CIL Metadata as Scala source, and other projects for the Scala.NET ecosystem

© Miguel Garcia, LAMP,
École Polytechnique Fédérale de Lausanne (EPFL)
`http://lamp.epfl.ch/~magarcia`

May 18th, 2010

Contents

Abstract

Besides adding support for Generics, other extensions and additions to the Scala.NET compiler would also increase its value for the community. These notes describe some of these projects, in a format suitable for compiler hackers.

1 Interpreting C# extension methods as defining Scala views

In terms of surface syntax, a C# *extension method* can be invoked as any other method, although its declaration is not part of the receiver's type (similar to Scala *implicits*, extension methods are resolved based on the static type of the receiver). In terms of its metadata encoding, an extension method `M` is declared in a static class, where the first argument of `M` encodes a `[System.Runtime.CompilerServices.Extension]` attribute on its type. This marker attribute instructs .NET compilers to handle that method argument as future receiver of `M`. The C# surface syntax prefixes this argument with the `this` keyword.

In this subsection, we explore how to make a C# extension method behave as a Scala implicit within the compiler boundaries i.e., from the time its metadata encoding is parsed till an assembly is emitted. The emitted code has to follow the original calling convention (i.e., a static method invocation is to be emitted).

1.1 Metadata-level representation of extension methods

As reported at The metadata of extension methods fulfills the following conditions:[1],

```
static IEnumerable<MethodInfo> GetExtensionMethods(Assembly assembly, Type extendedType)
{
    var query = from type in assembly.GetTypes()
                where type.IsSealed && !type.IsGenericType && !type.IsNested
                from method in type.GetMethods(BindingFlags.Static
                    | BindingFlags.Public | BindingFlags.NonPublic)
                where method.IsDefined(typeof(ExtensionAttribute), false)
                where method.GetParameters()[0].ParameterType == extendedType
                select method;
    return query;
}
```

BTW, trying to "manually encode" the metadata above will not work, as the example in p. 14 of LINQ Esstentials[2] shows:

```
[System.Runtime.CompilerServices.Extension]
public static int CompareTo(string origStr, string compareStr)
{   . . .
/* Triggers the error message
   Do not use System.Runtime.CompilerServices.ExtensionAttribute. Use the this keyword instead.
*/
}
```

As an illustration of the metadata encoding, Figure 1 shows the metadata of SQO extension methods obtained from disassembling `System.Core.dll`. For example, one of the overloadings of `Average` is:

```
.method public hidebysig static float64 Average<TSource>( /*- static method in static class */
     class [mscorlib]System.Collections.Generic.IEnumerable`1<!!TSource> source, /*- receiver */
     class System.Func`2<!!TSource,float64> selector
) cil managed
{
  .custom instance void /*- marker attribute */
     System.Runtime.CompilerServices.ExtensionAttribute::.ctor() = ( 01 00 00 00 )
  // Code size       13 (0xd)
  .maxstack 8
  IL_0000: ldarg.0
  IL_0001: ldarg.1
  IL_0002: call       class [mscorlib]System.Collections.Generic.IEnumerable`1<!!1>
                      System.Linq.Enumerable::Select<!!0,float64>(
                              class [mscorlib]System.Collections.Generic.IEnumerable`1<!!0>,
                              class System.Func`2<!!0,!!1>
```

[1]http://stackoverflow.com/questions/299515/c-reflection-to-identify-extension-methods
[2]http://www.mhprofessional.com/downloads/products/0071597832/0071597832_chap01.pdf

```
                    )
  IL_0007: call      float64 System.Linq.Enumerable::Average(
                            class [mscorlib]System.Collections.Generic.IEnumerable`1<float64>
                    )
  IL_000c: ret
} // end of method Enumerable::Average
```

1.2 Declaring implitics to achieve equivalent semantics

TODO Take into account the idioms proposed at `http://scala-programming-language.1934581.n4.nabble.com/scala-Class-overloading-vs-implicit-conversion-td1997797.html`

Refresher: From `Predef.scala`:

```
// views --------------------------------------------------------------
implicit def intWrapper(x: Int)   = new runtime.RichInt(x)
```

Given a C# extension method:

```
class C {
  public static retType M(this TargetType target, arg1Type arg1, ... ) { ... }
}
```

Symbols should be entered as if the following two classes had been defined:

```
class C {
  implicit def C2RichC(target : TargetType) = new RichC(target)
}

class RichC(target : TargetType) { // target is now in scope
  def M(arg1 : arg1Type, ... ) {
    // method body copied verbatim from original C# definition
  }
}
```

1.3 Emitting call sites for the original encoding

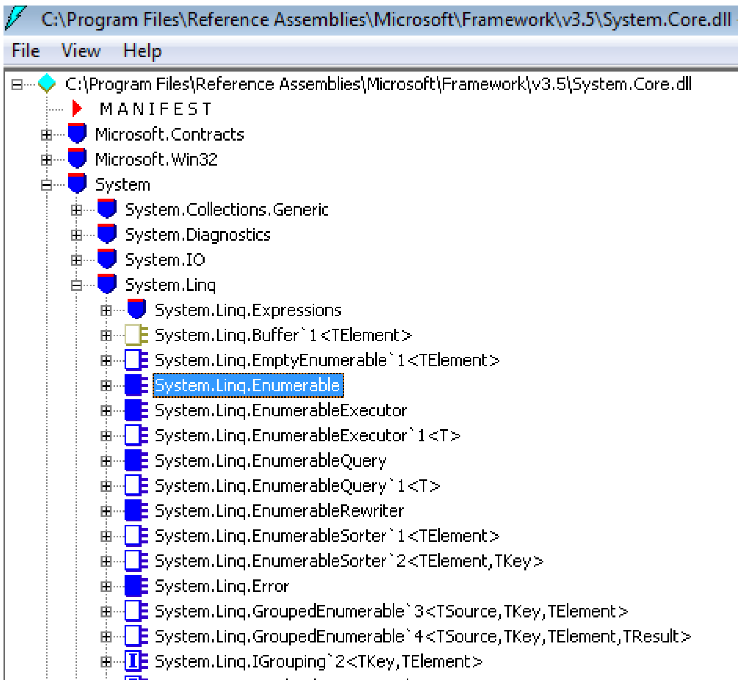
TODO During (or right after?) `GenICode`, trees where `RichC` methods are invoked have to be desugared into invocations on static methods of `C`

2 Using JVM-based Scala IDEs to develop Scala.NET apps

Right now, the development of .NET applications lacks IDE support. Before a VisualStudio plugin for Scala becomes available, an interim solution is explored in this section, relying on the JVM-based IDEs for Scala in use today.

When developing for .NET, the compiler has to *discover*[3] types in a bunch of assemblies. Is there a way to represent those .NET types in a format consumable by current IDEs supporting Scala? If so, then the IDE in question could be used

[3] *Decoding external types on JVM and CLR*`http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/TypeDecoding.pdf`



```
C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.5\System.Core.dll
File   View   Help
⊟ C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.5\System.Core.dll
    MANIFEST
  ⊞ Microsoft.Contracts
  ⊞ Microsoft.Win32
  ⊟ System
      ⊞ System.Collections.Generic
      ⊞ System.Diagnostics
      ⊞ System.IO
      ⊟ System.Linq
          ⊞ System.Linq.Expressions
          ⊞ System.Linq.Buffer`1<TElement>
          ⊞ System.Linq.EmptyEnumerable`1<TElement>
          ⊞ System.Linq.Enumerable
          ⊞ System.Linq.EnumerableExecutor
          ⊞ System.Linq.EnumerableExecutor`1<T>
          ⊞ System.Linq.EnumerableQuery
          ⊞ System.Linq.EnumerableQuery`1<T>
          ⊞ System.Linq.EnumerableRewriter
          ⊞ System.Linq.EnumerableSorter`1<TElement>
          ⊞ System.Linq.EnumerableSorter`2<TElement,TKey>
          ⊞ System.Linq.Error
          ⊞ System.Linq.GroupedEnumerable`3<TSource,TKey,TElement>
          ⊞ System.Linq.GroupedEnumerable`4<TSource,TKey,TElement,TResult>
          ⊞ System.Linq.IGrouping`2<TKey,TElement>
```

Figure 1: Metadata of SQO extension methods

to develop Scala.Net apps (with all navigation and editing goodies running on the JVM-based IDE, but offloading the compilation to the Scala.Net compiler).

The premiere "consumable representations of external types" are Scala sources themselves, containing type definitions with method bodies that just return default values, i.e., *stubs*. These stubs are generated from the referenced assemblies in a manner similar to how *.NET Reflector* does its thing. If C#'s surface syntax can express those types, even more so Scala's: all capabilities of the CLR type system are supported by Scala's type system (and then some). Granted, a convention is needed to represent in Scala sources the "calling convention" of events and properties (same as under the hood of the C# compiler[4]) but this does not compromise type safety. Additionally, the generation of stubs need be performed only upon recompilation of the referenced assemblies, i.e. not that often.

Assuming this option has been made to work, one issue remains: the Scala IDE should be instructed not to automatically import the types in `rt.jar` (in particular `java.lang.*`) but instead those in the `System.bla.bla` stubs available in the Scala sources obtained by "CIL Metadata as Scala sources".

Although IDE-based editing and navigation would be convenient, no IDE-based support would be available for cross-compiling nor debugging. For debugging, one would have to take the `pdb` file (which includes source locations, generated by Scala.Net) and the `.scala` files to an IDE for Mono or .NET to debug there.

[4]`http://www.mhprofessional.com/downloads/products/0071597832/0071597832_chap01.pdf`

2.1 Obtaining `.scala` stubs from `.dlls` and `.exes`

BTW, a similar functionality is called "Metadata as Source"[5] in C#

> *Metadata as source enables you to view metadata that appears as C# source code in a read-only buffer. This enables a view of the declarations of the types and members (with no implementations). You can view metadata as source by running the Go To Definition command for types or members whose source code is not available from your project or solution.*

TODO What API to call to obtain the C# snippets in question?

TODO Using `compiler.msil.TypeParser` to parse the metadata

- TODO: add URL to write-up *Parsing metadata for Generics, Round 2*
- *Parsing type-metadata about Generics on .NET*[6]

2.2 Conventions followed by the C# compiler to encode syntax sugar into metadata

In addition to the discussions in Sec. 1 (Extensions Methods), the following resources are useful:

- `http://connect.microsoft.com/VisualStudio/feedback/details/293271/ c-compiler-does-not-emit-specialname-metadata-for-getter-setter-methods-used-to-wrap`
- `http://www.mhprofessional.com/downloads/products/0071597832/0071597832_ chap01.pdf`

2.3 Printing the typing environment

Listing 1 shows a utility (a compiler patch, actually) to print (after the typing phase) not the compilation units, but the contents of the assemblies given by `-Xassem-extdirs` (this utility can be extended to print them out as *stubs*).

Listing 1 is reproduced from `http://permalink.gmane.org/gmane.comp. lang.scala.internals/3164`.

3 Refinement types can handle C# anonymous types (LINQ scenario)

In Scala, it is possible to declare structural types using type refinements (§3.2.7), including a shorthand form for creating values of structural types. For instance,

```
new { def getName() = "aaron" }
```

is a shorthand for

```
new AnyRef { def getName() = "aaron" }
```

[5]`http://msdn.microsoft.com/en-us/library/ms236403(VS.80).aspx`

[6]`http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/ScalaNetRound2.pdf`

Listing 1: Printing the typing environment

```
class SymbolPrinter(out: PrintWriter) extends treePrinters.Printer(out) {

  def printClass(sym: Symbol) {
    print(sym.toString)
    if (!sym.info.typeParams.isEmpty)
      printRow[Symbol](sym.info.typeParams, printTypeParam, "[", ", ", "]")
    print(" extends ")
    printRow[Type](sym.info.parents, (t => print(t.toString)), "", " with ", "")
    printColumn(sym.info.decls.toList, printMember, "{", "", "}\n")
    flush
  }

  def printTypeParam(sym: Symbol) {
    print(sym.name); print(sym.info.bounds.toString)
  }

  def printMember(sym: Symbol) {
    if (sym.isClass) printClass(sym)
    else print(sym.defString)
  }
}
```

Regarding *anonymous types*[7] and the role they play in LINQ queries [1], a Scala formulation that preserves their semantics yet does not require AST rewriting is . . . refinement types, of course!

```
val lst = List("a", "b", "c")
val res = lst map ( arg => new { val bb = arg.length } ) map ( arg2 => arg2.bb + 1 )
```

In the example above, "`new { val bb = arg.length }`" is an occurrence of *refinement types* (§3.2.7 in the Scala 2.8 spec) and captures the semantics of the C# version: an anonymous type inherits implicitly from `System.Object`.

TODO reformulate the snippet above in C#, compile, decompile with ILAsm, and see what the base class of the (synthetic, compiler-generated) anonymous type is.

Well, in fact, there's more to it than the example above might suggest. But not too much. Another example of anonymous types in C# is:

```
var query = from c in listOfCustomers
          select new {FirstName = c.Name, c.City};
```

Notice the "`c.City`" instead of "`City = c.City`"? We'll, a LINQ expander could take care to expand that shorthand.

Now, do we really want to follow verbatim the C# syntax? The issue at play is whether (a) we let the Scala.NET compiler accept the C# syntax for anonymous types; or (b) keep consistency with Scala refinement types. We choose (b), obviously.

LINQ *Expression Trees* in the context of Scala.NET are discussed in the write-up *Adding LINQ-awareness to Scala.Net* at `http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/ScalaNetLearnsLINQTricks.pdf`.

[7]`http://blogs.msdn.com/wesdyer/archive/2006/12/22/transparent-identifiers.aspx`

4 Emitting CIL for Serialization, Cloning, and Remoting

In general, the .NET framework relies more on attributes than marker interfaces, for example with the class-level `[Serializable]` and the field-level `[NonSerialized]` rather than `implements java.io.Serializable`. As a reminder, serialization, cloning, and remoting appear at the level of IClasses as Scala annotations:

```
for (annot <- c.symbol.annotations) annot match {
  case AnnotationInfo(tp, _, _) if tp.typeSymbol == SerializableAttr =>
    parents = parents ::: List(definitions.SerializableClass.tpe)
  case AnnotationInfo(tp, _, _) if tp.typeSymbol == CloneableAttr =>
    parents = parents ::: List(CloneableClass.tpe)
  case AnnotationInfo(tp, Literal(const) :: _, _) if tp.typeSymbol == SerialVersionUID =>
    serialVUID = Some(const.longValue)
  case AnnotationInfo(tp, _, _) if tp.typeSymbol == RemoteAttr =>
    parents = parents ::: List(RemoteInterface.tpe)
    remoteClass = true
  case _ => ()
}
```

Serialization

- Ins and Outs, `http://msdn.microsoft.com/en-us/library/7ay27kt9(VS.80).aspx`
- Common cases, `http://www.codeguru.com/csharp/csharp/cs_syntax/serialization/article.php/c7201`

Cloning. Quoting from `http://weblogs.asp.net/esanchez/archive/2008/05/18/cloning-objects-in-net.aspx`

> *Perhaps surprisingly the CLR doesn't offer a general cloning method, of course you could use `MemberwiseClone()` but this is a protected method, so it can be invoked only from inside the class of the object being cloned, which makes it difficult to use it in a general method, besides, `MemberWiseClone()` does just a shallow copy and what we really need is a deep copy.*

- Brad Abrams: Do not implement `ICloneable`.
  `http://blogs.msdn.com/brada/archive/2003/04/09/49935.aspx`
- `http://msdn.microsoft.com/en-us/library/system.icloneable.aspx`

Remoting

- From .NET Remoting to the Windows Communication Foundation (WCF), `http://msdn.microsoft.com/en-us/library/aa730857(VS.80).aspx`

TODO What methods and fields are currently being generated in `GenMSIL` for each of the above concerns? Are the respective design patterns complete or is sthg missing?

5 Default values for fields in CLR ("Object initialization semantics")

Quoting from [2]:

> *Default values reside in the* `Constant` *metadata table. Three kinds of metadata items can have a default value assigned and therefore can reference the Constant table: fields, method parameters, and properties. . . . The ILAsm syntax is:*
>
> `<field_def_const> ::= .field <flags> <type> <name> = <const_type> [( <value> )]`
>
> *The value in parentheses is mandatory for all constant types except* `nullref`*. . . . Suppose that we define a member field as follows:* `.field public static int32 ii = int32(12345)` *What will the value of the field be when the class is loaded? Correct answer: 0. Why? Default values specified in the Constant table are not used by the loader to initialize the items to which they are assigned. If you want to initialize a field to its default value, you must explicitly call the respective Reflection method to retrieve the value from metadata and then store this value in the field.*

TODO See §5.1.6 (*5.1.6 Early Definitions*) in the Scala Language Spec.

Early Member Definitions (draft), `http://www.scala-lang.org/sites/default/files/sids/nielsen/Tue,%202009-06-02,%2013:16/early-defs.pdf`

TODO What ICode is generated for early member defs, as e.g.

```
// phaseName = "tailcalls"
object tailCalls extends {
  val global: Global.this.type = Global.this
  val runsAfter = List[String]("uncurry")
  val runsRightAfter = None
} with TailCalls
```

6 Other projects for the Scala.NET ecosystem

- Scalify.NET and joint compilation of Scala and C#, `http://www.sts.tu-harburg.de/people/mi.garcia/ScalaCompilerCorner/MixedSourceScalaCSharp.pdf`

References

[1] Miguel Garcia. Compiler plugins can handle nested languages: AST-level expansion of LINQ queries for Java. In Moira C. Norris and Michael Grossniklaus, editors, *Proc. of ICOODB 2009*, pages 41–58, July 2009. `http://www.sts.tu-harburg.de/people/mi.garcia/pubs/2009/icoodb/compplugin.pdf`.

[2] Serge Lidin. *Expert .NET 2.0 IL Assembler*. Apress, Berkely, CA, USA, 2006.