



A Lean Mean DSL Machine

Rodrigo B. De Oliveira
rodrigobamboo@gmail.com

Mono Summit 2007

boo is a statically typed language

```
i as int
```

```
i = "Las Llamas Son Mas Grandes Que Las Ranas"
```

```
-----^
```

```
ERROR: Cannot convert 'string' to 'int'
```

static typing not to be confused with baby sitting needy compilers

```
class Convoluted {  
    Map<String, List<Map<String, Object>>> mappings =  
        new HashMap<String, List<Map<String, Object>>>();  
  
    public Map<String, List<Map<String, Object>>>  
        getMappings() {  
            return mappings;  
        }  
}
```

boo has type inference

```
i = 42  
i = "Cuidado, Llamas!"
```

```
-----^
```

```
ERROR: Cannot convert 'string' to 'int'
```

boo supports duck typing

if it walks like a duck...

if it quacks like a duck....

duck typing for talking to objects whose type is unknown

```
ie as duck = CreateInstance("InternetExplorer.Application")  
ie.Visible = true  
ie.Navigate2("http://www.go-mono.com/monologue/")
```

duck typing for dynamic dispatching (multimethods)

```
// Eval will be dispatched to the right overload  
// which could not be determined in compile time  
value = (evaluator as duck).Eval(expression)
```

boo - inspired by Python - uses indentation to delimit blocks

```
if SeeLlamas():  
    Scream("Cuidado, Llamas!")  
    Scream("Cuidado, Cuidado, Cuidado, Llamas!")
```

boo is object oriented (as much as allowed by the CTS)

```
class Dessert:  
    public name as string  
  
    override def ToString():  
        return name  
  
obj = Dessert(name: "Crunchy Frog")
```

boo is functional

```
function = WebClient().DownloadFile  
call = function.BeginInvoke(url, localFile)
```

boo is functional

```
callable Malkovich() as Malkovich
```

```
def malkovich() as Malkovich:  
    print "Malkovich!"  
    return malkovich
```

```
malkovich() () ()
```

boo is sweet

```
l = [42, "Silly", 1.618]
h = { "life, the universe and everything": 42,
      "phi": 1.618 }
t = 55ms
```

boo is sweet - interpolation

```
print "Total time is ${t + 20ms}"
```

boo is sweet generators (a.k.a. comprehensions)

```
deadParrots = parrot.Name for parrot in parrots \  
                if parrot.IsDeceased
```

boo is wrist friendly

```
import System
import System.Net
import System.Threading

url, local = argv

client = WebClient()
call = client.DownloadFile.BeginInvoke(url, local)
while not call.IsCompleted:
    Console.Write(".")
    Thread.Sleep(50ms)
Console.WriteLine()
```

boo is interactive

DEMO

**"The limits of my language
mean the limits of my world."
Ludwig Wittgenstein**

boo is extensible (macros)

```
using reader=File.OpenText("llamas.txt") :  
    print reader.ReadLine()
```

boo is extensible (attributes)

```
def Scream([required] message as string):  
    print message.ToUpper()
```

boo is extensible (steps)

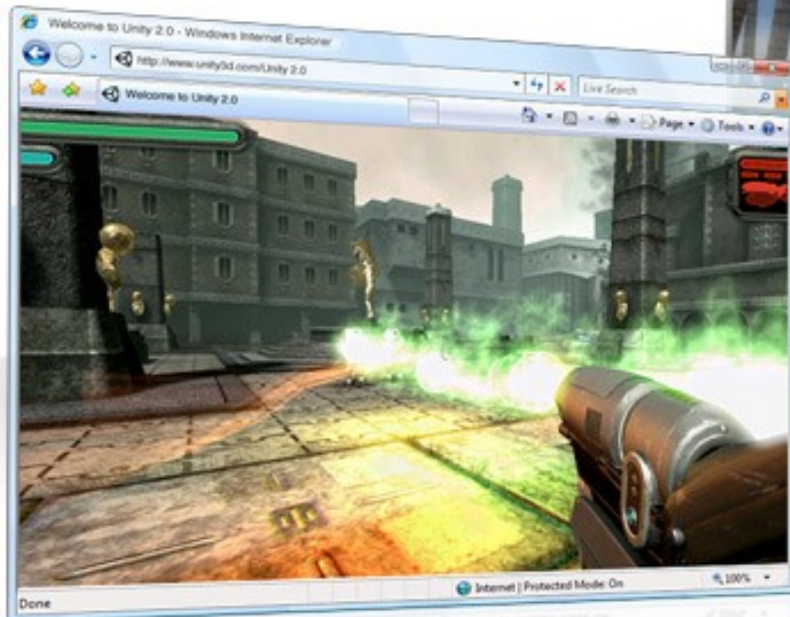
```
class StyleCheckerStep(AbstractVisitorCompilerStep):  
    override def Run():  
        Visit(CompileUnit)  
  
    override def LeaveClassDefinition(node as ClassDefinition):  
        if char.IsUpper(node.Name[0]): return  
  
        msg = "Class name '${node.Name}' should start with uppercase letter!"  
        Errors.Add(CompilerError(node, msg))
```

boo is an infrastructure for building languages

```
var a = 3;
eval( "class Foo { "
      + "    public function Run() { "
      + "        a = 42;"
      + "    } "
      + "}; "
      + "new Foo().Run();" );

print(a);
```

boo is an infrastructure for building languages



General Purpose Languages
or
Domain Specific Languages

Specter is a testing DSL (BDD)

```
import Specter
import NUnit.Framework

# a context is effectively a class that specifies the
# behaviour of an object that is in a given state.
context "Empty stack":
    # object the spec is defining
    stack as Stack

    # optional setup will be run before executing each specify block
    setup:
        stack = Stack()

    # specify can just be a single line
    specify stack.Count.Must == 0

    # Or it can a block of statements
    specify "Stack must accept an item and count is then one":
        stack.Push(42)
        stack.Count.Must == 1

    # You can specify that a block of code must
    # throw an exception when executed.
    specify { stack.Pop() }.Must.Throw(StackUnderflowException)

    # optional teardown is called after each specify
    # perform clean up actions here.
    teardown:
        stack.Dispose()
```

Binsor is a configuration DSL (Dependency Injection)

```
import Rhino.Commons
```

```
Component("default.repository",  
          IRepository, NHRepository)
```

versus

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <components>  
    <component id="default.repository"  
      service ="Rhino.Commons.IRepository`1, Rhino.Commons"  
      type="Rhino.Commons.NHRepository`1, Rhino.Commons"/>  
  </components>  
</configuration>
```

MockDSL

(Testing with Mocks)

```
[Test]
def Get_data_objects_for_nonexistent_company_throws():
  with_mocks:
    database = mocks.CreateMock[of IDatabase]()
    userProvider = StubUserProvider("andrew", "bad corp")
    userProvider.MakeCurrent()

    execute:
      uds = UserDataService(database, userProvider)
      expect_throw FaultException[of GetDataObjectsFault]:
        uds.GetDataObjects()
      assert thrown_exception.Detail.Type.Equals(
        GetDataObjectsFaultType.InvalidCompany)

    assuming:
      database.GetUserID("andrew", "bad corp")
      returned 1

    assuming:
      database.GetCompanyID("bad corp")
      # returning 0 from database implies nonexistent company.
      returned 0
```

Resources

<http://boo.codehaus.org/>

<http://boo.codehaus.org/BooManifesto.pdf>

[googlegroups://boolang](http://googlegroups.com/boolang)

<irc://irc.codehaus.org/boo>

The Future

More Type Inferencing
(specially for generic constructs)

.NET 3.5 (consolidate extension methods)

Meta methods

DLR integration (duck typing)

Split Language Infrastructure and Boo Language

Extensible Parsing?

boojay?

Questions?

THANKS!