Ambientes Virtuais de Execução

ISEL – ADEETC – LEIC

Inverno 2014/2015

Trabalho Final, 02 de Fevereiro 2015

Desenvolvimento de API fluente

Grupo 19:

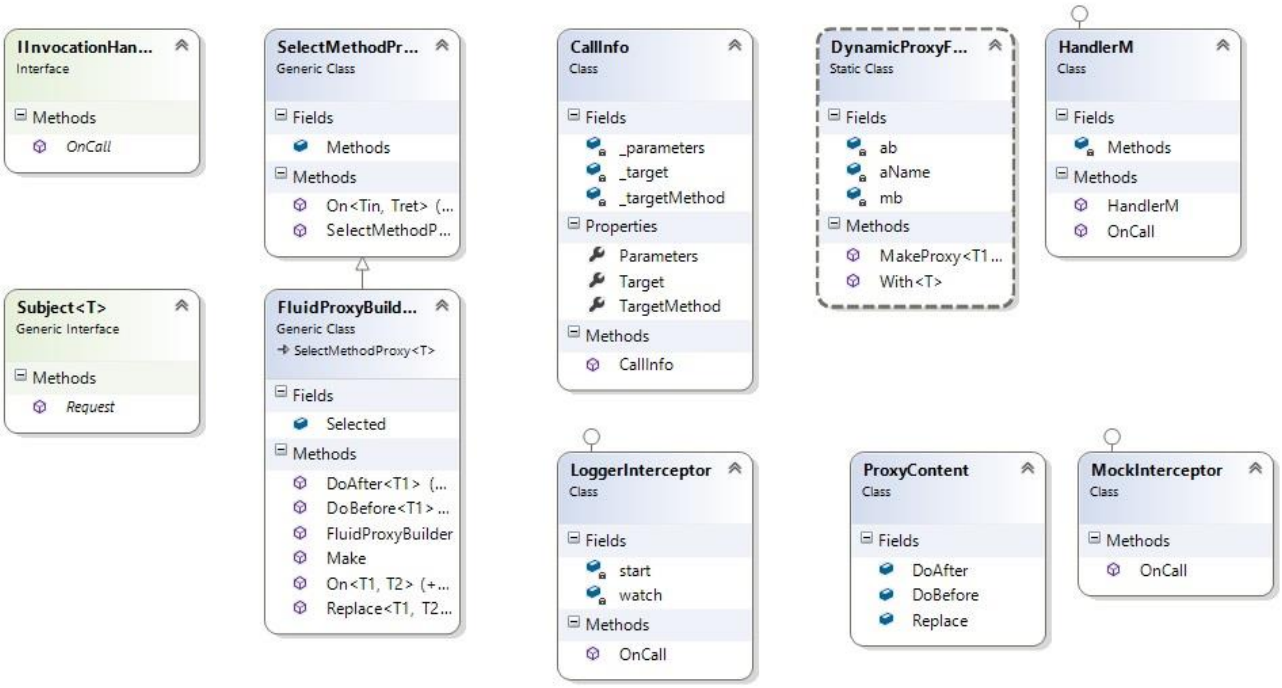
- Hilton Keitty Dias, Nº29973
- Homilzio Trovoada, Nº39368
- Mário Lourenço, Nº 39378

Introdução

O objectivo deste trabalho é o desenho e o desenvolvimento de uma API fluente pondo em prática os mecanismos apresentados nas aulas, nomeadamente a API de reflexão, *delegates* e linguagem intermédia(*CIL*).

Na implementação da API tivemos como base o padrão *Proxy* que consiste numa classe que serve de representante (ou fachada) para outro tipo. O representante e o tipo real têm a mesma interface.

Diagrama de Classes



Descrição das classes

- DynamicProxyFactory
- SelectMethodProxy
- ProxyContent
- FluidProxyBuilder
- HandlerM
- CallInfo

DynamicProxyFactory - é responsável por criar as proxys, seja através de um interceptor ou da API fluente.

SelectMethodProxy - garante que seja selecionado um método antes de se chamar um *DoBefore*, *Replace* ou *DoAfter*, esta classe contém um dicionário em que todos os métodos que podem ser interceptados são a chave, e sempre que se chama o método *On<>* é iniciado uma instância de *ProxyContent* a esse método no dicionário.

ProxyContent - contém 3 *delegates*:

DoBefore: nele são realizadas todas as acções antes de se executar o corpo do método.

Replace : contém o *delegate* que substituirá o método original.

DoAfter: contém a lista de todas as acções a serem executadas depois do método.

FluidProxyBuilder - contém os métodos *DoBefore*, *DoAfter*, *Replace*, *Make* e ainda redefine o *On<>*. No caso do *On<>* ser chamado mais do que uma vez para o mesmo método apenas tem efeito as alterações da última chamada.

HandlerM - o interceptor responsável por interceptar os métodos das proxys criadas através da API fluente.

CallInfo- Classe utiliária com informação sobre o método, o objecto (caso seja de instância) a que pertence e a lista dos parâmetros deste.

Utilização da biblioteca Reflection.Emit

A biblioteca *Reflection.Emit* foi uma ferramenta que nos ajudou a criar e guardar os tipos em runtime,(dinamicamente), gerando código *IL*, utilizando as classes definidas aí como *AssemblyName, AssemblyBuilder, ModuleBuilder, FieldBuilder, ConstructorBuilder, ILGenerator.*

Conclusão

Trabalho desenvolvido com sucesso após algumas anomalias intermédias no que toca a parte relacionada com a (*CIL*) por ser muito difícil fazer *debug* dos métodos onde é utilizada essa linguagem e também na definição dos campos da classe *ProxyContent* acabando por definir *Delegates* para o espaço, fazendo o lugar dos métodos de extensão, definidos na etapa 2.

Este trabalho foi uma ferramenta extremamente expressiva para consolidar a matéria apresentada durante o semestre nas aulas de Ambientes Virtuais de Execução.

Extra: Mais testes serão adicionados ao projecto assim que possível.