**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e de Computadores

Semestre de Inverno 2014/2015



Trabalho Final  
Ambientes Virtuais de Execução

**Trabalho elaborado por:**

Pedro Duarte Nº36832

Miguel Serejo Nº36864

Ruben Santos Nº36925

Fase 1

Nesta primeira fase foram implementados dois métodos como requerido no enunciado, um que cria um *proxy*, que contem todos os métodos virtuais e públicos, com um determinado comportamento especificado por uma implementação da interface *IInvocationHandle* (handler), para uma instância de uma determinada instância de uma classe (*oBase*).

public static T MakeProxy<T>(T oBase, IInvocationHandler handler) where T : class

O Segundo método devolve uma implementação de uma interface com o comportamento dos seus métodos também definidos por uma implementação da interface *IInvocationHandle* (*handler*). Este segundo método apenas chama o primeiro com parâmetro *oBase* a *null*.

public static T MakeProxy<T>(IInvocationHandler handler) where T : class

O primeiro método tem em consideração se o parâmetro oBase é *null*, fazendo uso de uma variável booleana para indicar se é para criar um proxy a partir de uma interface ou não. Se for de interface construtor do *proxy* criado ira ter só o parâmetro do tipo *IInvocationHandler*, ao invés deste e mais o do tipo de *Obase*. Também serão criados os campos do *proxy* dependendo se é para criar de interface ou não, caso seja *de* interfacesó terá o campo IInvocationHandler.

O *proxy* retornado foi implementado usando a biblioteca *Reflection*.Emit.

É começado por ser criado o construtor, que simplesmente afeta os campos do *proxy* *IInvocationHanle* e o com a instancia oBase, dependendo se esta é *null* ou não com foi explicado anteriormente. O seguinte excerto de código mostra a implementação do constructor usando Reflection.Emit:

ConstructorBuilder cb = tb.DefineConstructor(MethodAttributes.Public, CallingConventions.Standard, constructorParameters);

ILGenerator cbIL = cb.GetILGenerator();

cbIL.Emit(OpCodes.Ldarg\_0);

if (!isInterface)

cbIL.Emit(OpCodes.Call, oBase.GetType().GetConstructor(Type.EmptyTypes));

else

cbIL.Emit(OpCodes.Call, typeof(object).GetConstructor(Type.EmptyTypes));

cbIL.Emit(OpCodes.Ldarg\_0);

cbIL.Emit(OpCodes.Ldarg\_1);

if (!isInterface)

{

cbIL.Emit(OpCodes.Stfld, fReal);

cbIL.Emit(OpCodes.Ldarg\_0);

cbIL.Emit(OpCodes.Ldarg\_2);

}

cbIL.Emit(OpCodes.Stfld, fHandler);

cbIL.Emit(OpCodes.Ret);

De seguida serão implementados todos os métodos que o *proxy* conterá, estes métodos criarão uma instância de *CallInfo* e retornaram a chamada ao método *OnCall* de *IInvocationHandle* passando como parâmetro a instância de *CallInfo*.

Por fim é tido o cuidado de especificar que o métodos de proxy reimplementam o método da base sobre o qual o proxy está a ser criado.

O método que cria o proxy por fim cria o tipo proxy, grava para um ficheiro o IlCode definido usando a biblioteca Reflectio.Emit e chama através de reflecção o construtor do proxy criado retornando a instância criada.

Type finishedType = tb.CreateType();

ab.Save(asn.Name + ".dll");

ConstructorInfo typeConstructor = finishedType.GetConstructor(constructorParameters);

object o = typeConstructor.Invoke(constructorArguments);

return o as T;

Se pudéssemos visualizar em C# o proxy criado, a partir da listagem 2 do relatório, ficaria com a seguinte estrutura:

public class Proxy : Foo

{

IInvocationHandler handler;

Foo oBase;

public Proxy(Foo oBase, IInvocationHandler handler)

{

this.oBase = oBase;

this.handler = handler;

}

public override int DoIt(string v)

{

CallInfo ci = new CallInfo(oBase.GetType().GetMethod("DoIt"), oBase, new object[] {v});

return (int)handler.OnCall(ci);

}

}

Fase 2