An approach to implementing Aspect-Oriented Modeling designs in C#

Farshad Saeidi

Islamic Azad University, Science and Research Branch Sanandaj, Iran, farshad.saeidi@live.com

1. Introduction

Mainstream languages (such as Java and C#) often do not have an inbuilt facility for aspect-oriented programming (AOP). However, sometimes we need aspectual decomposition without details. In this article, we propose an approach to support aspectual decomposition via OOP features in languages that build on .net framework.

2. Background

2.1. aspect-oriented modeling

Objects support modularity, encapsulation, and inheritance. However, some features of objects are shared between them such as a security checking, performance monitoring, and logging. Hence, they being scattered all over the application. Aspect was introduced as a response to limitations of OOP. According to the Kiczales and others [3], component and aspect could define as:

- *A component, if it can be cleanly encapsulated in a generalized procedure (i.e., object, method, procedure, API).*
- *An aspect, if it cannot be cleanly encapsulated in a generalized procedure.*

2.2. Extension Method

Extension methods are a special kind of static method in .net, which allow adding functionality to already defined classes without modifying them [2]. Extension methods defined as static methods but should call in the instance of class. The type of the first parameter is a class that will extend, and has "this" modifier. Extension methods must define inside a non-nested, non-generic static class [3]. The most common extension methods are the Language Integrated Query (LINQ), which is native data querying mechanism of .net languages. The Listing 1 shows an extension method defined for the String type in C#.

```
public static class testExtensions
{
  public static int WordCount(this String str)
  {
    char[] separators = { ' ', '.', '?' };
    var WordList = str.Split(separators);
    return WordList.Length;
  }
}
```

Listing 1. An example of for string type

Extension methods are limited to the namespace. For example, Parallel LINQ are defined in the System.Linq.ParallelEnumerable class. Without including this namespace (by using keyword) in your class, PLINQ extension methods of will not add to the targeted class. Thus, you could develop your Owen PLINQ and choose between them only by select intended namespace.

2.3. Custom Attributes

An attribute is a type of metadata that associating with code elements such as class and methods. In addition of predefined attributes (e.g., Obsolete attribute), developers could define their own attribute. In the Listing 2, class1 has the aspect1 attribute. Attribute class must be driven from Attribute class and declare by "Attribute" Suffix.

```
[Aspect1]
class class1
{ … }

class Aspect1Attribute : Attribute
{ … }
```

Listing 2. An example of for custom attribute in C#

2.4. Computational Reflection

The high level, safe and sufficient mechanism of processing code by itself is a computational reflection. "Computational reflection by definition allows an artifact to introspect and to intercede on its own structure and behavior endowing." [62] it already know as mechanism that influence AOP [93, 57].

3. Aspect-Oriented Modeling

By aforementioned feature, we could implement AOM design in C# or other .net languages. The first step is to write attributes` class; the second is decomposing the crosscutting functionalities and develop them as extension methods. Reflection could be used to retrieve custom attributes of code elements as well as other elements of code even private members. Furthermore, it could change values or invoke methods. Finally, we must tag target classes by attributes. The extension methods use attributes to find which behavior should perform on target class even change it by reflection. The Listing 3 is a pseudocode of AOM impanation in C#. The log method associate with object class; thus, it being available for every type even "int" data type. The log mrthod looking for Aspect1 Attribute by using "Type.CustomAttributes" property and decide about the appropriate operation.

```
class Program{
    static void Main()
    {
        var c1 = new class1();
        var c2 = new class2();
        var c3 = new class3();
        int i = 10;
        this.log();
        c1.log();
        i.log();
        c2.log();
    }
}

[Aspect1]
class class1 { }

class class2 { }

[Aspect1]
class class3 { }

class Aspect1Attribute : Attribute { }

public static class AspectExtension {
  public static void log(this object target){
    var c=target.GetType().CustomAttributes;
    var a=c.FirstOrDefault(
      X=>
      X.AttributeType.Name=="Aspect1Attribute"
      );
    If    (a!=null)   { }
    else              { }
  }
}
```

Listing 3. Pseudocode of Log aspect in C#

4. Conclusion

This article attempt to explain how decompose crosscutting concerns. Unlike another work that develop complex artifact or use third-party approaches, prosed approach focus on using inbuilt facilities of .net framework include Extension Method, Computational Reflection, and Custom Attributes.

References

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier and J. Irwin, "Aspect-Oriented Programming," in *ECOOP 97*, Finland, 1997.

[2] G. M. Hall, Adaptive Code via C#: Agile coding with design patterns and SOLID principles, Pearson Education, 2014.

[3] B. Wagner, L. Latham, S. Addie and M. Wenzel, "Extension Methods (C# Programming Guide)," 23 08 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods. [Accessed 25 08 2017].

[4] W. Cazzola, "Evolution as «Reflections on the Design» , Models@run.time," 2014.

[5] R. Pawlak, L. Seinturier and Jean-Philipp, Foundations of AOP for J2EE Development, New York, NY, USA: Apress, Springer, 2005.

[6] É. Tanter, From metaobject protocols to versatile kernels for aspect-oriented programming, PhD thesis, University of Nantes and University of Chile, 2004.