Programação com Boo

Espante a dificuldade

A linguagem Boo oferece o melhor de três mundos: a sintaxe amigável do Python, a arquitura .NET e a tipagem forte de C#.

por Martin Streicher

Diferentemente das outras plataformas de desenvolvimento, o framework *.NET* consegue misturar e combinar código a partir de qualquer linguagem de programação. Para quem programa em .NET, a forma de código universal conhecida como *Microsoft Intermediate Language* (MSIL) é uma *lingua franca*. Após traduzir seu código-fonte para o formato MSIL, é possível combiná-lo com, por exemplo, *Visual BASIC .NET ou C#*, para assim produzir um executável.

Com efeito, dada toda essa flexibilidade, os programadores têm adaptado muitas linguagens populares para o .NET. O *IronPython* [1] é uma implementação completa do Python para .NET, enquanto que o *IronRuby* [2] é uma proposta de implementação do Ruby para .NET. Além disso, podemos encontrar versões de *Java*, *Lisp*, e *Smalltalk* para .NET. Mais ainda, se você não gostar de nenhuma das atuais linguagens de programação, pode criar a sua própria. Se você consegue consumir código-fonte e produzir MSIL, o céu é o limite.

Na verdade, essa é a gênese do *Boo* [3]. Baseado na sintaxe do Python – gentil com as articulações –, mas também apaixonado pela arquitetura .NET e pela tipagem forte de C#, o desenvolvedor Rodrigo Barreto de Oliveira resolveu combinar as melhores características dessas linguagens – com a quantidade certa de Ruby

Listagem 1: Reformatação de texto

```
01 import System.Text.RegularExpressions
02
03 def fimDePalavra(t as string):
04  return m.Index for m as Match in /\b|$/.Matches(s)
05
06 def quebraLinhas(t as string, colunas as int):
07
08   linhas = []
09
10   proxFim = colunas
11   ultFim = 0
12   inicioLinha = 0
13   for fp as int in fimDePalavra(s):
14     if fp > proxFim:
15       linha = t[inicioLinha:ultFim]
16       linhas.Add(linha.Trim())
17       inicioLinha = ultFim
18       proxFim = ultFim + colunas
19     ultFim = fp
20
21   linhas.Add(t[inicioLinha:].Trim())
22
23   return linhas
24
25 def testa(t):
26   print(join(quebraLinhas(t, 12), "*\n*"))
```

– para formar algo facilmente adaptável ao desenvolvimento iterativo.

Este artigo ajuda a entender e se iniciar no Boo e mostra como utilizar a IDE *MonoDevelop* para escrever e testar alguns códigos em Boo [4][5]. O Boo está disponível sob uma licença no estilo da MIT, que oferece ampla margem para a utilização do código e a criação de trabalhos derivados.

Python + Ruby + C# = Boo

O Python possui uma sintaxe mínima e utiliza espaços em branco para formar a estrutura do programa. O Boo segue a mesma linha. A **listagem 1** é um programa em Boo que reformata as linhas de texto para caber dentro de uma determinada largura.

A saída de `teste("este texto é mesmo bem comprido")` é um texto quebrado com no máximo 12 caracteres por linha:

Quadro 1: Boo-as ideias

O *Boo Primer* [6] é uma introdução muito bem escrita à linguagem Boo. As construções e a sintaxe de cada elemento são descritas em detalhes. Também se aprende um bocado navegando pelo código de exemplo fornecido com o compilador do Boo e suas suítes de teste. À medida que se estuda a linguagem, começa-se a ter uma noção de como ela funciona.

O Boo oferece tanto um tipo de lista quanto um de vetor. A lista pode variar de tamanho, enquanto o vetor tem tamanho fixo. É possível usar tanto uma lista quanto um vetor para extrair um, alguns ou todos os elementos, e ambos podem conter elementos heterogêneos. A **listagem 2** mostra algumas semelhanças e diferenças entre uma lista e um vetor.

O código da **listagem 2** produz a seguinte saída:

```
[0, 1, 2, 3, 4]
['hello', 1, true, System.Object]
[10, 'hello']
[1, True]

(0, 1, 2, 3, 4)
(0, 1, 5, 3, 4)
(1, 5)
```

Tanto a lista quanto o vetor utilizam colchetes (`[]`) para indexação e "fatias" (mais conhecido como *slicing*); uma lista é apresentada com os colchetes, enquanto um vetor possui parênteses, para diferenciá-lo. Uma "fatia" (*slice*) pode ser um elemento único ou uma faixa. Como exemplo, `m [1:6]` inclui os elementos desde o de índice 1 até o de índice 5. Já `m[:5]` é o conjunto de todos os elementos até o de índice 4. Acessos a índices além dos limites do vetor ou lista produzem uma exceção.

Curiosamente, uma cadeia de caracteres (*string*) se comporta como uma faixa e uma lista.

```
print (s = List('abcdefghi'))
[a, b, c, d, e, f, g, h, i]
print s[0:4]
[a, b, c, d]
print (s.Add('z'[0]))
[a, b, c, d, e, f, g, h, i, z]
```

Tal como Perl e Ruby, Boo também oferece um hash para armazenar pares no estilo `(chave, valor)`.

Uma `struct` (estrutura, na abreviação) é como uma classe – ela pode conter variáveis e métodos, mas é armazenada como um valor, em vez de uma referência. Um método também é um objeto e tem seus próprios métodos. Tornar cada função seu próprio objeto tem sua praticidade. Por exemplo, é possível executar instantaneamente uma função em sua própria thread chamando o seu próprio método `BeginInvoke()`. Também se pode encerrar a thread chamando o método `EndInvoke()`.

O Boo fornece ainda uma macro chamada `lock` para conceder a uma thread acesso mutuamente exclusivo a um objeto durante uma operação.

Também são fornecidas outras macros, mas o fato mais atraente é que qualquer pessoa pode criar uma nova macro utilizando qualquer linguagem. É claro que as macros salvam tempo de digitação, mas também reduzem erros e compartimentalizam seções de código. Por exemplo, com uma macro é possível transformar a sequência verbosa abaixo:

```
instanciaMuitoComprida = Foo()
instanciaMuitoComprida.f1 = 100
instanciaMuitoComprida.f2 = "abc"
instanciaMuitoComprida.FazerAlgo()
```

em uma declaração mais sucinta:

```
with instanciaMuitoComprida:
        _f1 = 100
        _f2 = "abc"
        _FazerAlgo()
```

Boo possui outras características agradáveis, mas o uso de macros, que permite a criação de linguagens para usos específicos, talvez seja seu recurso mais poderoso.

Listagem 2: Lista e vetor

```
01 l = List(range(5))
02 m = [‘hello’, 1, true, object]
03 n = []
04 n.Add(10)
05 n.Add(‘dez’)
06 print l
07 print m
08 print n
09 print m[1:3]
10
11 a = array(range(5))
12 print a
13 a(2) = 5
14 print a
15 print a[1:3]
```

```
este texto é*
*mesmo bem*
*comprido
```

Como se pode ver na **listagem 1**, o Boo é bastante espartano: não possui pontos-e-vírgulas, chaves, classes (se não forem necessárias) e não exige a declaração de variáveis. Basta atribuir e continuar. Mas o Boo é simples apenas na forma. O tipo é aplicado tanto de forma implícita quanto explícita. Argumentos formais podem ser tipados, como se pode ver na definição do método `quebraLinhas()`. Caso contrário, o tipo é inferido.

Por exemplo, a atribuição `linhas = []` deixa implícito que `linhas` é uma lista. Da mesma forma, `proxQuebra = colunas` deduz que `proxQuebra` é um `int` (inteiro, evidentemente).

Ruby também infere tipos. Tal como em Ruby, tudo no Boo é um objeto. O interessante trecho a seguir veio do *Boo Primer* [6] e ilustra isso (após compilar o shell interativo de Boo, como descrito posteriormente neste artigo, tente utilizar esse código).

```
o as object
o = ‘12’
o = 4
```

Por `o` estar instanciado como um objeto, a subclasse de todos os tipos, ele pode fazer referência a qualquer outro tipo. Este exemplo canônico demonstra o polimorfismo. Entretanto, se você declarar `i` de uma forma mais restrita, o Boo irá capturar uma atribuição errada.

```
i as int
i = 4
i = ‘12’
```

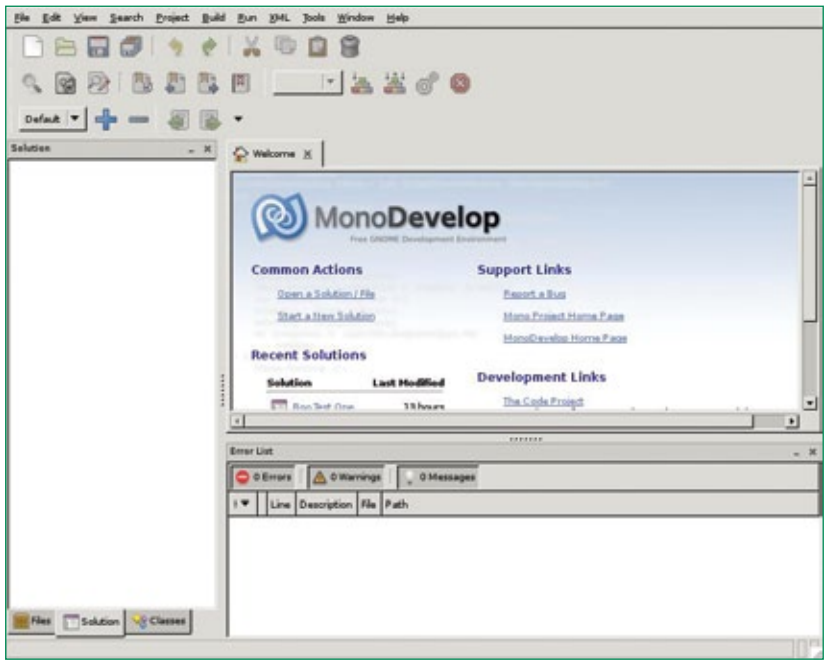

Figura 1 Janela principal do MonoDevelop.

```
a = ‘12’
--------
ERROR: Cannot convert ‘string’ to
➥ ‘int’.
```

Ruby e Boo possuem outro recurso em comum: ambos suportam tipagem por semelhança: se um objeto tem aparência de string, age como string e reage aos mesmos métodos que um objeto do tipo string, ele deve ser ou uma string ou pelo menos um mímico sagaz o suficiente para se passar por uma string.

Para utilizar a tipagem por semelhança em Boo, utilize o tipo `duck` (em inglês, chama-se a tipagem por semelhança de *duck typing*). O Boo pula a verificação dos tipos de variáveis cujo tipo declarado é duck.

```
fudd as int
daffy as duck
fudd = 0
daffy = 1
daffy = “quack”
fudd = “fique quieto”
ERROR: Cannot convert ‘string’
➥ to ‘int’.
```

Normalmente, só é preciso especificar um tipo quando vantajoso. Por exemplo, o Boo oferece a sobrecarga de métodos, um benefício adicional da declaração de tipo dos argumentos. Caso contrário, não declare um tipo; deixe a inferência fazer o trabalho pesado.

```
# x é int e pode somar
x = 5
x += 5
# agora, x é string
x = ‘hello’
print x.ToUpper()
‘HELLO’
```

Os tipos mais comuns de programação são sucintos e possuem nomes em minúsculas, tais como *object*, *int*, *string*, *double* e *bool*. Já os tipos mais complexos são capitalizados, como *List* e *Match*.

Iniciar o Boo

Para utilizar o Boo, é preciso instalar o ambiente de desenvolvimento integrado (IDE, de *Integrated Development Environment*) *MonoDevelop*, uma suíte de ferramentas de codificação elaborada para C# e outras linguagens .NET. A versão mais recente do MonoDevelop é a 1.9.2, uma prévia da versão 2.0. Os pacotes binários do Mono e do MonoDevelop estão disponíveis para várias distribuições Linux e outras plataformas, incluindo Mac e Windows, mas também é possí-

Listagem 3: Configuração do MonoDevelop

```
01 # Cria um local de trabalho
02 $ mkdir $HOME/mono
03 $ cd $HOME/mono
04
05 $ # Baixa, descompacta e compila os fontes
06 $ ## Mono
07 $ wget http://ftp.novell.com/pub/mono/sources/mono/mono-2.2.tar.bz2
08 $ tar xjf mono-2.2.tar.bz2
09 $ ( cd mono-2.2; ./configure; make; sudo make install )
10
11 $ ## GDI+
12 $ wget http://ftp.novell.com/pub/mono/sources/libgdiplus/libgdiplus-2.2.tar.bz2
13 $ tar xjf libgdiplus-2.2.tar.bz2
14 $ ( cd libgdiplus-2.2.; ./configure; make; sudo make install )
15 $ # Acrescenta /usr/local/lib ao caminho das bibliotecas
16 $ sudo echo '/usr/local/lib' >> /etc/ld.so.conf.d/win32.conf
17 $ sudo ldconfig
18
19 $ ## GTK#
20 $ wget http://ftp.novell.com/pub/mono/sources/gtk-sharp212/gtk-sharp-2.12.7.tar.bz2
21 $ tar xjf gtk-sharp-2.12.7.tar.bz2
22 $ ( cd gtk-sharp-2.12.7; ./configure; make; sudo make install )
23
24 $ ## GNOME#
25 $ wget http://ftp.novell.com/pub/mono/sources/gnome-sharp220/gnome-sharp-2.20.1.tar.bz2
26 $ tar xjf gnome-sharp-2.20.1.tar.bz2
27 $ ( cd gnome-sharp-2.20.1; ./configure; make; sudo make install )
28
29 $ ## Ferramentas do Mono
30 $ wget http://ftp.novell.com/pub/mono/sources/mono-addins/mono-addins-0.4.zip
31 $ unzip mono-addins-0.4.zip
32 $ ( cd mono-addins-0.4; ./configure; make; sudo make install )
33
34 $ ## IDE MonoDevelop
35 $ wget http://ftp.novell.com/pub/mono/sources/monodevelop/monodevelop-1.9.2.tar.bz2
36 $ tar xjf monodevelop-1.9.2.tar.bz2
37 $ ( cd monodevelop-1.9.2; ./configure; make; sudo make install )
38
39 $ ## Ferramenta de compilação Nant
40 $ wget http://superb-east.dl.sourceforge.net/sourceforge/nant/nant-0.86-beta1-src.tar.gz
41 $ tar xzf nant-0.86-beta1-src.tar.gz
42 $ (cd nant-0.86-beta1; make; sudo make install )
43
44 $ ## Depurador Mono
45 $ wget http://ftp.novell.com/pub/mono/sources/mono-debugger/mono-debugger-2.2.tar.bz2
46 $ tar xzf mono-debugger-2.2.tar.bz2
47 $ ( cd mono-debugger-2.2; ./configure; make; sudo make install )
48
49 $ ## Suporte ao Depurador Mono no MonoDevelop
50 $ wget http://ftp.novell.com/pub/mono/sources/monodevelop-debugger-mdb/monodevelop-debugger-mdb-1.9.2.
➥tar.bz2
51 $ tar xjf monodevelop-debugger-mdb-1.9.2.tar.bz2
52 $ (cd monodevelop-debugger-mdb-1.9.2; make; make install )
```

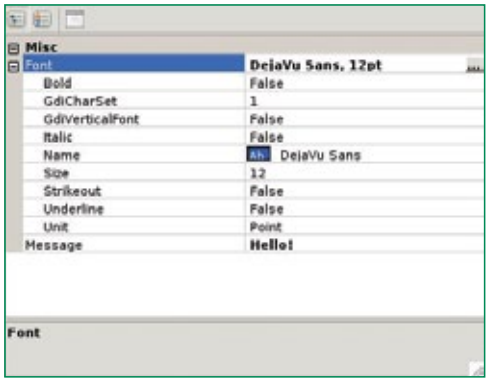

Figura 2 Fontes e Boo – resultados da **listagem 6**.

vel compilar o software a partir dos fontes.

Para compilar o código-fonte, certifique-se de que a máquina possui as ferramentas de desenvolvimento obrigatórias para compilar aplicativos Gnome.

Alguns outros utilitários e bibliotecas de desenvolvimento também são necessários: uma máquina virtual Java, o gerador de *parsers Bison*, o utilitário *pkg-config*, a biblioteca de renderização *Pango*, a biblioteca de acessibilidade *ATK*, as bibliotecas *GTK+* 2,0, a biblioteca *Curses*, alguns *bindings* de linha de comando e a Glib versão 2.0 ou posterior.

Em seguida, baixe e descompacte os *tarballs* do MonoDevelop e suas dependências (veja a lista de componentes completa no site do Mono [7]). São nove os pacotes exigidos (**listagem 3**).

Preste atenção especial às instruções do GDI+: é necessário configurar o carregador para encontrar as bibliotecas em `/usr/local/lib`. Se você tentar executar alguns aplicativos do Boo que fazem uso de *Windows Forms* e obtiver um erro como *System.DllNotFoundException: Gdiplus.dll*, provavelmente é porque você pulou essa etapa (veja o site do projeto Mono para mais informações [8]).

A construção completa leva cerca de 30 minutos e produz o "motor" do Mono, a ferramenta de construção *nant* (semelhante ao *make* em finalidade), a IDE MonoDevelop, um depurador e várias bibliotecas do Linux e do .NET. Por padrão, todo o software é instalado em `/usr/local/` e seus subdiretórios. Para usar outro diretório, use o comando `./configure --prefix /outro/diretório` com cada comando.

O próximo passo é baixar e compilar o compilador Boo e seu respectivo ambiente para MonoDevelop:

```
$ wget http://dist.codehaus.org/
➥boo/distributions/
➥boo-0.9.0.3203-2-src.zip
$ mkdir boo; unzip ../
➥boo-0.9.0.3203-2-src.zip
```

Imediatamente, edite o arquivo `default.build` e altere a linha `<property name="skip.vs2005" value="False" />` para `<property name="skip.vs2005" value="True" />`. Este passo pula a parte da compilação específica para o Visual Studio. Agora, compile o Boo:

```
$ cd boo
$ /usr/local/bin/nant rebuild
$ /usr/local/bin/nant update-bin
$ /usr/local/bin/nant
➥ compile-tests && nunit-console
➥ tests/build/*Tests.dll
$ /usr/local/bin/nant install
$ cd ..
```

Agora já temos uma cópia perfeitamente funcional do compilador Boo. Para experimentá-lo, crie e execute um dos programas de exemplo do Boo disponíveis em `./examples/`.

```
$ # Compile e execute o código
$ booc examples/arrayperformance.
➥boo
$ mono arrayperformance.exe
153.613 elapsed.
250.573 elapsed.
216.785 elapsed.
$ # Interprete o código
$ booi arrayperformance.boo
153.613 elapsed.
250.573 elapsed.
216.785 elapsed.
```

Sucesso! O programa de exemplo `arrayperformance.boo` copia um gran-

Listagem 4: Uso do booish

```
01 Welcome to booish, an interpreter for the boo programming
➥ language.
02 Running boo 0.9.0.3203 in CLR v2.0.50727.1433.
03
04 Enter boo code in the prompt below (or type /help).
05 >>> print "Hello, world"
06 Hello, world
```

Listagem 5: Inclusão do Boo

```
01 $ # Os dois passos a seguir são necessários enquanto os
02 $ # makefiles do Boo usados antes não forem corrigidos.
03 $ sudo mkdir -p /usr/local/lib/boo
04 $ sudo cp /usr/local/lib/mono/boo/*.dll /usr/local/lib/boo
05
06 $ wget http://ftp.novell.com/pub/mono/sources/monodevelop-boo/
➥monodevelop-boo-1.9.2.tar.bz2
07 $ tar xjf monodevelop-boo-1.9.2.tar.bz2
08 $ ( cd monodevelop-boo-1.9.2; ./configure; make; sudo make
➥ install )
```

de vetor e mede o tempo decorrido. Além disso, podemos testar o Boo diretamente em seu interpretador interativo, o *booish* (**listagem 4**).

Finalmente estamos prontos para incluir no MonoDevelop o suporte ao Boo (**listagem 5**). O site da Linux Magazine [9] possui um link para scripts que compilam o software.

Boo via MonoDevelop

Primeiro inicie a IDE MonoDevelop. A janela principal se assemelha à **figura 1**. Para criar um aplicativo Boo, clique em *Start a New Solution | Boo | Empty Project*. Em seguida, escolha um nome para o projeto e um local para armazená-lo e clique em *OK*. Na tela seguinte, marque *Unix Integration* (integração com Unix, totalmente opcional) e clique em *OK*. Agora a lista à esquerda deve mostrar uma "solução".

Para escolher a solução, clique com o botão direito do mouse sobre ela e escolha *Add | New File...*. Em seguida, escolha *Empty File*, dê um nome ao arquivo e depois clique em *New*. Neste ponto, já podemos escrever código em Boo no painel principal. O editor colore a sintaxe conforme se digita e automaticamente identa com base no contexto. E como estamos usando o Mono, é possível utilizar o framework .NET ou qualquer outro que possua *bindings* de CLI, como o *Gtk#*. Agora digite no MonoDevelop o código da **listagem 6**, salve o arquivo e escolha *Run | Run*.

A **listagem 6** produz a saída mostrada na **figura 2**. Este exemplo demonstra também as classes do Boo. A diretiva `classPropertyEditor(Form)` define a classe `PropertyEditor` e a deriva a partir da classe `Form` dos *Windows Forms*. O método `constructor()` é auto-explicativo. As declarações `[property(Message)]` e `_message as string` criam um método *getter* (de consulta) e outro *setter* (de definição) com nomes para a `property_message`, que é uma string Boo. A declaração `[property(Font)]` é semelhante, embora seja uma fonte.

Listagem 6: Teste das classes do Boo

```
01 import System.Windows.Forms from System.Windows.Forms
02 import System.Drawing from System.Drawing
03
04 class PropertyEditor(Form):
05     def constructor([required]obj):
06         grid = PropertyGrid(Dock: DockStyle.Fill, SelectedObject:
➥ obj)
07         Controls.Add(grid)
08
09 class Options:
10     [property(Message)]
11     _message as string
12
13     [property(Font)]
14     _font as System.Drawing.Font
15
16 options = Options(Message: "Ola!", Font: Font("Lucida Console",
➥ 12.0))
17 editor = PropertyEditor(options)
18 editor.ShowDialog()
19 print(options.Message)
```

Economia de letras e sanidade

A página do projeto Boo oferece receitas para acesso a bancos de dados, soluções gráficas e muito mais. Os desenvolvedores do Boo se socializam em um grupo do Google, um canal de IRC, uma lista e um wiki. Atualmente, a equipe está focada em melhorar o suporte ao Boo no MonoDevelop.

Se você estiver no Windows, Linux ou qualquer outra plataforma com Mono e estiver cansado de escrever chaves e definir tipos, conheça o Boo. Ele oferece o melhor de pelo menos três mundos. ■

Mais informações

[1] IronPython: http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython

[2] IronRuby: http://www.ironruby.net/

[3] Boo: http://boo.codehaus.org/

[4] Mono: http://mono-project.com/

[5] MonoDevelop: http://monodevelop.com/

[6] Boo Primer: http://boo.codehaus.org/Boo+Primer

[7] Lista completa de pacotes para o Mono: http://ftp.novell.com/pub/mono/sources-stable

[8] Como resolver bibliotecas perdidas: http://mono-project.com/DllNotFoundException e http://mono-project.com/Config_DllMap

[9] Listagens deste artigo: http://www.lnm.com.br/arquivos/LM/56/boo