Dynamische Webseiten mit XML

International PHP Conference 2001
07.11.2001, Frankfurt-Mörfelden

Stephan Schmidt

GUTEN MORGEN!

Inhalt der Session

- Über den Redner
- Evolution von Websites
- Kurze Einführung in XML
- Trennung von Content und Layout
- Transformation von XML in HTML
- XML Parsing und Transformation mit PHP 4
- Transformation mittels expat
- Einfügen dynamischer Inhalte
- patXMLRenderer

Stephan Schmidt

- Web Application Developer bei Metrix Internet Design GmbH in Karlsruhe
- Spezialisiert auf Frontends und Community Anwendungen
- Nebenbei: LGPL Tools auf
  http://www.php-tools.de

Evolution von Websites

Statische HTML Seite als Prospekt im Internet

Einfache dynamische Elemente

Webapplikationen wie Shops

Webbrowser nicht mehr einziges Anzeigemedium
XML als Format zur Datenhaltung, XSLT zur Transformation

Aber was ist mit dynamischen Elementen?

Kurze Einführung in XML

- Beschreibung strukturierter Daten wie z.B. Bücher, Preislisten, usw.
- Verwendung von Tags mit Attributen, wie bei HTML
- großer Unterschied zu HTML: freies Tag Set, dass der Anwender selbst definieren kann
- Validierung durch eine DTD oder XML-Schema
- Interne und externe Entitäten (z.B. ä)

Wichtige XML Regeln

- Jedes Dokument besitzt eine Wurzel
- Jeder Tag muss wieder geschlossen werden, Also `<p>...</p>` statt `<p>` und `<br/>` statt `<br>`
- Attribute immer in Anführungszeichen
- Encoding für Sonderzeichen wie "**<**" und "**>**"

Trennung von Content und Layout

- XML beschreibt den Inhalt, nicht das Layout
- Layout weiterhin in HTML (Browserunterstützung)
- Definition von Tags, die logische Elemente eines Dokuments beschreiben
  - <headline> für eine Überschrift
  - <quote> für ein Zitat
  - usw...
- Beispiel? » nächste Folie...

Beispiel für ein XML Dokument

```
<page>
    <headline>Dynamische Webseiten mit XML</headline>
    <section title="Über den Autor">
        <para>Der Autor ist so früh morgens noch müde.</para>
    </section>
    <section title="Kurze Beschreibung">
        <para>Der Vortrag enthält</para>
        <list>
            <listelement>Beispiele</listelement>
            <listelement>wenig PHP Code</listelement>
        </list>
    </section>
</page>
```

Transformation von XML in HTML

Transformation jedes Tags in seine HTML Repräsentation, z.B.:

```
<headline>Dynamische Webseiten mit XML</headline>
```

...wird zu:

```
<font size="+1">
<br><b> Dynamische Webseiten mit XML</b><br><br>
</font>
```

Transformation des ganzen Dokuments

- Für jeden Tag wird ein Template benötigt
- Richtige Reihenfolge bei Verschachtelungen (z.B. Listenelemente in einer Liste)

...wer macht das?

[XSLT]

Transformation mittels Stylesheet

und XSLT Prozessor

XML Parsing mit PHP 4

Es existieren drei Möglichkeiten:

- Verwendung der Sablotron Extension und XSLT
  - Wenig Programmieraufwand
  - Keine Kontrolle während des Parsingprozesses
- Verwendung des SAX-basierten Parsers
  - Mehr Programmieraufwand
  - Volle Kontrolle während des Parsens
- Verwendung der XML-DOM Extension
  - Einlesen der ganzen Datei
  - Hoher Speicherverbrauch

PHP 4 und expat

- Sehr einfache API
- Dokument wird von oben nach unten durchlaufen
- Registrieren von Callback Handlern
  - Start Element Handler
  - End Element Handler
  - Cdata Handler
  - ...und andere (im Moment unwichtig)

Transformation mittels expat

- Verwendung einer Template Klasse

- **Start Element »**
  Holen des Blocks und ersetzen der Platzhalter durch Attributwerte
- **CDATA Handler »**
  Anhängen an den Inhalt des Tags
- **End Element »**
  Parsen des Templates und HTML Code an den Inhalte des übergeordneten Tags anhägen

Beispiel

XML

```
<section title="XML Transformation">
    <para>XML kann mit PHP in HTML transformiert werden.</para>
</section>
```

Template für <section>

```
<table border="0" cellpadding="0" cellspacing="2" width="500">
        <tr><td><b>{TITLE}</b></td><tr>
        <tr><td>{CONTENT}</td></tr>
</table>
```

Template für <para>

```
<font face="Arial" size="2">{CONTENT}<br></font>
```

Einfügen dynamischer Inhalte

Jede PHP Funktion kann zu jeder Zeit verwendet werden.

Einfaches Beispiel: <time:current/> soll Uhrzeit einsetzen.

Lösung: switch/case Anweisung in den Element Handlern, die Code ausführen, statt Template zu parsen.

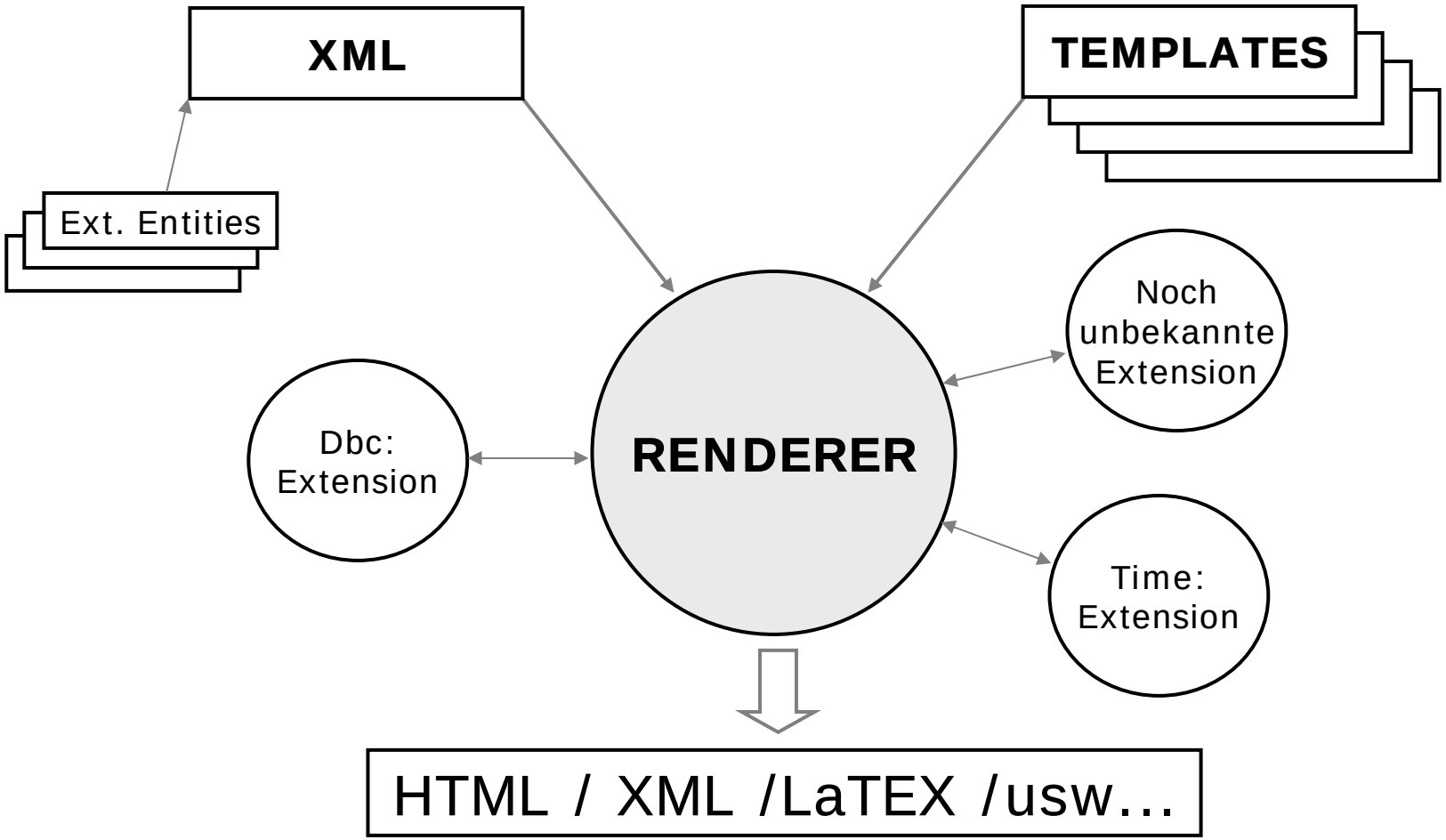
» unübersichtlich, sehr viel Code im Renderer

Besser: beliebig erweiterbarer Parser, der je nach Namespace den Tag an eine Erweiterung übergibt.

Randy - patXMLRenderer

- Open Source (LGPL License)
- Verwendet die Klasse patTemplate
- Unterstützt externe Entitäten
- Beliebig viele Extensions, Tags werden je nach Namespace an die Extensions weitergereicht
- Renderer delegiert nur und übernimmt Transformation
- Beliebig viele Designs oder Ausgabeformate
- Weitere Features: Caching, Logging, usw...

Arbeitsweise



Funktionsweise

- Erzeugen einer Instanz
- Referenz auf ein Template Objekt
- Setzen der Dateinamen der Templates
- Setzen des Dateinamens der XML Datei
- Starten des Parsers und Ausgabe des erzeugten Dokuments
  - Öffnen der XML Datei
  - Zeile für Zeile Einlesen und dem XML Parser übergeben

Dynamischer Content und Extensions

- Öffnender Tag wird gefunden
  mit Namespace » weiter an Extension und Namespace auf Stack

- Daten werden gefunden
  innerhalb eines Namespace » weiter an Extension?

- Schließender Tag wird gefunden
  mit Namespace » weiter an Extension und Rückgabewert an Inhalt anhägen
  ohne Namespace » normal parsen.

Struktur einer Extension

Jede Extension hat drei Handler:

- Starthandler legt Tags und Attribute auf einen Stack
- CData Handler speichert Daten in einer Variable
- End Element Handler führt switch/case Anweisung aus und gibt XML oder CData zurück.

Beispiel: DBC Extension

Verbindung herstellen (keine Rückgabe an den Renderer)

```
<dbc:connection name="foo">
    <dbc:host>localhost</dbc:host>
    <dbc:name>myDB</dbc:name>
    <dbc:user>myUser</dbc:user>
    <dbc:pass>myPass</dbc:pass>
</dbc:connection>
```

Query abschicken (Rückgabe » auf nächster Folie...)

```
<dbc:query connection="foo">
    SELECT * FROM myTable ORDER BY id
</dbc:query>
```

Rückgabe des Queries

```
<result>
    <row>
        <field>1</field>
        <field>Stephan Schmidt</field>
    </row>
    <row>
        <field>5</field>
        <field>Sebastian Mordziol</field>
    </row>
</result>
```

Zurückgegebenes XML wird vom Renderer mit Templates transformiert (könnte auch wieder Extension Tags enthalten).

Schreiben neuer Extensions

Am besten durch Ableitung von einer Extension-Basisklasse:

```
class patXMLRendererExtension
{
    var $cacheAble = array(); // Welche Tags können gecached werden?
    var $attStack = array(); // Stapel für Attribute
    var $tagStack = array(); // Stapel für Tags
    var $dataStack = array(); // Stapel für CDATA

    function getExtensionName()
    {
        .....
    }

    function getExtensionVersion()
    {
        .....
    }
```

Schreiben neuer Extensions

```
    function  startElement( $parser, $ns, $tag, $attributes )
    {
            array_push( $this->attStack, $attributes );
            array_push( $this->tagStack, $tag );
            unset( $this->data[( count( $this->tagStack ) - 1 )] );
            return;
    }
    function  characterData( $parser, $ns, $data )
    {
            if( trim( $data ) )
                $this->data[( count( $this->tagStack ) - 1 )] .= $data;
            return;
    }
    function   getData()
    {
            return   $this->data[( count( $this->tagStack ) - 1 )];
    }
}
```

Beispiel der time: Extension

Basisklasse erweitern und Caching Verhalten für die Tags setzen:

```
class patXMLRendererTimeExtension extends patXMLRendererExtension
{
var $name       = "patXMLTimeExtension";
var $version    = "0.9";
var $cacheAble  = array(    "FORMAT"      => true,
                            "CURRENT"     => false,
                            "COUNTDOWN"   => false,
                            "ELAPSED"     => false );
```

Danach muss nur eine Methode implementiert werden:

Beispiel der time: Extension

```
function  endElement( $parser, $ns, $tag )
{
  $data       = $this->getData();
  $tag        = array_pop( $this->tagStack );
  $attributes = array_pop( $this->attStack );

  switch( $tag )
 {
    case  "FORMAT":
      return date( $attributes["FORMAT"], strtotime( $data ) );
      break;
    case  "CURRENT":
      return date( $attributes["FORMAT"], time() );
      break;
    .....
  }
}
```

Intelligentes Caching

Bisher nur Unterstützung eines intelligenten Caching Mechanismus:

- Überprüfen aller externen Entitäten
- Überprüfen aller verwendeten Templates
- Überprüfen aller Extensions

```
<!--
cache generated=2001-11-01 21:15:33
checkExternal=teasers.xml
checkExternal=projectnav.xml
checkExternal=headernav.xml
checkTemplate=templates/blue/content.tmpl
checkTemplate=templates/blue/pat.tmpl
action=StartCache
-->
```

Möglichkeiten

- Extensions können beliebig verschachtelt werden

```
<dbc:query connection="foo">
SELECT * FROM myTable WHERE id=<var:get scope="global">userid</var:get>
</dbc:query>
```

- Extensions für alle wichtigen PHP Befehle
- Extensions, mit denen sich ganze Applikationen steuern lassen, z.B.:
  `<news:add/>` oder `<news:display amount="5" mode="latest"/>`

- XML als Metasprache: Tags als Befehle, Attribute als Parameter

Geplante Features

- Optional Transformation mit XSLT
- Weitaus mehr Extensions
- Festes Vorgeben einer "Expires"-Zeit für den Cache
- Logfile Analyser (fast fertig)
- Einfaches Erzeugen von Offline-Versionen

Ende

Vielen Dank für Ihre Aufmerksamkeit.

Weitere Informationen:

- Im Open Source Raum (bis 15:00)
- http://www.php-tools.de
- schst@php-tools.de