

# It's A User Group Thing

- As an INETA member, your User Group gets:
  - Visits from Speaker Bureau members at no charge
  - INETA even pays for the pizza
  - Discounts on software, conferences, subscriptions and more
  - Newsletter, content repository and forums of use to User Group members
  
- INETA needs volunteers
  - Benefit your User Group and yourself by getting involved
  
- INETA Web Site: [www.ineta.org](http://www.ineta.org)



# Aspect Oriented Programming in .NET



Tom Barnaby  
Intertech Training



- What is Aspect Oriented Programming?
  - The failings of OOD/OOP
  - Separating concerns and scattering
  - Survey of AOP lingo
  - A look at AspectJ
  
- AOP Implementations
  - Dynamic weaving versus static weaving
  - Language extensions versus XML versus metadata
  - Interception versus AOP
  - Survey of .NET AOP solutions
  
- Implementing AOP in .NET
  - Attribute review
  - Context review
  - A custom AOP framework demo



# A Short Programming History

- In the beginning there was 0 and 1 ...

Paradigm	Languages
Imperative	Assembly, BASIC
Procedural	C, PASCAL, FORTRAN
Object Oriented	Simula, C++
Managed OO	Smalltalk, Java, C#
<b>What's next?</b>	<b>New language required?</b>



## ■ Motivations ...

Paradigm Shift	Motivation
Imperative to Procedural	Must break down app into small digestible pieces
Procedural to OOP	Minimize global access, object metaphor models real world.
OOP to Managed OOP	Memory leaks, thrashing the heap
Managed OOP to ??	What's wrong with OOD/OOP?



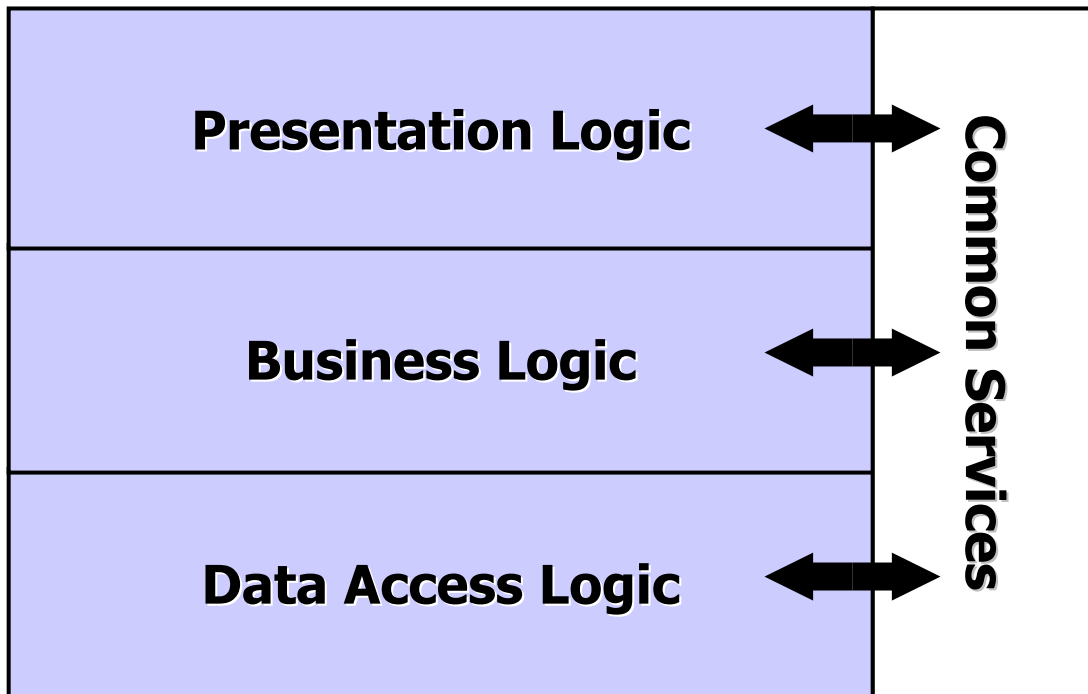
## ■ What's wrong with OOP?

OOP Problem	Solution
Relational data / OO impedance mismatch	O/R mapping tools, OO Databases
Distributed OO suffers from performance, scalability and coupling issues	Service Oriented Architecture (SOA)
Difficult to model business processes as objects	Service Oriented Architecture (SOA)
<b>Difficult to modularize system-wide requirements</b>	<b>Aspect Oriented Programming (AOP)</b>



# AOP Motivations

- Consider standard n-tier application.
  - N-tier apps usually have many “cross-cutting” concerns, that is, functionality required by all layers of the application.
  - This functionality is typically encapsulated within a services layer.



## **Common Services:**

Event Logging

User tracking

Exception Handling



## ■ The canonical AOP example: Logging

- Requirement: *Each domain class method will log the start and end of the method.*

```
public class CustomerService : BusinessObject
{
    public Customer GetCustomerByEmail(string email)
    {
        // Log method start
        Logger logger = new Logger("mylogfile.txt");
        logger.WriteMethodStart("GetCustomerByEmail");

        // Go get the customer
        Customer cust = FetchTheCustomerFromDB(email);

        // Log method end
        logger.WriteMethodEnd("GetCustomerByEmail");

        return cust;
    }
}
```

These code blocks deal with the logging *aspect* of the method and are not germane to retrieving a customer by email.

- The logging code is *tangled* within the domain logic
  - It, therefore, obscures the essence of the domain logic
  
- The logging code is *scattered* throughout the application
  - In a large application, how do you ensure the logging requirement is met in every method, today and in the future?
  - A new logging requirement may require a rewrite of each logging block of code.



# AOP Motivations – Tangling And Scattering

- For example: *Each domain class method should log the incoming and outgoing parameters.*
  - Step 1: Update the WriteMethodStart/End methods to accept additional parameters (easy).
  - Step 2: Update the logging logic of every domain method (not so easy, tedious, and error prone).

```
public Customer GetCustomerByEmail(string email)
{
    // Log method start
    Logger logger = new Logger("mylogfile.txt");
    logger.WriteMethodStart("GetCustomerByEmail", email);

    // Go get the customer
    Customer cust = FetchTheCustomerFromDB(email);

    // Log method end
    logger.WriteMethodEnd("GetCustomerByEmail", cust);

    return cust;
}
```



# AOP Motivations – Tangling and Scattering

- At its core, AOP addresses this "tangling and scattering" of code.
  - It provides a means to encapsulate crosscutting concerns into a single module (i.e. an aspect).
  - Since the logic is centralized, it is much easier to update.
  
- AOP compliments OOP
  - Unlike PP to OOP, it is not a complete paradigm shift.
  - Used correctly, AOP + OOP can create more modular solutions.
  - Modularity (of course) brings many benefits: more reuse, clearer code, easier maintenance, more flexibility.



- For sake of comparison, a **component**:
  - Is a system property that can be encapsulated in a single procedure or class.
  - Think GUI widgets, business objects, etc.
  
- In contrast, an **aspect**:
  - Is a system element that cannot be cleanly encapsulated in a single procedure or class.
  - Aspects tend to include system elements that affect the performance, behavior, or semantics of the *entire* application.
  - Think memory access, object synchronization, etc.



## ■ (#1) A **join point** is

- Where a component and aspect intersect. For example:

```
public Customer GetCustomerByEmail(string email)
{
    // Log method start
    Logger logger = new Logger("mylogfile.txt");
    logger.WriteMethodStart("GetCustomerByEmail");

    // ...
}
```

## ■ (#2) A **join point** is

- A well-defined point in the program execution that allows the insertion of aspect logic.
- For example: method calls, constructor calls, setting a field, etc.
- Future uses will refer to this definition.



## ■ A **point cut** is:

- A mechanism for finding specific join points and inserting **advice**.
- Specifying point cuts is a differentiating factor between various AOP implementations.

## ■ **Advice** is:

- Aspect logic that executes before, after, or instead of a join point.
- For example, the advice body could contain the logging logic that executes before and after a method call.



- **Weaving** refers to:
  - A process that evaluates the point cuts and injects advice at the specified join points.
  - For example, it would insert the logging logic before and after a method call.
  
- How/when weaving occurs varies between AOP implementations.
  - Compile/JIT time (AspectJ) - tends to be fast and type safe.
  - Dynamic/Runtime (JBoss and most .NET implementations) – slower, but more flexible.



## ■ AspectJ

- is currently the most popular AOP framework
- extends the Java language with AOP keywords (aspect, pointcut, etc.)
- uses static weaving
- is tested and proven with real world applications
- is NOT part of the standard Java language



# An Overview of AspectJ

## ■ Example: logging in AspectJ

```
aspect Logger {  
  
    pointcut log():  
        execution(public * BusinessObject+.*(..));  
  
    before() : log() {  
        System.out.println("  Logger: " +  
            thisJoinPoint.getSignature());  
    }  
  
}
```

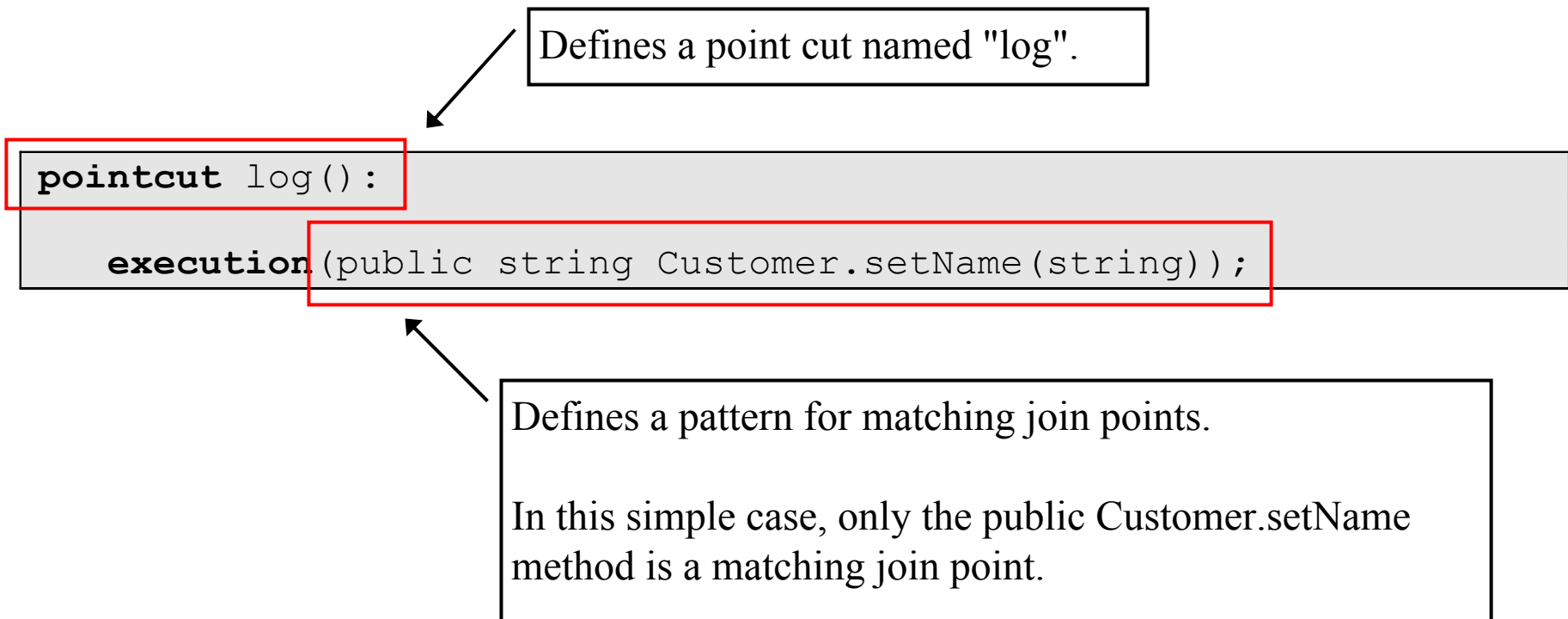
A pointcut definition

Advice to inject before execution  
of the join point.



# An Overview of AspectJ

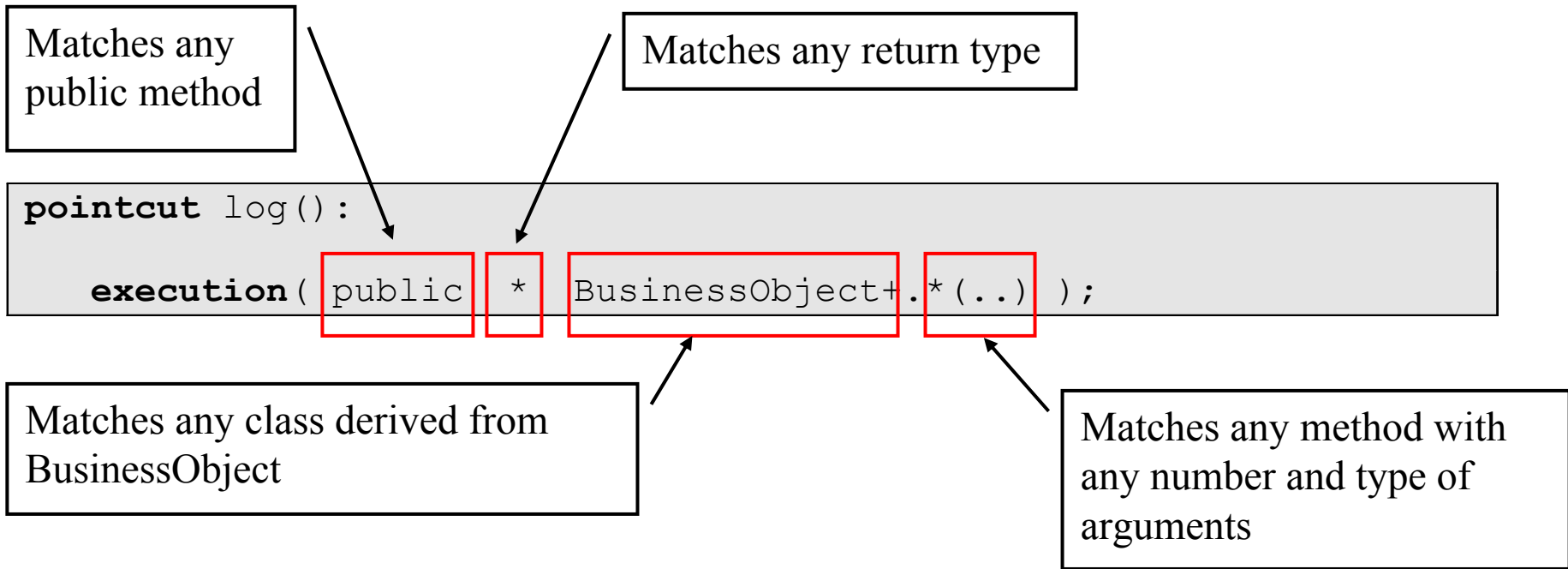
- The power of AspectJ lies in its flexible point cut model.





# An Overview of AspectJ

- Here's a more interesting point cut:



```
public class CustomerService : BusinessObject  
{  
  public Customer GetCustomerByEmail(string email)  
  { // ...  
  }  
}
```

See the match?



- AspectJ is an extremely robust AOP implementation, but has shortfalls.
  - Its point cut mechanism is powerful, but complex and easy to introduce unintended side effects.
  - Visual tools and IDE integration are required for mere mortals to understand the aspect flow and the point cut matches.
  - AspectJ employs static (compile-time) weaving that maximizes performance at the cost of flexibility.
  - It adds several new keywords and syntax to the Java language. Some argue that this is not desired nor necessary.



- Use dynamic (runtime) weaving (ala JBoss)
  - Provides greater flexibility.
  - For example, the logging aspect may be switched on or off by changing a config file and restarting the application.
  - Must use some type of interception mechanism that in most cases will be slower than static weaving.
  - However, a good runtime implementation should have minimal impact on performance.

- Define join points, point cuts, etc. in an XML file (ala JBoss).
  - This keeps the language "pure".
  - Most flexible option when used with runtime weaving.
- Define join points, point cuts, etc using metadata tags (some .NET implementations).
  - Keeps the language pure.
  - Makes applying aspects more declarative.
  - Some implementations eschew the regexp-like pattern matching complexities of AspectJ point cuts.
  - For best results, platform/language must have native support for metadata tags. Currently true for .NET, not for Java (but on its way).



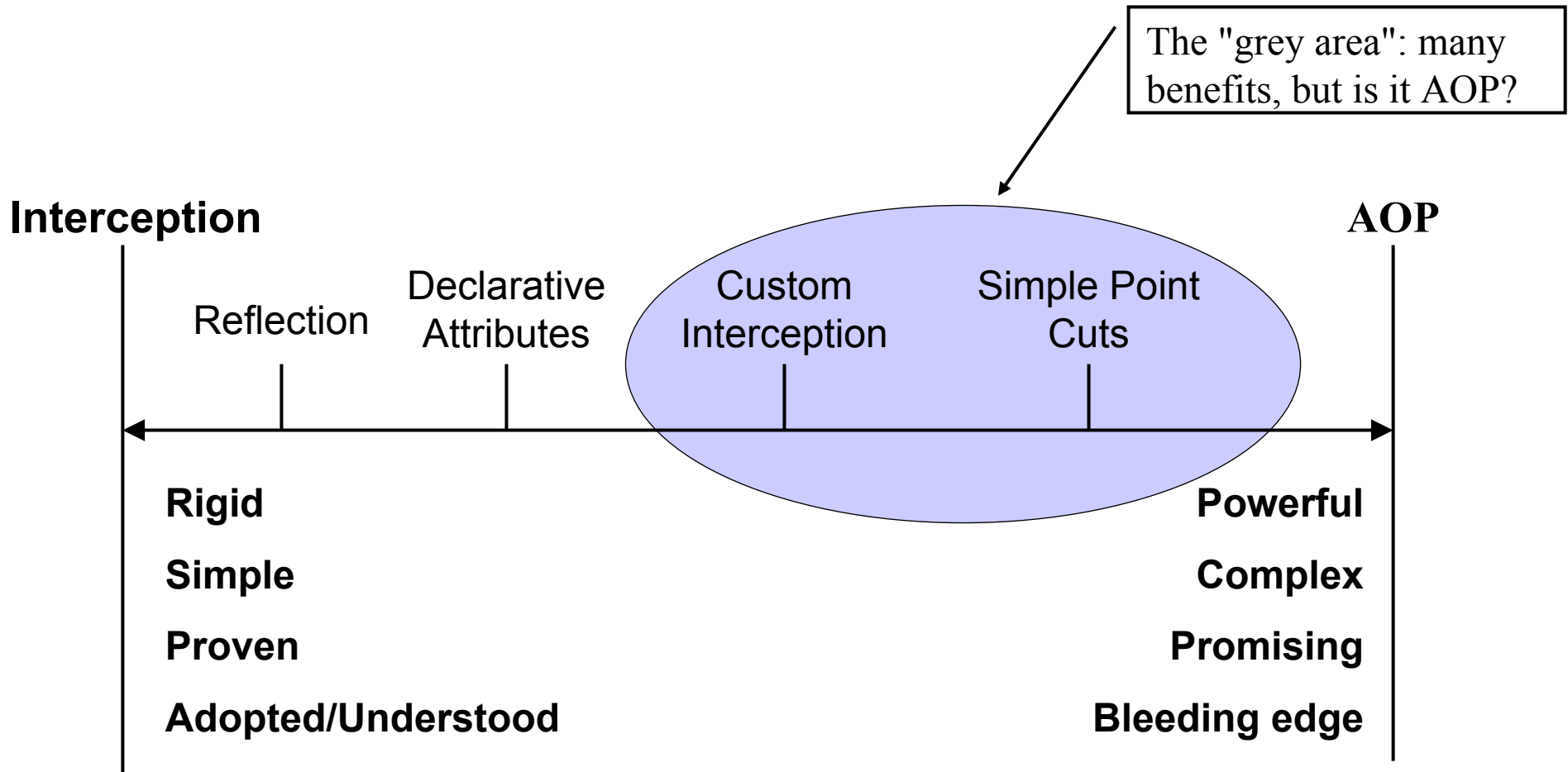
# Interception versus AOP

- Dynamic AOP tools weave using runtime interception techniques.
- Interception is a common design pattern modulo AOP.
  - Application servers: EJB, COM+
  - Web services: SOAP extensions, WS-Security, WS-Transactions, etc.
- Where does interception end and AOP begin?
  - More importantly (for AOP adoption) at what point along the continuum is ROI maximized?



# Interception versus AOP

- The interception to AOP spectrum





- .NET is a promising AOP platform. .NET
  - Supports reflection, dynamic component loading
  - Has a robust set of code generation APIs (CodeDOM)
  - Provides an interception framework
  - Supports custom metadata (termed *attributes*)
  - Supports many languages. So a non-language specific AOP implementation would work in C#, VB.NET, MC++, etc.



- Despite this AOP proclivity, .NET lags behind Java in AOP adoption
  - The .NET community is not as AOP aware
  - With AspectJ, AOP germinated in the Java world
  - AOP tools starting to emerge – but are fairly primitive compared to AspectJ
  
- Microsoft is focused on SOA, Web, tools.
  - *May* view attributes + flexible interception as "good enough".
  - This leaves .NET AOP R&D to the independents / academics



- Still many .NET AOP initiatives are in the works:
  - Aspect C#: Similar to AspectJ.
  - Weave .NET: Extends Aspect C#
  - Aspect .NET: Provides cross-language AOP using .NET attributes for to specify join points. Has some MS backing.
  - Loom .NET/Rapier-LOOM.NET: static and dynamic AOP tools
  
- And my own pet project: Rumpelstiltskin
  - Based on attributes and .NET context interception (big performance penalty)
  - Mind numbingly simple join points and advice
  - For educational use only



- Assemblies contain self-describing data termed *metadata*.
  - Includes assembly version, dependent assemblies, and detailed info about each type defined in the assembly.
  - .NET compilers generate metadata in a standard, predictable manner.
  - Used for many services: assembly binding, versioning, reflection, serialization.
  
- Attributes extend the assembly metadata.
  - Compilers handle attributes by recording additional metadata in the assembly.
  - The metadata is consumed at runtime and can change the runtime behavior of the application.



- Attribute syntax is straightforward.
  - You can apply attributes to any code item: class, structure, assembly, field, method, event, parameter, etc.
  - The attribute always precedes the code item it describes.
  - By convention, name attributes with an **Attribute** suffix.
  - You can omit the **Attribute** suffix when applying the attribute.
  
- .NET compilers, tools, and runtime look for specific attributes in the code and metadata.
  - When they detect an attribute, they respond accordingly.
  - Applying attributes ultimately affects their behavior.



# Attribute Basics

- Example: applying the **Obsolete** attribute:

```
// Compiler issues the specified error if code uses the
// HorseAndBuggy class.
[Obsolete("Use the Car class instead!", true)]
class HorseAndBuggy
{ // ...
}
```

```
static void Main(string[] args)
{
    HorseAndBuggy hb = new HorseAndBuggy();
}
'Obsolete.HorseAndBuggy' is obsolete: 'Use the Car class instead!'
```



# Attribute Basics

## ■ The resulting CIL:

`.custom` directive flags a custom attribute in the CIL code.

```
.class private auto ansi beforefieldinit HorseAndBuggy
  extends [mscorlib]System.Object
{
  .custom instance void [mscorlib]System.ObsoleteAttribute::.ctor(string, bool) =
  ( 01 00 1A 55 73 65 20 74 68 65 20 43 61 72 20 63 // ...Use the Car c
    6C 61 73 73 20 69 6E 73 74 65 61 64 21 01 00 00 ) // lass instead!...

  // snip ...
} // end of class HorseAndBuggy
```

Attribute data is serialized and stored here.



- The .NET framework uses attributes extensively
  - **Platform Invoke / COM Interop:** DllImport, StructLayout, FieldOffset, MarshalAs, ClassInterface, Guid
  - **Enterprise Services (COM+):** Transaction, JustInTimeActivation, ObjectPooling, InterfaceQueuing
  - **Serialization:** Serializable, NonSerialized, XmlElement, XmlAttribute, XmlRoot
  - **Security:** FileIOPermission, PrincipalPermission, RegistryPermission, etc.
  - **Web Services:** WebService, WebMethod, SoapHeader, XmlInclude, SoapExtension
  - **Visual Studio .NET:** Description, Browsible, DefaultValue, Category, DefaultValue



# Building Custom Attributes

- .NET allows you to create your own custom attributes.
  - If Microsoft can use attributes so effectively, so can you.
  - With custom attributes, you can add your own custom metadata to the assembly.
  - Like the built-in attributes, consuming code can read the metadata and respond as needed.



# Building Custom Attributes

- All attributes (including custom) derive from **System.Attribute**.
  - To create a custom attribute simply derive a class from the **System.Attribute** class.
  - Remember, follow convention and use the **Attribute** suffix.
  - Apply the **AttributeUsage** attribute and **AttributeTargets** enumeration to specify what code items the attribute can decorate.

```
enum AttributeTargets
{
    Assembly, Module, Class, Enum, Struct, Constructor, Method,
    Property, Field, Event, Interface, Parameter, Delegate, ReturnType,

    All = Assembly | Module | Class | Enum | Struct | Constructor |
        Method | Property | Field | Event | Interface | Parameter |
        Delegate | ReturnType
}
```



# Example: DeveloperInfo Attribute

- The **DeveloperInfo** attribute is intended to adorn types with information regarding the developer.
  
- The **DeveloperInfo** attribute requirements are:
  - Document the developer name (required)
  - Document the developer email (required)
  - Document the developer work phone (optional)
  - Document the developer mobile phone (optional)
  - Can decorate classes, structures, and interfaces.



# Example: DeveloperInfo Attribute

```
[AttributeUsage(AttributeTargets.Class |
    AttributeTargets.Struct | AttributeTargets.Interface)]
public sealed class DeveloperInfoAttribute : Attribute
{
    private string mName, mEmail, mWorkPhone, mMobilePhone;

    public DeveloperInfoAttribute(string name, string email)
    {
        mName = name;
        mEmail = email;
    }

    public string WorkPhone
    {
        get { return mWorkPhone; }
        set { mWorkPhone = value; }
    }
    public string MobilePhone
    {
        get { return mMobilePhone; }
        set { mMobilePhone = value; }
    }
    // Snipped Name and Email readonly properties ...
}
```

Constructor params are termed *positional parameters*.

Read/write properties are termed *named parameters*.



# Example: DeveloperInfo Attribute

- You specify positional parameters just like any constructor parameter.

```
[DeveloperInfo("Homer", "hs@atomic.com")]  
public class SomeClass  
{}
```

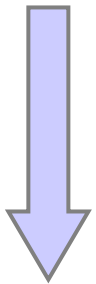
- You specify named parameters using the following syntax.  
Named parameters:
  - Are optional.
  - Must appear after all positional parameters.
  - Can appear only once.

```
[DeveloperInfo("Homer", "hs@atomic.com",  
    WorkPhone="555-5555", MobilePhone="777-7777")]  
public class AnotherClass  
{}
```



# Example: DeveloperInfo Attribute

```
[DeveloperInfo("Homer", "hs@atomic.com",  
              WorkPhone="555-5555", MobilePhone="777-7777")]  
public class AnotherClass  
{}
```



Compiles To

Serializes To

```
.class public auto ansi beforefieldinit AnotherClass  
  extends [mscorlib]System.Object  
{  
  .custom instance void DeveloperAttribute.DeveloperInfoAttribute::.ctor(  
    string, string) =  
    ( 01 00 05 48 6F 6D 65 72 0D 68 73 40 61 74 6F 6D // ...Homer.hs@atom  
      69 63 2E 63 6F 6D 02 00 54 0E 09 57 6F 72 6B 50 // ic.com..T..WorkP  
      68 6F 6E 65 08 35 35 35 2D 35 35 35 35 54 0E 0B // hone.555-5555T..  
      4D 6F 62 69 6C 65 50 68 6F 6E 65 08 37 37 37 2D // MobilePhone.777-  
      37 37 37 37 ) // 7777  
}  
// end of class AnotherClass
```



- Custom attributes become assembly metadata just like built-in attributes.
  - The .NET runtime, compiler, and tools discover built-in attributes and act upon the metadata.
  - For example, when it sees the **StructLayout** attribute, the runtime applies the specified memory layout to the structure.
  
- What about custom attributes?
  - The compilers, tools, and runtime ignore custom attributes.
  - *Therefore, custom attributes are useful only to the extent that consuming code discovers and acts upon them.*
  - You use .NET reflection to discover custom attributes.



# Reflecting for Attributes

- Many framework classes provide methods that retrieve custom attributes from the type metadata.
  - Most implement the **ICustomAttributeProvider** interface.

```
public interface ICustomAttributeProvider
{
    // Retrieves all the custom attributes
    object[] GetCustomAttributes(bool inherit);

    // Retrieves all the custom attributes of the given type
    object[] GetCustomAttributes(
        Type attributeType, // Type of attribute to retrieve
        bool inherit);      // If true, retrieves inherited
                           // attributes

    // Returns true if the given attribute type is applied
    bool IsDefined(Type attributeType, bool inherit);
}
```



# Reflecting for Attributes

Class	Notes
Assembly	Implements the <b>ICustomAttributeProvider</b> interface to retrieve custom attributes applied to the current assembly.
Attribute	Provides two static methods for custom attribute retrieval: <b>GetCustomAttribute</b> and <b>GetCustomAttributes</b> . Each are overloaded many times to support retrieval for any of the code items that accept attributes.
MemberInfo	Implements the <b>ICustomAttributeProvider</b> interface and is a base class for <b>MethodInfo</b> , <b>PropertyInfo</b> , <b>EventInfo</b> , <b>FieldInfo</b> , <b>ConstructorInfo</b> and <b>Type</b> . Each of these overrides the <b>GetCustomAttributes</b> method to retrieve the custom attributes on its respective code item.
Module	Implements the <b>ICustomAttributeProvider</b> interface to retrieve custom attributes applied to the current module.
ParameterInfo	Implements the <b>ICustomAttributeProvider</b> interface to retrieve custom attributes applied to the current parameter.



# Reflecting for Attributes

```
[DeveloperInfo("Homer", "hs@atomic.com")]  
[DeveloperInfo("Marge", "hs@atomic.com")]  
[Obsolete("Might as well use VB6 buddy")]  
public class SomeClass  
{}
```

```
static void Main(string[] args)  
{  
    Type t = typeof(SomeClass);  
    foreach(object attr in t.GetCustomAttributes(true))  
    {  
        // Display the attribute type  
        Console.WriteLine("{0}, {1}", attr,  
            attr.GetType().FullName);  
    }  
}
```

```
C:\WINDOWS\System32\cmd.exe - developerattribute.exe  
Name=Marge;Email=hs@atomic.com;Work Phone=;Mobile Phone=, DeveloperAttribute.DeveloperInfoAttribute  
System.ObsoleteAttribute, System.ObsoleteAttribute  
Name=Homer;Email=hs@atomic.com;Work Phone=;Mobile Phone=, DeveloperAttribute.DeveloperInfoAttribute
```



# Aspect-Oriented Programming (AOP)

- The goal of AOP is to minimize the “join points” between aspect logic and domain logic.
  - .NET combines attributes and context to implement aspect functionality.
  - Context provides the interception mechanism, attributes provide the means to encapsulate the aspect.
  - The end result allows you to write code like this:

```
[LogMethodCalls("mylogfile.txt")]
public class CustomerService : ContextBoundObject
{
    public Customer GetCustomerByEmail(string email)
    {
        // Go get the customer
        Customer cust = FetchTheCustomerFromDB(email);
        return cust;
    }
}
```



# Reviewing Context

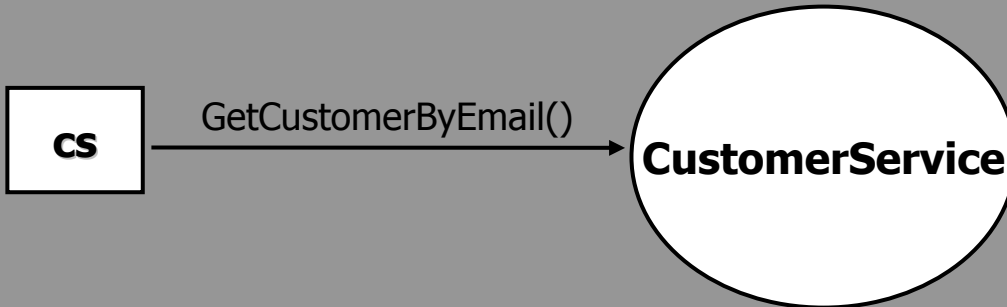
- The typical object reference is a direct reference.

```
public class CustomerService  
{ // ...  
}
```

```
static void Main(string[] args)  
{  
    CustomerService cs = new CustomerService();  
    cs.GetCustomerByEmail("hs@atomic.com");  
}
```

## Application Domain

### Context 0 (default)





# Reviewing Context

- Context provides an interception layer.

```
[Synchronization()]
```

```
public class CustomerService : ContextBoundObject  
{ // ...  
}
```

```
static void Main(string[] args)
```

```
{
```

```
    CustomerService cs = new CustomerService();  
    cs.GetCustomerByEmail("hs@atomic.com");  
}
```

## Application Domain

### Context 0 (default)

cs

GetCustomerByEmail()

**Proxy to  
CustomerService**

### Context 1 (Synched)

**CustomerService**

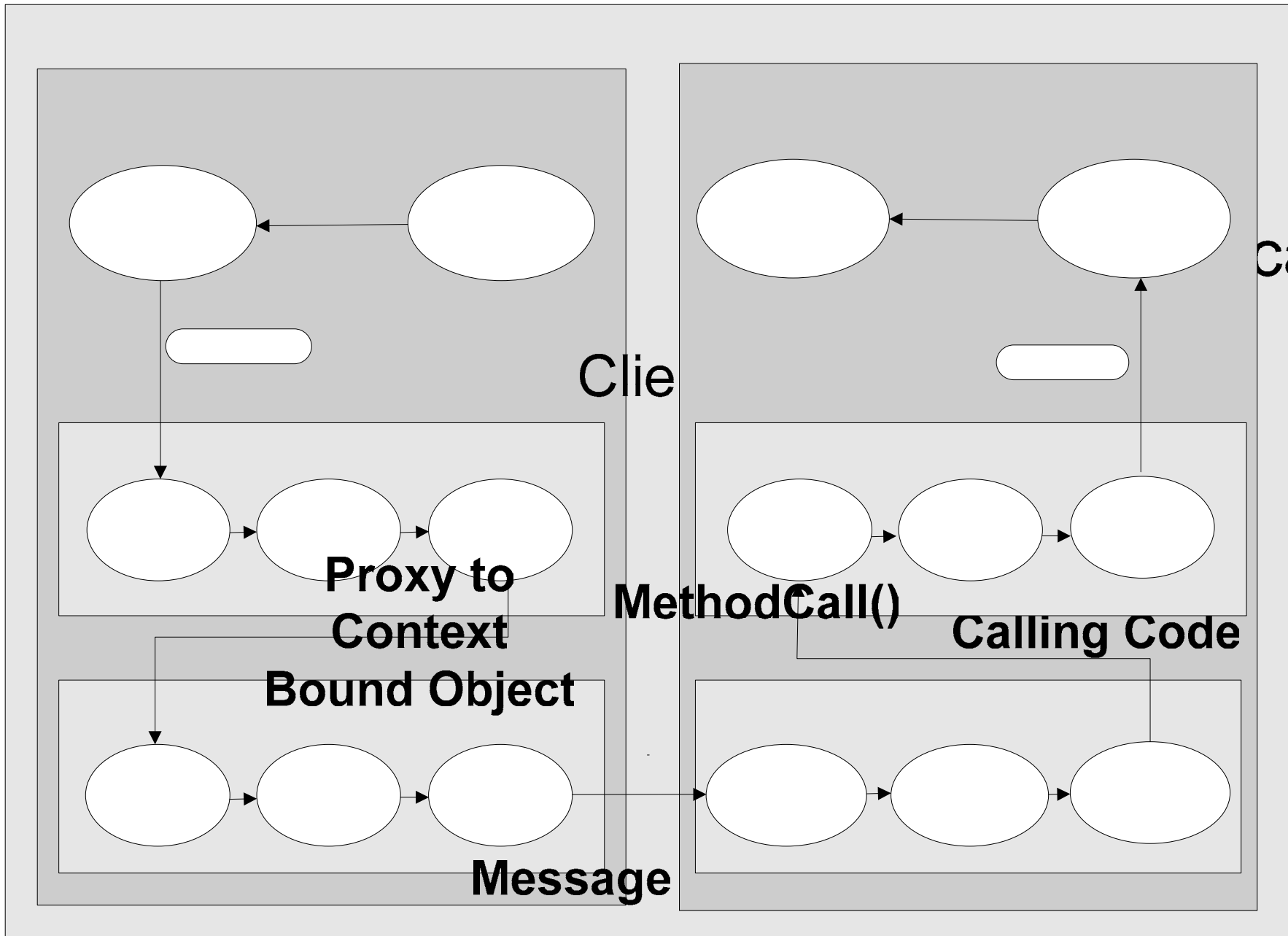
Aspect logic is inserted here



- When a method is called on a CBO:
  - The proxy converts the method call to an **IMessage** object.
  - The message is passed through a series of **IMessageSink** objects, which are linked to form a conceptual chain.
  - Each sink object can investigate the incoming message and perform required logic (such as logging).
  - A sink object can change the message.
  - When finished, the sink object passes the message to the next sink object in the chain.
  - Eventually the message arrives at the **StackBuilder** object, which applies the method to the CBO.
  - The return value (if any) passes through the same sink objects in reverse.



# The Context Sink Chain



ication



# The Context Sink Chains

Sink Chain	Executes In	Notes
Envoy	Client context	Intercepts calls from any client to the context bound object. Can be used to validate calls before doing any other expensive marshaling.
Client Context	Client context	Intercepts all calls leaving the context of a context bound object. You can use this chain to log all outgoing method calls.
Server Context	Server context	Intercepts all calls entering the context. You can use this chain to provide per context logging of incoming method calls.
Object	Server context	Intercepts all calls entering the context and invoked on a given object in the context. The runtime establishes one object sink chain for each object bound to the context. You can use this chain to provide per object logging of incoming method calls.



- How do you use context and attributes to implement AOP?
  - You develop a custom message sink that encapsulates your aspect logic.
  - You develop a custom *context attribute* that is a factory for your custom message sink.
  - You adorn CBOs with your context attribute.
  
- When the runtime creates a CBO:
  - It checks for any context attributes.
  - If found, it calls factory methods on the context attribute. Each factory method creates and returns a message sink.
  - The runtime inserts the message sink in the appropriate chain (envoy, client, server, or object).



# Developing a Message Sink

- Message sinks implement **IMessageSink**.

```
public interface IMessageSink
{
    // Implement to return the next message sink in the chain
    IMessageSink NextSink { get; }

    // Implement with the required aspect logic for a
    // synchronous method call.
    IMessage SyncProcessMessage(IMessage msg);

    // Implement with the required aspect logic for an
    // asynchronous method call.
    IMessageCtrl AsyncProcessMessage(IMessage msg,
        IMessageSink replySink);
}
```



## ■ Example: LogMethodCallsSink

```
public class LogMethodCallsSink : IMessageSink
{
    private IMessageSink mNextSink;
    private LogMethodCallsAttribute mLogInfo;

    public LogMethodCallsSink(IMessageSink nextSink,
        LogMethodCallsAttribute logInfo)
    {
        mNextSink = nextSink;
        mLogInfo = logInfo;
    }

    public IMessageSink NextSink
    {
        get { return mNextSink; }
    }
    // ...
}
```

The sink ctor should accept a reference to the next sink in the chain and store it in a field for later use.



# Developing a Message Sink

```
public class LogMethodCallsSink : IMessageSink
{
    public IMessage SyncProcessMessage(IMessage msg)
    {
        PreProcess(msg);
        IMessage returnMsg = mNextSink.SyncProcessMessage(msg);
        PostProcess(returnMsg);
        return returnMsg;
    }

    private void PreProcess(IMessage msg)
    {
        IMethodCallMessage callMsg = (IMethodCallMessage)msg;
        Logger logger = new Logger(mLogInfo.FileName);
        logger.WriteMethodStart(callMsg.MethodName, callMsg.Args);
    }

    private void PostProcess(IMessage msg)
    {
        IMethodReturnMessage rtnMsg = (IMethodReturnMessage)msg;
        Logger logger = new Logger(mLogInfo.FileName);
        logger.WriteMethodEnd(rtnMsg.MethodName, rtnMsg.ReturnValue);
    }
}
```

Aspect logic



# Developing the Context Attribute

- You must implement a context attribute to insert the custom sink in the context sink chain.
  - The primary role of the context attribute is to add the appropriate *context properties* to the new context.
  - A context attribute implements **IContextAttribute**.

```
public interface IContextAttribute
{
    // The runtime calls this on each context attribute applied
    // to the CBO. If false, the runtime creates a new context and
    // calls GetPropertiesForNewContext to establish the new
    // context's properties.
    bool IsContextOK(Context ctx, IConstructionCallMessage msg);

    // The context attribute is expected to use the provided
    // construction call message to add context properties to
    // the new context.
    void GetPropertiesForNewContext(IConstructionCallMessage msg);
}
```



# Developing the Context Property

- A context property defines the behavior of the new context.
  - It creates the message sinks which the runtime inserts in the context sink chain.
  - A context property implements **IContextProperty**.

```
public interface IContextProperty
{
    // Returns the properties name
    string Name { get; }

    // Once this is called, additional context properties cannot
    // be added to the context.
    void Freeze(Context newContext);

    // Called by the runtime to verify that the new context
    // satisfies the context property. For example, the context
    // property may check for conflicting context properties in
    // the context.
    bool IsNewContextOK(Context newCtx);
}
```



# Developing the Context Property

- Context properties may also implement one or more of the following interfaces.
  - Each interface provides a single factory method to return a specific type of message sink.
  - The runtime calls these methods (if present) and inserts the returned sink in the appropriate chain.

Sink Factory Interface	Factory Method
IContributeEnvoySink	GetEnvoySink
IContributeClientContextSink	GetClientContextSink
IContributeServerContextSink	GetServerContextSink
IContributeObjectSink	GetObjectSink



# Deriving from ContextAttribute

- The **ContextAttribute** class makes life easier.
  - It provides default implementations for **IContextAttribute** and **IContextProperty**.
  - You can derive your custom context attribute from **ContextAttribute**.
  - You must implement the appropriate sink factory interface(s).

```
[AttributeUsage (AttributeTargets.Class)]  
public sealed class LogMethodCallsAttribute : ContextAttribute,  
    IContributeObjectSink  
{ // ...  
}
```

This attribute contributes message sinks to the server context's object sink chain.



# The LogMethodCalls Attribute

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class LogMethodCallsAttribute : ContextAttribute,
    IContributeObjectSink
{
    private string mFileName;

    public LogMethodCallsAttribute(string fileName)
        : base("LogMethodCalls") // Context property name is
        { // required to construct base class
        mFileName = fileName;
    }

    public string FileName
    {
        get { return mFileName; }
    }

    public IMessageSink GetObjectSink(MarshalByRefObject obj,
        IMessageSink nextSink)
    {
        return new LogMethodCallsSink(nextSink, this);
    }
}
```



# The LogMethodCalls Attribute

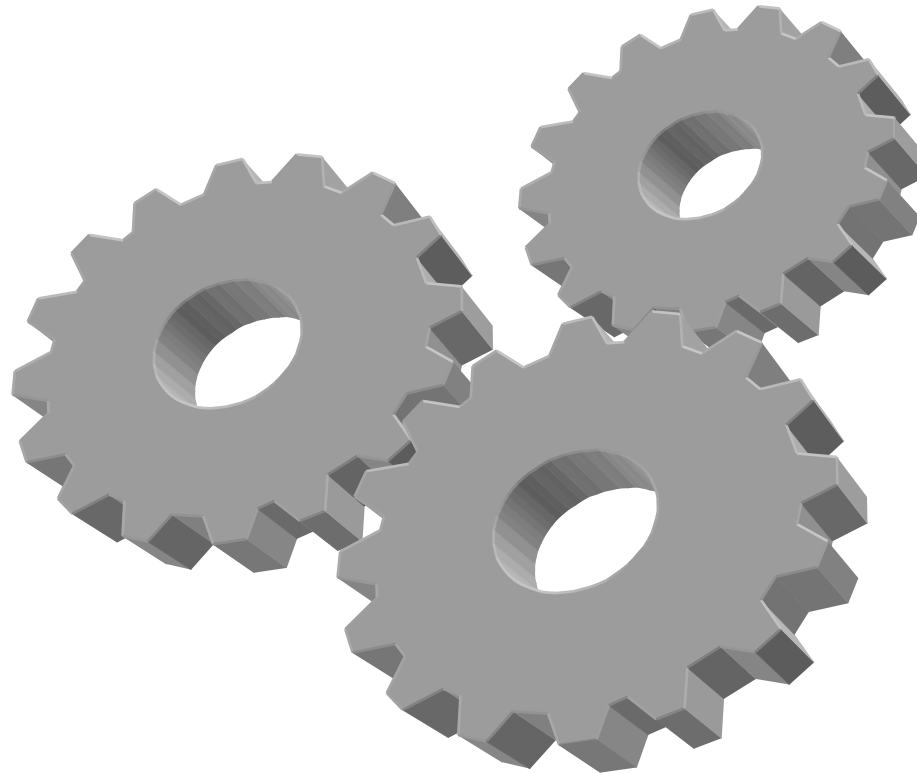
```
[LogMethodCalls("mylogfile.txt")]  
public class CustomerService : ContextBoundObject  
{  
    public Customer GetCustomerByEmail(string email)  
    {  
        // Go get the customer  
        Customer cust = FetchTheCustomerFromDB(email);  
        return cust;  
    }  
}
```

```
static void Main(string[] args)  
{  
    CustomerService cs = new CustomerService();  
    cs.GetCustomerByEmail("hs@atomic.com");  
}
```

```
mylogfile.txt - Notepad  
File Edit Format View Help  
7/31/2003 12:35:36 AM: Method GetCustomerByEmail starting  
7/31/2003 12:35:36 AM: Method GetCustomerByEmail ending
```



- The Rumpelstiltskin AOP Framework





## More Information

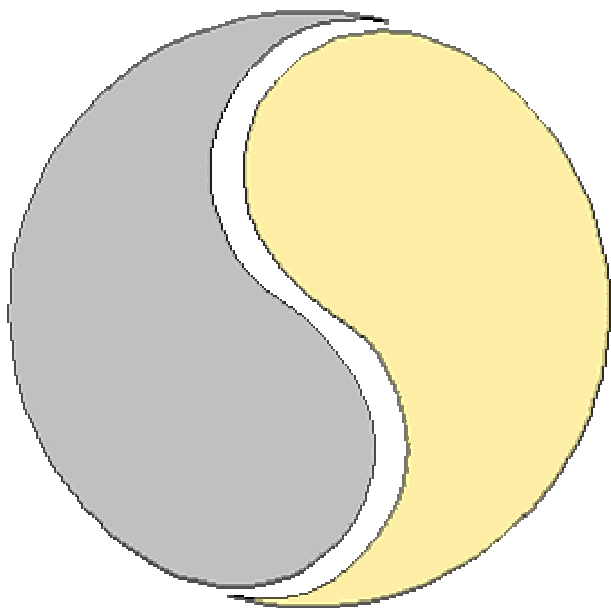
- AOP web site: <http://aosd.net>
- *I Want My AOP* - <http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html>
- *Interview with Gregor Kiczales:*  
<http://www.theserverside.com/talks/videos/GregorKiczalesText/interview.tss>
- AspectJ: <http://eclipse.org/aspectj/>



## Presentation Complete!

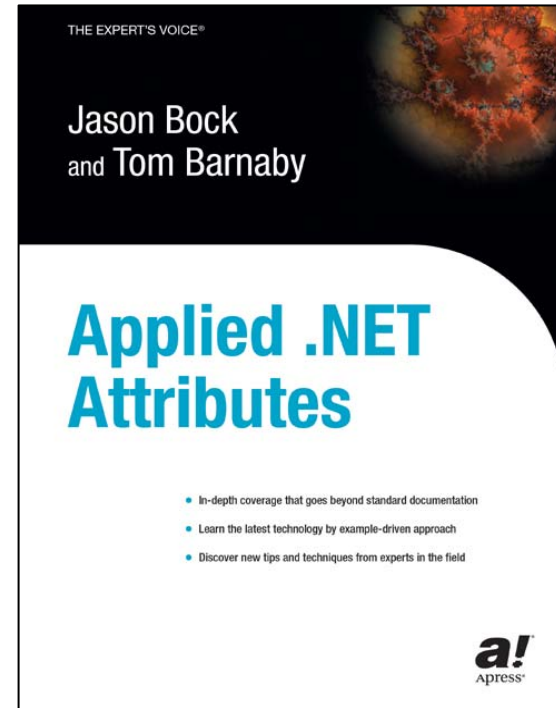
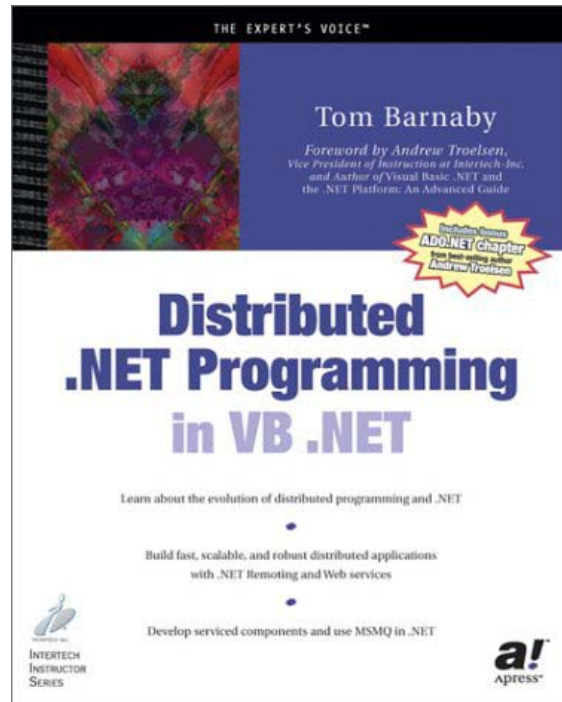
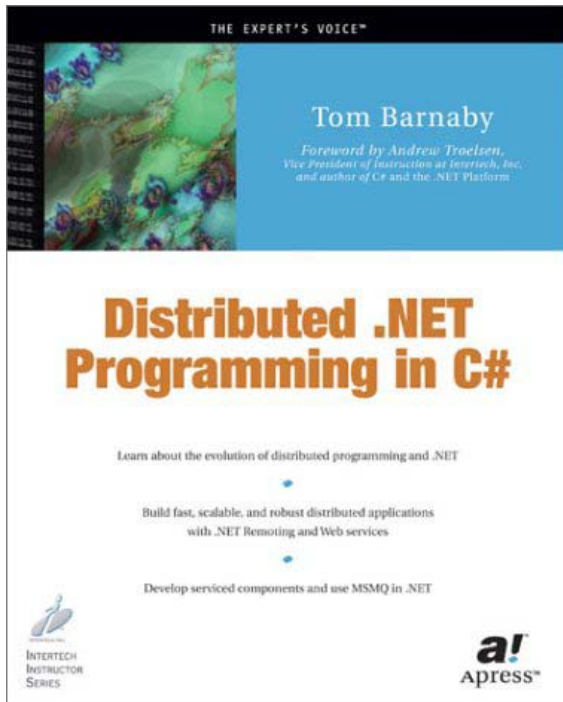
Thanks for Attending!

Check out our .NET classes  
at [www.intertech-inc.com](http://www.intertech-inc.com)



*Extreme .NET*  
*Complete VB.NET*  
*Complete C#*  
*Complete ASP.NET*  
*Expert Distributed .NET*

## Shameless Self-Promotion



***“Rating this book based solely on what I learned from reading it, this is one of my top picks. Add to that the fact that it is also an enjoyable read--you can't go wrong.”***

**- Steve Sharrock, [aspalliance.com](http://aspalliance.com)**