Dynamic Programming in C#

Bart J.F. De Smet
bartde@outlook.com

pluralsight
hardcore dev and IT training

Why Dynamic Typing?

- **Static at heart**
  - Design decision for CLR and C# in the v1.0 days (late 90s)
  - Benefits of static typing:
    - Great compile-time error checking
    - Performance: exact object layouts etc.
  - Every memory location in managed code has a type
    - Verifiability of code

- **Dynamic typing has its use**
  - Lots of data is weakly typed in nature
    - XML, JSON
  - Popularity boost of dynamic languages
    - Python, Ruby, JavaScript, etc.
  - Perceived developer productivity
    - No need to compile code, schematize data, etc.
    - Less barriers to experimentation

C#'s Dynamic Type

- **Static type for dynamic dispatch**
  - Compiles away to System.Object
  - Instructs compiler to emit dynamic call sites

```
string  s = "Foo";
string  u = s.ToUpper();
```

- Strongly typed
- Statically typed

```
object  s = "Foo";
string  u = s.ToUpper();
```

- Weakly typed
- Statically typed

```
dynamic s = "Foo";
string  u = s.ToUpper();
```

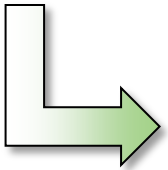
- Weakly typed
- Dynamically typed

No IntelliSense here!

The Role of Language Binders

- **Compile-time versus runtime**
  - Statically typed code
    - Compiler performs type checking, overload resolution, etc.
    - Compiler emits precise code to execute the logic
  - Dynamically typed code
    - Microsoft.CSharp.dll performs similar tasks at runtime
    - Dynamic dispatch sites capture language-specific information
    - The DLR is used to generate code at runtime

```
dynamic s = "Foo";
string  u = s.ToUpper();
```



```
object s = "Foo";
CallSite conv = siteContainer.site1;
CallSite call = siteContainer.site2;
string u = conv.Target(conv,
   call.Target(call, s));
```

DLR call sites populated with C# binders

Invocation of dynamic operations

The Role of Language Binders

- **Binders for various operations**
  - Method, indexer, property, function invocation
  - Binary, unary, conversion operations
  - Object creation (not supported in C#)

```
object s = "Foo";
if (c.site1 == null) {
  c.site1 = CallSite<Func<CallSite, object, string>>.Create(
    Binder.Convert(CSharpBinderFlags.None, typeof(string), …)
  );
}
if (c.site2 == null) {
  c.site2 = CallSite<Func<CallSite, object, object>>.Create(
    Binder.InvokeMember(CSharpBinderFlags.None, "ToUpper", …);
  );
}
string u = c.site1.Target(c.site1, c.site2.Target(c.site2, s));
```

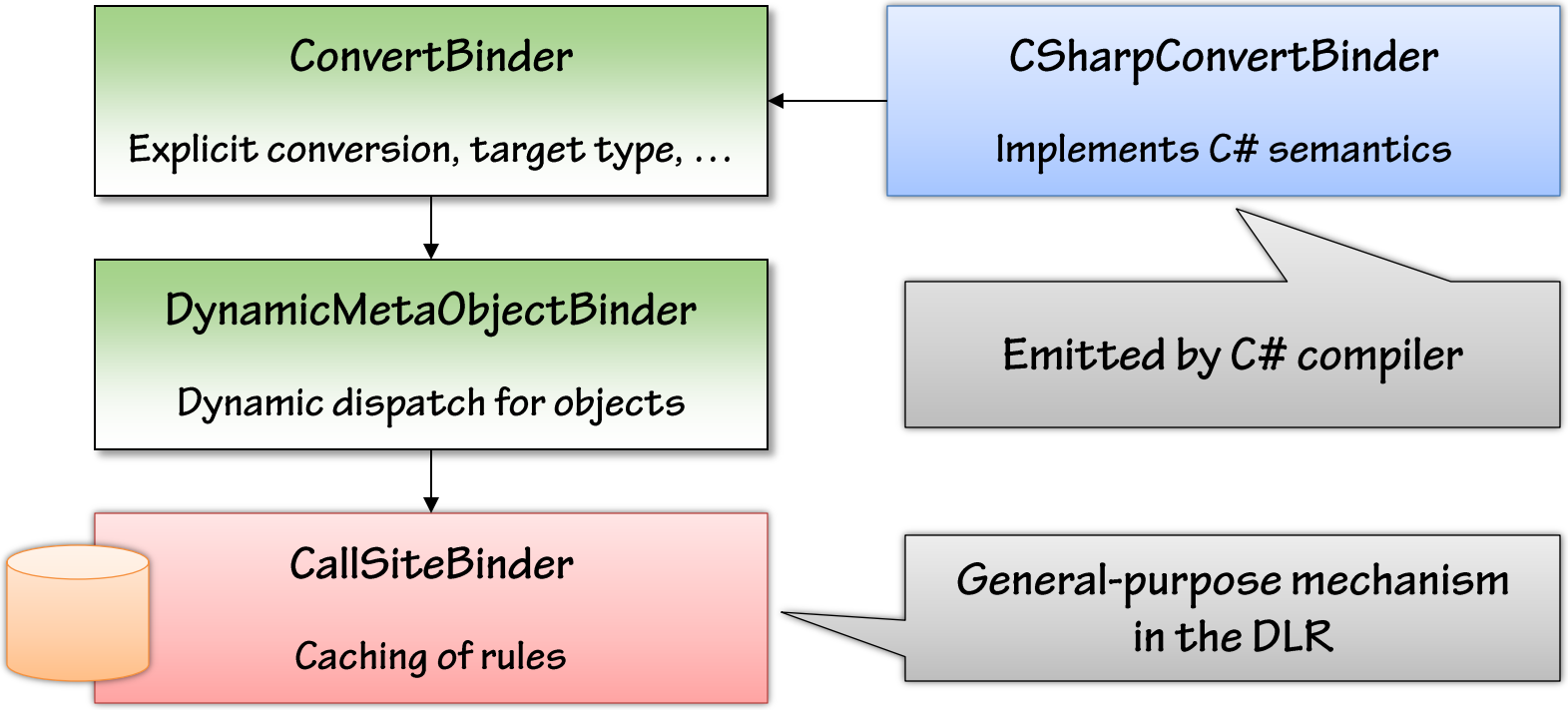
Site can update itself

C# binder

Delegate property

Internals of Language Binders

- **Binders provided by languages**
  - Microsoft.CSharp.RuntimeBinder.Binder class
  - Methods return CallSiteBinder objects
    - Derived from base classes in System.Dynamic
    - Override Bind methods to return logic to perform operations



Call Site Caches

- **Site container generated by compiler**
  - One site per dynamic operation
  - Lazily instantiated at runtime
- **Polymorphic inline caching**
  - Caches operations returned from the binder
  - Dispatch based on types of operands

```
dynamic x = Add(1, 2);
dynamic y = Add("Hello, ", "C#");
dynamic z = Add(DateTime.Now, TimeSpan.Zero);
```

```
dynamic Add(dynamic a, dynamic b) {
  return a + b;
}
```

Binary operation call site

Call Site Caches

```
dynamic x = Add(1, 2);
dynamic y = Add("Hello, ", "C#");
dynamic z = Add(DateTime.Now, TimeSpan.Zero);
```

```
[return:Dynamic]object Add([Dynamic]object a, [Dynamic]object b) {
  if (c.site1 == null)
    c.site1 = CallSite<Func<CallSite, object, object, object>>.Create(
      Binder.BinaryOperation(…, ExpressionType.Add, …));
  return c.site1.Target(c.site1, a, b);
}
```

Code returned by C# binder

```
Target = (CallSite s, object a, object b) => {
  if (a is int && b is int)
    return (int)a + (int)b;
  else if (a is string && b is string)
    return string.Concat((string)a, (string)b);
  else
    s.Update(a, b); // calls the C# binder
};
```

Dynamic Quirks

- **Erasure of "dynamic" type**
  - No special runtime type, represented as System.Object
  - [DynamicAttribute] metadata used by compilers
  - Overload resolution restrictions:

```
void Foo(dynamic d) { … }
void Foo(object o) { … }
```

Compile error because of same signature

- **Cannot implement generic interfaces using dynamic**
  - Pretty much the same as implementing it with System.Object
  - No good place for compiler to put [Dynamic]
  - Conflicts with the System.Object-based implementation

```
class C : IEnumerable<dynamic> {
  public IEnumerator<dynamic> GetEnumerator() { … }
}
```

Would erase to object with no place for [Dynamic]

Summary

- **Dynamic typing**
  - Escape valve for dynamic language interop
  - Enables accessing weakly typed data structures
- **Performance of dynamically typed code**
  - Below statically typed code
  - Optimization with call site caching
  - Leverages DLR and expression trees to JIT compile
- **Moving parts**
  - Language binders provide semantics
  - Call sites provide gateway to DLR
  - Implementation provides caching etc.