

Topics

- What is AOP
- AOP in the CLR
- How AOP works
- Existing AOP implementations
- Pros & Cons
- Q & A



What is AOP?

Aspect-oriented programming (AOP) is a programming paradigm that increases modularity by enabling improved separation of concerns.

Source: Wikipedia



```
void Transfer(Account fromAccount, Account toAccount,
              int amount)
{
    if (fromAccount.Balance < amount)
    {
        throw new InsufficientFundsException();
    }

    fromAccount.Withdraw(amount);
    toAccount.Deposit(amount);
}
```



```
void Transfer(Account fromAccount, Account toAccount, int amount)
{
    if (!CurrentUser().CanPerform(OP_TRANSFER))
    {
        throw new SecurityException();
    }

    if (amount < 0)
    {
        throw new ArgumentException();
    }

    Transaction tx = database.NewTransaction();
    try
    {
        if (fromAccount.Balance < amount)
        {
            throw new InsufficientFundsException();
        }
        fromAccount.Withdraw(amount);
        toAccount.Deposit(amount);

        tx.Commit();
        SystemLog.LogOperation(OP_TRANSFER, fromAccount, toAccount, amount);
    }
    catch (Exception e)
    {
        tx.Rollback();
        throw e;
    }
}
```



```
[SecurityRequirement(OP_TRANSFER)]
[Transaction]
[AuditLog(OP_TRANSFER)]
void Transfer(Account fromAccount, Account toAccount,
              [NonNegative] int amount)
{
    if (fromAccount.Balance < amount)
    {
        throw new InsufficientFundsException();
    }

    fromAccount.Withdraw(amount);
    toAccount.Deposit(amount);
}
```



Terminology

- Join Points
 - Locations in a program where behaviour can be added
- Advices
 - The operations permitted at join points (before, after, instead etc.)
- Point cuts
 - The specific join points that are being used
- Weaving
 - Combining the behaviour code with the mainline code



AOP in the CLR

- Attribute driven
- Limited
- But extensible via `ContextBoundObject` / `MarshalByRefObject`



Demo

- Synchronization
 - MethodImplAttribute



Very simple demo showing theMethodImpl attribute. This is detected by the CLR at runtime, which ensures that the method is only accessed by one thread at a time. Under the covers, it is effectively doing a lock(this). For static methods, it is doing a lock(type)

Demo

- CLR Security
 - `PrincipalPermissionAttribute` & `CodeAccessSecurityPermissionAttribute`



The `PrincipalPermission` attribute causes the CLR to check the `Thread.CurrentPrincipal` object in accordance with the requirements specified on the attribute. `PrincipalPermission` is a subclass of `CodeAccessSecurityPermission`, which you can override if you want to do something specific to your environment (i.e., some permission check that is not related to `Thread.CurrentPrincipal`)

AOP in the CLR

- There are some others, such as
 - Transaction (System.EnterpriseServices)
 - JIT Activation (System.EnterpriseServices)
 - STAThreadAttribute / MTAThreadAttribute
- But, luckily, it is extensible



Demo

- Adding new behaviour through ContextBoundObject



Two requirements:

- Object derives from ContextBoundObject
- Attribute derives from ContextAttribute

Attribute the provide ContextProperty subclass, which in turn provides a MessageSink. MessageSink gets called during method invocation process

Demo

- Cleanup the ContextBoundObject project
 - Can now write our extensions as simple attributes
 - And apply the attributes to individual methods
 - Could easily change the demo to use something other than attributes (XML config, DSL etc)
 - Oh, and we run a performance test, just out of interest...



ClassUnderTest derives from AttributeBasedAOP which derives from AOPEntabledType which derives from ContextBoundObject

AOPEntabledType has an AOPAttribute, which is a ContextAttribute. The presence of this causes the AOPAspect to be hooked into the execution pipeline. The AOPAspect, in turn, makes “PreCall”, “Call” & “PostCall” calls down to the AOPEntabledType that is being called.

These 3 methods are handled by AOPEntabledType & its subclass, AttributeBasedAOP, which is where the attribute processing takes place

Demo

- ContextBoundObject not so great
 - Leaves the “client” code very clean, which is good
 - But forces out inheritance hierarchy
 - And performs badly
- How else can we do call interception?



ClassUnderTest derives from AttributeBasedAOP which derives from AOPEntabledType which derives from ContextBoundObject

AOPEntabledType has an AOPAttribute, which is a ContextAttribute. The presence of this causes the AOPAspect to be hooked into the execution pipeline. The AOPAspect, in turn, makes “PreCall”, “Call” & “PostCall” calls down to the AOPEntabledType that is being called.

These 3 methods are handled by AOPEntabledType & its subclass, AttributeBasedAOP, which is where the attribute processing takes place

Demo

- AOP through Subclassing
 - Let's try virtual methods...



Create a proxy class which derives from the ClassUnderTest and overrides it's virtual methods. Within each override, do any AOP processing required.

Demo

- AOP through interface decorating
 - Same idea as subclassing, but using interface-based proxies instead



Demo

- Subclassing with runtime IL generation
 - Subclassing / interface approach was fine
 - But writing the proxy class wasn't
 - Let's generate it...



A little complex because we used a lambda expression which lifted the parameters. Should make it clear how that works, though 😊

Demo

- Introduction to Castle Windsor Dynamic Proxy
 - DP does the heavy-lifting of the IL generation for us
 - **Much** easier for most requirements



Demo

- Using DP to create a simple AOP Runtime
 - A simple fluent interface, and supporting code, to implement a basic AOP implementation



Demo

- Adding Interception via IoC containers
 - Using the Windsor Container



Demo

- AOP in Windsor Container – Facilities
 - Simple way of getting hooks into the Windsor Container



Demo

- AOP with StructureMap
 - Combining the DP AOP with StructureMap



Demo

- AOP with Spring.NET
 - Example of AOP using something other than attributes
 - In this case, XML configuration



Demo

- AOP with PostSharp
 - AOP implemented using a post-compilation hook
 - Provides significant advantages
 - Performance
 - More Join Points
 - new()
 - field get / set



Demo

- Code Contracts
 - A bit more real world AOP than the simple Trace example
 - Although would need a **lot** of additional attributes to make it useable!



Demo

- Mixins
 - Not really AOP, but most AOP frameworks support it since the implementation is so close



Pros & Cons of AOP

- Pros
 - Remove cross-cutting concerns from core logic
 - Can add transparency
- Cons
 - Errors in AOP have a large impact
 - Non-obvious performance implications
 - Can remove transparency
 - Possible security risks?



Q&A



Contact Details & Useful Links

- sstrong@imeta.co.uk
- <http://blogs.imeta.co.uk/sstrong>
- <http://twitter.com/srstrong>
- http://en.wikipedia.org/wiki/Aspect-oriented_programming
- <http://www.castleproject.org/>
- <http://www.springframework.net/>
- <http://structuremap.sourceforge.net/Default.htm>
- <http://www.postsharp.org/>
- <http://en.wikipedia.org/wiki/Mixin>
- <http://kozmic.pl/archive/2008/12/16/castle-dynamicproxy-tutorial-part-i-introduction.aspx>

