

Open Source .NET?!

7. Juni 2004

Rechtlicher Hinweis

Dieser Beitrag ist lizenziert unter der UVM Lizenz für die freie Nutzung unveränderter Inhalte.

Zusammenfassung

Beim (ideologiefreien) Vergleich zwischen .NET und Open-Source-Technologien liegt der Fokus häufig auf dem Webbereich; hier lautet das Duell in der Regel ASP.NET vs. PHP. Häufig werden Beispiele zitiert, die mit ASP.NET sehr einfach sind und mit der Open-Source-Konkurrenz nur schwer durchführbar ist. Beispiele sind das Formular-Handling und -Validierung, Web Services und XML-Unterstützung. Der Vortrag zeigt, was Sache ist: Wie einfach ist es wirklich mit ASP.NET, und hat PHP wirklich nichts dagegen zu setzen?

1 Open Source .NET?!

Mit ASP.NET ist das so eine Sache. Die eine Fraktion der ideologisch Verblendeten scheuen Proprietäres im Allgemeinen und die Microsoft-Technologie im Speziellen. Die andere notorische Fraktion der ideologisch Verblendeten vergöttert prinzipiell alles aus Redmond.

Trotz einiger kritischer Mails empfindlicher Anhänger der einen oder anderen Fraktion im Vorfeld, versucht dieser Beitrag, die gesetzten Ziele zu erfüllen:

- Ist ASP.NET wirklich so gut wie viele sagen?
- Ist ASP.NET wirklich so unbrauchbar wie viele sagen?
- Wie sieht es bei relevanten Standardanwendungen aus

Als Vergleichstechnologie wird PHP genommen. Im Vorfeld haben die Autoren an verschiedenen Stellen recherchiert und unter anderem kleine, mittlere und große Firmen befragt, mit Schulungsunternehmen und Verlagen gesprochen, was wirklich im breiten Einsatz ist. Eines ist klar: Enthusiasten und selbsternannte Chef-Evangelisten bevorzugen natürlich immer „ihre“ Technologie, wogegen ja prinzipiell nichts spricht. Und es lässt sich für jede der Alltagsanwendungen eine Technologie finden, in der etwas subjektiv besser oder schlechter geht. Es geht aber nicht darum, für alles jeweils den besten zu finden, sondern einen Vergleich zu ziehen. Dafür muss eine Technologie gewählt werden. Als Maßstab wird nicht nur der Funktionsumfang gewählt, sondern auch die Verbreitung der Sprache (deswegen Schulungsunternehmen und Verlage). Die Wahl ist schließlich auf PHP gefallen. Anhänger anderer Sprachen können ihre Technologien gerne anhand der selben Maßstäbe mit ASP.NET „vergleichen“. Dann aber bitte auch die Marktanteile und aktuellen Buchabsätze der jeweiligen Technologie mit PHP.

Eine weitere Einschränkung gibt es noch: Es wird in der Regel nur die Standard-Distribution der Technologien gewertet. Mit Zusatzmodulen lassen sich natürlich viele Anwendungen besonders einfach oder schön implementieren. Wenn allerdings externe Module mit aufgenommen werden dürften, würden sich daraus zwei Nachteile ergeben: Zum einen würde dann der Fokus weg von der Technologie hin zu Drittanwendungen gehen, zum anderen wäre der Vergleich dann genauso unübersichtlich wie der Markt für Zusatzmodule selbst.

Wenn es um Features geht, wird immer alles in den Ring geworfen, was zur Verfügung steht. Allerdings sollte immer hinterfragt werden, ob diese Features auch wirklich nützlich sind und in der Praxis eingesetzt werden (deswegen die Firmen verschiedenen Größe). Mit zu den häufigsten Anwendungen gehören:

- **Formularhandling:** Die Abfrage und Auswertung von Formulardaten ist das A und O einer Webanwendung. Je einfacher es geht, desto besser.
- **Arbeiten mit XML:** Nach dem Hype folgt (endlich) die Praxisanwendung. XML wird immer häufiger eingesetzt, eine Web-Technologie sollte es möglichst einfach machen, auf XML-Daten zuzugreifen
- **Web Services:** Seit Jahren ein Modethema, mit immer mehr Unterstützern. Insbesondere für den B2B-Bereich eine wichtige Technologie.

ASP.NET hat den Ruf, dass viele dieser Aufgaben besonders einfach zu erledigen sind:

- Formulardaten lassen sich so einfach abfragen, dass man nicht mehr auf eine andere Technologie umsteigen möchte.
- XML ist zentraler Bestandteil der .NET-Strategie, ASP.NET ist also für die Verarbeitung von XML geradezu prädestiniert.
- Microsoft hat Web Services quasi erfunden und tief in .NET integriert; die Verwendung ist ein Klacks.

Diese Aussagen werden wir auf Gültigkeit hin überprüfen. Dennoch: Komplette Objektivität kann natürlich nicht erzielt werden; persönliche Präferenzen und Ansichten der Autoren spielen an vielen Stellen mit. Das Ziel soll es aber keineswegs sein, einen Sieger zu küren. Stattdessen geht es darum, mit zum einen mit Vorurteilen aufzuräumen und auch eine gewisse Form der Entscheidungshilfe zu bieten. Wer sich darauf einlassen kann und will, wird hoffentlich einen Nutzen aus diesen Ausführungen ziehen. Außerdem ist auch das Aufzeigen von Defiziten ein Ansatz, die weitere Entwicklung von Open-Source-Projekten in eine Richtung zu lenken.

1.1 Allgemeines

ASP.NET ist die Web-Komponente der .NET-Strategie von Microsoft. Es handelt sich dabei um die Weiterentwicklung von ASP (Active Server Pages). Vom Programmier-Ansatz hat ASP.NET mit dem alten ASP allerdings nur noch relativ wenig zu tun. Ein wichtiger Bestandteil von .NET ist das .NET Framework, eine umfangreiche Klassenbibliothek.

Das .NET Framework und damit ASP.NET kann kostenlos bezogen werden, es ist allerdings eine Windows-Lizenz erforderlich. Zwar gibt es mit Mono (<http://go-mono.org/> <<http://go-mono.org/>>) ein vielversprechendes Open-Source-Projekt, das sich eine Portierung von .NET auf andere Plattformen zum Ziel gesetzt hat, doch das befindet sich noch kaum im Produktiveinsatz. Deswegen wird als Webserver fast ausschließlich das Microsoft-Produkt IIS eingesetzt. Ein weiteres interessantes Projekt ist dotGNU (<http://www.gnu.org/projects/dotgnu> <<http://www.gnu.org/projects/dotgnu/>>).

PHP (das rekursive Akronym steht für PHP: Hypertext Preprocessor) wurde Mitte der 90er Jahre ersonnen und hat in den letzten Jahren einen rasanten Siegeszug hinter sich. PHP ist Open Source und steht damit auf (fast) jeder Plattform zur Verfügung. In der Regel wird PHP zusammen mit dem Marktführer im Webbrowsermarkt, dem Apache-Webserver eingesetzt. Anfang Mai aktuell ist PHP-Version 4.3.6 sowie der zweite Release Candidate des lange erwarteten Nachfolgers PHP 5, der unter anderem eine deutlich verbesserte Unterstützung von OOP sowie eine runderneuerte XML-Unterstützung bietet.

1.2 Formularhandling

Die Abfrage von Formulardaten ist eine klassische Webanwendung - und sie wird besonders häufig falsch gemacht. Das Auslesen der Nutzerangaben ist eine Sache, viel wichtiger ist jedoch die Usability des Formulars. Problematisch ist nämlich häufig die Formularvalidierung. Zunächst ist auch dieses eine eher einfache Aufgabe - die Eingaben werden geprüft und gegebenenfalls wird eine Fehlermeldung ausgegeben. Was aber, wenn ein Formular mit 50 Feldern ausgefüllt wird und bei einem ein Fehler auftritt? Die Angaben in den restlichen 49 Feldern sollten gefälligst beibehalten werden. Konsequenz: Unter Umständen müssen Formularfelder vorausgefüllt werden.

In den meisten serverseitigen Technologien und auch in PHP ist die Vorausfüllung prinzipiell machbar, aber mühsam. An Vieles muss gedacht werden, beispielsweise an die korrekte Kodierung der Daten. Hier ein Beispiel in PHP, bei dem ein Textfeld vorausgefüllt wird:

```
<input type=\grqq{}text\grqq{} name=\grqq{}Feld\grqq{} value=\grqq{ }<?php
    if (isset($_POST["Feld\grqq{}"])) {
        echo htmlspecialchars($_POST["Feld\grqq{}"]);
    }
?>\grqq{} />
```

Also: Relativ aufwändig. Noch komplexer wird es bei Mehrfach-Auswahlfeldern (*<select multiple>*).

Bei ASP.NET sieht es anders aus. Die hohe Einstiegshürde ist unter anderem damit begründet, dass eine Trennung von Code und Content quasi erzwungen wird. Es gibt zwei Konzepte für Steuerelemente:

- **HTML Controls:** Durch den Zusatz *Runat="Server"* werden herkömmliche HTML-Elemente serverseitig „sichtbar“ gemacht. Sie können dann objektorientiert modifiziert werden.
- **Web Controls:** Neue Steuerelemente werden von ASP.NET in HTML umgewandelt. Einige dieser Steuerelemente ähneln HTML Controls, andere wiederum erzeugen gleich eine Fülle von HTML-Elementen.

Das Vorgehen ist nun das Folgende: Im HTML-Code befinden sich die HTML Controls und Web Controls. In einer externen Datei (das nennt man dann *Code Behind*) oder am Anfang der Seite können dann diese Steuerelemente modifiziert oder einfach nur ausgelesen werden. In der speziellen Methode *Page_Load()* wird Code platziert, der beim Laden der Seite ausgeführt wird:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load() {
    Platzhalter.InnerText = DateTime.Now.ToShortDateString();
    Link.NavigateUrl = "http://www.linuxtag.de/";
}
</script>
<html>
    <head>
        <title>HTML Controls und Web Controls</title>
    </head>
    <body>
        <p>Seite erzeugt am
            <span Id="Platzhalter" Runat="Server" />.</p>
        <p><asp:HyperLink Id="Link"
            Text="LinuxTag"
            Runat="Server" /></p>
    </body>
</html>
```

Beim Laden wird also zum einen die Eigenschaft *InnerText* des ** -Elements gesetzt; es ist hier (von unterschiedlicher Groß-/Kleinschreibung) genauso wie bei JavaScript. Bei dem Web Control *<asp:HyperLink>* , der einen Link darstellt, wird die Eigenschaft *NavigateUrl* gesetzt, die das Linkziel angibt. Beim Laden der Seite landet der folgende HTML-Code beim Client:

```
<html>
    <head>
        <title>HTML Controls und Web Controls</title>
    </head>
    <body>
```

```

    <p>Seite erzeugt am
      <span id="Platzhalter">02.05.2004</span>.</p>
    <p><a id="Link" href="http://www.linuxtag.de/">LinuxTag</a></p>
  </body>
</html>

```

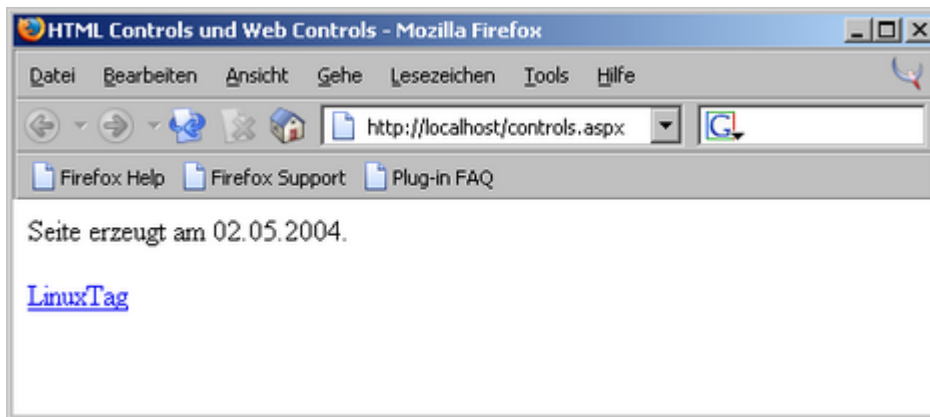


Abbildung : Die Ausgabe der HTML Controls und Web Controls Bei Formulardaten ist das ganz ähnlich. Auch hier genügt der Zusatz Runat="Server" (in allen Formularelementen und im <form>-Tag ebenfalls). Dann können die Daten ausgelesen werden. Die Besonderheit: Beim Neuladen des Dokuments behält das Formularelement seinen Wert. Es ist also nicht mehr notwendig, sich um eine Vorauffüllung zu kümmern, das macht ASP.NET alleine. Hier ein Beispiel:

```

<%@ Page Language="C#" %>
<script runat="server">
void Page_Load() {
    Ausgabe.InnerText = Eingabe.Value;
}
</script>
<html>
  <head>
    <title>Formular</title>
  </head>
  <body>
    <form Runat="Server">
      <p>Eingabe:
        <input type="text" Id="Eingabe" Runat="Server" />
        <input type="submit" Value="Abschicken" />
      </p>
    </form>
    <p>Ausgabe: <span Id="Ausgabe" Runat="Server" /></p>
  </body>
</html>

```

Wie in Abbildung zu sehen, werden auch Sonderzeichen korrekt dargestellt.

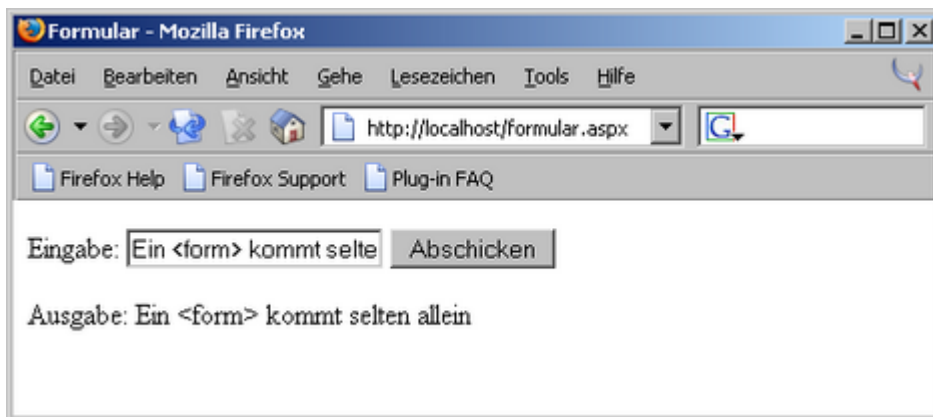


Abbildung : Das ASP.NET-Formular

In Hinblick auf die Formularvalidierung bietet ASP.NET ebenfalls etwas Besonderes: Validation Controls sind Steuerelemente, die mit einem Formularelement verbunden werden. Wird das Formularelement nicht so ausgefüllt, wie gewünscht (möglich ist eine profane Pflichtfeldprüfung, aber auch Komplexeres wie etwa eine Validierung mit einem regulären Ausdruck), wird an der Stelle, an der das Validation Control steht, eine Fehlermeldung ausgegeben. Auch hierzu ein Beispiel:

```
<html>
  <head>
    <title>Formular</title>
  </head>
  <body>
    <form Runat="Server">
      <p>Eingabe:
        <input type="Text" Id="Eingabe" Runat="Server" />
        <asp:RequiredFieldValidator
          ControlToValidate="Eingabe"
          ErrorMessage="Bitte ausfüllen!"
          Runat="Server" /><br />
        <input type="submit" Value="Abschicken" />
      </p>
    </form>
  </body>
</html>
```

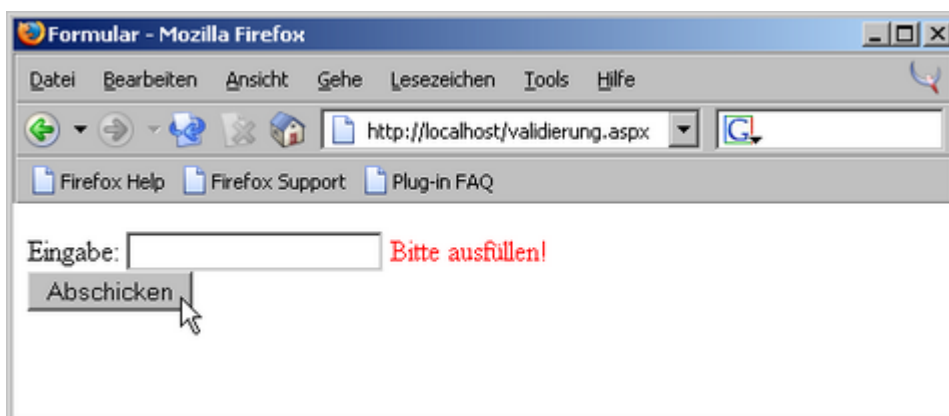


Abbildung : Die automatisch ausgegebene Fehlermeldung

Beim Versuch, das Formular leer abzuschicken, wird die Fehlermeldung ausgegeben. Beim Microsoft Internet Explorer sofort (sofern JavaScript aktiviert ist), also clientseitig; bei allen anderen Browsern wird zunächst das Formular verschickt, die Fehlermeldung serverseitig eingefügt und dann das ganze zurück an den Client geschickt. Es wäre wohl technisch ohne Weiteres möglich, auch bei anderen Browsern per JavaScript die Fehlermeldung schon beim Versuch, das Formular zu versenden, auszugeben; man möge sich überlegen, wieso das nicht so implementiert wurde.

Ist also ASP.NET in diesem Bereich der PHP-Konkurrenz klar überlegen? Klares Nein. PHP selbst bietet nichts Vergleichbares. Eine gewisse Form von Automatismus bietet jedoch das PEAR-Modul *HTML_QuickForm*. Damit ist all das möglich, was ASP.NET hier so attraktiv macht: Vorausfüllung und Validierung. Allerdings muss hier das komplette Formular objektorientiert aufgebaut werden, während bei ASP.NET immerhin eine (vor allem in größeren Firmen häufig so durchgeführte) Arbeitsteilung möglich ist: Ein Designer entwirft das Formular, der Entwickler fügt *Runat="Server"* und ein wenig Programmierung hinzu.

Was bei dieser Argumentation aber völlig außer Acht gelassen wird: Automatismen sind schön, gehen aber meist mit etwas Kontrollverlust daher. Man sieht das beispielsweise an den ASP.NET Validation Controls: Sie funktionieren im Internet Explorer besser als in anderen Browsern. Wenn man außerdem nicht aufpasst, arbeitet das komplette Formular nur mit aktiviertem JavaScript, für professionelle Anwendungen natürlich ein No-Go. Deswegen sollte an dieser Stelle PHP nicht abgeschrieben werden, eine genaue Kontrolle über den ausgegebenen Quellcode hat durchaus Vorteile. Und auch wenn wir externe Module nicht in die „Wertung“ mit aufnehmen wollten, sei dennoch erwähnt, dass sich diverse Module in Entwicklung oder Test befinden, die ein ähnliches Verhalten wie ASP.NET bieten sollen. Es bleibt also weiter spannend.

2 XML

Was vor Jahren noch eine echte Lachnummer war - jeder unterstützte es, keiner konnte erklären wozu es gut ist - ist heute tatsächlich in breiter Verwendung: XML. Und tatsächlich bietet ASP.NET eine umfangreiche XML-Unterstützung. Ist sie aber tatsächlich besser? Um das zu beantworten, zunächst ein (einfaches) Beispieldokument:

```
<?xml version=\grqq{ }1.0\grqq{ } encoding=\grqq{ }UTF-8\grqq{ } ?>
<vortraege>
  <vortrag>
    <titel>Open Source .NET?!</titel>
  </vortrag>
  <vortrag>
    <titel>The GIMP auf den zweiten Blick</titel>
  </vortrag>
</vortraege>
```

ASP.NET bietet nun vielfältige Möglichkeiten. Beispielsweise könnten alle Vorträge ausgegeben werden:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="server">
void Page_Load() {
  XmlDocument d = new XmlDocument();
  d.Load(Server.MapPath( "./vortraege.xml" ));
  XmlNodeList l = d.GetElementsByTagName( "vortrag" );
  foreach (XmlNode n in l) {
    Response.Write( Server.HtmlEncode( n.InnerText ) + "<br />" );
  }
}
```

```

</script>
<html>
  <head>
    <title>XML auslesen</title>
  </head>
  <body>
  </body>
</html>

```

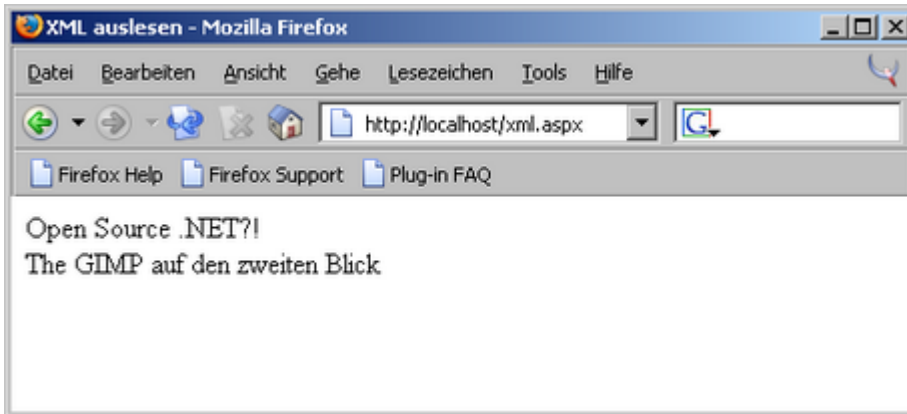


Abbildung : Die Vortagstitel

werden ausgelesen Der Zugriff funktioniert also via DOM. Es sind natürlich noch viele weitere Ansätze möglich, beispielsweise Validierung gegen DTD oder Schema, Die üblichen X-Technologien wie XSLT, XPath etc. sind ebenfalls implementiert. Eine weitere Besonderheit: Der Zugriff auf XML-Daten, sowohl lesend als auch schreibend, kann ähnlich wie der Datenbankzugriff unter ASP.NET erfolgen. Damit ist es ohne Weiteres möglich, XML-Dateien ohne großen Umarbeitungsaufwand als Datenbank zu verwenden.

Und wie sieht es bei PHP aus? Die XML-Unterstützung von PHP 4 ist, sagen wir mal, mäßig. Auf dem Papier sieht sie gut aus, allerdings gilt sie nicht gerade als stabil. Mit PHP 5 hat sich das grundlegend geändert. Eine komplett neue XML-Unterstützung kann mit ASP.NET locker mithalten. Außerdem steht mit *SimpleXML* eine Erweiterung zur Verfügung, die ihren Namen wirklich verdient. Mit diesem, bei einem Perl-Modul entlehnten Ansatz ist ein objektorientierter Zugriff auf XML-Daten möglich, ohne dass groß programmiert werden muss: simpel eben. In PEAR gibt es zudem viele weitere XML-Klassenbibliotheken, die unter anderem eine DTD-Validierung bieten.

Die XML-Unterstützung von ASP.NET ist funktional umfangreich und bietet Zugriff aus einem Guss. Bei PHP sind die verschiedenen PHP-Bibliotheken verteilt, mal als Erweiterung, mal als PEAR-Paket. Dafür steht mit *SimpleXML* eine unglaublich nützliche und praktische Erweiterung zur Verfügung, für die es bei ASP.NET noch kein Pendant gibt.

3 Web Services

Zum Abschluss noch ein Blick auf Web Services. Insbesondere im professionellen Bereich wird diese Technologie immer stärker eingesetzt. Referenzimplementationen bei Google, Amazon, Dell und vielen anderen belegen das.

Microsoft hat Web Services natürlich nicht erfunden, wie das am Anfang des Beitrags noch augenzwinkernd in den Raum gestellt worden ist; allerdings wirkten die Redmonder bei vielen Standardisierungsgremien mit (was sie nicht daran hindern sollte, selbst auch eigene, proprietäre „Standards“ zu setzen). Und in der Tat ist die Erstellung eines Web Services mit ASP.NET sehr einfach. Im Wesentlichen beschränkt es sich darauf, nach der Implementierung das Schlüsselwort *[WebService]* voranzustellen:

```

<%@ WebService language="C#" class="WS" %>
using System;

```

```

using System.Web.Services;
using System.Xml.Serialization;

public class WS {

    [WebMethod]
    public string Datum() {
        return DateTime.Now.ToShortDateString();
    }
}

```

Wird die Datei mit der Endung *.asmx* gespeichert, erledigt ASP.NET den Rest. Ein Aufruf der Datei im Webbrowser liefert automatisch generierte Informationen; außerdem kann per Mausklick der Dienst auch getestet werden. Abbildung zeigt das.

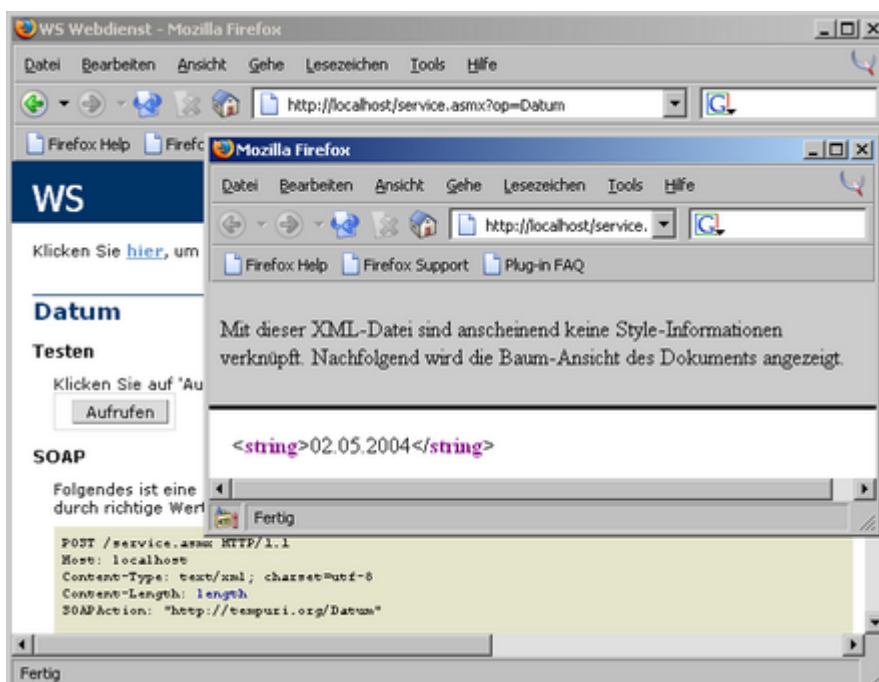


Abbildung : Infos zum Web Service, inklusive Test Und noch besser: Wird *?WSDL* an den URL angehängt, wird automatisch eine WSDL-Beschreibung des Dienstes erzeugt - erneut ohne Programmieraufwand.

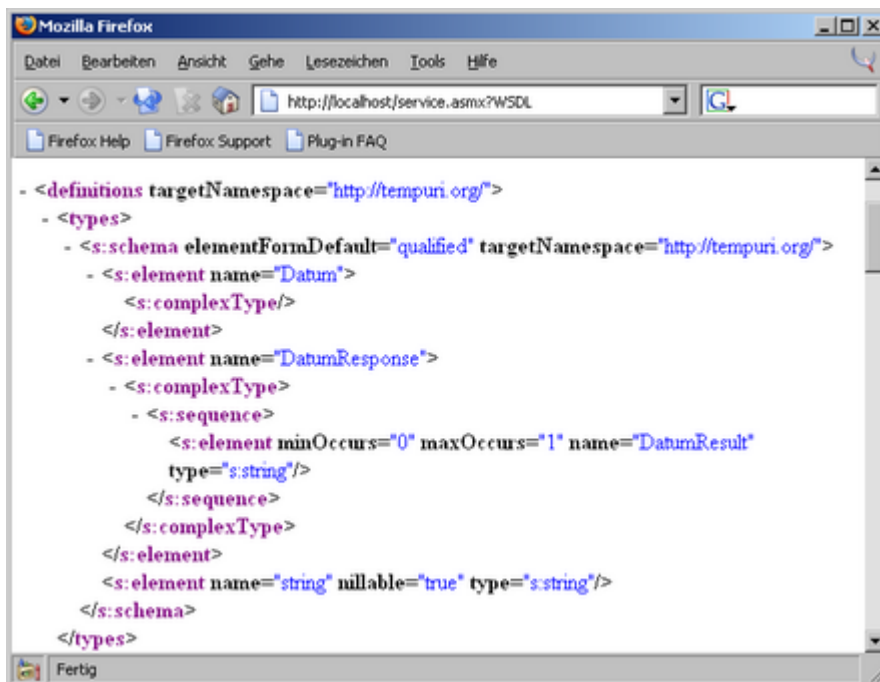


Abbildung : Automatisch erzeugtes WSDL

Ist also PHP hier klar unterlegen? Wie immer: Jein. Es stimmt schon, bei PHP ist es etwas aufwändiger, einen Web Service anzubieten und ihn zu konsumieren. Allerdings steht mit dem PEAR-Paket *SOAP* eine recht ausgereifte Möglichkeit zur Verfügung, die auch automatische WSDL-Generierung erzeugt. PHP 5 bringt eine neue SOAP-Erweiterung gleich mit. Diese ist zwar noch nicht ganz auf dem Stand von *PEAR::SOAP*, aber das wird sich in Zukunft sicherlich ändern.

4 Fazit

Dieser kleine Ausschnitt hat gezeigt: ASP.NET ist keineswegs unschlagbar. Doch um den „Feind“ beurteilen zu können, sollte man ihn schon anschauen. In manchen Bereichen, beispielsweise Web Services, hat ASP.NET die Open-Source-Community angeregt, die Funktionalität nachzubauen und möglicherweise langfristig zu übertreffen. Und das ist eine positive Sache.