ESTUDO DO FRAMEWORK MONO

Quéli Giaretta, Jacques Duílio Brancher

Universidade Regional Integrada do Alto Uruguai e das Missões - Campus de Erechim

Av. Sete de Setembro, 1621 - Centro - Erechim - RS

queli@erechim.psi.br, jacques@uricer.edu.br

Resumo: *Este tutorial apresenta uma visão geral do Projeto Mono. A idéia básica é mostrar o desenvolvimento da plataforma, a estrutura de funcionamento, especificando as vantagens e novidades que o Mono disponibiliza junto com as novas tecnologias que o mesmo implementa. As linguagens de programação que podem ser utilizadas para a criação de novas aplicações, citando os principais projetos em desenvolvimento que utilizam desse framework, com uma breve descrição do Projeto Mono Brasil. Introduzindo a linguagem C#, uma das linguagens mais completas para criação de aplicações utilizando Mono.*

Palavras Chave: *framework, Mono, C#.*

Conteúdo

1. Introdução

1.1. O que é o Mono?

O Projeto Mono é uma iniciativa da comunidade para desenvolver uma versão baseada em GNU/Linux, como Software Livre, da plataforma de desenvolvimento .NET da Microsoft. Ele incorpora componentes-chave do .NET, incluindo um compilador para linguagem de programação C#, um compilador just-in-time para o Common Language Runtime, e um *suite* completo de bibliotecas de classe.

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 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.

Miguel de Icaza criou este projeto para facilitar o desenvolvimento, assim automaticamente todos os programas desenvolvidos sobre a plataforma .NET passam a serem híbridos, podendo ser executados em qualquer ambiente que possua um *framework* .NET instalado.

Esta disponível a versão beta do Mono, com a versão 1.0 do compilador C# e da máquina virtual, suporte x86, PowerPC, SPARC e S390, Java VM para rodar aplicativos JAVA e .NET, suporte para Linux, MacOS X, Windows, Solaris e HP-UX, e uma série de outras novidades. Sendo que a versão atual do Mono já é compatível com a versão 1.0 e 1.1 do .NET .

A plataforma .NET propõe o conceito de portabilidade, podendo desenvolver aplicativos inteiros tendo a certeza que irão ser compatíveis com outros sistemas operacionais sem precisar escrever nenhuma linha de código adicional (em muito breve). Outro benefício muito interessante desta plataforma é o desprendimento do *client*, podendo ele ser um pc, pockect pc, palm, etc....

Vários projetos já estão em andamento, sendo que uns para o desenvolvimento da própria ferramenta e outros para desenvolver aplicações especificas. Um exemplo disso é o MonoBasic que a comunidade brasileira ajuda desenvolver, trata-se de um compilador .NET dentro do projeto Mono.

Outro projeto interessante, é o Mono A.I. compreende uma série de ferramentas voltadas a programação de agentes e sistemas de inteligência artificial. O objetivo do projeto é criar uma plataforma para desenvolvimento de sistemas multi-agente inteligentes, com a definição de uma API de programação completa, suporte a *cluster* de máquina Mosix, processamento de agentes de forma distribuída, visualização em 3D de agentes em forma de constelações e comunidades de agentes.

1.2. 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 totalmente inter operacional com a plataforma Windows.

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.

O WebSite do projeto já possui diversos contribuintes, oferece sessões de documentação, artigos, palestras, how-tos, exemplos e, explicações detalhadas de como instalar o Mono.

1.3. Estrutura do *framework*

O Mono, que é um ambiente multi-linguagem e orientado a objetos, possui algumas características que facilitam a reutilização de classes entre as linguagens além de facilitar a estruturação de arquiteturas e soluções.

Organiza as APIs em divisões lógicas (*namespaces*) claras e consistentes, sem risco de dessincronia entre *headers* e a implementação, pois os metadados que substituem os *headers* são completos e estão dentro das montagens (*assemblies*) junto com o código. Além disso há um esquema de versionamento garantido pelo *runtime*, isso evita o uso de versões incompatíveis de classes.

A linguagem C# é uma boa migração para quem vem do C padrão do GCC, ou esteve brigando com o C++ do mesmo compilador. Traz orientação a objetos plena, sintaxe para metadados embutidos (atributos), distinção sintática para definição de propriedades e de campos e unicidade sintática para o uso de ambos.

O mecanismo de coleta de lixo (*garbage collection*), evita uma série de problemas (basicamente com ponteiros perdidos, e desperdício de memória) que normalmente abundam no desenvolvimento C/C++, pela dificuldade em eliminá-los sem um longo processo de depuração.

Em termos gerais se compararmos com o Java o fato de que mesmo com o JCP (Java *Community Process*) quem detém o poder sobre o seu futuro é a SUN. O Mono por sua vez baseia-se em padrões ISO (e ECMA) e é uma implementação livre, de excelente qualidade. O Mono engloba uma série de tecnologias em seus *class libraries* grosseiramente equivalente aos *class libraries* disponíveis para Java, com vantagens específicas para um lado e para outro em tópicos isolados. Mas no geral é mais fácil encapsular funcionalidades já prontas (como o GTK ou o QT) em Mono do que em Java (via JNI), e normalmente obtendo-se melhor performance.

Ao relacionarmos a estrutura do .NET temos uma visão do que esta sendo feito através do Mono. A (Figura 1) fornece o contexto dos ajustes da aplicação na estrutura Mono/Net.

As bibliotecas de classe fornecem uma gama de facilidades para o desenvolvimento das aplicações. Escritas primeiramente em C#, podem ser usadas com qualquer linguagem. A biblioteca de classe é estruturada com *namespaces*, e desenvolvida com bibliotecas compartilhadas denominadas *assemblies*. Quando falamos da estrutura do .NET, estamos consultando primeiramente estas bibliotecas de classe.

A C*ommon Language Infrastructure* (CLI), conhecido como *runtime*, implementa a execução do Mono. O *runtime* também é usado para executar a compilação de uma aplicação .NET. Para fazer funcionar uma aplicação, é preciso fazer chamada ao *runtime* definindo os parâmetros necessários.

A CLI é uma *máquina virtual* que contém um carregador (*loader*) para as classes (carrega as classes que um programa usa para executar sua funcionalidade), um compilador JIT (*just-intime*), que compila o programa na sua primeira utilização gerando um código para a CPU nativa e um ambiente de coleta de lixo (*garbage collection*), que gerencia o uso da memória do computador, eliminando os famosos

GPFs e similares. A CLI tem a mesma função que a Java *Virtual Machine* (JVM) da linguagem Java, é o componente que irá permitir que programas escritos em C# sejam executados em sistemas operacionais não-Windows, como o Linux.

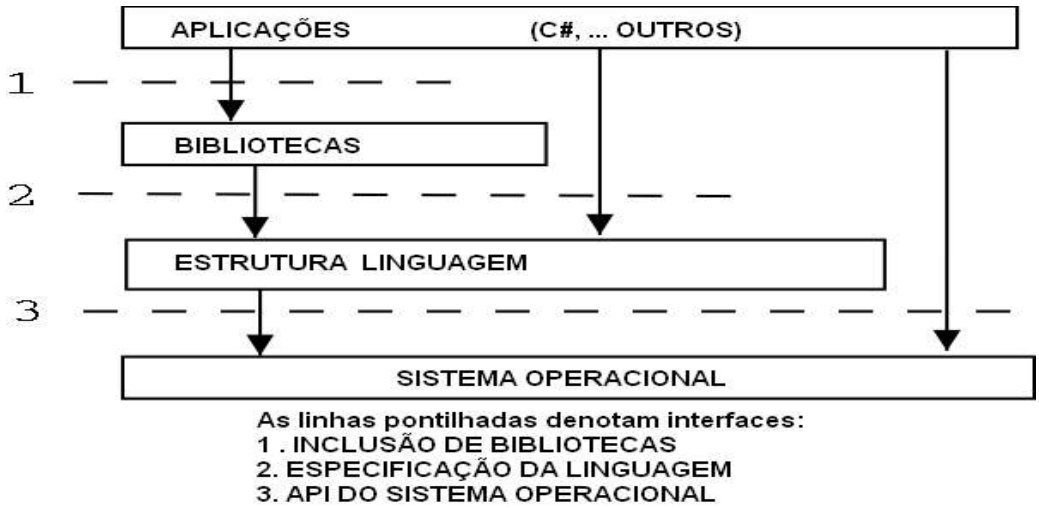

Figura 1. Mostra uma visão extremamente simplificada da estrutura da plataforma, deixando de considerar o ASP.NET e ADO.NET.

A biblioteca de classes é o elemento que permitirá que as aplicações realizem funções vitais, tais como processamento XML, entrada/saída e conexão a redes. O projeto Mono pretende criar uma biblioteca que seja compatível com a CLI da Microsoft, permitindo que os programas escritos para Linux/Mono sejam executados com a mesma funcionalidade no Windows/.NET. Além disso, existe uma grande expectativa de que a comunidade código livre crie bibliotecas adicionais que executem funcionalidades extras e que também sejam compatíveis com a CLI.

A *Common Language Specification* (CLS) define a interface para o CLI; por exemplo convenções tais como os tipos underlying para Enum. O compilador Mono gera uma imagem que se adapta ao CLS. Esta é a linguagem intermediária, *Common Intermediate Language.* O *runtime* do Mono verifica esta imagem e faz com que ela execute.

1.4. Tecnologias Implementadas

1.4.1. C#

Compilador C# (Padrão ECMA 334)

Ambiente de Execução CLI - *Common Language Infrastructure* (Padrão ECMA 335). Permite que as aplicações rodem independentes de plataforma.

Mono inclui uma implementação da linguagem C#, uma linguagem desenvolvida originalmente pela Microsoft . O C# emprega vários elementos dos sistemas Java, de C++ e *smalltalk.*

A linguagem C# é a aposta da Microsoft na concorrência com o Java, oferecendo todos os recursos de uma linguagem de programação moderna, como orientação a objetos, suporte para interfaces/componentes e, principalmente, código gerenciado (*managed code*).

O código gerenciado criado pelo C# não é um código específico para um determinado tipo de máquina, portanto ele independe de CPU. Isso permite que o mesmo código seja executado em uma grande variedade de máquinas, tais como

computadores pessoais (PCs), máquinas RISC, handhelds ou palms e celulares. É necessário, que exista uma máquina virtual (CLI) para cada tipo de máquina em que o código será executado.

Outra grande vantagem que resulta do código gerenciado é o aumento na segurança e na estabilidade do código gerado. Isto é obtido pelo modo como o código gerenciado e a máquina virtual tratam a execução do programa. Eles impedem qualquer operação que ponha a máquina em risco, mesmo que o código tente realizar este tipo de operação (as permissões do que pode ou não ser feito são atribuídas pelos administradores de rede). Este conjunto de recursos foi um dos principais motivos para a mudança de rumo da Ximian, que antes desenvolvia usando linguagens tradicionais como C.

Devido ao seu elegante projeto orientado a objeto, C# é uma escolha excelente para arquitetar uma ampla gama de componentes – de objetos de negócio de alto nível a aplicações no nível do sistema. Usando construções de linguagem C# simples, estes componentes podem ser convertidos em serviços Web, permitindo que eles sejam invocados pela Internet, a partir de qualquer linguagem rodando em qualquer sistema operacional.

C# é projetado para trazer desenvolvimento rápido para o programador C++ sem sacrificar o poder e o controle que têm sido a característica fundamental do C e C++. Devido a esta herança, C# possui um alto grau de fidelidade com C e C++. Desenvolvedores familiarizados com estas linguagens podem se tornar produtivos em C# rapidamente.

A linguagem é projetada para ajudar os desenvolvedores a fazer mais com um número menor de linhas de código e menos oportunidades de erro.

O novo modelo para desenvolvimento de aplicações significa que mais e mais soluções requerem o uso de padrões Web emergentes tais como *Hypertext Markup Language* (HTML), *Extensible Markup Language* (XML) e *Simple Object Access Protoco*l (SOAP). Ferramentas de desenvolvimento existentes hoje foram desenvolvidas antes da Internet existir ou quando a Web conforme nós a conhecemos hoje estava na sua infância. Como resultado, elas nem sempre proporcionam a melhor forma de trabalhar com as novas tecnologias Web.

Programadores C# podem utilizar um *framework* extensivo para construção de aplicações da plataforma Microsoft .NET ou então Mono. C# inclui suporte pré-construído para transformar qualquer componente em um serviço Web que pode ser invocado pela Internet – a partir de qualquer aplicação rodando em qualquer plataforma.

O que é ainda melhor, o *framework* dos serviços Web pode tornar os serviços Web existentes iguais a objetos C# nativos para o programador, permitindo assim que os desenvolvedores utilizem os serviços Web existentes com as habilidades de programação orientada a objeto que eles já possuem.

Existem mais recursos sutis que tornam o C# uma excelente ferramenta de programação Internet. Por exemplo, o XML está emergindo como a forma padrão de passar dados estruturados pela Internet, que freqüentemente são muito pequenos. Para performance melhorada, C# permite que os dados XML sejam mapeados diretamente em um tipo de dados *struct* ao invés de uma classe. Esta é uma forma mais eficiente de tratar pequenas quantidades de dados.

Mesmo os programadores C++ mais experientes podem cometer o mais simples dos erros – esquecer de inicializar uma variável, por exemplo – e freqüentemente estes erros simples resultam em problemas não previstos que podem permanecer não

descobertos por longos períodos de tempo. Uma vez que um programa está em uso na produção, pode ser muito custoso corrigir mesmo o mais simples erro de programação.

O projeto moderno do C# elimina os erros de programação C++ mais comuns. Por exemplo:

- Garbage collection libera o programador da sobrecarga do gerenciamento de memória manual.
- Variáveis no C# são inicializadas automaticamente pelo ambiente.
- Variáveis são *type-safe.*

O resultado final é uma linguagem que torna bem mais fácil para os desenvolvedores escrever e manter programas que resolvam problemas complexos.

Reduz os custos de desenvolvimento do dia a dia com suporte pré-construído para versionamento. Atualização de componentes de software é uma tarefa com alta probabilidade de erro. Revisões feitas no código podem alterar não intencionalmente a semântica de um programa existente. Para auxiliar o desenvolvedor neste problema, C# inclui o suporte a versionamento na linguagem. Por exemplo, a sobrescrita de método deve ser explícita; ela não pode acontecer de forma inadvertida como no C++ ou Java. Isto ajuda a evitar erros de codificação e preserva flexibilidade de versões. Um recurso relacionado é o suporte nativo para interfaces e herança de interface. Estes recursos habilitam *frameworks* complexos a serem desenvolvidos e evoluídos com o tempo. Em conjunto, estes recursos tornam mais robusto o processo de desenvolvimento de versões posteriores de um projeto, reduzindo assim os custos gerais de desenvolvimento para as versões sucessivas.

Melhor mapeamento entre processos de negócio e implementação:

Com o alto nível de esforço gasto pelas corporações no planejamento de negócio, é imperativo ter uma forte conexão entre os processos abstratos de negócio e a implementação real de software. Mas a maioria das ferramentas de linguagem não possui uma forma fácil de unir lógica de negócio ao código. Por exemplo, os desenvolvedores provavelmente utilizam comentários de código hoje em dia para identificar quais classes formam um objeto de negócio abstrato particular.

A linguagem C# permite metadados extensíveis *typed* que podem ser aplicados a qualquer objeto. Um arquiteto de projeto pode definir atributos específicos de domínio e aplicá-los a quaisquer classes de elemento da linguagem, interfaces e assim por diante. O desenvolvedor pode então examinar via programação os atributos de cada elemento. Isto torna fácil, por exemplo, escrever uma ferramenta automatizada que garantirá que cada classe ou interface está identificada corretamente como parte de um objeto de negócio particular, ou simplesmente criar relatórios com base nos atributos específicos de domínio de um objeto. A forte união entre os metadados customizados e o código do programa ajuda a fortalecer a conexão entre o comportamento pretendido do programa e a implementação real.

Interoperabilidade extensiva:

O ambiente *type-safe* gerenciado é apropriado para a maioria das aplicações corporativas. Mas a experiência mostra que algumas aplicações continuam a exigir código "nativo", ou por razões de performance ou para interoperar com interfaces de programação de aplicação (APIs) existentes. Estes cenários podem forçar os desenvolvedores a usar C++ mesmo quando eles prefeririam usar um ambiente de desenvolvimento mais produtivo.

C# endereça estes problemas da seguinte forma:

- Incluindo suporte nativo para o *Component Object Model* (COM) e APIs baseadas em Windows.
- Permitindo o uso restrito de ponteiros nativos.

Com C#, cada objeto é automaticamente um objeto COM. Desenvolvedores não têm mais que implementar explicitamente interfaces *IUnknown* e outras interfaces COM. Ao invés disto, estes recursos são pré-construídos. De forma similar, programas C# podem usar nativamente objetos COM existentes, independente da linguagem usada para sua criação.

Para os desenvolvedores que precisam disto, C# inclui um recurso especial que habilita um programa a chamar qualquer API nativa. Dentro de um bloco de código marcado em especial, desenvolvedores podem usar ponteiros e recursos C/C++ tradicionais tais como memória gerenciada manualmente e aritmética de ponteiros. Esta é uma enorme vantagem sobre os outros ambientes. Isto significa que programadores C# podem construir sobre sua base de código C e C++ existente, ao invés de descartá-la.

Nos dois casos – suporte a COM e acesso API nativo – a intenção é proporcionar ao desenvolvedor o poder e o controle essenciais sem ter que deixar o ambiente C#.

C# é uma linguagem orientada a objeto moderna que habilita os programadores a construir rápida e facilmente soluções para a plataforma Microsoft .NET. O *framework* proporcionado permite que os componentes C# se tornem serviços Web que estão disponíveis pela Internet, a partir de qualquer aplicação rodando em qualquer plataforma.

A linguagem melhora a produtividade do desenvolvedor e serve para eliminar erros de programação que podem levar a custos de desenvolvimento aumentados. C# traz o desenvolvimento Web para o programador C e C++ e mantém o poder e flexibilidade que estes desenvolvedores demandam.

1.4.2. VB

VisualBasic.NET trás muitas adições, Basic se tornando tão poderoso quanto C# e outras línguas. Mono funciona com um compilador de CLI, fazendo essa linguagem ser tão digna quanto C#. Graças às CLIs escritas em MonoBasic pode ser usado da mesma maneira que C#.

O MonoBasic (mbas) é o compilador .NET dentro do projeto Mono. Inicialmente contido dentro do mcs (o compilador C#), hoje evolui independentemente. A idéia é suportar 100% da sintaxe .NET (arquivo .vb) e oferecer extensões para fontes MonoBasic (arquivos .mbs), que podem ser misturados na mesma compilação.

1.4.3. LOGO

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

1.4.4. ADO.NET

Mono oferece uma relação comum às bases de dados. Dificilmente existira uma base de dados não suportada, um *directly*, com ODBC ou *libgda*. Muito portátil, funciona em todas as plataformas.

ADO.NET está sendo implementado e já está bastante funcional

1.4.5. ASP.NET

ASP.NET é o melhor *framework* que se tem para desenvolvimento de WebSites e WebServices. É uma linguagem independente. Imagine uma WebPage escrita em Pascal, LOGO, Basic, C #, C++.... Você pode acessar todas as características das bibliotecas de classe e isso é muito rápido, porque todas as bibliotecas são compiladas. ASP.NET apresenta também WebForms, que criam elementos HTML acessando objetos e organizando no HTTP.

ASP.NET já permite a criação de várias tipos de páginas Web e também WebServices.

1.4.6. 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.

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 Object Model 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.

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.

1.4.6.1. Arquitetura do GTK+

Foi projetado de forma modular, a biblioteca GTK é baseada em outras quatro bibliotecas:

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.

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.

Um aspecto importante de um *framework* GUI é a forma como a apresentação e organização dos componentes gráficos em uma janela é feita. 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 *suite* 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*. Tanto o GTK+ como as GUIs 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.

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 armazenar 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.

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 janela. Como os componentes não tem 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.

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 são suportadas por sistemas Unixes, 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.

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 está disponível, através de ferramentas externas que manipulam o arquivo XML gerado por ele, um exemplo é a interface C++ do GTK+, conhecida como Gtkmm.

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.

A medida que cresce o número de usuários do *framework* Mono, as ferramentas se tornam mais funcionais, e conseqüentemente 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.

1.4.7. QT#

O QT# foi criado pela comunidade KDE e fornece uma ligação às bibliotecas de desenvolvimento do KDE. A base das bibliotecas estão disponíveis também para Windows e MacOS X, mas são livres somente no Unix.

1.4.8. XML

Mono tem o suporte perfeito para XML, com muitos *parsers* e bibliotecas de processamento.

1.5. Plataformas suportadas

Mono pode ser utilizado no Windows, MacOS X, FreeBSD e Linux e por isso envolve 99% do mercado desktop. Usuário Windows e Linux em x86 devem ganhar 70%. Entretanto é necessitado mover a SPARC e a Solaris.

Atualmente suporta Gnome desktop, KDE e o Windows desktop (em processo).

2. Instalação

2.1. 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:

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

2.2. Instalando o Mono no Slackware

Pacotes:

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:

Utilizando os pacotes .gz, para fazer a instalação da maneira tradicional, "*configure, make* e *make install*".

Primeiro o *runtime*, descompacte o pacote mono-026.tar.gz em uma pasta qualquer.

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.*

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.

2.3. Instalação no Windows

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

Executar o instalador:

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:

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:

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

Instalando o GTK#

Buscar o 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*".

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 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:

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*".

2.4. 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 dos 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/.

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 coloca-la dentro do arquivo *.bashrc* (este arquivo encontra-se no seu diretório do seu usuário ou do /etc/.bashrc)

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

Para efetuar o download do manual você deve realizar o *login* no servidor digitando:

```
cvs login
```

O servidor do CVS irá solicitar uma senha, mas como estamos realizando o *login* como *anonymous* basta informar o seu endereço de email quando ele solicitar a senha

Agora a seguinte linha faz do *check out* (*co* é *check out* abreviado) e ira ser realizado o download do Guia do mono.

```
cvs -z3 co monkeyguide
```

2.4.1. 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:

```
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:

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:

```
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:

```
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. IDEs

3.1. 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 E*nterprise 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 mais melhor ao ambiente.

3.2. #Develop

#develop (SharpDevelop) é uma IDE livre para C# e projetos do VB.NET da plataforma do NET da Microsoft. É *open-source* (GPL). Posssui 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.

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*
- ... e mais

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. Runtime

O *runtime* Mono executa um mecanismo de JIT para a máquina virtual CLI, o carregador de classes, o *carbage collector, threading* de sistema e as bibliotecas de acesso aos metadados.

Temos atualmente dois *runtimes*:

mono: O compilador *Just In Time* sua implementação usa um seletor de instruções BURS. Suporta somente as máquinas x86 no JIT, até agora.

mint: O interpretador mono. Este é um mecanismo de *runtime.*

O *runtime* mono pode ser usado como um processo *stand-alone*, ou ser encaixado em aplicações.

Encaixar o *runtime* mono permite que as aplicações existentes em C ou C++ sejam estendidas em C# ao reúso de todo o código.

Ambiente Gerenciado:

Todo código desenvolvido para .NET *framework* é dito “gerenciado”. As principais características deste ambiente gerenciado são:

O executável não contém código Intel x86 e sim código MSIL com chamadas à biblioteca de classes;

O gerenciamento de memória é feito automaticamente com o uso de um coletor de lixo (*garbage collector*);

As atribuições têm o tipo validado;

A princípio o sistema de tipos é inviolável;

Operações que podem comprometer a segurança, como abrir um arquivo, exigem permissões especiais.

As qualidades acima permitem rodar programas de origem duvidosa, por exemplo da Internet, sem que estes programas possam comprometer a segurança e integridade do sistema.

Verificabilidade:

Todo programa criado pelo compilador C# é dito “verificável”. Isto quer dizer que o compilador JIT pode, em tempo de execução / compilação, verificar e garantir que o programa não faça nenhuma operação que possa comprometer a segurança e integridade do sistema.

Pode parecer estranho, mas existem instruções MSIL capazes de abrir brechas na segurança do sistema, como por exemplo, para manuseio direto de ponteiros ou “*casts*” inseguros. Estas instruções são necessárias em alguns casos, como por exemplo para que a própria biblioteca chame a API do Windows. Programas que contém estas instruções são ditos “não-verificáveis”.

O compilador C# pode criar programas não-verificáveis, incluindo manipulação direta de ponteiros, com a opção “*/unsafe*”. Já o compilador C++ sempre gera código não-verificável. Evidentemente é necessário um privilégio especial de segurança para rodar programas não-verificáveis.

É perfeitamente possível criar programas bastante úteis sem violar os critérios de “verificabilidade” e, conseqüentemente, segurança.

5. 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.

O Mcs não compila o código original, mas um tipo do bytecode, que possa ser processado pelo *runtime* do Mono.

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.

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.

O Assembler Mono é, o contrário do disassembler Mono, mas também é escrito em C#.

6. Ferramentas de Apoio

6.1. 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.

NUnit é distribuído com mono.

Nunit-gtk é a versão GUI do ambiente de teste Nunit.

6.2. 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 intrinsicamente uma boa estrutura aos ficheiros. As IDEs 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.

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.

6.3. Mono debugger

O 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.

7. Técnologias Implementadas

7.1. 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. Um bom exemplo seria um aplicativo com funcionalidade da busca, que deixa o usuário trabalhando, enquanto procura.

7.2. 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*.

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.

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 importar-se 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*.

Em cada aplicação existem dois tipos diferentes:

Um é o *Server*

Outro é *Client*

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

7.3. XML

XML *(**EX**tensible **M**arkup **L**anguage*), 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 as 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.

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.

É possível:

- 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.

7.4. ADO.NET

O *framework* .NET envia com o namespace, *System.Data*, que não é parte da biblioteca da classe de ECMA. Entretanto tem implementação Mono. Este namespace é chamado de "ADO.NET". O nome vem de ADO, o *ActiveX Data Objects*, a última camada do Database da Microsoft. Parcialmente construída sobre o ADO.NET.

ADO.NET é uma API de acesso a dados, em especial bases de dados relacionais. A Microsoft prove ao .NET *framework* os provedores de base de dados para MS SQL Server, Oracle, ODBC, Ole-DB e XML.
Mono pode acessar as mesmas bases, mas adicionalmente a Database MySQL, PostgreSQL, e IBM DB2.

As bases de dados não disponíveis podem ser utilizadas através de ODBC ou OLE-DB. Ambos são executados em sistemas Windows e Unix. Todas essas bases de dados tem uma interface em comum, são fáceis de se usar. Há também um provedor de XML, ele permite acessar arquivos de XML em uma base de dados.

Databases Suportadas:

Quando a Microsoft lançou o ADO.NET disponibilizou MS SQL Server, SqlXML, OLE DB e – desde o 1.1 - ODBC e Oracle no namespace System.Data, mono está na frente da MS nisso. Além de fornecer System.Data, e o namespace. Mono.Data

tem atualmente servidores para: Firebird, IBM DB2, Postgres, MySQL, SqlLite, Sybase, TDS Generic.

A implementação Mono's OLE DB, baseia-se na LibGDA, tem suporte à (e outros já listados) MS Access.

Outras bases de dados não listadas, no *unixodbc* dos sistemas Unix podem ser acessadas (e outros já listados) com os d*rivers* apropriados.

7.5. Utilizando o servidor ASP.NET

Além das típicas WebPages, mais e mais aplicações são baseadas na Web. A razão é, são fáceis de manter, atualizar, e muito utilizadas pelos usuário. Também o HTML evoluiu e com CSS, tornando possível projetos maiores. Aplicações Web geralmente são escritas na maior parte em Java, ASP, PHP ou - agora - em ASP.NET. Mono traz o ASP.NET para o Linux - junto com Apache uma das plataformas mais importantes usadas na web, hoje.

Parte integrante do .NET *framework*, ASP.NET é um conjunto de objetos, em tempo de desenvolvimento, e serviços, em tempo de execução, utilizados para desenvolver e rodar aplicações Web.

Vantagens do ASP.NET

ASP.NET é código compilado, a invés de interpretado como em plataformas anteriores;

Simplicidade no desenvolvimento, separando a lógica da aplicação da interface do usuário;

Suporte a aplicações Web ASP.Net com WebForms;

É possível desenvolver no Windows e rodar no Linux.

Existem duas formas de usar o ASP.NET do Mono:

Servidor em C# xsp

Módulo para o Apache 1.3 e 2.0

7.6. WebServices

O que são WebServices? Como se comunicam através da rede? Como trabalham? A idéia por trás do WebServices é a seguinte, você pode chamar as funções de seu código, que serão executadas em um WebServer. Assim você pode passar parâmetros à função e o WebServer retorna o valor.

Clientes e servidores podem ser desenvolvidos.

7.7. Gnome.NET

7.7.1. GTK#

GTK é uma abreviação para GIMP Tool Kit. O GIMP (GNU Image Manipulation Program) é um poderoso editor gráfico, ao estilo do Adobe Photoshop, com licença GPL, o que significa que ele é gratuito, e pode ser distribuído livremente. O GTK tem esse nome pois foi originalmente concebido para servir de ferramenta no desenvolvimento do GIMP. Devido à versatilidade das funções do GTK, hoje este é utilizado na produção de diversos outros programas além do GNU *Image Manipulation*

Program, que variam desde pequenos utilitários, como o GTK-ICQ, até grandes projetos, como o gerenciador de Desktop GNOME.

O GTK é na verdade um conjunto de *widgets* (você verá o significado desta palavra mais adiante), que usa funções de outra biblioteca chamada GDK (GIMP *Drawing Kit*), que por sua vez é um conjunto de funções que chamam outras funções de baixo nível do ambiente gráfico em que o programa é compilado. Todo o conjunto do GTK ainda depende de uma biblioteca chamada GLib (GNU *Library*) de funções úteis, comuns a vários programas GNU, e que aumentam portabilidade.

Existem interfaces de GTK para várias linguagens, embora este documento trate apenas da versão para C. Programar em GTK exige o entendimento de conceitos de orientação a objetos, que não são tão complicados para quem não conhece nada deste estilo de programação, e ainda servem como uma boa introdução para o assunto.

GTK# é a ligação desde *toolkit* ao Mono, quer dizer é usado por qualquer linguagem Mono incluindo MonoBasic e C#.

8. Desenvolvimento Utilizando C#

8.1. Estrutura, Interpretador/Compilador?

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 interpretado 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 “*garbage 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*”.

8.2. Aplicação Exemplo

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!");
    }
}
```

8.3. Namespace/Definição de classes/Main()

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.

8.4. Utilizando o Compilandor C#

Os fontes em C# tem extensão "*cs*". Todos os fontes em um "projeto" são compilados diretamente para um único "*assembly*" (.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 '). Uma biblioteca de ligação dinâmica contém conjunto das classes, que podem então ser instaciadas 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).

Tabela 1 - Principais interruptores do compilador.

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 executavel 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.

8.5. Tipos de Dados

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.

8.5.1. Tipo Valor

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 tipo pré-definidos. Porque em C# todos os tipos de valores aparentemente fundamentais são definidos sobre um tipo de objeto. Então temos que os tipos .*Systenm* definidos no *framework* .NET correspondem a tipos pré-definidos.

Tabela 2 – Tipos pré-definidos em C#.

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 $\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$ com 7 figuras significativas
Dobro	System.Double	Sim	8	Aproximadamente $\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$ com 15 ou 16 figuras significativas
Decimal	System.Decimal	Sim	12	Aproximadamente $\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$ com 28 ou 29 figuras significativas
Char	System.Char	N/A	2	Algum caráter de *Unicode* (bocado 16)
Bool	System.Boolean	N/A	1/2	verdadeiro ou falso

Exemplo de declaração de duas variáveis e definidas como inteiro:

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

8.5.2. Tipo de Referência

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");}
```

8.6. Operadores

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.

Tabela 3 – Operadores Padrão/Sobrecarga.

Categoria	**Nome**	**Exemplo**	**Sobrecarga de Operadores**
Primary	Grouping	(a+b)	Não
	Member	A.B	Não
	Struct pointer member access	A->B	Não
	Method call	f(x)	Não

Categoria	**Nome**	**Exemplo**	**Sobrecarga de Operadores**
	Post increment	c++	Sim
	Post decrement	c--	Sim
	Constructor call	c = new Coord();	Não
	Array stack allocation	int* c = stackalloc int[10]	Não
	Struct size retrieval	sizeof (int)	Não
	Arithmetic check on	checked {byte c = (byte) d;}	Não
	Arithmetic check off	unchecked {byte c = (byte) d;}	Não
Unary	Positive value	+10	Sim
	Negative value	-10	Sim
	Not	!(c==d)	Sim
	Bitwise complement	~(int x)	Sim
	Pre increment	++c	Sim
	Pre decrement	--c	Sim
	Type cast	(myType)c	Não
	Value at address	int* c = d;	Não
	Address value of	int* c = &d;	Não
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

Categoria	Nome	Exemplo	Sobrecarga de Operadores
	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

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}
```

8.7. Ponteiros

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;
		}
	}
}
```

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

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.

8.8.1. Controle De Fluxo: Indicações de Laço

8.8.0.2. while

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++;
}
```

8.8.0.3. do-while

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);
```

8.8.0.4. for

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);
}
```

8.8.0.5. foreach

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.

8.8.2. Controle de Fluxo: Jump

8.8.0.7. break

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;
}
```

8.8.0.8. continue

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++;
}
```

8.8.0.9. goto

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;
```

8.8.3. Controle de Fluxo: Estruturas de Seleção

8.8.0.11. if - else

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");

8.8.0.12. switch - default

A intrução ' *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;
```

8.9. Arrays

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 arrays 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 arrray. 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]);
```

9. Referências

BINHARA, A., TEIXEIRA R. *Projeto Mono Brasil* (2003) http://monobrasil.softwarelivre.org/. Março, 2004.

GLADE GTK+ User Interface Builder (2004). http://glade.gnome.org/. Último acesso, Abril (2004).

GTK# the Free .NET GUI (2004). http://gtk-sharp.sourceforge.net/. Último acesso, Maio 2004.

GTK+ - The Gimp Toolkit (2004). http://www.gtk.org/. Último acesso, Maio 2004.

GtkSharpWiki - Home Page (2004). http://www.nullenvoid.com/gtksharp/wiki/. Último acesso, Maio 2004.

KUHN, Bradley M. *Free Software Foundation* (2003). http://xmundo.net , Abril, 2004.

Microsoft .Net Framework Home. Disponível em: http://msdn.microsoft.com/netframework/. Abril, 2004.

ROITH, J. e HANSEN, M. W. *Mono - Doc* (2003) http://Mono.ximian.com:8080/index.html, Fevereiro, 2004

ROITH, J. e HANSEN, M. W. The Mono Handbook (2003) http://www.gotmono.com/docs. Fevereiro, 2004.

SANCHES, André L. Programação para GTK/GNOME com C++ (2003).

Softsteel Solutions - C# Tutorial (2003). Disponível em: http://www.softsteel.co.uk/tutorials/cSharp/cIndex.html. Fevereiro, 2004.

Website of the Eclipse Foundation. Disponível em: http://www.eclipse.org. Março, 2004.