

The road to Aspect Oriented Programming **Another way to orient your code?**

By Dr. Neil Roodyn

Over the last few years there has been a noise I've noticed that has been getting louder. Upon listening more intently it appeared to be a quiet voice whispering "AOP". A few months ago the sound was becoming loud enough to get me wondering and I thought it was time to find out more. In 1995 I worked for a small software company developing real time 3D graphics software, they were called the Art Of the Possible, but I was sure AOP had nothing to do with these guys. First step get onto Google, a quick search and I downloaded a paper entitled "Aspect-Oriented Programming" that was presented in 1997 at the European Conference on Object-Oriented Programming by a bunch of guys from Xerox PARC. So it's been around for a while, in fact the paper was written after the authors had been working on the ideas for a couple of years.

The conference paper looked like a hard place to start; just because I have a Dr. in front of my name doesn't mean I want to make life difficult for myself! A quick look through the Google results and I came across a few articles that looked a little gentler as an introduction.

I was right about the less intense nature of these articles although they also seemed a little fluffy without much real discussion as to what AOP really is; oh well back to the source and the original paper from Xerox PARC. With examples in a LISP like language and a whole new vocabulary to get to grips with, the paper certainly doesn't make it easy for the average developer to get to grips with AOP. What I'm going to attempt to do in this article is take you (obviously an above average developer if you're reading this) through what aspect oriented programming is all about and how you can get going doing some of it now.

Programming History 101 – the business case

In the beginning man wrote big programs in one file with lots of goto statements, mans head hurt. Some genius suggested breaking the problem down into smaller pieces and writing code to solve all the small pieces then 'compiling' the small pieces together to get the program to work. Writing the small pieces was easy and man was happy, then the pieces had to be connected together and mans head hurt again! Over the years the nature of these small pieces has been refined, from functions and procedures through to modules and more recently to classes. The aim has been to keep logically related code together and in object oriented programming to create classes that represent something that we humans understand, with state and functionality.

Different programming environments have presented different mechanisms to help us create 'better' software. C++ popularized ideas like type safety and class inheritance, Java popularized the virtual machine and reflection, the .NET framework has begun to popularize ideas such as attributes and cross language support. All of these and the plethora of other development tools try to make life easier for the developer, the tester, the poor guy who has to maintain the code in 10 years time and ultimately provide greater business value for the purse holder paying for the software development. AOP is another idea to help us developers do a better job and increase the value of what we produce.

What's wrong with what we do now?

A good place to start is to examine what issues exist with our current model of development, assuming we are using good OOP practices (you did say you were above average didn't you?). Think about some of the software you've written lately, has there been any duplicate code? Of course not you are above average and refactor all duplicate code into its own methods and classes, right? But what about all those traces, the error logging, the exception handling, the security checks you're still doing all those across the board in multiple classes, if not in the classes that are being directly instantiated then in their parent or bases classes. Let's take a small example, imagine we're building a banking system, the classes we might create would be account, employee and customer; Figure 1.

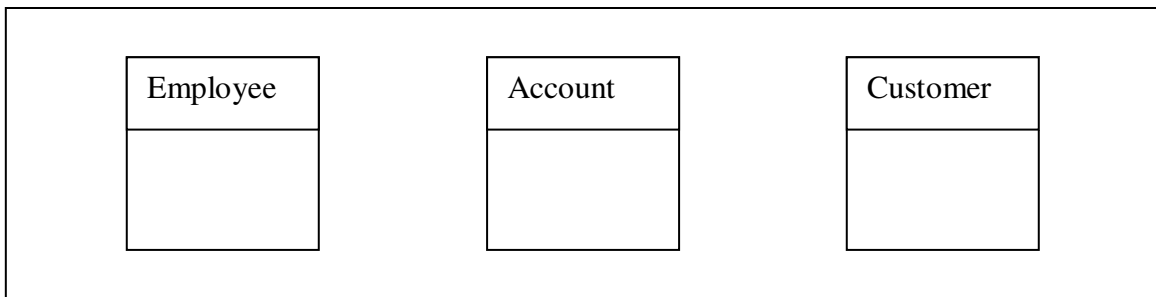


Figure 1

An account object might be restricted to access by only certain employees and the customer to which is it related. Of course we'd need to keep records of every transaction in an account and also log the activity of all employees and customers on the system. In an OOP world how would we do this? Create a base class or a static class that contains the functionality for security, logging and transactional support? Does it make sense to have our Employee, Account and Customer classes derive from this general purpose base class? Figure 2

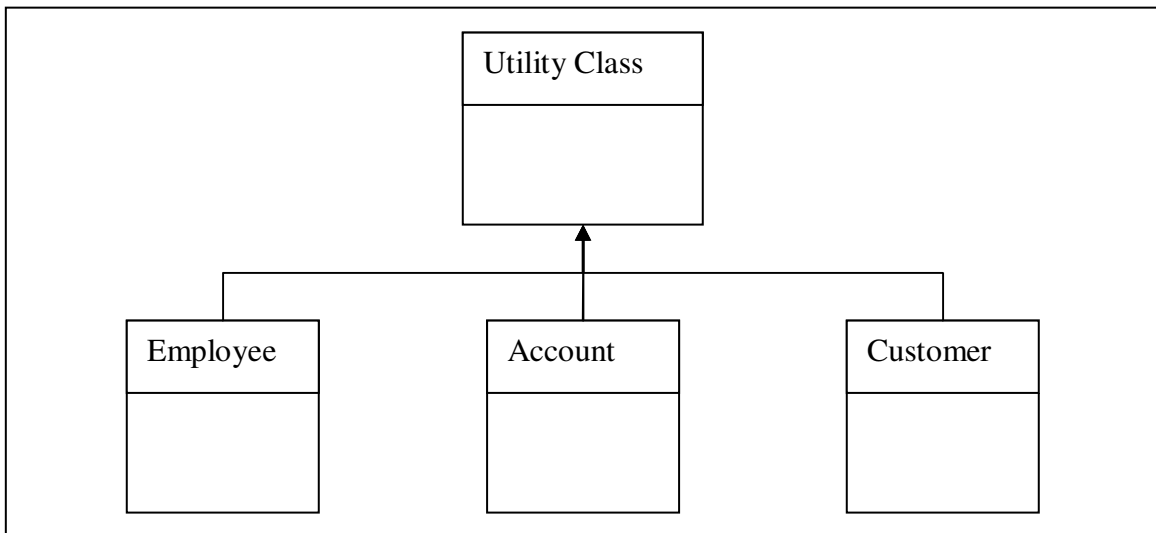


Figure 2

The very fact we have a utility class raises a bad odour in the design and the code, we have some functionality which cuts across the problem domain, known as cross-cutting in the AOP world. Even if we are happy to go along with this sour smell in our design lets think about how the code might look in one of our methods, the withdraw funds method in the account class (I have used explicit namespaces to make a point here), Listing 1 :

```
void WithdrawFunds(Customer c, Decimal amount)
{
    Utility::LogMethodCall(c);
    Try
    {
        if (Utility::HasPermissionToWithdraw( c ) )
            try
            {
                Utility::BeginTransaction();
                // Do the actual withdrawl
            }
            Finally
            {
                Utility::EndTransaction();
            }
    }
    Catch( Exception e )
    {
        Utility::HandleException(e);
    }
}
```

Listing 1

Notice that I haven't actually written the business code, all the code in Listing 1 is the infrastructure code that supports the business logic that will go in the function. How many of you have written code that looks like this? I know I have, and it gets very repetitive and boring. All you want to do is insert the actual functionality to get the system working and yet in order to deliver a robust, secure and well behaved application you have to reinsert this infrastructure code time and again.

Some languages have clever mechanisms that can help solve this problem, for example templates and multiple-inheritance in C++. Unfortunately they tend to introduce another problem, namely complexity. The code tends to get tangled and much harder to maintain, especially for those other average programmers (who don't read these articles).

It is this cross-cutting problem, where functionality exists in a more generic sense for the system to operate correctly and needs to be available across the domain of objects that are instantiated in the running system, which we need to solve.

Aspects a possible solution?

Imagine we could break all that repeated infrastructure code out from the methods and apply it to the methods in some other way without having to rewrite it into the method each time. These units of functionality are what are termed aspects. This definition is clarified in the Xerox PARC paper; a property that must be implemented is:

A component, if it can be cleanly encapsulated in a generalized procedure (i.e. object, method, procedure, API). By cleanly, we mean well localized, and easily accessed and composed as necessary. Components tend to be units of the system's functional decomposition, such as image filters, bank accounts and GUI widgets.

An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. Examples of aspects include memory access patterns and synchronization of concurrent objects.

So an aspect is a unit that provides general functionality that cross-cuts the classes (components) in our system. For our banking example we may have a security aspect, a logging aspect and a transactional aspect, Figure 3.

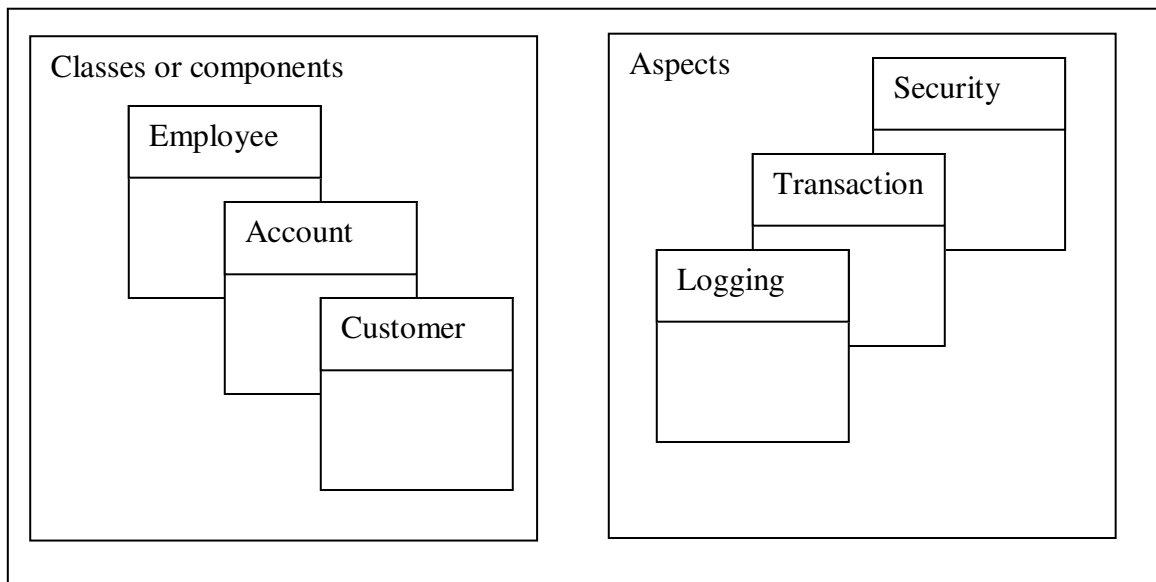


Figure 3

Weaving the code together

Ignoring how we will actually write these aspects (we will return to this later) we need some mechanism to intertwine the code in the aspects with the code in the classes. In aspect oriented terminology this is called weaving. Weaving can be done either at compile time, generating a woven code that gets executed, or at run time, intercepting the methods in the classes and weaving the aspect code through them as they run. There may be some performance issues that you need to consider with run time weaving, but the chances are high you wouldn't notice any issue and you might even get a performance benefit if you currently have very tangled code.

At first glance weaving may seem an easy thing to do; surely we simply need to intercept all method calls and fire off some code before and after the methods run? If we think a little more about it we realise that doesn't quite meet our requirements.

We need to be able to:

1. identify what aspects to weave into which methods
2. have aspect code woven into the code within a method, not just before and after it runs

As this is a non trivial task several Aspect Weavers are available to help with this process. The most well known is called AspectJ, no prizes for guessing it's targeted at the Java developer. AspectJ comes from Xerox PARC and is unsurprisingly one of the most mature tools for weaving. If you are lucky enough to be using Eclipse as your IDE then you can download a plug-in to integrate AspectJ with Eclipse, this is the slickest combination of aspect oriented development tools I've seen. Also for the Java guys is HyperJ from the research boys and girls at IBM. For the .NET developers out there, you guys will have to wait before we see any mature aspect oriented tools, there are a couple of academic projects that have yielded .NET weaving tools; Weave.NET and Loom.NET, of these the Loom.NET project is the more accessible. There are however technologies built into the .NET Framework that will allow you to get started on some aspect oriented development without the need for an additional aspect weaver, as we shall see. A great article by Dharma Shukla, Simon Fell and Chris Sells covers how to get the fundamentals of AOP rolling with .NET.

Once we have a tool that can do the weaving we still need to define the points at which the weaving should be done; where to interleave the aspect code with the class code. Here I need to introduce you to some more new terms in the aspect oriented world. The first is a **join point**; a join point is any place in the execution of the code that can be well-defined. Some simple examples might be a particular method call, any method call in a class or any usage of a resource. All of these are places you may wish to join or weave in some aspect code. The aspect code we will execute at a particular join point is called an **advice**, the advice code can execute either before, after or around the join point. Finally we need some mechanism to 'weave' our advice code into the join points, a **pointcut** defines how our advice code will interact with the join points, for example on execution of a method, on setting a field or where a particular expression evaluates to true. Going back to our banking example the join point may be the WithdrawFunds method in the Account class and the pointcut could be defined as any time the WithdrawFunds method is executed and the advice could be the code to check the access privileges prior to running the method.

Phew! Did you get all that? If not then read it again, if you are still have trouble then take a break, surf the web, Google for other related material and generally give it time to settle. While I don't believe AOP is as large a jump as OOP was it often takes some time to reach the point of aspect enlightenment, don't rush a good thing!

The Nitty-gritty

Now we should have a basic understanding of aspect oriented development let's take a look at some code. Working with the same simple example we started with, the bank account and weaving in some security aspects. The first set of example code is in Java written using Eclipse and AspectJ.

Listing 2 shows a simple account class, notice I have excluded any code that may be handled by an aspect. So this leaves a neat and clean piece of functionality in the methods of a class.

```
public class Account
{
    private double balance;

    ...

    public void WithdrawFunds(double amount )
    {
        balance -= amount;
    }
    .
    .
    .
}
```

Listing 2

```
public aspect Security {

    pointcut access(): within(Account) && execution(* Withdraw*(..));

    before(): access()
    {
        System.out.println(
            "Checking security for account access");
        boolean hasAccess = false;
        //hasAccess = CheckAccessRights();
        if (hasAccess)
        {

        }
        else
        {
            System.out.println("Access not permitted");
            //throw (new SecurityException)
        }
    }
}
```

Listing 3

Listing 3 contains some of the code from the Security aspect. Notice how the **pointcut** is defined as the execution of a method name that begins with Withdraw AND is within the Account class. The pointcut is named 'access' so that an advice can be weaved into it within this aspect. The advice in this example is code to weave **before** the pointcut, so before any method name starting with Withdraw in the Account class is executed. The AspectJ compiler does the rest of the work; it weaves the code together at compile time to produce the functionality we require.

Now for all the .NET developers we can try the same thing, but different. In the above example AspectJ allows us to do compile time weaving, the code that is generated is woven code. In .NET we are going to try some dynamic weaving with C# (it's not the only way to do it, see the LOOM.NET static aspect weaver, Google LOOM .NET). As we can see, in Listing 4, we have an Account class similar to the one we examined earlier in Java. Notice the SecurityAttribute that has been added and the class is derived from ContextBoundObject. We will find out more about the SecurityAttribute shortly. The ContextBoundObject base class allows our Account class to 'weave' with a class derived from ContextAttribute, and guess what the SecurityAttribute is derived from?

```
[SecurityAttribute()]
public class Account : ContextBoundObject
{
    private double balance;

    public Account()
    {
        balance = 0.0;
    }

    public void WithdrawFunds(double amount )
    {
        balance -= amount;
    }
}
```

Listing 4

```
[AttributeUsage(AttributeTargets.Class)]
public class SecurityAttribute : ContextAttribute
{
    public SecurityAttribute() : base("BaseSecurity") {}

    public override void GetPropertiesForNewContext
        (IConstructionCallMessage msg)
    {
        msg.ContextProperties.Add(new SecurityProperty());
    }
}
```

Listing 5

As Listing 5 illustrates the SecurityAttribute class is derived from the ContextAttribute class, but there is now also mention of a SecurityProperty class that needs to be implemented. This SecurityProperty class (Listing 6) gets instantiated when a new context is created, which equates in this example to each time an Account class is created. The SecurityProperty class supports an interface called IContributeObjectSink, it is this that does the real weaving work, the GetObjectSink method that the SecurityProperty exposes creates a new SecurityAspect object which supports the IMessageSink interface. In this way all method calls can be trapped in the SecurityAspect and handled as we desire.

```

public class SecurityProperty : IContextProperty, IContributeObjectSink
{
    public IMessageSink GetObjectSink(MarshalByRefObject o,
        IMessageSink next)
    {
        return new SecurityAspect(next);
    }

    public bool IsNewContextOK( Context ctx )
    {
        return true ;
    }

    public void Freeze(Context newContext){}

    public string Name
    {
        get
        {
            return "SecurityProperty";
        }
    }
}

```

Listing 6

```

class SecurityAspect : IMessageSink
{
    ...

    public IMessage SyncProcessMessage(IMessage msg)
    {
        if ((msg is IMessageMessage))
        {
            IMessageMessage method = msg as IMessageMessage;
            if (method.MethodName.StartsWith("Withdraw"))
            {
                accessAdvice(msg);
            }
        }
        IMessage returnMsg = _next.SyncProcessMessage(msg);
        return returnMsg;
    }

    private void accessAdvice(IMessage msg)
    {
        Trace("Checking security for account access");

        bool hasAccess = false;
        //hasAccess = CheckAccessRights();
        if (!hasAccess)
        {
            Trace("Access not permitted");
            //throw (new SecurityException)
        }
    }
}

```

```
    ...  
}
```

Listing 7

The SecurityAspect class shown in part (the important part!) in Listing 7 demonstrates how the SyncProcessMessage method can use reflection technologies to determine if a class method beginning with Withdraw is being executed and then calling the accessAdvice method.

As you can see having an aspect weaver certainly helps and the AspectJ toolkit coupled with an IDE such as Eclipse makes life a whole lot easier for the aspect oriented developer. I know that work is underway to build more complete aspect weavers for the .NET environment and the way that metadata is handled in .NET should facilitate these projects.

The Dark Side

There is a dark side to this whole wonderful AOP solution that we have discovered so far, be warned, not all is as rosy as it may first appear. Firstly think about what happens if you want an aspect to maintain state? There are issues when the aspect gets called and with state management. The recommendation is don't maintain state in any advice code, so therefore anywhere in your aspect code.

Aspect code generally has not got any concept of causality or cause and effect. You need to be careful about one aspect conflicting with another aspect or causing effects that are unpredictable. Worse than this is the fact that the code being acting upon in the classes has no notion of the advice code that is woven in. A developer maintaining this code might be very surprised at the functionality being exposed by a function if they are not aware that any code weaving is happening every time they compile/run the program. I have seen this happen and it is an incredibly frustrating experience.

Final Advice

Excuse the pun, but I couldn't resist! AOP is a growing movement, like all new ideas in the software world it is taking its time to get going, but I think the pace of change is faster now than it was when OOP was first suggested (at the end of the 1960's) so expect to see some aspect oriented development in the workplace in the coming few years. I know some developers are already just doing it and using the ideas to solve problems right now. Learn AOP and make sure you have an understanding of when it might be useful, and then put it away in your toolbox. Just because you have a hammer it doesn't make everything a nail!

SIDEBAR

The FAQ

Q: Is AOP going to replace object oriented programming?

A: Certainly not! It complements the OO way of doing things and is not as big jump as OOP was from the functional or procedural programming models developers were working with before.

Q: Is AOP going to solve all our programming problems?

A: Again, certainly not! The aspect oriented approach is another tool for the developer to keep in his toolbox to use when required.

Q: Is it really worth learning about AOP?

A: Damn right it is! This may not be as big a leap as OOP but it represents another great way to solve certain problems, namely those crosscutting concerns. A developer that ignores AOP is likely to end up with more tangled code that is harder to maintain.

Q: Where can I find out more?

A: Point your browser to <http://aosd.net/> and do some experiments, play around with an Aspect Weaver and get something running. Nothing beats doing it.

English born, Dr. Neil has been travelling the world working with software companies. He loves Australia, where while enjoying the Sydney lifestyle he helps software development teams get more productive. Over the last couple of decades Neil has worked with a variety of companies including Synon, ComputaCenter, SLK, EDI, Citect, Microsoft and Rogue Wave. Neil has also been involved in the formation of several software start-ups. Neil brings his business and technical skills to the companies he works with to ensure he has happy customers.

Neil has been closely associated with leading edge technologies for the last few years. This started with adopting, using and then teaching C++ in the early 1990's. Then in 1995 getting to grips with the details of COM and travelling around Europe teaching and mentoring development teams. Since 1999 Neil has been involved with the agile development movement, especially eXtreme Programming, which he has been using to help teams get more productive. In the last 3 years Neil has been leading the push to .NET, by running several .NET projects and teaching .NET courses. The latest technology curves that Neil is helping developers with are Tablet PC and Smartphone development. Neil studied Software Architectures for Real Time Systems at University College London, for which he received a PhD.

Dr. Neil can be contacted on Neil@Roodyn.com