_intro, acessado em outubro de 2004.

[DUR 2004] Durães, Ramon. ASP.Net WebServices para Iniciantes, publicado em http://www.linhadecodigo.com.br/artigos.asp?id_ac=393 , acessado em novembro de 2004.

[HAD 2004] Haddad, Renato. Visão Geral do .Net Compact Framework, publicado em http://www.microsoft.com/brasil/msdn/Tecnologias/netframework/framework_visaogeral.aaspx, acessado em setembro de 2004.

[MOU1 2004] Moura, Sanyo Capobiango S. de. Delphi 8 e ASP.Net – Acesso a dados com ASP.Net e BDP - Revista ClubeDelphi, edição 53.

[MOU2 2004] Moura, Sanyo Capobiango S de. ASP.Net – Desenvolvimento para a Web com Delphi 8. Revista ClubeDelphi , edição 51.

[MOU3 2004] Moura, Sanyo Capobiango S de. Delphi 8 e ASP.Net – Server Controls, Sessões e Validações. Revista ClubeDelphi , edição 52.

[QEL & JAC 2003] Giaretta, Quéli & Brancher, Jacques Duílio. Estudo do Framework Mono. Universidade Regional Integrada do Alto Uruguai e das Missões – Campus Erechim. Artigo publicado em 2003.

[SAN 2001] Santos, Izabel C. de Mendonça & Tavares, Ana Beatriz - tradutoras. Microsoft .Net, Framework e aplicativos WEB / Microsoft Corporation. Editora Campus. Rio de Janeiro, 2001.

[SAN 2004] Sant'Anna, Mauro. Artigo ".Net Framework", publicado em http://www.microsoft.com/brasil/msdn/colunas/net/col_net_fevereiro01.aspx, acessado em setembro de 2004.

[TEI 2004] Teixeira, Rafael. Prevalência em Mono, publicado em http://monobrasil.softwarelivre.org/twiki/bin/view/Main/BambooPrevalence, acessado em novembro de 2004.

[TOL 2004] Tolomelli, Leonardo. Código Compartilhado do Common Language Infraestruture, publicado em

http://www.microsoft.com/brasil/msdn/colunas/batepapo/col_batepapo_6.aspx , acessado em setembro de 2004.

12 Anexo A – Links

Esta seção lista endereços da Internet com assuntos relacionados com este tutorial.

12.1 Microsoft .Net Framework

MSDN Brasil: http://www.msdn.com.br *(em português)*
MSDN Internacional: http://www.msdn.com *(em inglês)*
Microsoft .Net Framework: http://www.microsoft.com/net/ *(em inglês)*
ASP.Net: http://www.asp.net *(em inglês)*
DotNet-fr: http://www.dotnet-fr.org/ *(em francês)*

12.2 Mono

Projeto Mono Internacional: www.mono-project.com *(em inglês)*
Projeto Mono Brasil: http://monobrasil.softwarelivre.org *(em português)*
The Mono Community: http://www.gotmono.net/ *(em inglês)*
GUI ToolKits for Mono: http://primates.ximian.com/~miguel/toolkits.html *(em inglês)*

12.3 dotGNU Portable.Net

Site Oficial: http://www.dotgnu.org *(em inglês)*

12.4 Ferramentas e Tecnologias .Net

MonoDevelop: http://www.monodevelop.com *(em inglês)*
Microsoft ASP.Net WebMatrix: http://www.asp.net *(em inglês)*
Bamboo.Prevalence: http://bbooprevalence.sourceforge.net/ *(em inglês)*
NpgSQL: http://gborg.postgresql.org/project/npgsql *(em inglês)*
Advanced Data Provider: http://advanced-ado.sourceforge.net/ *(em inglês)*

12.5 Linguagens .Net

P#: http://www.dcs.ed.ac.uk/home/stg/Psharp/ *(em inglês)*
Boo: http://boo.codehaus.org/ *(em inglês)*

12.6 Java

IKVM.Net: http://jilcsourceforge.net *(em inglês)*
JILC: http://jilcsourceforge.net *(em inglês)*
GNU Java Compiler: http://gcc.gnu.org/java/ *(em inglês)*
GNU Classpath: http://www.gnu.org/software/classpath/classpath.html *(em inglês)*
Java Portal: http://www.portaljava.com.br *(em português)*

12.7 Outros

ECMA (European Computer Manufacturing Association): http://www.ecma.ch *(em inglês)*
Guia da Conectiva sobre CVS: http://bazar.conectiva.com.br/~godoy/cvs/admin/ *(em português)*
Prevayler: www.prevayler.org *(em inglês)*
Revista ClubeDelphi: http://www.clubedelphi.net *(em português)*
WebMobile Magazine: http://www.portalwebmobile.com.br *(em português)*
Linha de Código: www.linhadecodigo.com.br *(em português)*

13 Índice Remissivo

E

F

G

I

J

L

M

N

P

Q

R

S

T

V

W

X