A Study of Handling Cross-Cutting Concerns in Software Applications

Dr. Kamlendu Kumar Pandey[1]* Dr. Devendra Pandey[2]

[1,2] Assistant Professor Department of Information and Communication Technology, Veer Narmad South Gujarat University, Surat, India

Abstract – Cross-Cutting Concerns are the part of design pattern and application architecture in software development process where several system and allied concerns are handles in an independent manner saving time of application development. Programming languages tend to focus more on Business Logic implementation and no other aspects like input validation, access control, transaction management, input/output validation, error handling, and in some cases session management. Here in this paper we consider how cross-cutting concerns is implemented in different from object oriented programming languages and platforms.

Keywords: Cross-Cutting Concerns, Point-Cuts, Aspects, Concerns

-----------------------------------x-----------------------------------

1. INTRODUCTION

The software application development has to be done in context with the current trends, practices and challenges. The productivity is very important as no of man hours decide the cost incurred in the project. All the current practices are trying to reduce the development time. It has been observed that most of the developer's time is consumed in writing system level concerns rather than the core business logic. *Cross-Cutting Concern [1]* are the design pattern to be followed in the modern application design. A pizza order application which needs inventory checking and payment has to deal with lot of cross cutting concerns which can affect the running of various modules. The concerns require handling at many places and developers have to infuse a repetitive coding in all the modules. If a concern changes than also the changes have to be done in various modules. The concerns in pizza application are, user verification, transaction processing, status updation, concurrency handling, customize resources in real time, data validation, inter thread communication handling and creating user log with timestamps of all the operations . It can be used to extract an audit trail for the application if asked by the company or user. This can make an application trustworthy. It is shown in Fig 1 as how cross cutting concerns are handled.

Cross-Cutting Concerns (CCC) as a design pattern focus on modularizing the application in such a way that various implementations used in CCC can be materialized and the programmers have a clean way to put their system level and application level concerns. The CCC is neatly implemented in Java with terms called aspect and that is why they name it as Aspect Oriented Programming [2]. .Net do not have a defined API for handling CCC but it can be made to do so. This paper covers both case of Java and .NET. Java Spring API has incorporated Aspect as mandatory implementation.

Cross Cutting Concern

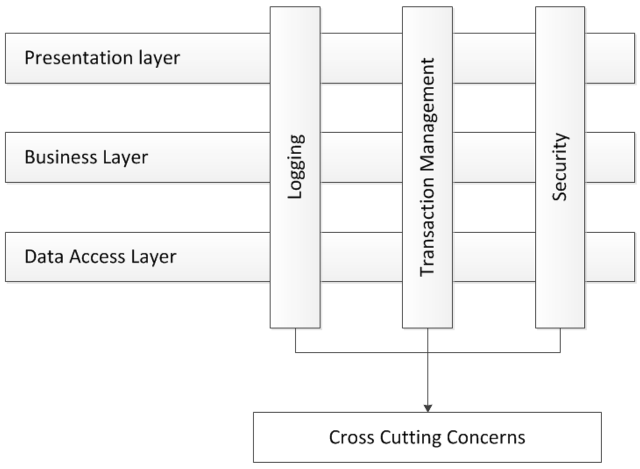

***Fig.1 Cross cutting concerns in application (courtesy :* https://www.codejava.net)**

Some terms used in Aspect Oriented Programming

a) **Aspect:** This a module in the project that cuts across various implementations in a software implementation. Events like logging can be one of the aspects.

b) **Join point:** This is a point in the code to which the concerns are bound and execution on this point will trigger the aspects

c) **Advice:** The are places on join points as an additional and conversational annotation which tells in what condition the aspect should be executed (execution before or after)

d) **Pointcut:** This is a kind of declaration in an execution cycle as how many point cuts are to be considered

2. IMPLEMENTATIONS OF CROSS-CUTTING CONCERNS

As this is part of design patterns we will see how aspects are considers in various programming languages for reference we have selected Java and .Net frameworks.

2.1 Java Implementation of Cross-cutting Concerns using AspectJ

AspectJ [3] is the implentation of CCC in Java language. It comes out with various terms like aspects, advices, join points, point cuts and using them in a specific way is called weaving. The rules and the syntax to be followed in the is shown in the example given below . We have covere java in two ways. A command line application and a web application to show its real power of handling cross cutting concerns.

AspectJ - SampleHello Example to show CCC

First we will see how a Java class of simply printing a message with a suitable additional greeting behaves in a Non Aspect way.

```
// A class SampleHello in file SampleHello.java

public class SampleHello {

private void speak(String somemessage) {

sop(somemessage);

}

private void speakToPerson(String greet_message, String somenane) {

Sop(somenane + ", " +greet_message);

}

private sop( String str) { System.out.println(str);}}
```

We can perform the task in AspectJ way also in the following way

```
// This is a aspect driven class - GreetingManners.java

public aspect GreetingManners {

pointcut call speakGreetingMessage() :

try{

call(public void SampleHello.speak*(..));

before() :

callApectMessage() {

sop("Hi!");

}

after() : callAspectMessage() {

sop("Happy to see you again!");} catch(Exception e) {}

}

private sop( String str) { System.out.println(str);}}
```

2.2 Cross cutting concerns with Spring Framework

Java Spring framework has got excellent and easy-to-use AOP capabilities[7]. Spring follows the concept of Inversion of Control which is bound to the web server / container. So all the Aspect based configurations are directly executed by container execution environment. Aspect comes bundled in with the Spring frme work which is based on MVC design pattern. The beans and the methods are using aspects to do address many system level and application level concerns. Then the Spring Aspect class is written as

```
public class MySpringAspect

{

public Object monitor(ProceedingJoinPoint myaspect)

try{ {

sop("Recording the call in log [" + myaspect.toString()+

"] using values :"+ myaspect.getArgs()[0] );

Object placeholder = myaspect.proceed();

sop("Spring Recording Aspect : Spring End call [" +

myaspect.toString() + "returning :" +placeholder);

} catch (Exception e)

return placeholder;
}

private sop( String str) { System.out.println(str);} }
```

This aspect has to be registered with spring beans as follow

```
<bean id=Recording"  class = "MySpringAspect"/>

<! AOP code --><aop:config>

<aop:aspect ref="MySpringAspect">

<aop:pointcut id="ptRecording"

expression="execution(* SampleHello*.*(..))"/>

<aop:around pointcut-ref="ptrecording" method="monitor"/>

</aop:aspect>

</aop:config>
```

2.3 Implementation of Cross-cutting concerns using .NET frameworks

Though .NET framework does not use AOP explicitly but there are many was by which cross-cutting concerns can be dealt with. To do this .NET heavily relies on reflection and introspection aspects :-

- CRUD. operations
- Track Sales of customer.
- Print information of sales.
- Get information of emails and sales.

Let us see the coding in .NET to handle this

Let us assume that we have a Customer class and some operations mentioned above are to be performed on Customer class. Below are the .Net Implemntation of the same

```
public void addCustomer()

{ Email custEmail = new mail();

custEmail.Send();

Print custPrint = new Print();

custPrint.Print(); }
```

Above is the example of tangled code . The customer is to be added first in the database using standard CRUD operations then same data is to be Emailed using a using Email class and then the entire data is to be printed on the web interface. This kind of coding is quite taxing and becomes repetitive in all such entities while doing CRUD operations.

We have two ways to deal with the situation.

a) Weaving : Way to deal with cross cutting concerns

We can categorize the concerns as core concerns and Cross cutting concerns. With the API available address this concern with appropriate methods, logic and use Point cuts and aspects to deal with it. Once the core and cross cutting concerns are written they must be packaged in one with appropriate annotation so that they behave a same block. This is called weaving. Java uses this way to handle cross cutting concerns. Advices Re one of the major part and they should be placed on events as to execute after or before the method calls. .

b) The Attribute Programming

.Net as such do not have a separate API for handling cross cutting concerns but with the help of Attribute based programming it can separate the concerns. There are no aspect, point cust and advises as in mature Aspect API but with the help of generic .Net API and Attributes it is possible to handle the concern. There is of course no comparison directly with Java as .Net has its own comprehensive way to implement different software architectures. Here we will see how attributes can be used to handle cross cutting concerns

Cross cut Email code :

The namespace : namespace CustomerSalesProject

{

// A brief look into this

using Attribute : [AttributeUsage(AttributeTargets.All)]

A class : public sealed class Email : System.Attribute

```
{

private string emailStr;

public Email(string xemailStr)

{

emailStr = xEmailStr;  dispatch();  }

public void dispatch()
{

Console.WriteLine("The Email is dispached to Administrator");} } }
```

Handling the Aspects and Cross Cutting concerns of printing

```
/*Defining name space */namespace CustomerSalesClassProject

{

[AttributeUsage(AttributeTargets.All)]

class ourPrint : System.Attribute

{

string strPrinterName; public Print(string pstrPrinterName)

{

strPrinterName = pstrPrinterName; OurPrint();

}

public bool sayPrint()

{

output("Printed to printer");   return true; } }

public output(str) { Console.WriteLine(str))}
```

All the concerns mentioned over will be called by the customer now.

```
/*Customer Class*/ /public class RCustomer : ContextBoundObject

{

public RCustomer()  {  }

[Email("someperson@ernet,in")]

[Print("Our Printed")]

public void AddRecord()

{

Console.WriteLine("Record is added to Customer");   } }
}
```

3. DISCUSSION

The cross cutting concerns find their ways in all the programming languages. Some languages have inbuilt mechanism to deal with it while in someone needs to write some system level code to handle the same. We saw that AOP is the integral part in Java. It is implemented in stand-alone as well as Spring based web application frameworks. We can that in case of .Net no advice is being used. We can see in the above code that both the cross cutting concerns are actually fired before the data of customer is added int o the table. The constructors are invoked in every call that means the all logic will be invoked throgh constructor. This is not a good way of writing code. In .NET the main class will always need a change of code whenever a concern regaring this is to be invoked. Like we have to add Atttibute annotation on all the method in which concerns are raised. In practice AOP implementation has to be clean and the main class need not be disturbed in any case. We must have separate concern handling classes to be invoked on a particular event. The aspects are of great use when we want to maintain audit trail of our applications and each and every event need to be logged into with proper time stamp and user details. It has been observed that it a often ignored task left to be done later which we feel that this is the integral part of design and must be interwooven neatly in the application.

4. CONCLUSION

We have tried to investigate as how the cross cutting concerns are handled by various application development frameworks. We have explored Core Java, Spring framework and .Net framework to study this. Java is having an build mechanism to handle cross cutting concerns. In Spring the concerns are based on container's property of Inversion Of Control where aspects are the integral part to handle system level and transaction level concerns without disturbing the core business logic. In .Net The cross cutting behavior is achieved by Attribute programming and weaving activity. All languages like Ruby, Python and Angular too have ways to deal with cross cutting concerns. The purpose is to make the reader know that we can do away with all clutter in our business logic using aspects in our applications

REFERENCES:

1. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda Aspect-oriented programming, 1997 – Springer

2. Shigeru Chiba, Michihiro Horie, Kei Kanazawa, Fuminobu Takeyama, and Yuuki Teramoto. 2012. Do We Really Need to Extend Syntax for Advanced Modularity?. In Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development (AOSD '12). 95—106

3. David Robinson, "An introduction to Aspect Oriented Programming in Java"

4. Robert E. Filman and Daniel P. Friedman (2000). Aspect-Oriented Programming is

Quantification and Obliviousness. Technical Report

5. P. Alves, E. Figueiredo, F. Ferrari (2014). **Avoiding code pitfalls in aspect-oriented programming** Proc. of the Brazilian Symposium on Programming Languages, SBLP, Brazil (2014), pp. 31-46

6. http://www.onjava.com/pub/a/onjava/ 2004/01/14/aop.html

7. http://docs.spring.io

8. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold (2001). **An overview of AspectJ** Proc. of the 15th European Conf. on Object-Oriented Programming, ECOOP, pp. 327-353

Corresponding Author

Dr. Kamlendu Kumar Pandey*

Assistant Professor Department of Information and Communication Technology, Veer Narmad South Gujarat University, Surat, India

kspandey@vnsgu.ac.in