

## syngenio in der Fachpresse

Medium: Javamagazin

**Thema: Struts mit Hilfe von AspectJ, XSLT-Stylesheets und Ant anpassen**

Autor: Arno Schumacher

Ausgabe : 11. 2003

### **Aspekte verstreben**

Für das Web-Framework Struts ist bis heute eine Vielzahl kommerzieller und nichtkommerzieller Erweiterungen verfügbar, die sich leicht einbinden lassen. Mittlerweile findet man auch eine Reihe von Publikationen, die sich mit Struts befassen und die zugrunde liegenden Design-Konzepte beschreiben. Dies macht Struts zu einem idealen Ausgangspunkt für die Entwicklung von Web-Applikationen.



### **Kontakt & weitere Informationen:**

syngenio AG  
Ivonne Machnacz  
Andreas-Hermes-Straße 3  
D-53175 Bonn  
Fon +49 (0)2 28-6 20 95-100  
Fax +49 (0)2 28-6 20 95-150  
ivonne.machnacz@syngenio.de  
www.syngenio.de

## Struts mit Hilfe von AspectJ, XSLT-Stylesheets und Ant anpassen

von Arno Schumacher

# Aspekte verstreben

Für das Web-Framework Struts ist bis heute eine Vielzahl kommerzieller und nichtkommerzieller Erweiterungen verfügbar, die sich leicht einbinden lassen. Mittlerweile findet man auch eine Reihe von Publikationen, die sich mit Struts befassen und die zugrunde liegenden Design-Konzepte beschreiben. Dies macht Struts zu einem idealen Ausgangspunkt für die Entwicklung von Web-Applikationen.

Struts ist ein leichtgewichtiges Framework. In vielen Fällen können mit Struts nicht alle Anforderungen an ein JSP-Framework abgedeckt werden, weshalb Struts häufig noch erweitert wird: Beispielsweise um sicherzustellen, dass die auf Grundlage von Struts entstehenden Anwendungen den speziellen Sicherheitsanforderungen von Online-Transaktionslösungen genügen.

Die Architektur von Struts ist sehr modular. So lassen sich in der Regel Erweiterungen und Ergänzungen durch Bereitstellung eigener Implementierungen von Struts-Interfaces und durch die Anpassungen der Struts-Konfiguration leicht bewerkstelligen. Direkte Veränderungen am Struts-Kernel, also durch Modifikation an den Struts-Sourcen und die Erzeugung einer eigenen Struts-Distribution können daher oft vermieden werden. Somit kann der Übergang auf eine neue Struts-Version, wenn Apache eine neue Struts-Release freigibt, einfach und unkompliziert verlaufen. In einigen Fällen macht die Modifikation des Struts-Kernels dennoch Sinn:

- Es müssen weitere Third Party Libraries benutzt werden, die eigene Plugin-Implementierungen bereitstellen oder Struts-Klassen durch Ableitung erweitern und so die Verwendung selbst entwickelter Struts-Plugins limitieren oder unmöglich machen.

- Es wird eine IDE oder ein Wizard verwendet, der die Benutzung eigener Interface-Implementierungen nicht unterstützt.
- Erhöhte Sicherheitsaspekte bedingen eine Modifikation des Struts-Kernels.
- Zur Implementierung eines gewünschten Features bietet Struts keinen geeigneten Plugin-Mechanismus an.

Führt man selbst Änderungen am Kernel durch, so müssen diese Modifikationen ausreichend dokumentiert sein, um eine erneute Einpflege in Updates zu ermöglichen. Eine manuelle Einpflege ist jedoch in der Regel recht zeitaufwändig, umständlich und auch fehleranfällig.

Im Folgenden wird gezeigt, wie sich Modifikationen am Struts-Kernel so gestalten lassen, dass bei einem Übergang zu einem neuen Update die Kernel-Modifikationen weitgehend automatisiert nachgezogen werden können. Der Ansatz beruht auf der Verwendung von AspectJ als Java-

Präprozessor sowie Java-Compiler und XSLT-Scripte zur Modifikation von XML-Sourcen, wie etwa Taglib-Deskriptoren. Dabei wird die Build-Infrastruktur der Struts-Source-Distribution auch verwendet, um die modifizierte Struts-Version zu generieren. In einem vorgelagerten Schritt werden Ant *build.xml*-Dateien der Source-Distribution mit XSLT-Stylesheets so gepatcht, dass auch die Erweiterungen in den Build-Prozess mit einbezogen werden. In Abbildung 1 wird der Ablauf des Build-Prozesses in schematischer Form dargestellt.

Im Rahmen eines Kundenprojektes wurden so alle Erweiterungen an Struts durchgeführt, unter anderem:

- Request-Synchronisation: Eingehende HTTP-Requests modifizieren den Zustand der Web-Applikation. Im Falle einer Multiframe-Anwendung oder eines Doppel-Clicks des Anwenders könnten zwei oder mehr HTTP-Requests zur glei-

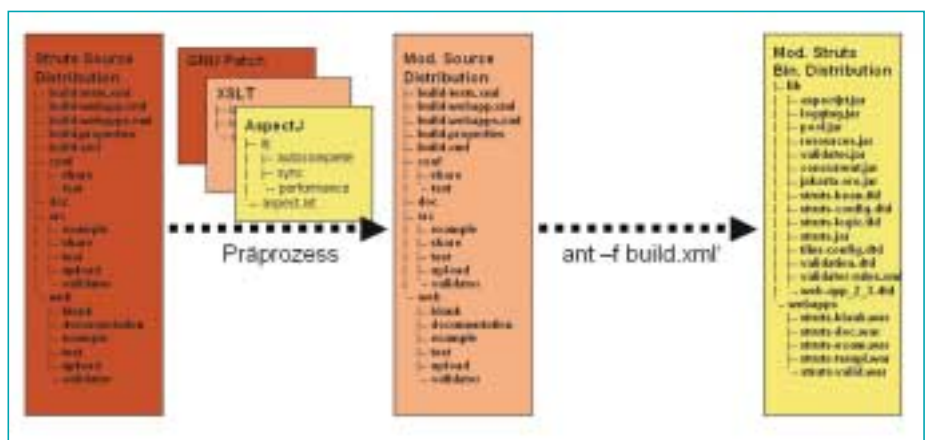


Abb. 1: Schematische Darstellung des Build-Prozesses



Quellcode auf CD!

chen Session durch die Servlet Engine verarbeitet werden. Eine Synchronisation der HTTP-Request Verarbeitung kann daher zur Vermeidung von Race Conditions notwendig sein.

- Patch der Struts-Taglib zur Einpflege eines *Autocomplete*-Flags.
- **Interceptor-Framework:** Ähnlich wie beim *javax.servlet.Filter*-Mechanismus können für wohldefinierte Schnittstellen im Struts-Code Interceptor-Ketten definiert und zur Laufzeit der Anwendung ausgetauscht und rekonfiguriert werden.
- **Zwischenseiten-Mechanismus** zur Verhinderung von Mehrfach-Clicks.
- **Performance-Messpunkte.**
- **Sicherstellung der Integrität der Struts-Anwendung.**
- **Browsernavigations-Detektion und Requestreplay.**

Eine dieser Erweiterung soll in diesem Artikel exemplarisch dargestellt werden. Dabei werden die verwendeten Technologien, wie AspectJ und XSLT, kurz skizziert. Auf der Heft-CD finden sich neben der im Folgenden dargestellten Erweiterung noch Code-Fragmente zu zwei weiteren Beispielen.

Bevor es aber losgeht, möchte ich eine knappe Bemerkung zu Microsofts Internet Explorer einschieben: Dieser bietet ein Autocomplete-Feature an, welches aus Sicherheitsgründen unterbunden werden sollte. Setzt man im HTML *Input*-Tag das Attribut `AUTOCOMPLETE="OFF"`, so speichert der Internet Explorer die Benutzereingaben nicht:

```
<input AUTOCOMPLETE="OFF" name="pin" type="password" />
```

Zunächst noch etwas Theorie, bevor demonstriert wird, wie sich die Struts-Distribution mittels AspectJ und zur Lösung des Autocomplete-Problems anpassen lässt.

### AspectJ

Im Rahmen des Softwareentwicklungsprozesses werden heute eine Vielzahl von Werkzeugen, Techniken und Methoden eingesetzt, um die Komplexität des mittels Software zu lösenden Problems zu beherrschen und um Software effizient erstellen und warten zu können.

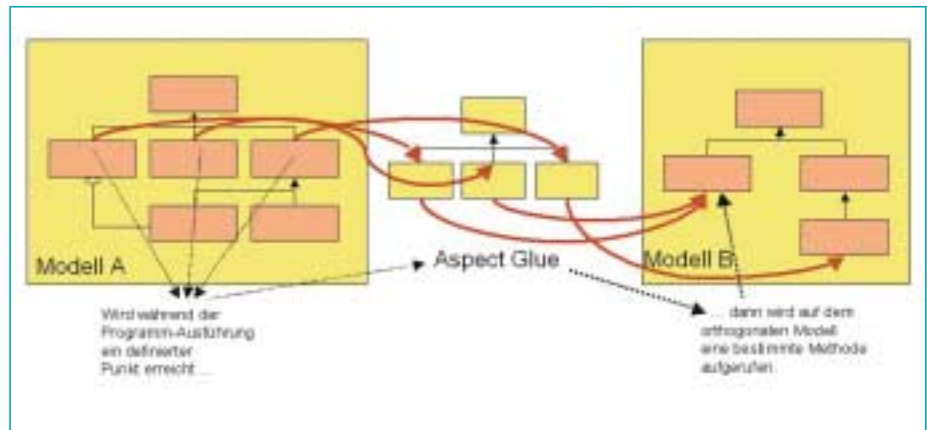


Abb. 2: JoinPoints bei Aspekt Orientierter Programmierung

Bei der objektorientierten Modellierung eines Problems werden relevante Elemente aus der Problemdomäne identifiziert und in Form von Klassen beschrieben. Querschnittliche Eigenschaften, so genannte Crosscutting Concerns, werden dabei nicht innerhalb einer einzelnen Klasse lokalisiert, sondern klassenübergreifend modelliert. Beispiele für Crosscutting Concerns sind:

- Authentifizierung,
- Zugriffskontrolle,
- Synchronisation,
- Testen von Vor- und Nachbedingungen,
- Fehlerbehandlung,
- Performanceoptimierung,
- Transaktionsverwaltung,
- Persistenz sowie
- Einhaltung von Architektur- und Designregeln.

Problematisch bei dieser klassenübergreifenden Modellierung ist die fehlende Modularität der Concerns. Notwendige Änderungen erfordern das Auffinden des tangled Codes und eine konsistente Modifikation unter Berücksichtigung des jeweiligen Kontextes. Dies macht Modifikation von Crosscutting Codes aufwändig, schwierig und damit auch fehleranfällig.

Ausgangspunkt für Aspect Oriented Programming (AOP) ist die Überlegung, dass es „die“ modulare Struktur eines Softwaresystems nicht gibt. Verschiebt man die Gewichtung der Concerns bei der Modellierung eines Systems, so erhält man unterschiedliche, orthogonale Modelle (Crosscutting Models). Crosscutting ist also eine

inhärente Eigenschaft eines Softwaresystems. AOP stellt Werkzeuge und Methoden zur Verfügung, um Crosscutting Concerns/Models zu beschreiben und miteinander zu verknüpfen. Die Verknüpfung der Crosscutting Models wird durch Aspects beschrieben. Ein Aspect spezifiziert, an welchen statischen Punkten im Programmtext bzw. an welchen dynamischen Punkten im Aufrufgrafen eines Modells, welche Objektmethoden eines orthogonalen Modells aufgerufen werden. Die oben angeführten Punkte werden auch als JoinPoints bezeichnet (Abb. 2).

AspectJ von *aspectj.org* ist ein Werkzeug des Computer Science Lab des Palo Alto Research Centers (PARC) zur Modellierung und Realisierung von Crosscutting Concerns in Java. Seit Anfang diesen Jahres erfolgt die Entwicklung von AspectJ innerhalb des Eclipse-Projekts. AspectJ ist Open Source und unter der Mozilla Public License Version 1.1 erhältlich. Plugins für IDEs wie Eclipse und JBuilder sind verfügbar.

AspectJ ist eine Erweiterung des Java-Standards: Ein spezieller Java-Compiler, mit dem es möglich ist, Aspekte in Java-Klassen hineinzuweben. Seit Version 1.1 können nicht nur in Java-Sourcen mit AspectJ behandelt werden, sondern auch Java CLASS-Dateien mit Aspekten angereichert werden. Im Rahmen dieses Artikels werden wir uns das Hineinweben von Aspekten in Java-Sourcen konzentrieren.

In AspectJ werden JoinPoints mit Hilfe so genannter Pointcuts beschrieben. Aus einfachen Pointcuts lassen sich über Bool'sche Operatoren wiederum komplexe Pointcuts beschreiben. Pointcuts defi-

### Listing 1

```
package struts;

import org.apache.struts.taglib.html.*;
import org.apache.struts.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.*;
import java.io.*;

privileged aspect BaseFieldTagAspect
{
    private boolean BaseFieldTag.autocomplete = false;
    public boolean BaseFieldTag.getAutocomplete()
    {
        return this.autocomplete;
    }
    public void BaseFieldTag.setAutocomplete(final boolean b)
    {
        this.autocomplete = b;
    }

    [...]
}
```

### Listing 2

```
privileged aspect BaseFieldTagAspect
{
    [...]

    StringBuffer around( BaseFieldTag tag )
    : withincode( public * BaseFieldTag.doStartTag() )
    && call( StringBuffer.new( String ) )
    && this( tag )
    {
        return new StringBuffer(
            tag.getAutocomplete()
            ? "<input AUTOCOMPLETE=\"OFF\" type=\""
            : "<input type=\"" );
    }
}
```

nieren auch Variablen. An diese werden Objekte im Kontext des JoinPoints, gebunden. Advices – das sind Code-Segmente, die an Pointcuts geknüpft werden können, um Methoden auf Objekte eines orthogonalen Modells zuzugreifen – können dann diese Variablen verwenden. Mittels AspectJ ist es auch möglich, neue Methoden in bestehende Klassen zu weben. Dies wird auch als Inter Class-Deklaration bezeichnet.

Doch nun genug der Theorie, zurück zum Beispiel. Das Beispiel ist relativ einfach, daher wird AspectJ primär als Java-Präprozessor verwendet, d.h. der Advice Code wird direkt verwendet, um die benötigte Modifikation an Struts zu realisieren.

In Struts ist die Klasse *org.apache.struts.taglib.html.BaseFieldTag* für die Generierung des HTML *Input*-Tags zuständig. Eine Unterstützung für das *Autocomplete*-Attribut ist jedoch nicht vorhanden. Die *BaseFieldTags*-Klasse soll nun so angepasst werden, dass es das *Autocomplete*-Attribute unterstützt. Das *BaseFieldTag* wird dazu in einem ersten Schritt um ein Bean Property *Autocomplete* ergänzt, dies ist der Schalter des JSP-Tags mit dem sich die Autocompletion ein- bzw. ausschalten lässt. Die Ergänzung geschieht mit einer AspectJ Inter Class-Deklaration, die in Listing 1 dargestellt ist.

Im Code-Fragment sieht man, dass sowohl die Variable *autocomplete* als auch die beiden Accessor-Methoden *getAutocomplete* und *setAutocomplete* das Präfix *BaseFieldTag* besitzen. Dies veranlasst den AspectJ-Präprozessor sowohl die Variable als auch die beiden Metho-

den in die Klasse *BaseFieldTag* hineinzuweben.

Im nächsten Schritt soll nun die neue *Autocomplete* JavaBean-Property bei der Erzeugung der HTML-Ausgabe Verwendung finden. Das *BaseFieldTag* aus der Struts-Source Distribution verwendet einen *java.lang.StringBuffer*, der mit dem String "*<input type=\"*" initialisiert wird, um das HTML-Fragment des Tags zu generieren:

```
public int doStartTag() throws JspException
{
    StringBuffer results = new StringBuffer("<input type=\"");
    results.append(type);
    results.append("&name=\"");
    [...]
}
```

Übergibt man dem Konstruktor des *StringBuffers* den Wert "*<input AUTOCOMPLETE=\"OFF\" type=\"*", dann ist man bereits am Ziel. Mit AspectJ lässt sich dies recht einfach durchführen. Dazu wird der *BaseFieldTagAspect* noch um einige Zeilen ergänzt (Listing 2). Die Zeilen

```
call( StringBuffer.new( String ) )
&& withincode( public * BaseFieldTag.doStartTag() )
&& this( tag )
```

spezifizieren einen Pointcut: Alle Aufrufe eines *StringBuffer*-Konstruktors innerhalb der *doStartTag*-Methode der *BaseFieldTags*-Klasse. Der Pointcut definiert zudem noch die Variable *tag* und bindet sie an das Objekt, in dem die Methode aufgerufen wird, also an das *BaseFieldTag*-Objekt. Innerhalb des Advices:

## Anzeige

```
{
return new StringBuffer(
    tag.getAutocomplete()
    ? "<input AUTOCOMPLETE=\"OFF\" type=\""
    : "<input type=\"" );
}
```

wird nun die ursprüngliche Codesequenz `new StringBuffer("<input type=\""` wie gewünscht ausgetauscht. Der Vollständigkeit halber sei noch erwähnt, dass inner-

halb der `release`-Methode des `BaseFieldTags` noch die Variable `autocomplete` auf ihren Default-Wert zurückgesetzt werden sollte. Dies kann relativ leicht auch mit Hilfe von AspectJ geschehen.

### XSLT

Was noch fehlt, ist die Anpassung des Tag-Library-Descriptors, um das `autocomplete`-Attribut des Struts `Input`-Tags sichtbar zu machen. Innerhalb der Struts-Distribu-

tion finden wir die Quellen für den Tag-Library-Descriptor in der Datei `userGuide/struts-html.xml`.

Die Modifikation von Konfigurationsdateien, Tag-Library-Descriptors oder Build-Dateien der Struts-Source-Distribution erfolgt über XSLT-Stylesheets. XSLT ist eine Sprache zur Transformation von XML-Dokumenten des W3Cs. Für Apaches Ant Build-Framework existieren Tasks, die es ermöglichen, XML-Transformationen auch innerhalb eines Build-Prozesses durchzuführen. In Listing 3 ist das Stylesheet zur Modifikation des Tag-Library-Descriptors abgebildet.

### Fazit

In diesem Artikel wurde gezeigt, wie sich mit Hilfe von AspectJ, XSLT-Stylesheets und Ant eine Struts-Distribution anpassen lässt. Durch die hier skizzierten Technologien können Änderungen an der Struts-Distribution nicht nur durchgeführt werden, sondern werden auch in einem dokumentiert. Innerhalb eines Kundenprojektes wurden auf diese Art alle Modifikationen an Struts erfolgreich implementiert. Am Ende des Projektes wurde von Apache ein Struts-Update freigegeben. Aus dieser Struts-Distribution ließ sich sicher und ohne größeren Zeitaufwand eine neue modifizierte Struts-Distribution erstellen. Die hier aufgezeigten Techniken lassen sich auch ohne weiteres auch auf andere Projekte übertragen. ■

### Links & Literatur

- [eclipse.org/aspectj/](http://eclipse.org/aspectj/)
- [www.w3.org/TR/xslt/](http://www.w3.org/TR/xslt/)

### Listing 3

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Diese Regel erweitert die Definition des
  Struts Passwort Tags um das autocomplete
  Attribute. -->
  <xsl:template match="tag[name='password']">
  <xsl:copy>
  <xsl:call-template name="copy-attributes"/>
  <xsl:apply-templates select="@*|node()"/>
  <attribute>
  <name>autocomplete</name>
  <required>>false</required>
  <rtexprvalue>>true</rtexprvalue>
  </attribute>
  </xsl:copy>
  </xsl:template>

  <!-- Diese Regel erweitert die Definition des
  Struts Form Tags um das autocomplete
  Attribute. -->
  <xsl:template match="tag[name='form']">
  <xsl:copy>
  <xsl:call-template name="copy-attributes"/>
  <xsl:apply-templates select="@*|node()"/>
  <attribute>
  <name>autocomplete</name>
  <required>>false</required>
  <rtexprvalue>>true</rtexprvalue>

  </attribute>
  </xsl:template>

  <!-- Diese Regel kopiert den aktuellen Knoten
  und seine Attribute in das Zieldokument und
  initiiert eine rekursive Verarbeitung der
  Kindelemente (apply-templates). Dies ist die
  Default-Regel die angewendet wird, wenn keine
  spezielleren Regeln greifen. -->
  <xsl:template match="@*|node()">
  <xsl:copy>
  <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
  </xsl:template>

  <!-- Die 'copy-attributes' Regel kopiert alle
  Attribute des aktuellen Knotens in das
  Zieldokument.-->
  <xsl:template name="copy-attributes">
  <xsl:for-each select="@*">
  <xsl:attribute name="{name()}">
  <xsl:value-of select="."/>
  </xsl:attribute>
  </xsl:for-each>
  </xsl:template>
  </xsl:stylesheet>
```

# Anzeige