MSI

AOP in anderen Sprachen

Seyed-Nabiollah Mirhosseini-Zamani
Email: NabiZamani@aol.com

Winfried Mosler
Email: winfried.mosler@gmail.com

Übersicht

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- Einleitung zu AOP (5 min)
- Verschiedene Sprachen (25 min)
  - Common Lisp
  - JavaScript
  - Ruby / Python
- .Net (30 min)
  - Einleitung (ca. 3-5 min)
  - S2AOP.Net (5 min)
  - AspectDNG (mit Demo 20 min)
  - Eventuell .Spect (Demo)
- Ausblick & Erfahrungen (5 min)

http://en.wikipedia.org/wiki/Aspect-oriented_programming

Recherche: AOP in anderen Programmiersprachen

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- .NET Sprachen
- Ada
- ABAP
- Common Lisp
- JavaScript
- Ruby
- Python
- Perl
- Delphi/Pascal

- In den weniger verbreiteten Sprachen wenige und häufig nur wenig weit entwickelte AOP-Projekte

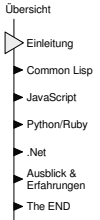
http://en.wikipedia.org/wiki/Aspect-oriented_programming

Ausgewählte “andere” Programmiersprachen

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Common Lisp
  - schon lange AOP-ähnliche Elemente
  - Gregor Kiczales, Lead Designer des CLOS ist mitverantwortlich für AspectJ
- JavaScript
  - einfaches Konzept
  - einfache und leicht nachvollziehbare Implementierung
- Ruby, Python, Perl
  - ähnliche Umsetzungen
- .NET Sprachen
  - Unterstützung in vielen Projekten
  - Ähnlichkeiten zu Java

“Dynamische” Programmiersprachen

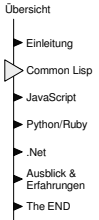


- “Dynamisch“ meint hier vor allem zur Laufzeit Klassen, Objekte, Methoden etc. verändern oder hinzufügen zu können
- Common Lisp, JavaScript, Ruby, Python, PHP, ...
  - haben typischerweise dynamische Typzuweisung
  - ermöglichen Metaprogramming
    - → mit Hilfe von Metaklassen
    - → oder/und Reflection
  - sind in der Regel interpretierte Skriptsprachen
- Aspekteigenschaften durch Metaprogramming
  - Introduction durch dynamisches Erweitern von Klassen (Mixin)
  - Advices umleiten von Methoden
- andere Denk-/Sichtweise:
  - technisch: AOP kann mit Metaprogramming umgesetzt werden

“Is AOP Metaprogramming?
Can Metaprogramming Do AOP?“

http://www.ddj.com/dept/architect/184415220

Common Lisp



- Lisp-Dialekt
  - 1981 entwickelt
  - standardisiert in ANSI X3.226-1994
  - Common Lisp Object System (CLOS)
    - → Method Combinations (before, after, around)
    - → Metaklassen, die das Verhalten jeder Klasse beschreiben und verändert werden können

http://www.lisp.org/

Sprach Definition:
http://www.lisp.org/HyperSpec/FrontMatter/index.html

Common Lisp

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Vorgehensweise für AOP-Introduction
  - Mixin-Ansatz
    - → Mixin, eine Klasse, die wie Interfaces Variablen und Methoden definiert
    - → Mixin als Spezialisierung der Hauptklasse
- Vorgehensweise für AOP-Advices
  - Method Combination
    - → Schlüsselwörter before, after und around für Methodenerweiterungen

http://c2.com/cgi/wiki?MixIn

Common Lisp

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Vorgehensweise für PointCuts
  - mit Metaklassen
    - → z.B. invokeMethod(..) der Metaklasse überschreiben, die bei jedem Methodenaufruf bei einem Objekt der Klasse aufgerufen wird
    - → Metaklassen können auch spezialisiert werden
- Versuch von PointCut Definitionen im AspectL Projekt

http://www.voelter.de/data/articles/aop/aop.html

http://common-lisp.net/project/closer/aspectl.html

Common Lisp - Beispiel

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Klassendefinitionen:

```
(defclass window ()
        ((width
            :accessor window-width
            :initarg :width
            :type integer
            :initform 200)))

(defclass border-mixin ()
        ((border-width
            :type integer
            :initarg :border-width
            :initform 1)))
```


http://c2.com/cgi/wiki?MixIn

Common Lisp - Beispiel

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Methodenerweiterung:

```
(defmethod window-width :around ((window border-mixin))
        (+ (call-next-method)
        (* 2 (slot-value window 'border-width))))
```

- neue window-Klasse mit Mixin:

```
(defclass my-window-with-border (border-mixin window)())
```

- Instanz:

```
(defvar *my-window* (make-instance 'my-window-with-border
                    :width 400
                    :border-width 10)
```

- window-width Methodenaufruf:

```
(window-width *my-window*)  ->  420
```


http://c2.com/cgi/wiki?MixIn

Common Lisp - Beispiel

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Beispiel zur besseren Demonstration in Pseudo-Javasyntax

```
class Shape {
   public void paint() {
      // paint shape
} }

class ColoredShape extends Shape {
   public before void paint() {
      // set brush color
} }

class Test {
   public void run() {
      Shape shape = new Shape();
      shape.paint();
      Shape shape2 = new ColoredShape();
      shape2.paint();
} }
```


http://www.voelter.de/data/articles/aop/aop.html

Common Lisp - Beispiel

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Beispiel zur besseren Demonstration in Pseudo-Javasyntax

```
public class LoggingClass extends StdMetaclass {
    public void invokeMethod( Object dest, String name, Object[]
   params ) {
       System.out.println( name+" called on "+dest+" with "+params);
       super.invokeMethod( dest, name, params );
    }
}

public class LogTest extends Object metaclass LoggingClass {
    public void doSomething() {
       // do something
    }
}
```


http://www.voelter.de/data/articles/aop/aop.html

Common Lisp

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Vorteile Metaprogramming
  - viele Möglichkeiten
- Nachtiele in Bezug auf AOP
  - Expertenwissen notwendig
  - kein Standard

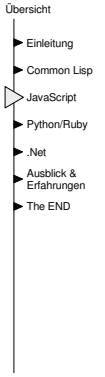
JavaScript

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- prototypbasiert
  - keine Klassen
  - Prototyp-Objekte
- in der Regel in Web-/HTML-Seiten eingebaut
- interpretierte Skriptsprache
- Vorhandene Objekte + Data Object Model (DOM)
- dynamisch
  - Objekte erweitern
  - Funktionen umbiegen
  - etc.

http://en.wikipedia.org/wiki/JavaScript

JavaScript - AOP Projekte

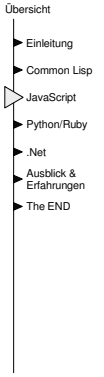


- ”AOP Fun with JavaScript“
  - ein simples Post in einem Forum 2003
  - publiziert ein Aspect Objekt, das Methoden-wrapping übernimmt
- “AOP and JavaScript“
  - verbessertes Aspect Objekt
  - kann auch Introduction

AOP Fun
http://www.jroller.com/page/deep?entry=aop_fun_with_javascript

AOP and JavaScript
http://www.dotvoid.com/view.php?id=43

Beispiel - ”AOP Fun“



- Konfiguration
  - <script src="Aspects.js"></script>
- Vorgehensweise
  - Implementieren wie gewöhnlich
  - Advices als normale Methoden definieren
  - Weben über die Methoden des „Aspect“-Objekts
    - → addBefore, addAfter, addAround
    - → jeweils mit der Advice Methode und den betroffenen JoinPoint

http://www.jroller.com/page/deep?entry=aop_fun_with_javascript

Beispiel - ”AOP Fun“

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

```
<script src="Aspects.js"></script>
<script>

   function makeGreeting(text) {
        return "Hello " + text + "!";
   }

   alert(makeGreeting("world"));

</script>


// Ergebnis ist eine Dialogbox mit dem Text:
/*
     Hello world!
*/
```


Beispiel - ”AOP Fun“

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

```
<script>

   function aopizeAdvice(args) {
        args[0] = "AOP " + args[0];
        return args;
   }
   function shoutAdvice(result) {
        return result.toUpperCase();
   }

   Aspects.addBefore(this, "makeGreeting", aopizeAdvice);
   alert(makeGreeting("world")); // Hello AOP world!

   Aspects.addAfter(this, "makeGreeting", shoutAdvice);
   alert(makeGreeting("world")); // HELLO AOP WORLD!

</script>
```

Beispiel – “AOP Fun“

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

```
Aspects = new Object();
Aspects.addBefore = function(obj, fname, before) {
   var oldFunc = obj[fname];
   obj[fname] = function() {
      return oldFunc.apply(this, before(arguments, oldFunc, this));
};};
Aspects.addAfter = function(obj, fname, after) {
   var oldFunc = obj[fname];
   obj[fname] = function() {
      return after(oldFunc.apply(this, arguments), arguments, oldFunc, this);
};};
Aspects.addAround = function(obj, fname, around) {
   var oldFunc = obj[fname];
   obj[fname] = function() {
      return around(arguments, oldFunc, this);
};};
```


Beispiel - ”AOP Fun“

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Vorteile:
  - Sehr einfache und kleine Methode
  - Nutzt vorhandene Sprachkonstrukte
- Nachteile:
  - Händisches Weben

JavaScript - Behaviour

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- "Behaviour Klasse"
  - Problem: JavaScript is häufig direkt in HTML-Seiten und Elemente eingebaut.
  - Mit Aspekten/Concerns/Behaviours sollen Funktionen dynamisch an HTML-Elemente gebunden werden.
  - Ziel ist die Trennung von Darstellung von Logik.
  - Kein “echtes” AOP

http://bennolan.com/behaviour/

http://lambda-the-ultimate.org/node/816

JavaScript - Behaviour

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Konfiguration:
  - <script type="text/javascript" src="behaviour.js"></script>
- Vorgehensweise:
  - Normale Erstellung der HMTL Seite
  - Definition der JavaScript-Funktionen in einem assoziativen Array
  - dessen IDs sind CSS-Selektoren
  - Weaving-Methode, die zur Laufzeit jederzeit erneut aufgerufen werden kann

http://bennolan.com/behaviour/

JavaScript - Behaviour

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Anstatt:

```
<ul id="example">
    <li>
        <a onclick="this.parentNode.removeChild(this)" href="#">
            Click me to delete me
        </a>
    </li>
</ul>
```


http://bennolan.com/behaviour/

JavaScript - Behaviour

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Schreibt man:

```
<ul id="example">
    <li>
        <a href="/someurl">Click me to delete me</a>
    </li>
</ul>

var myrules = {
    '#example li' : function(el){
        el.onclick = function(){
            this.parentNode.removeChild(this);

        }
    }
};
Behaviour.register(myrules);
```


http://bennolan.com/behaviour/

JavaScript - Behaviour

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Kein echtes AOP, aber eine ähnliche Idee
- Vorteile:
  - Trennung von Ansicht und Logik
  - Einfache kleine Klasse
  - Hohe Wiederverwendbarkeit
- Nachteile:
  - Overhead
- Kritik:
  - Bei dynamischer Webseitengenerierung ist es auch serverseitig möglich den Code zu trennen und JavaScript an den gewünschtgen Stellen einzubetten.

http://bennolan.com/behaviour/

Python / Ruby

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- Python
  - Projekte/Module, die ähnlich wie im JavaScript-Beispiel umgesetzt sind / benutzt werden
    - → Logilab Aspects
    - → Light-weight Approach to AOP in Python
- Ruby
  - Rühmt sich als eine der dynamischsten Sprachen
    - → starkes Mixin Konzept
  - AspectR: Module das ähnlich wie das JavaScript-Beispiel funktioniert
  - Ruby Change Request (RCR): Vorschlag Ruby um AOP Sprachelemente zu erweitern
    - → verglichen mit AspectJ auch sehr techniknah

Logilab

http://www.logilab.org/projects/aspects

Light-weight Approach to AOP in Python

http://www.cs.tut.fi/~ask/aspects/aspects.html

AspectR:

http://aspectr.sourceforge.net/

AspectOrientedRuby:

http://www.rubygarden.org/ruby?AspectOrientedRuby

Ruby Mixin Beispiel:

http://c2.com/cgi/wiki?MixIn

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

Folie 26

http://en.wikipedia.org/wiki/Aspect-oriented_programming

Übersicht

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

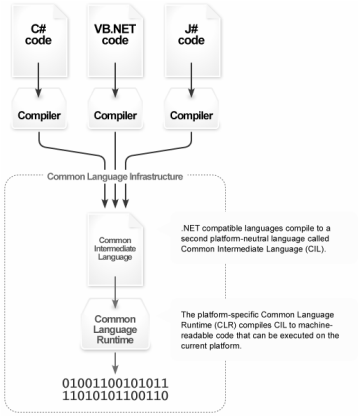
- Einleitung (5 min)
- Verschiedene Sprachen (25 min)
  - Common Lisp
  - JavaScript
  - Ruby / Python
- .Net (30 min)
  - Einleitung (ca. 3-5 min)
  - S2AOP.Net (5 min)
  - AspectDNG (mit Demo 20 min)
  - Eventuell .Spect (Demo)
- Ausblick & Erfahrungen (5 min)

Folie 27

Überblick .Net

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- Unterstützt verschiedene Sprachen, z.B. C# und VB.Net
- Anwendungen werden zu einem "Zwischen-Code" kompiliert (CIL)
- Quasi "plattformunabhängig"
- "Zwischen-Code" wird von einer virtuellen Laufzeitumgebung ausgeführt (CLR)
- Ist dem "Java-Konzept" also sehr ähnlich



Folie 28

Die .Net Plattform unterstützt verschiedene Sprachen wie z.B. C#, J#, VB.Net und weitere. Das heißt der Entwickler hat die Möglichkeit, sich eine Sprachen herauszusuchen, die er angemessen beherrscht. Die Quelltexte werden von Compilern (abhängig von der Wahl der Sprache(n)) in einen "Zwischen-Code" kompiliert (CIL). Das Ergebnis kann dann auf einer virtuellen Laufzeitumgebung (CLR) ausgeführt werden. An dieser Stelle spricht man von Plattformunabhängigkeit, auch wenn .Net derzeit lediglich für Microsoft-Betriebssysteme verfügbar ist (Projekte zur Portierung von .Net auf andere Betriebssysteme gibt es bereits).

Es ist erkennbar, dass das .Net Konzept sehr stark dem Java-Konzept ähnelt.

CIL = Common Intermediate Language (das Pendant zum Bytecode in Java), früher MSIL = Microsoft Intermediate Language

CLR = Common Language Runtime (das Pendant zur Java Virtual Machine)

Die Abbildung stammt vom http://en.wikipedia.org/wiki/.NET_Framework

AOP-Tools für .Net

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

Welche .Net Tools für AOP gibt es?

- Aspect# (Aspect Sharp)
- NKalore
- Spring.Net AOP
- NAop
- SetPoint
- PostSharp
- Encase
- S2AOP.Net
- AspectDNG
- .Spect (dotSpect)
- ...

Es gibt also sehr viele!

Folie 29

Es gibt viele konkurrierende (Open Source) Projekte, die AOP-Funktionalitäten unter .Net gewähren. Sie sind sich an vielen Stellen sehr ähnlich. Unterschiedlich sind vor allem die unterstützten .Net Sprachen, die AOP-Funktionalitäten und die Art und Weise der "Aspekt-Konfiguration". Häufig dient AspectJ als Vorbild, z.B. bei Aspect# (Quelle: http://wwwse.fhs-hagenberg.ac.at/se/berufspraktika/2002/se99047/contents/german/aop_net.html).

Dieser Vortrag beschäftigt sich lediglich mit

- AspectDNG
- S2AOP.Net
- .Spect (falls noch ausreichend Zeit bleibt)

AOP-Tools für .Net

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Welche .Net Tools für AOP gibt es?

- Aspect# (Aspect Sharp)
- NKalore
- Spring.Net AOP
- NAop
- SetPoint
- PostSharp
- Encase
- S2AOP.Net
- AspectDNG
- .Spect (dotSpect)
- ...

Es gibt also sehr viele!

Es gibt viele konkurrierende (Open Source) Projekte, die AOP-Funktionalitäten unter .Net gewähren. Sie sind sich an vielen Stellen sehr ähnlich. Unterschiedlich sind vor allem die unterstützten .Net Sprachen, die AOP-Funktionalitäten und die Art und Weise der "Aspekt-Konfiguration". Häufig dient AspectJ als Vorbild, z.B. bei Aspect# (Quelle: http://wwwse.fhs-hagenberg.ac.at/se/berufspraktika/2002/se99047/contents/german/aop_net.html).

Dieser Vortrag beschäftigt sich lediglich mit

- AspectDNG
- S2AOP.Net
- .Spect (falls noch ausreichend Zeit bleibt)

S2AOP.Net und C#

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- http://www.seasar.org/en/dotnet/
- Vorgehensweise
  - Anwendung wie gewöhnlich implementieren
  - Interceptor implementieren
    - → Kann als "Aspekt" angesehen werden
    - → Eine gewöhnliche Klasse, die das Interface IMethodInterceptor implementiert
    - → IMethodInterceptor ist das einzig zu implementierende Interface
  - Konfiguration
    - → über XML (Aspekte bzw. Advices werden Klassen, Methoden zugewiesen)
    - → programmatisch (hier nicht weiter behandelt)
  - Instanz der Klasse über die GetComponent-Methode des Containers beschaffen

- Statisches Weaving
- Konfiguration
  - Aspekte bzw. die Advices werden Klassen, Methoden usw. zugewiesen. Diese Zuweisung wird manchmal als "Konfiguration" bezeichnet.
- Auf http://www.seasar.org/en/dotnet/s2dotnet/aop.html findet man kurz beschrieben, wie vorgegangen werden muss

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

Anwendung wie gewöhnlich implementieren:

HelloWorld.cs

```
using System;

class HelloWorld {

    public void PrintHelloWorld(){
        Console.WriteLine("HelloWorld");
    }
    public static void Main(){
        HelloWorld hw = new HelloWorld();
        hw.PrintHelloWorld();
    }
}
```

Jeder Aufruf von PrintHelloWorld() soll protokolliert werden.

Ohne AOP würde man entweder im Rumpf der Methode PrintHelloWorld() oder vor und nach dem Aufruf der Methode PrintHelloWorld() entsprechende Anweisungen einfügen.

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

Interceptor implementieren:

MyInterceptor.cs

```
using S2.NET.Framework.Aop.Interceptors;

public class MyInterceptor : IMethodInterceptor {

    public object Invoke(IMethodInvocation invocation) {

        Console.WriteLine("Before PrintHelloWorld()"); // before invoking a method
        object ret = invocation.Proceed();
        Console.WriteLine("After PrintHelloWorld()"); // after invoking a method

        return ret;
    }
}
```

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Konfigurationsdatei erstellen:

MyAOPConfig.dicon

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container//EN"
   http://www.seasar.org/dtd/components.dtd">
<components>

   <component name="traceInterceptor“ class=„MyInterceptor"/>

   <component class=“HelloWorld">
      <aspect pointcut=“PrintHelloWorld">traceInterceptor</aspect>
   </component>

</components>
```


Über diese Config-Datei wird die Beziehung zwischen Aspekt (bzw. Interceptor) und Klasse hergestellt.

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Zusätzlich Anweisungen in Main()-Methode, z.B. in HelloWorld.cs

```
using S2.NET.Framework.Container;
using S2.NET.Framework.Container.Factory;
…
private const string PATH = “MyAOPConfig.dicon";
…
public void Main() {

      IS2Container container = S2ContainerFactory.Create(PATH);

      HelloWorld hw = (HelloWorld)container.GetComponent(typeof(HelloWorld));

      hw.PrintHelloWorld();

}
…
```


Ausgabe:

Before PrintHelloWorld()

HelloWorld

After PrintHelloWorld()

Nachteil: Die Entwickler müssen sich die Instanzen über GetComponent(…) beschaffen, ansonsten kommt es nicht zu der obigen Ausgabe.

S2AOP.Net und C#

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- **Vorteile**
  - AOP wird unterstützt
  - Alternativ auch ohne XML möglich (programmatisch)
  - Es muss nur ein Interface implementiert werden
  - Keine “neuen“ Sprachkonstrukte
  - Bestehende Klassen müssen nicht verändert werden
- **Nachteile**
  - Statisches Weaving
  - Unterstützt lediglich C#
  - Jeder Entwickler muss sich an Regeln halten (➔ Erzeugung von Instanzen)
  - Konfiguration über XML, hierbei zusätzliches Coding nötig
  - Programmatische Lösung schafft hier auch keine Vorteile
  - Dokumentation lässt sehr zu wünschen übrig (besonders für “nicht-Japaner“)
  - Aktuelle Version 1.0.0 beta2

AspectDNG und C#

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- http://www.dotnetguru.org/sarl/aspectdng/index.xml
- Vorgehensweise
  - Anwendung wie gewöhnlich implementieren (z.B. mit C#)
  - Aspekte implementieren
    - → Gewöhnliche .Net Klasse
    - → Eine Methode als Interceptor definieren (Rückgabetyp ist object, ein Parameter vom Typ JoinPoint)
    - → Ein .Net Attribut dieser Methode zuweisen (Achtung: sprachabhängig)
  - Kompilieren (exe oder dll)
  - Ergebnis mit AspectDNG “verweben“ (Weaving)

Von der AspectDNG Homepage entnommen:

1.1. What's the use of AspectDNG?

AspectDNG is a .NET aspect weaver, that's to say a tool that can "transplant" code into an existing assembly. This transplant is made after the standard .NET compilation, which means that both aspect and the so called "base" code (the one the transplant will be operated on) can be developped in any programming language that is compatible with .NET CLS. Another way to say that: AspectDNG works on assemblies (EXE or DLL) that may have been created out of C#, VB.NET, Eiffel.NET, Mananged C++...

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Kompilieren und Weaving

- Sample.cs wie gewöhnlich kompilieren
  - → C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc /out:HelloWorld.exe HelloWorld.cs
- Aspects.cs kompilieren
  - → C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc /out:myaspect.dll /r:aspectdng.exe MyAspect.cs
- Ergebnis
  - → Es stehen 2 Assemblies zur Verfügung (HelloWorld.exe und myaspect.dll)
  - → HelloWorld.exe enthält noch kein "Aspect-Coding"
  - → Es muß noch noch "verwoben" werden (➔ Weaver)
- Weaving der beiden Assemblies mit AspectDNG (Konsolenaufruf)
  - → aspectdng.exe HelloWorld.exe myaspect.dll
  - → Dabei wird zunächst HelloWorld.exe in HelloWorld.exe.backup gesichert
  - → Anschliessend ist das Ergebnis des Weavings in HelloWorld.exe verfügbar

Ohne die Compileroption "/r:aspectdng.exe" meldet der C#-Compiler, dass er die Attribute bzw. die Klassen der Attribute nicht kennt.

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Anwendung wie gewöhnlich implementieren:

HelloWorld.cs

```
using System;

class HelloWorld {

    public void PrintHelloWorld(){
        Console.WriteLine("HelloWorld");
    }
    public static void Main(){
        HelloWorld hw = new HelloWorld();
        hw.PrintHelloWorld();
    }
}
```

Jeder Aufruf von PrintHelloWorld() soll protokolliert werden.

Dieses Beispiel kennen wir ja schon.

AspectDNG und C#

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Interceptor implementieren:

MyAspect.cs

```
using DotNetGuru.AspectDNG.Joinpoints;
using System;

public class MyAspect{

    [AroundCall("* HelloWorld::PrintHelloWorld(*)")]
    public static object myMethodCallInterceptor(JoinPoint jp) {

        Console.WriteLine("Around Advice: before ==> "+jp);
        object result = jp.Proceed();
        Console.WriteLine("Around Advice: after  ==> "+jp);

        return result;
    }
    public static void Main(){  //do nothing }
}
```


•Dieses Beispiel kennen wir ja schon.

•Mögliche Attribute sind zum Beispiel:

- •[AroundCall("* HelloWorld::PrintHelloWorld(*)")]
- •[AroundCall("* HelloWorld::.ctor(*)")]
- •[AroundBody("* HelloWorld::*HelloWorld(*)")]
- •…

•Möglich ist auch das Hizufügen von mehreren Attributen zu einer Methode

```
•[AroundBody("* HelloWorld::*HelloWorld(*)")]
[AroundCall("* HelloWorld::.ctor(*)")]
public static object myMethodCallInterceptor(JoinPoint jp) {
        …
}
```

Ausgabe:

Around Advice: before ==> instance method HelloWorld::PrintHelloWorld()

HelloWorld

Around Advice: after ==> instance method HelloWorld::PrintHelloWorld()

AspectDNG

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- **Vorteile**
  - Multi-Language Weaver für .Net
  - Keine “neuen“ Sprachkonstrukte
  - Aspekte sind gewöhnliche Klassen
  - Attribute für Methoden sind einfach verständlich
  - Bestehende Klassen müssen nicht verändert werden
  - Kein zusätzlicher Quelltext nötig
  - “Weaving“ benötigt lediglich exe/dll, anstatt “Base“-Code
- **Nachteile**
  - Aktuelle Version 0.9.80
  - Derzeit noch Probleme mit VB.Net Aspekten
  - Schlechte Fehlermeldungen

AspectDNG

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

Demo

Von der AspectDNG Homepage entnommen:

1.3. How does AspectDNG work?

It sounds weird, but is a child's play. It's a three phase process:

1. First, the base assembly and the one containing aspects to be weaved are disassembled. What we mean by disassembled is that their binary representation (CIL or Common Intermediary Language, aka ECMA-335) is translated into a higher level language, easier to work on. This language is based on XML, we call it ***ILML***.
2. Second, the base assembly ILML reprensentation is transformed (thu XSLT) so that aspect ILML instructions are added in the right place (what we call transplant zones or pointcuts). Each XSLT transform creates a new ILML document, which means you can chain as many transforms as you want (of course, you'll wait longer)
3. Then, the last ILML document must be transformed back into a binary assembly (EXE or DLL): this process is symetrical to the disassembly process. Let's call it reassembly.

Ausblick & Erfahrungen

Übersicht
Einleitung
Common Lisp
JavaScript
Python/Ruby
.Net
Ausblick & Erfahrungen
The END

- .Net
  - Sehr viele Auswahlmöglichkeiten wenn es um AOP geht
  - Leider oft unterschiedliche Interpretationen von AOP
  - AOP-Projekte häufig sehr klein (2-3 Personen)
  - Häufig französische AOP-Projekte
  - AspectDNG in Zukunft integrierbar in VS .Net
- AOP allgemein
  - Was bedeuted AOP genau?
    - → was gehört dazu?
    - → was gehört nicht dazu?
  - Wer sorgt für Standards?
  - AspectJ gilt oft als Vorbild
  - AOP erschwert möglicherweise das Debugging
  - AOP nicht immer mit OO-Paradigma vereinbar

Ausblick & Erfahrungen

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- “dynamische“ Sprachen
  - AOP-Idee wenig verbreitet
  - Ermöglichung von AOP mit eigenen Sprachelementen
  - AOP-Projekte bieten nur “primitive“ Möglichkeiten
    - → Keine zusätzlichen AOP-Sprachelemente
- sonstiges
  - Java 1.5
    - → Reflection API stark erweitert
    - → unterstützt auch Attribute
  - AspectDNG basiert auf .Net Attribute, Reflexion API, XML/XSLT
  - Portierung von AspectDNG auf Java?

The END

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

Fragen?

Quellen

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- Wikipedia
  - http://en.wikipedia.org/wiki/Aspect-oriented_programming
    + weiterführende Links
- AOP und Metaprogramming
  - http://www.ddj.com/dept/architect/184415220
- Common Lisp
  - http://www.lisp.org/HyperSpec/FrontMatter/index.html
  - http://c2.com/cgi/wiki?MixIn
- AspectL
  - http://common-lisp.net/project/closer/aspectl.html
- JavaScript und “AOP”
  - http://www.jroller.com/page/deep?entry=aop_fun_with_javascript
  - http://www.dotvoid.com/view.php?id=43
  - http://bennolan.com/behaviour/
  - http://lambda-the-ultimate.org/node/816
  - http://calculist.blogspot.com/2005/07/aop-in-javascript.html

Quellen

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- Logilab
  - http://www.logilab.org/projects/aspects
- Light-weight Approach to AOP in Python
  - http://www.cs.tut.fi/~ask/aspects/aspects.html
- AspectR:
  - http://aspectr.sourceforge.net/
- AspectOrientedRuby:
  - http://www.rubygarden.org/ruby?AspectOrientedRuby
- Ruby Mixin Beispiel:
  - http://c2.com/cgi/wiki?MixIn

Quellen

Übersicht
- Einleitung
- Common Lisp
- JavaScript
- Python/Ruby
- .Net
- Ausblick & Erfahrungen
- The END

- Wikipedia
  - http://en.wikipedia.org/wiki/.NET_Framework
- AspectDNG
  - http://www.dotnetguru.org/sarl/aspectdng/index.xml
- S2AOP.Net
  - http://www.seasar.org/en/dotnet/s2dotnet/aop.html
- FH-Hagenberg
  - http://wwwse.fhs-hagenberg.ac.at/se/berufspraktika/2002/se99047/contents/german/aop_net.html
- .Spect
  - http://dotspect.tigris.org/
- Spring .Net AOP
  - http://www.springframework.net/doc/reference/html/aop.html