DAS MAGAZIN FÜR PROFESSIONELLE IT-LÖSUNGEN

Der Entwickler

<XML...> MAGAZIN

Sonderdruck 5.2001

# Musterung

## XSLT Entwurfsmuster

In der objektorientierten Softwareentwicklung sind Entwurfsmuster ein beliebtes Hilfsmittel. Ihr Einsatz erleichtert die Lösung von Problemen durch Kategorisierung, abstrakte Beschreibungen und Implementierungsbeispiele. Für XML und XSLT wurden ebenfalls Muster gefunden und beschrieben.

von Thomas Bayer, Tobias Kieninger, Özgür Kipik

### XML Muster

Die Idee, Design strukturiert zu beschreiben, wurde in die XML-Welt übernommen und langsam sind Muster für XML und XSLT im Netz oder in der Literatur zu finden. Zur Einführung stellen wir kurz das XML-Entwurfsmuster Head-Body vor, welches mit vielen anderen unter www.xmlpatterns.com nachgeschlagen werden kann. Eine Anwendung dieses Musters findet man im BMECat-Format, einem weit verbreiteten Standard für Produktkataloge. Im folgenden ein Auszug:

```
<BMECAT version="1.2">
 <HEADER>
  <CATALOG>
   <CATALOG_ID>f:3M739</CATALOG_ID>
   <CATALOG_NAME>Kurs</CATALOG_NAME>
   ...
  </CATALOG>
  ...
 </HEADER>
 <T_NEW_CATALOG>
  <ARTICLE>
  ...
  </ARTICLE>
 ...
 </T_NEW_CATALOG>
</BMECAT>
```

Die Namensgebung einiger Elemente lehnt sich an den Aufbau einer HTML-Seite an. Metainformationen über den Katalog werden im Element HEADER untergebracht. Dies ist analog zu einer HTML-Seite, die in ihrem Element HEAD ebenfalls Informationen wie den Titel der Seite, die Adressbasis oder logische Beziehungen zu anderen Dateien einschließt. Der Rumpf (Body), der durch das T_NEW_CATALOG-Element markiert wird, umschließt den Inhalt eines Kataloges. Hier kann wieder der Vergleich zu HTML herangezogen werden. Innerhalb des Elementes BODY einer HTML-Datei befindet sich ebenfalls der Inhalt der Seite.

Neben dem Head-Body-Pattern gibt es eine Reihe weiterer XML-Muster wie „Catch All Elements“ oder „Choice Reducing Container“.

### XSLT Muster

Die Systematik der formalen Beschreibung von XML-Entwurfsmustern kann auch auf die „eXtensible Stylesheet Language Transformations“ (XSLT) angewandt werden. Neben den Stylesheet Patterns aus „XSLT Programmers Reference“ von Michael Kay und einigen Ideen und Konzepten in der sehr ergiebigen XSLT Faq findet man erst wenige formal beschriebene Entwurfsmuster.

Wir möchten ein Muster für XSLT anhand eines praktischen Szenarios vorstellen. Die Struktur dieses Patterns erinnert an das Überladen von Methoden in der Objektorientierung, daher haben wir uns für den Namen „Overloaded Template Pattern“ entschieden.

Ein XML-Dokument mit einer Liste von CDs soll mittels XSLT in HTML transformiert werden. Einige CDs verfügen bereits über einen Preis und sollen in einer Tabelle mit einem orangefarbenen Hintergrund markiert werden (siehe Abb. 1).

Ein Stylesheet mit einem Template für das Element *cd* erfüllt die Aufgabe:

```
<xsl:template match="cd">
 <xsl:variable name="farbe">
  <xsl:choose>
   <xsl:when test="preis">orange</xsl:when>
   <xsl:otherwise>lightgrey</xsl:otherwise>
  </xsl:choose>
 </xsl:variable>

 <tr style="background: {$farbe}">
  <td><xsl:value-of select="titel"/></td>
 </tr>
</xsl:template>
```

Die Anweisung <*xsl:variable*> enthält einen *choose*-Block, der den Wert der Variable *farbe* abhängig von der Existenz des

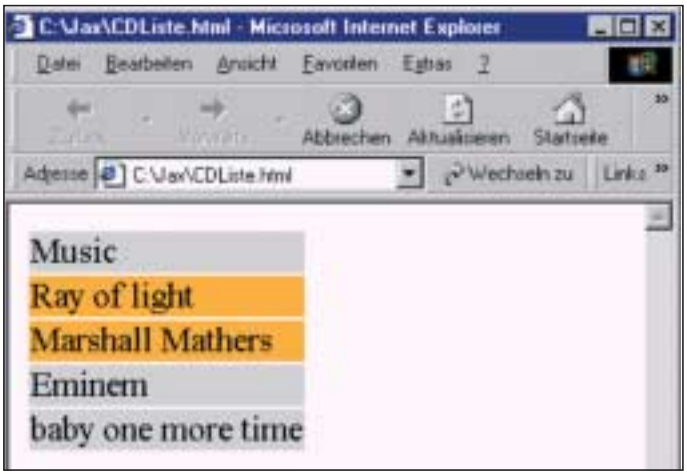


Abb. 1: CD-Liste

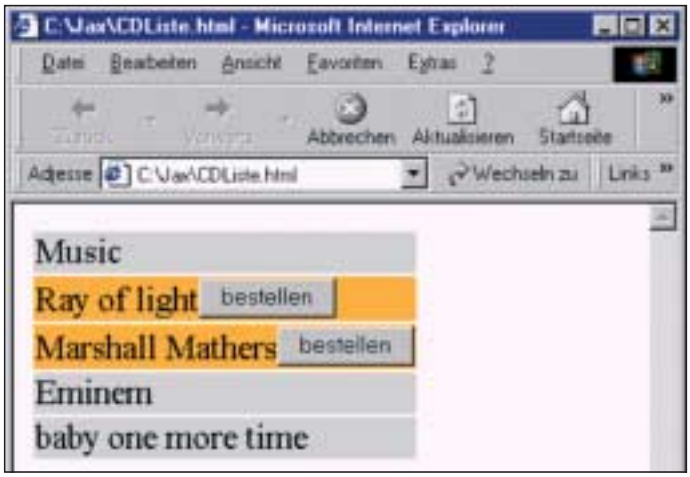


Abb. 2: CD-Liste mit Bestellknopf

Childelements *preis*, mit *orange* oder *lightgrey* initialisiert. Dem Attribut *style* wird über die Variable *farbe* der jeweilige Hintergrund zugewiesen.

Variablen haben Auswirkungen auf die Lesbarkeit und provozieren Seiteneffekte. In XSLT sollte man möglichst die Verwendung von Variablen vermeiden. Will man nicht auf Variablen verzichten, so können diese in XSLT zwar definiert, danach jedoch nicht mehr verändert werden.

Wir können auf die Variable *farbe* verzichten, indem wir den Wert des Attributs *style* direkt durch den *choose*-Block bestimmen. Dazu kann das Element <*xsl:attribute*> innerhalb von <*tr*> verwendet werden:

```
<xsl:template match="cd">
 <tr>
  <xsl:attribute name="style">
   background:
   <xsl:choose>
    <xsl:when test="preis">orange</xsl:when>
    <xsl:otherwise>lightgrey</xsl:otherwise>
   </xsl:choose>
  </xsl:attribute>
  <td>
   <xsl:value-of select="titel"/>
  </td>
 </tr>
</xsl:template>
```

Wir haben jetzt eine Variable eliminiert und den Code etwas vereinfacht. Noch stört der unhandliche *choose*-Block.

Dass Bedingungsselemente wie *choose* und *if* Schwierigkeiten bereiten können, wird deutlich, wenn die Anforderungen an das Stylesheet steigen. Zusätzlich zur farblichen Markierung von CDs soll ein Knopf BESTELLEN angezeigt werden, damit diese CD in einen Warenkorb aufgenommen werden kann. Das Resultat ist in Abbildung 2 sichtbar.

Eine zweite Bedingungsanweisung wird benötigt, da mit dem *choose*-Element bisher nur der Wert des Attributes *style* bestimmt wurde. Für den Button muss erneut geprüft werden, ob die CD über einen Preis verfügt oder nicht (siehe Listing 1).

Die zwei Bedingungsanweisungen blähen den Code auf. Weiterhin wird die Anzahl der Bedingungen bei komplexeren Problemstellungen weiter steigen. Ein Ausweg stellt das Overloaded Template Pattern dar (siehe Kasten „Overloaded Template Pattern (Auszug)“) Der Code wird vereinfacht und der Inhalt eines unübersichtlichen Templates mit Bedingungselementen auf mehrere einzelne aufgeteilt. Diese verfügen über ein zusätzliches Prädikat zur Spezialisierung. Das Template ohne Prädikat dient in diesem Fall als Standard-Template.

Das erste Template in folgendem Listing enthält die Standardausgabe der CDs ohne speziellere Behandlung.

```
<xsl:template match="cd">
 <tr style="background: lightgrey">
  <td><xsl:value-of select="titel"/></td>
 </tr>
</xsl:template>

<xsl:template match="cd[preis]">
 <tr style="background: orange">
  <td>
   <xsl:value-of select="titel"/>
   <input type="button" value="bestellen"/>
  </td>
 </tr>
</xsl:template>
```

Das Template mit dem Prädikat „preis“ wird nur für CDs instanziert,

**Listing 1**

```
<xsl:template match="cd">
 <tr>
  <xsl:attribute name="style">
   background:
   <xsl:choose>
    <xsl:when test="preis">orange</xsl:when>
    <xsl:otherwise>lightgrey</xsl:otherwise>
   </xsl:choose>
  </xsl:attribute>
  <td>
   <xsl:value-of select="titel"/>
   <xsl:if test="preis">
    <input type="button" value="bestellen"/>
   </xsl:if>
  </td>
 </tr>
</xsl:template>
```

welche ein Preis-Element enthalten. Die farbige Markierung und der Button werden in einem separaten Template gekapselt. Durch den Wegfall der Bedingungselemente wird der Aufbau der Templates wesentlich einfacher.

Das Entwurfsmuster ist ideal für die Behandlung von Sonderfällen mittels Prädikaten. Unter Umständen entsteht doppelter Code durch die Aufspaltung in mehrere Templates. Dieser Nachteil kann durch die Verwendung von <*xsl:apply-templates*> vermieden werden.

Ein weiteres Beispiel für den sinnvollen Einsatz des Patterns demonstriert die abwechselnde Färbung von Tabellenzeilen:

```
<xsl:template match="cd">
 <tr style="background: lightgrey">
  <td><xsl:value-of select="titel"/></td>
 </tr>
</xsl:template>

<xsl:template match="cd[position() mod 2 = 1]">
 <tr style="background: orange">
  <td><xsl:value-of select="titel"/></td>
 </tr>
</xsl:template>
```

Während das erste Template wieder als Standard-Template dient, behandelt das zweite lediglich die ungeraden Tabellenzeilen wie 1, 3, 5 usw. Dies wird über das Prädikat *[position() mod 2 = 1]* erreicht, welches auf den Restwert der Division zweier Zahlen prüft. Bei geraden Zeilennummern wird das Standard Template instanziert und die Zeile hellgrau hinterlegt. Die ungeraden werden entsprechend orange dargestellt.

Das Overloaded Template findet man häufig in der Praxis, z.B. in Stefano Mazzocchis Web Publishing Framework Cocoon 2. Das Cocoon Core Stylesheet verwendet dieses Pattern für das Erstellen der Klasse *Sitemap*. Der Javacode für Actions wird von unterschiedlichen Templates erzeugt, je nachdem, ob ein Actionset verwendet wird oder nicht:

```
<!—processing of an act element having a type
                                        attribute—>
<xsl:template match="map:act[@type]">
 ...
</xsl:template><!— match="map:act[@type]" —>

<!—processing of an act element having a set attribute—>
<xsl:template match="map:act[@set]">
 ...
</xsl:template><!— match="map:act[@set]" —>
```

## Fazit

Die Methodik, Designerfahrungen in Mustern zu beschreiben, ist auch für die eXtensible Stylesheet Language Transformations anwendbar. Muster können ein gemeinsames Vokabular für verschiedene Entwurfsansätze schaffen und etablieren.

Wer selbst bereits Stylesheets geschrieben hat, weiß, wie wichtig Erfahrung beim Entwickeln ist. Wir erwarten eine zunehmende Verbreitung, da Muster diese Erfahrungen vermitteln können.

### Overloaded Template Pattern (Auszug):

Die komplette Beschreibung ist unter www.xslt-patterns.com zu finden.

**Abstrakt**

Aufteilung von unübersichtlichen Templates mit Bedingungselementen auf mehrere einzelne Templates, die um ein zusätzliches Prädikat erweitert sind.

**Motivation**

Bedingungselemente in Templates können unübersichtlich und schlecht wartbar sein.

**Anwendbarkeit**

Verwendung, wenn verwandte Matchpatterns sich in der Instanziierung unterscheiden sollen. Vor allem für die übersichtliche Behandlung von Sonderfällen.
Anwendung, wenn unterschiedliche Verhaltensweisen durch Bedingungsanweisungen behandelt werden.

**Struktur**

Mehrere Templates unterscheiden sich im Matchpattern nur durch ein zusätzliches Prädikat. Ein Standard-Template kann durch Weglassen des Prädikates definiert werden.

**Teilnehmer**

Mehrere alternative Templates, die sich im Matchpattern bis auf ein Prädikat unterscheiden. Mindestens ein aufrufendes Template, welches die alternativen Regeln durch ein *apply-templates*-Element aufruft. Der XPath Ausdruck im *select* ist gegenüber dem zusätzlichen Prädikat unspezifisch.

## > Links & Literatur


- XSLT Patterns: www.XSLT-patterns.com
- XML Patterns: www.xmlpatterns.com
- Michael Kay, XSLT - Programmer´s Reference
- XSLT Faq: www.dpawson.co.uk/xsl/sect2/sect21.html
- Cocoon2: http://xml.apache.org/cocoon2
- BMECat: www.bmecat.de/deutsch/index.asp