

Dynamic: don't be afraid

Hadi Hariri

Developer & Technical
Evangelist JetBrains

<http://hadihariri.com>

<http://twitter.com/hhariri>



32MINUTOS.NET
...el podcast sobre desarrollo que dura 30 minutos.

Agenda

A tour around Dynamics. Why,
What, How, When, If...

There is a world outside static typing

A Dynamic world

- Types defined at runtime
- No Compiler (Usually)
- Late Binding
- Interpreted (Not always)

And it's not Evil



facebook®



shopify
e-commerce made easy



On the other hand...we live in a
Static World

A Static World

- Types can be implicit or explicit (var)
- Compiler Safety
- Early Binding

Best of both worlds?

DLR & C# 4 lets us taste the other side



GET YOUR STINKING
DYNAMIC TYPES OFF
OF MY STATIC
LANGUAGE





So what exactly do we get?

IronRuby

IronPython

C#

VB.NET

DLR

Bindings

.NET

Ruby

Python

Office

Hosting API

Debugging API

Interop
Binders

Dynamic
Objects

Call-Site
Caching

Expressions

Expression
Compiler /
Interpreter

IL Code Generator

C# added “Dynamic”

Dynamic Support

- **dynamic** keyword
- Classes/Binders and interfaces to work with dynamic types

Why the need for dynamic?

```
var assembly = Assembly.LoadFrom(@"..\..\..\..\ExternalTypes.dll");  
var typeInfo = assembly.GetType("ExternalTypes.CustomerService");  
var methodInfo = typeInfo.GetMethod("MakeCustomerPreferred");  
var customerService = Activator.CreateInstance(typeInfo);  
methodInfo.Invoke(customerService, null);
```



```
var customerService = new CustomerService();  
customerService.MakeCustomerPreferred();
```

```
object calculatorType = ruby.Runtime.Globals.GetVariable("Calculator");  
object calculator = ruby.Operations.CreateInstance(calculatorType);  
object sum = ruby.Operations.InvokeMember(calculator, "add", 20, 30);  
Console.WriteLine(String.Format("The sum is {0}", sum));  
Console.ReadLine();
```



Interoperability

- Talking to COM
 - Need a type-library beforehand
 - Use Method Invocation

DEMO

TALKING COM

Interoperability with other languages

- IronPython
 - Interpreted
 - Can be compiled
- IronRuby
 - Interpreted
- Your own language

DEMO

TALKING RUBY

Saving on unnecessary code

The case of the DTO

Creating Dynamic Objects in C#

Options

- ExpandObject
- DynamicObject
- IDynamicMetaObjectProvider

DEMO

MVC - DTO'S, VIEWBAGS AND EXPANDOS

Expando Object

- Built-in Dynamic Object. Works out of the box
- Benefits over Dictionary
 - More Fluent
 - Support for Methods
 - Supports Hierarchies
 - Implements INotifyPropertyChanged
- Limitations
 - Index Access

DynamicObject

- Moving Beyond an Expando
- Built-in class which implement **IDynamicMetaObjectProvider**
- Allows easy creation of Dynamic types

DEMO

DYNAMIC OBJECT

Saving on undetermined MetaProgramming

Aspects of MetaProgramming

- Adding / Removing Methods
- Creating Instance Methods
- Creating Static / Class Methods
- Query Classes

Defining Fluent API's and DSL's

DEMO

METHOD MISSING – SIMPLE DATA

Consuming the unknown or
ever-changing

DEMO

CONSUMING JSON / EASYHTTP / EASYCOUCHDB

A bit on how it all works...

The backbone of dynamic support

DLR

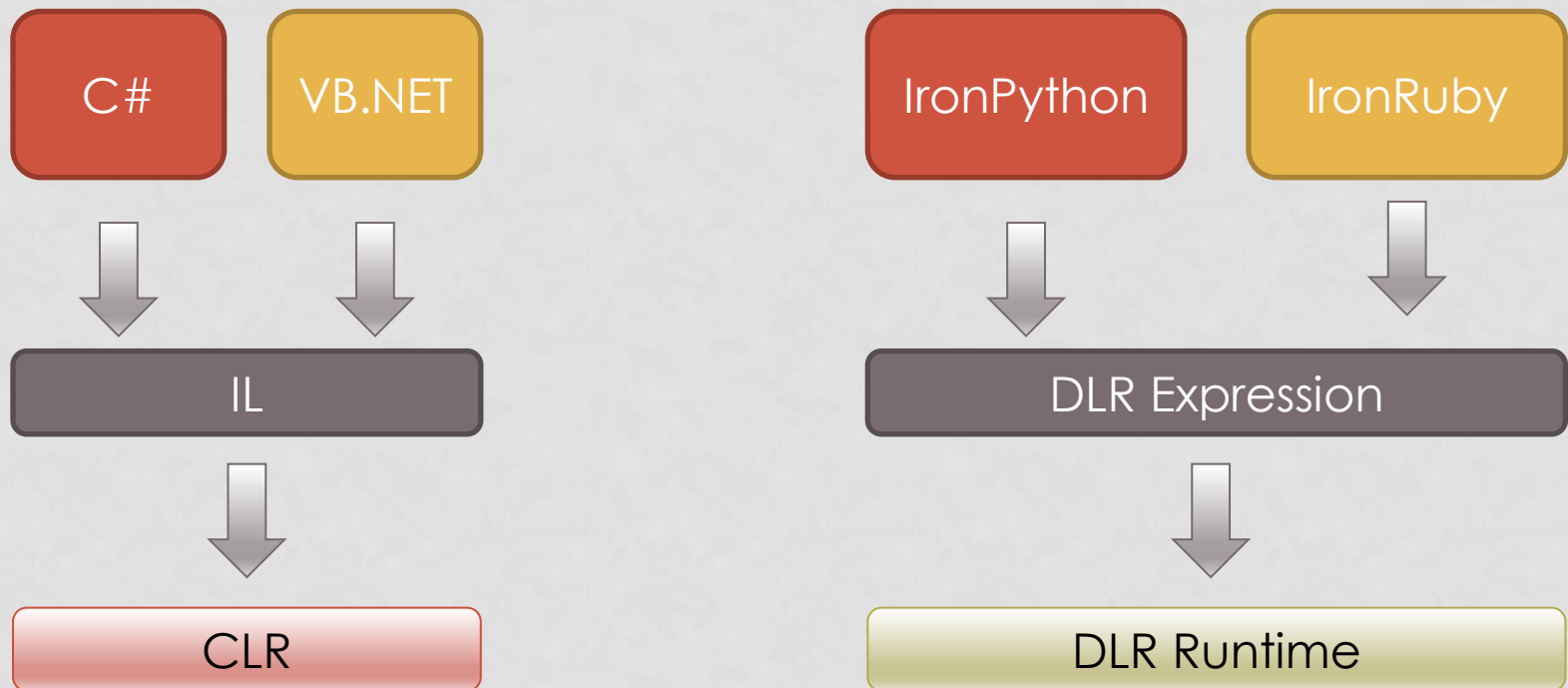
```
graph TD; DLR[DLR] --> Semantics[Language Semantics via DLR EXpression]; DLR --> Logic[Define Late Binding Logic];
```

**Language Semantics
via DLR EXpression**

**Define Late Binding
Logic**

DLR Expression

- Superset of Linq.Expression
- Common to multiple Languages
- DLR Expression is to DLR Languages what IL is to CLR languages



DLR Expression

Code As Data \leftrightarrow Data As Code

```
int x = 2 + 3;
```

```
Expression binaryExpression = Expression.Add(  
    Expression.Constant(2),  
    Expression.Constant(3)  
);
```

```
Func<int> call = Expression.Lambda<Func<int>>(binaryExpression).Compile();
```

DLR Expression

- Only consists of Expressions, no Statements
- Examples
 - BinaryExpression
 - IfThenElse
 - Switch
 - Extend with your own....

```
SwitchExpression switchExpression = Expression.Switch(  
  
    Expression.Constant(5), // switch discriminator  
    new SwitchCase[]  
    {  
        Expression.SwitchCase(// Expression for switch case ),  
        Expression.SwitchCase(// Expression for switch case ),  
  
    }  
);
```

A DLR Expression

Member	Description
NodeType	Type of Node (BinaryExpression, IfThenElse)
Type	The type it represents (String, Int32)
CanReduce	Boolean indicating if reducable
Accept(vistor: Expression Vistor): Expression	For visiting the expression in Visitor Pattern
Reduce(): Expression	For reducing to existing expression from custom

Up to now, we know the types

Early Binding

- We know the types
- We know what methods to call
- This is embedded in the IL: Call ToLower of String

```
string message = "Hello";
```

```
Console.WriteLine(message.ToLower());
```

What about when we don't?

```
dynamic x = 2 + 3;
```

Late Binding

- We only know the types at runtime
- We have to figure out how to call those methods at runtime
- It's not embedded in the "IL"
- It's potentially slower

Late Binding

```
static void Main(string[] args)
{
    dynamic value = "Hello";
    Console.WriteLine(value.ToString());
}
```

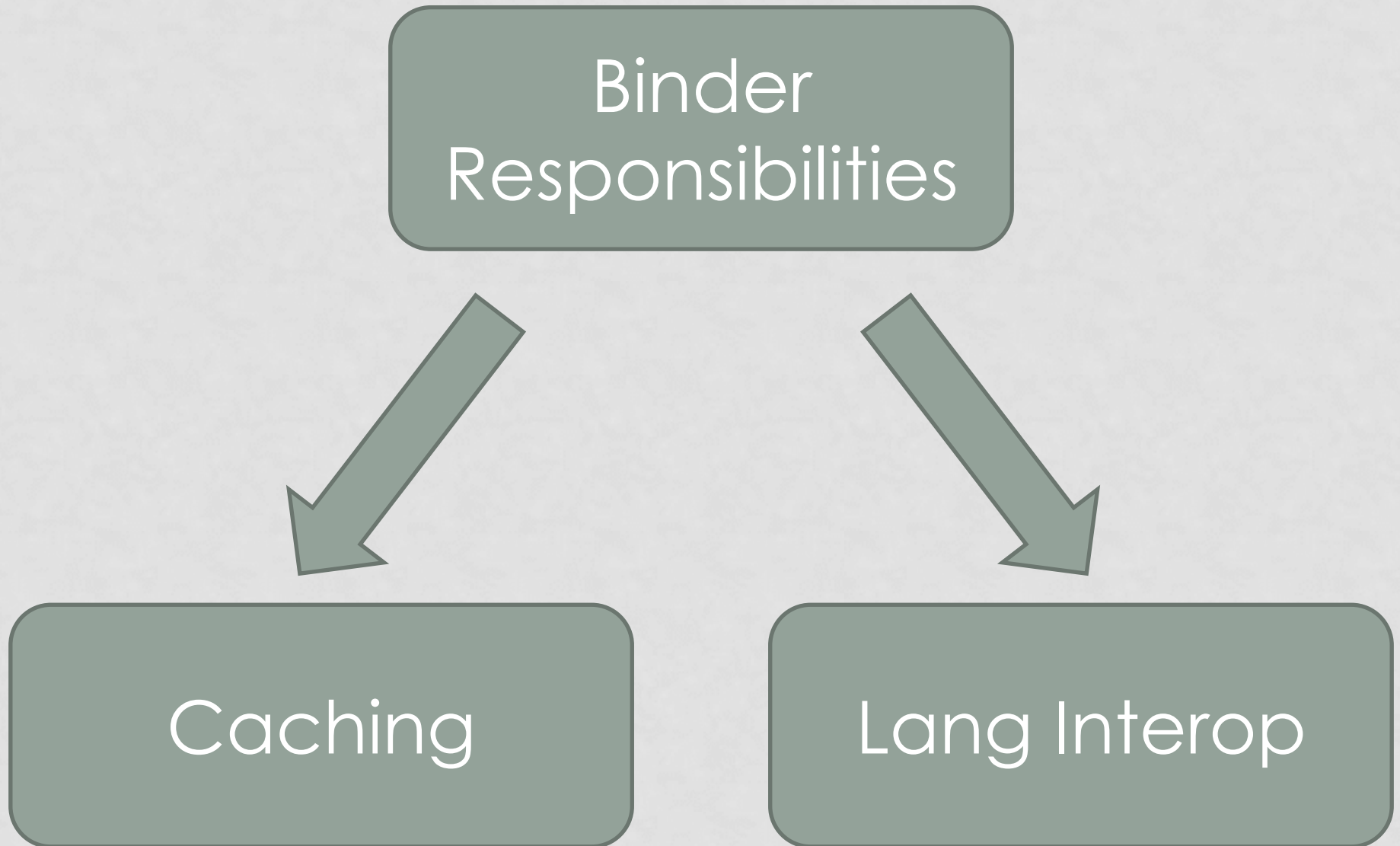


```
private static void Main(string[] args)
{
    object value = "Hello";
    if (<Main>o_SiteContainer0.<>p_Site1 == null)
    {
        <Main>o_SiteContainer0.<>p_Site1 = CallSite<Action<CallSite, Type, object>>.Create(Binder.InvokeMember(CSharpBinder
    }
    if (<Main>o_SiteContainer0.<>p_Site2 == null)
    {
        <Main>o_SiteContainer0.<>p_Site2 = CallSite<Func<CallSite, object, object>>.Create(Binder.InvokeMember(CSharpBinder
    }
    <Main>o_SiteContainer0.<>p_Site1.Target(<Main>o_SiteContainer0.<>p_Site1, typeof(Console), <Main>o_SiteContainer0
}
```

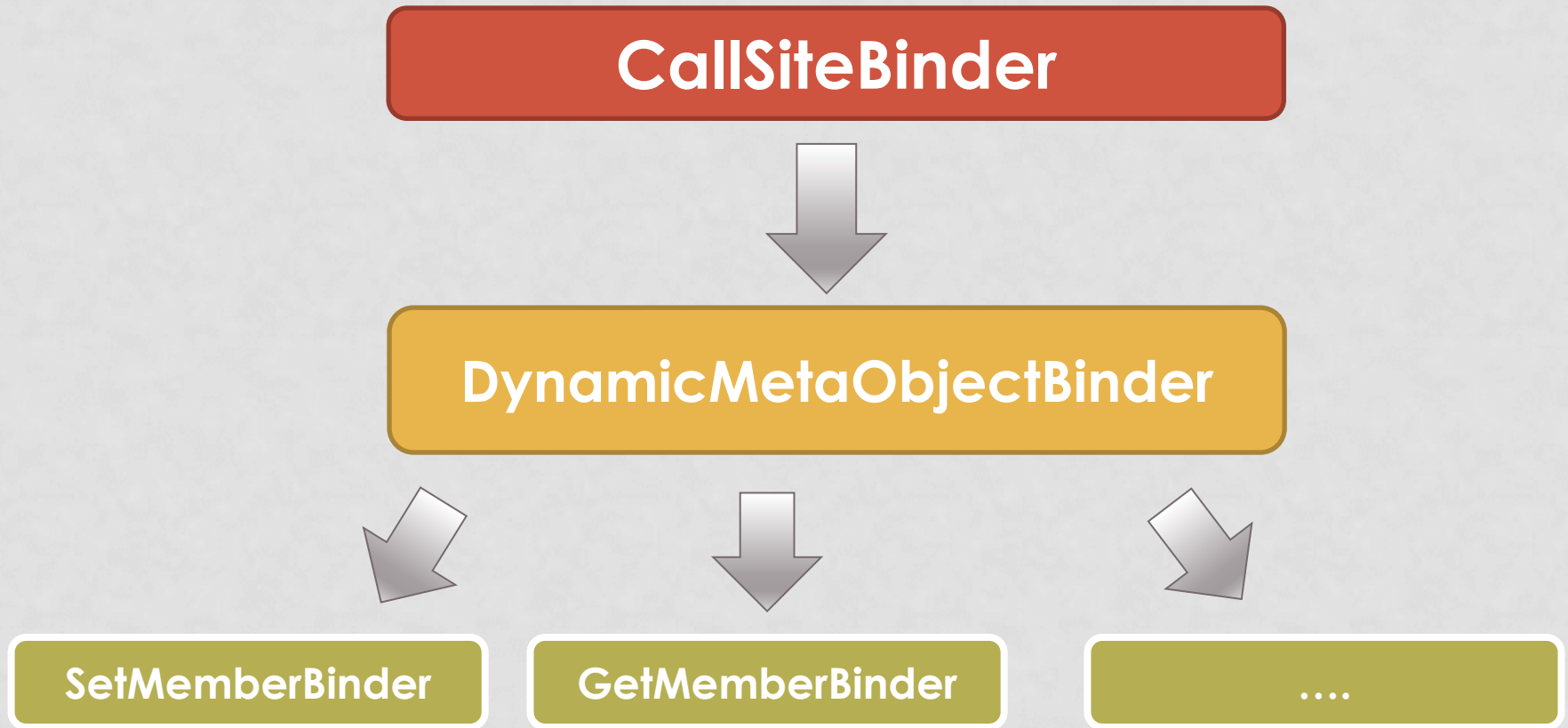
What are Call Sites

- Position in the code where a method is called
- In Static Languages (Early Binding)
 - This is done at compile time
- In Dynamic Languages (Late Binding)
 - This is done at run time
 - By Binders

Binder Responsibilities



Binders in DLR



DLR CallSites

- A Call Site is responsible for Caching and Binding
- **DynamicMetaObjectProvider** responsible for Language Interop
- Binding is defined in terms of **Rules**
 - **Restrictions:** Conditions under which the binding result is valid
 - **Result:** Actual result of the binding
- Rules allow Caching (3 Levels of Caching)

Late Binding with Binders

- Binders defined by Target Language
 - IronPython
 - IronRuby
 - C#
- Late Binding performed by the target language of Binder
- 12 Actions that can be performed defined by Common Type System

Binding with Dynamic Objects

- Use Dynamic Objects for Binding as opposed to Binders
- Dynamic Object responsible for Binding all operations (12 defined by CTS)
- `IDynamicMetaObjectProvider`
 - Meta Object that performs binding
 - Allows decoupling from class
 - Returns `DynamicObject`

DynamicMetaObject

- Two Responsibilities
 - Base Class for all classes that implement late binding
 - Carry result of Late Binding

Coming back to the DynamicObject

- Wrapper around 12 methods of **DynamicMetaObject**
- Implemented internally via **MetaDynamic** class
- Whereas **DynamicMetaObject** deals with DLR Expression, **DynamicObject** deals with C#
- **DynamicObject** falls back to language binder if it cannot bind

summing up...

The Disadvantages

- There is no compile type-checking
- Potentially slower (even with caching)
- There is no Intellisense (Partially incorrect)

What about Unit Testing?

Reasons to not use dynamic

- It's cool!
- It's the new thing!
- I feel like writing unit tests to make up for compiler

Reasons to not not use dynamic

- There's no compiler
- There's no intellisense
- You shouldn't mix dynamic and static languages

Reasons to use Dynamic

- Interoperability
 - COM
 - Consuming Dynamic Languages
 - Ruby
 - JavaScript
- Fluent API's and DSL
- Consuming the *unknown*
 - Dynamic Structures
- Avoiding unnecessary “class explosion”

Thank you



Hadi Hariri

Technology Evangelist

<http://hadihariri.com>
<http://twitter.com/hhariri>
e-mail: hadi@jetbrains.com

