Extending Aspects Support in .NET Framework

Satyender S. Duhan
Department of Computer Science & Applications,
Kurukshetra University, Kurukshetra, Haryana, India.
ssduhan@gmail.com

Abstract*: Aspect-oriented programming is the programming paradigm of next generation that is one step ahead to object oriented programming paradigm. It improves separation of concerns in software development. Aspect Oriented Software Development makes it possible to modularize crosscutting concerns of a software system more cleanly, thus results in lesser coupling and higher cohesion. In this paper I have done a review on aspect oriented software development with AspectJ language that is more often used for the purpose. An analysis has been made to find out the scope of the implementation of the AOP in .NET framework. I also proposed the enhanced compiler model to support AOP in .NET framework.*

Keywords: *Aspect-Oriented Software Development, Aspect Oriented Programming, AspectJ Crosscutting Concern, .NET framework, Enhanced Compiler Model.*

1. INTRODUCTION

Aspect-Oriented Programming (AOP) is a new technology for dealing explicitly with separation of concerns in software development [2,5]. Aspect Oriented Software Development makes it possible to modularize crosscutting concerns of a software system, thus making it easier to maintain and evolve [6]. By far, the most compelling argument for aspect-oriented programming is that it significantly increases modularity and cohesion, a cleaner separation of concerns, and improved locality of change, thereby increasing understandability, easing the maintenance burden. In particular, AOPs use abstractions representing *concerns* that *cross-cut* the program modules that implement the primary functionality [5].The development process are modified by using AOP. Classes and methods of core/main concerns are implemented and

tested as before. However, instead of embedding the code for crosscutting measures into method bodies, separate aspects are defined that enclose the code. For example, code that implements a particular security policy is commonly distributed across a set of classes and methods that are responsible for enforcing the policy. However, with AOPs, the code implementing the security policy can be factored out into one aspect. Later, the aspects are woven into the classes that represent the core concerns of the system. Once complete, the woven targets should be the composite of behavior of both core and cross-cutting concerns [1].

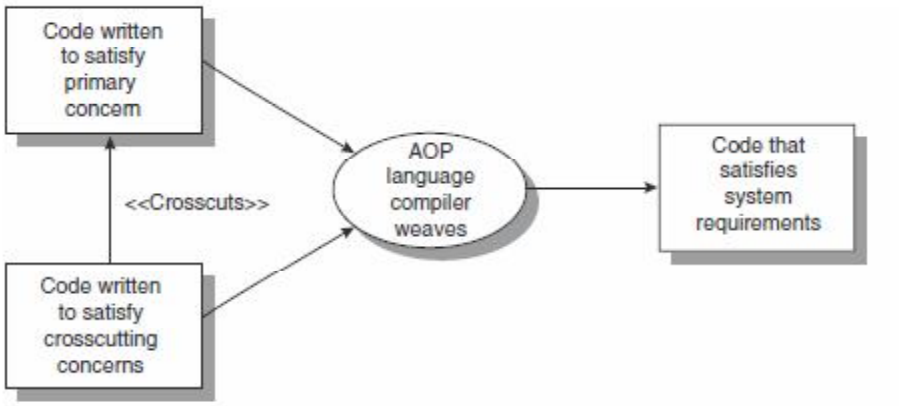

Figure 1: Aspect Oriented Programming Process [1].

2. ASPECT-ORIENTED PROGRAMMING CONCEPT

Aspect-Oriented Programming (AOP) complements object oriented programming by allowing the developer to dynamically modify the static object oriented model to create a system that can grow to meet new requirements. Following are some of the important features of aspect oriented programming [6,17]:

i. ***Cross-cutting concerns*****:** Even though most classes in an Object Oriented model will perform a single, specific function, they often share common, secondary requirements with other classes [19]. For example, we may want to confirm security strategy implemented to classes within the data-access layer and also to classes in the user interface layer whenever a thread enters or exits a method. Each class has a different primary functionality, but identical secondary functionality.

ii. ***Advice*****:** The supplementary code that you want to apply to your existing code. In our example, this is the security strategy code that we want to apply whenever the thread enters or exits a method.

iii. ***Join point:*** The events during execution at which advice may execute are called join points [16]. For example join points can be the method calls and execution, constructor call and execution, exceptions thrown, initialization of classes etc.

iv. ***Point-cut:*** The point of execution in the application at which cross-cutting concern needs to be applied [19]. In our example, a point-cut is reached when the thread enters a method, and another point-cut is reached when the thread exits the method.

v. ***Aspect:*** This is the combination of the point-cut and the advice.

vi. ***Weaving*****:** It is the process of linking aspects with base code or other application types. This can be done at compile time, load time, or at runtime.

3. AOP vs. OOP

Aspect Oriented programming can be used not only with functional programming, but also with object oriented programming. It should not be "vs."; it should be "Aspect Oriented Programming *with* Object Oriented Programming". Aspect oriented programming is not introduced to replace OOP, but complements it to eliminate its short comings. AOP may be seen as elder brother of OOP [9]. AOP may be considered as a kind of "meta-programming". The whole thing that AOP does could also be done without it by just adding more code that will off course increase the coupling effect. AOP saves you writing this code.

Where the components of OOP are inheritance, encapsulation, and polymorphism, the AOP are join points, point-cut, advice, and introduction [15]. To better understand these terms, consider the following simple example.

```
public class ClsFirst
{
        public void sayWelcome () {System.out.println ("Welcome to AOP"); }
        public void saySomething (String s) {   System.out.println (s); }
        public static void main (String[] args)
        {
                sayWelcome ();
                saySomething ("Hi");
        }
}
```

Listing 1: ClsFirst.java

We have our existing Java code in ClsFirst.java. Suppose, we want to add aspects to perform additional functions as follows:

i. To print a message *before* and *after* any call to the ClsFirst.sayWelcome() method.

ii. To test that the argument of the ClsFirst.saySomething() method is at least four characters.

The following is the AspectJ implementation for the desired functionality.

```
public aspect FirstAspect {
    public pointcut MethodCalled (): call (public void ClsFirst.say*() );
    public pointcut MethodCalledWithArg (String s): call(public void
    ClsFirst.saySomething(String)&&args(s);
    before(): MethodCalled() {
    System.out.println("\n ClsFirst." + thisJoinPointStaticPart.getSignature().getName() + "Begin..." ); }
    after(): MethodCalled() {
    System.out.println("\n ClsFirst." + thisJoinPointStaticPart.getSignature().getName() +" Finish...");}
    before(String s): MethodCalledWithArg(s) {
    if (s .length() < 4) {
        System.out.println ("Error: Message should have not less than four characters");
        return;
        }
      }
    }
```

Listing 2: FirstAspect.aj

AspectJ Compiler: For compiling and running a0bove example just write:

ajc FirstAspect.aj ClsFirst.java

Note that it is also allowed users to concurrently edit the same file. Inspired by the aspect-oriented transaction framework *AspectOptima* [8] an aspect is implemented to make users work on different copies of a document using aspect *versioning* feature [4].

4. INNOVATING ASPECT ORIENTED PROGRAMMING IN .NET

Aspect Oriented Programming is a technique to separate crosscut code across different modules in any software system. Every requirement is a concern. Software development exists because of business concerns [7,8]. Software development is nothing but addressing a collection of concerns in real life [9]. For instance, a retailer sales book has the concerns: sale details, facility to mail sales information to publisher,

taking print out of billing information etc. after each transaction. Following is the UML class diagram for this software system.

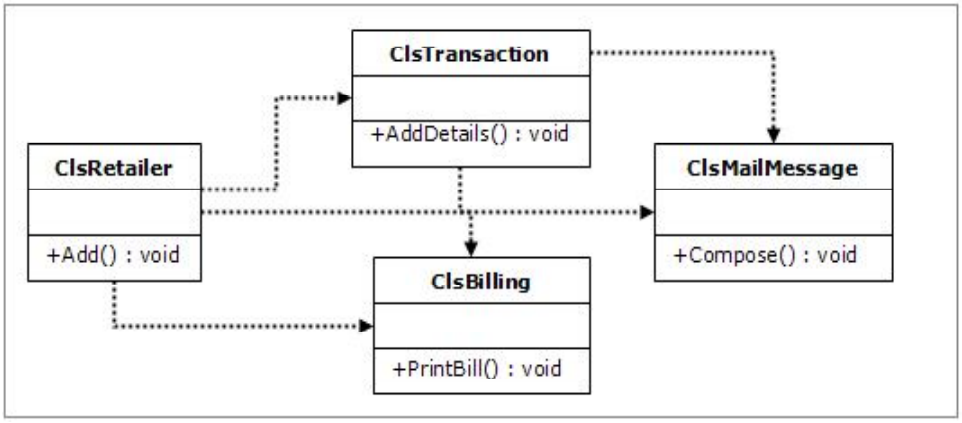

Figure 2: UML Class diagram for Retailer Sales Book.

4.1 Tangled code problem in .NET languages

Aspects can be identified at the design and implementation phases, though the inter-component tangling tends to occur at the implementation/coding phase [3]. Look at the *Add* method of the Retailer class.

```
public void Add()
{
        // method adds customer details, Transaction details; Adding code will go here
        ////////////////////////////////////////
        // after adding details to database, an Message is sent in the form of Mail
        ClsMailMessage objClsMailMessage = new ClsMailMessage();
        objClsMailMessage.Compose();
        // After sending mail message, bill is printed
        ClsBilling objClsBilling = new ClsBilling();
        objClsBilling.PrintBill();
}
```

Listing 3: Add method of class "ClsRetailer"

Note that class *ClsRetailer* is doing very bulky jobs some of which is not of its concerns. Example: sending mail message and printing billing information are not its concern at all. The same implementation has to be done with the *clsTransaction* class. The *Billing* and *MailMessage* span across more than one module. As we have discussed throughout previous sections that such types of concerns are called as *Crosscut concerns*. The code is quiet messed up as we are using lots of objects. These types of code are called as *"Tangled code"* in AOP terminology [9].

So all software applications have two types of concerns: *Core/Main* concern (Example: Retailer and Transaction maintenance concerns) and *Crosscut* concerns (Example: Printing, logging, sending mail, security and exception handling etc. which spans across modules).

4.2 Proposed solution for tangled code: Enhanced Compiler Model

Till now .NET compilers did not support actual AOP. This is a major deficiency of .NET framework. Compiler is AOP featured when it has keyword support for *Aspect*, *Join points*, *Pointcut*, *Advice* etc. [9]. So I support the arguments that C# and other .NET languages are not actual AOP languages. .NET does not support compile time and link time weaving. Run time weaving is done by using the .NET runtime. Code detects the core, crosscut etc. and executes them at run time. AOP in .NET framework can be achieved up to some extent through Attribute Programming [10,11] and Context-Bound objects [10,18]. Many AOP researchers do argue that it is not actual AOP [9].

In this section, I am going to present the proposed model for compilation/ execution process in .NET in order to support separation of concern for cleaner modularity and to convert C# (an OOP) to AOP. The figure below shows the proposed compiler model for .NET with C# as a base language. In this model separate modules for crosscut (aspects) and core concerns (base code) are developed and then all the modules are fed to the compiler. There will be a *weaver* attached to or be a part of compiler which weaves the aspects with base code. The extended (AOP supported) compiler then compiles both the modules and generates a single Intermediate Language (MSIL) and executable.

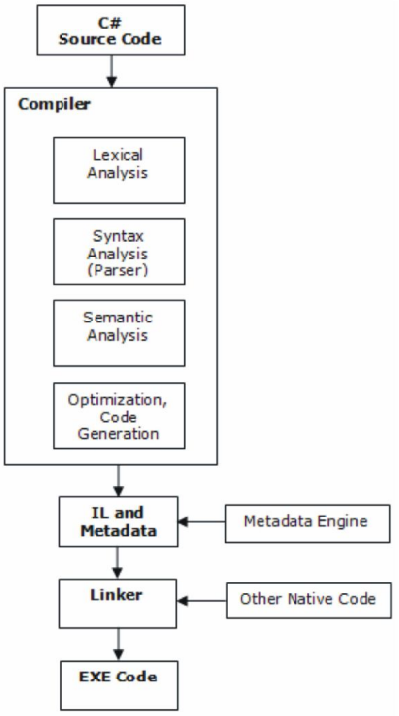

Figure 3: Conceptual structure of the typical compiler in C#.NET [12,13].

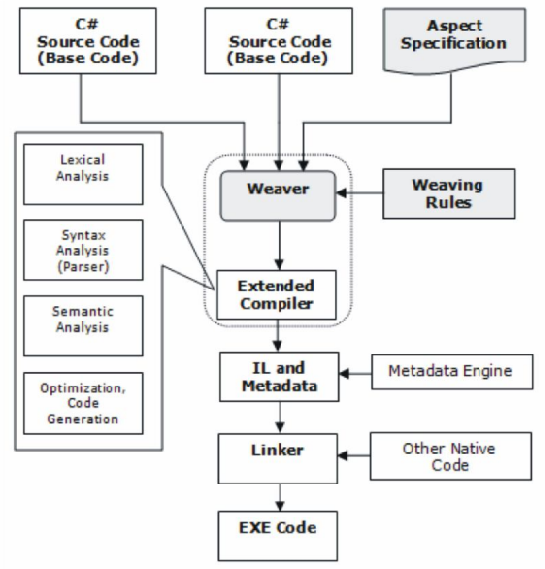

Figure 4: The Purposed Extended Compiler to support Aspects and Weaving.

5. APPROACHES TO IMPLEMENT AOP IN .NET

There are some alternating techniques to implement AOP in .NET at runtime. Although they are not the exact AOP implementation as discussed in previous section, a couple of them are attribute programming and using the third party compilers (EOS compilers).

5.1 Attributed Programming

In this section, I used custom attributes to implement AOP concern separation. Crosscut concern code is moved to attributes and method implements the main concern. The class should be derived from "*System.Attribute*" to be used as an attribute. Below is the *constructor* and *compose* method code for *Mailing message* crosscut concern.

The MailMessage Class:

```
using System;
using System.Reflection;
using System.Net.Mail;
using System.Net.MailMessage;
        [AttributeUsage(AttributeTargets.All)]
        public class ClsMailMessage : System.Attribute
        {
        private string EmailID;
        public ClsMailMessage(string strMailid)
        { // TODO: Add constructor logic here
        EmailID=strMailid;
        Compose();
        }
        public void Compose()
        {
        MailMessage message = new MailMessage();
        message.To=EmailID;
        message.body= txtmessage.Text;
        SmtpMail.Send(message);
        Response.Write("Email sent to Publisher successfully..");
        }
}
```

Listing 4: Crosscut "MailMessage" code

In the same manner we can define the *constructor* and *PrintBill* method for class *ClsBilling*. Finally, the Retailer class will call both the concerns as follows:

```
public class ClsRetailer : ContextBoundObject
{
public ClsRetailer()
{
        // TODO: Add constructor logic here
}
        // tangled code in "Listing 3" is replaced by following attributed code
// Attribute applied code
[ClsMailMessage("publisher@gmail.com")]
[ClsPrintBill("homeprinter")]
public void Add()
{       //adding details code will go here.
        Response.Write("Add method called of the class Retailer");
}
public static void Main(string[] args)
{
ClsRetailer ret = new ClsRetailer();
ret.Add();
Console.Read();
}
}
```

Listing 5: Core concern "Retailer" code

The main drawback of the attributed programming is that we cannot specify at what point the crosscut concern code should run. In the above class both the crosscut concerns are firing *before* the customer is added to the database that is before the Add method definition. Another drawback is that all properties are set using the constructor of the class, which is not the proper way to implement business logics [9].

5.2 Using the EOS compiler

The one of the best features of .NET framework is that is it is designed in such a manner that it can support third party software without causing interface problems. EOS compiler tool, a third party software tool is an aspect-oriented extension for C# on Microsoft .NET Framework. Latest version of EOS can be downloaded from [14]. EOS compilers have support for AOP keywords like *aspect*, *introduce*, *before*, *after* etc. So the AOP implementation can be made possible in .NET with the help of third party AOP compilers compatible with .NET framework.

Remoting Proxies could be another solution possible, but unfortunately Remoting Proxies can only be used on interfaces or MarshalByRefObjects.

6. CONCLUSION

I believe it is the right time to develop the full flagged concepts of Aspect Oriented Programming into software development processes. AOP can help to simplify typical existing systems, and will assist the programming of exceptionally complex systems of future.

We can not perform weaving using the current .NET compilers. So there is no real time implementation to be seen using the current compiler. This paper addresses the aspect support in the .NET framework, and proposes an extension to compiler to support aspect weaving and thus AOP. After few years, this paper will look completely unintelligent, when Microsoft would have made their compilers AOP compatible. This work increases the aspect reusability and interchangeability in .NET. I consider it a next logical step in the evolution of AOP in .NET framework.

REFRENCES

[1] Joseph D. Gradecki and Nicholas Lesiecki, *Mastering AspectJ Aspect-Oriented Programming in Java*, Published by Wiley Publishing, Inc., Indianapolis, Indiana, ISBN 0-471-43104-4.

[2] L. Bergmans and M. Aksits. *Composing crosscutting concerns using composition filters*. Communications of *ACM*, 44(10):51–57, 2001.

[3] K. Czarnecki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. Ph.D. Thesis, Technische Universität Ilmenau, Germany, 1998.

[4] Antoine Marot,Roel Wuyts, *Composing Aspects with Aspects, AOSD'10,* ACM Press 978-1-60558-958-9/10/03, March 2010.

[5] G. Kiczales, E. Hilsdale, , J. Hugunin, M. Kersten,J. Palm, and W. Griswold. *An Overview of AspectJ*. In 15th European Conference on Object-Oriented Programming, Budapest, Hungary, 2001.

[6] T. AspectJ. *The AspectJ (TM) Programming Guide*, 2002.

[7] G. Denaro and M. Monga. *An experience on verification of aspect properties*. In Proceedings of the 4th international workshop on Principles of software evolution, pages 186-189. ACM Press, 2002.

[8] J. Kienzle, E. Duala-Ekoko, and S. G´elineau. Aspectoptima: *A case study on aspect dependencies and interactions. In Transactions on Aspect-Oriented Software Development V*, pages 187–234. Springer-Verlag, 2009.

[9] Shivprasad Koirala, *An Introduction to Aspect oriented programming in .NET,* Article on: http://www.codeproject.com, 22 august 2005.

[10] Dharma Shukla, Simon Fell, and Chris Sells, *Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse,* MSDN Magazine, March 2002 issue.

[11] MSDN Help *Aspect oriented Programming in .NET, http://msdn.microsoft.com/en-us/library/aa288717(VS.71).aspx.*

[12] E. Balagurusamy, *Programming in C#,* Tata McGraw-Hill Publication Company Limited, ISBN: 0070667578, Nov. 2007.

[13] Kazuaki Maeda, *XML Externalization Built into Compiler Front -Ends Using a Parser Generator*,IAENG International Journal of Computer Science, Advance online publication: 15 August 2007.

[14] Third Party Compiler Tools to Implement AOP in .NET: http://www.cs.virginia.edu/~eos/.

[15] Yasser EL-Manzalawy, Systems & Computers Engineering Department, AZHAR University, Cairo, *OOPs vs. AOP*, http://www.Developer.com

[16] M. Wand, G. Kiczales, C. Dutchyn, *A semantics for advice and dynamic join points in aspect-oriented programming*. ACM Transactions on Programming Languages and Systems (TOPLAS), September 2004. Volume 26, Issue 5

[17] Filman, Robert E.; Tzilla Elrad, Siobhan Clarke, and Mehmet Aksit. *Aspect-Oriented Software Development*. ISBN 0-321-21976-7.

[18] T. Dinkelaker, M. Mezini, and C. Bockisch. *The art of the meta-aspect protocol*. In AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development, pages 51–62.ACM, 2009.

[19] Graham O'Regan, *Introduction to Aspect Oriented Programming*, article on http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html?page=1