

# Integration dynamischer Inhalte in Zope/Plone-Umgebungen

Christian Jungstand | christian.jungstand@mail.mdc.tu-dresden.de

Tobias Miosga | tobias.miosga@mail.mdc.tu-dresden.de

*Seit Beginn der Existenz des Applikations-servers Zope gibt es immer wieder Bestrebungen, nicht-python-basierte Anwendungen innerhalb von Zope nutzen zu können. Insbesondere an Hochschulen wird erst in jüngster Zeit ein Trend zu zentralen Web-systemen erkennbar. Wie aber können nun die vielen dezentralen Webserver und auch unterschiedlichen Technologien in einem zentralen System vereint werden? Im folgenden Artikel stellen wir anhand existierender PHP-Schnittstellen Zope-Produkte vor, die eine Einbindung von PHP erlauben. Im Weiteren wird eine Technologie vorgestellt, die an der TU Dresden eingesetzt wird und für externe Technologien universell einsetzbar ist.*

## Zope-Produkte

Derzeit sind zwei Zope-Produkte für die Einbindung von PHP-Skripten in Zope verfügbar. Das wohl bekanntere Produkt PHPParser/PHPGateway wurde ursprünglich als PHPParser von Maik Jablonski [1] entwickelt und ab 2002 von Wei He [2] übernommen. Ein weiteres Produkt ist PHPObject [3]. Letzteres ist aber seit 2002 als *old and unmaintained* markiert.

### PHPParser/PHPGateway

Für beide Funktionalitäten ist es notwendig, auf dem jeweiligen Renderingknoten das PHP-Kommandozeileninterface zu installieren. Für die Verwendung des PHParsers ist der Zugriff auf das Zope Management Interface (ZMI) notwendig. In diesem wird ein PHPParser-Objekt hinzugefügt und der Quelltext in PHP eingefügt. PHPGateway hingegen erlaubt es, ein Verzeichnis zu benennen, in dem die PHP-Skripte liegen. Beide Objekte sind innerhalb von Zope echte DTML-Subklassen, so dass DTML-Attribute und PHP-Anweisungen gemischt verwendet werden können. PHP wird als Postprozessor verwendet; damit werden alle DTML-Anweisungen zuerst ausgeführt und die Anfrage danach an das PHP-Comand Line Interface (PHPCLI) übergeben. POST- und GET-Parameter können hierbei von PHP genauso bearbeitet werden wie Zope-eigene Parameter, die mittels einer zusätzlichen Variablen übergeben werden (`$ZOE_VARS[]`). Der Hauptnachteil liegt darin, dass PHPCLI auf jedem Rendering-Knoten installiert sein muss. Daneben müssen die PHP-Skripte

auf jedem dieser Knoten verfügbar sein. Ein weiterer Nachteil des PHParsers ist, dass `include()` und `require()` nicht ohne zusätzliche Manipulationen funktionieren.

### PHPObject

Wie bereits erwähnt, handelt es sich um ein älteres Produkt, welches seit 2002 nicht weiter entwickelt und gepflegt wird. Es soll der Vollständigkeit halber trotzdem erwähnt werden. Auch bei PHPObject wird ein solches Objekt im ZMI angelegt und mit PHP-Code gefüllt. Dieser kann sich auch innerhalb von DTML-Code befinden und wird dort durch die bekannten PHP-Begrenzer (`<? ... ?>`) vom übrigen Code abgegrenzt. In der Abarbeitung wird zuerst der DTML-Code geparkt und das Ergebnis in einer temporären Datei gespeichert, welche im Anschluss an den PHP-Interpreter übergeben wird (`(php -q temp_file > result_file)`). Das Ergebnis wird dann zurückgegeben und die temporären Dateien werden gelöscht.

## TUD-Helpers

Die bisher vorgestellten Zope-Produkte erfüllen zwar ihre Aufgaben, haben aber einen zentralen Nachteil. Der PHP-Code wird auf Kommandozeilenebene ausgeführt. Die ohnehin schon höhere Lastsituation als Applikationsserver wird dadurch noch weiter verstärkt. Die PHPParser-Routinen im Cluster der TU Dresden zogen soviel Last, dass diese die Gesamtverfügbarkeit bedrohten. Das System

durch mehr Rechenleistung zu stabilisieren, ist dabei nur ein Teilerfolg. Große Portallösungen wachsen stetig und der Wunsch nach neuen Funktionalitäten oder Schnittstellen zu bereits vorhandenen Informationssystemen ist dabei stets vorhanden. Aus diesem Grund wurde nach alternativen Lösungen zur Anbindung externer Anwendungen und Informationssysteme gesucht und es wurde ein völlig neuer Weg eingeschlagen. Ziel war es, bestehende Anwendungen schnell und möglichst einfach in das Gesamtsystem einzubinden, ohne die ohnehin schon knappen Personalressourcen mit Produktentwicklung zu binden und ohne Leistung aus dem Servercluster abzuziehen. Die Lösung der Problemstellung lag letztlich in der Python-Bibliothek *urllib* [4]. Der Funktion *urlopen* wird eine URL übergeben. Diese öffnet dann eine auf dem Server befindliche lokale Datei oder öffnet einen Socket zum Zielsystem. Der Rückgabewert ist ein dateiähnliches Objekt, das weiter verarbeitet werden kann. So besteht nun zwar die Möglichkeit, eine externe Anwendung einzubinden, die Notwendigkeit der Interaktion bleibt damit aber noch verwehrt. Hier wird das optionale *data*-Argument der Funktion notwendig, es unterliegt aber zwei notwendigen Voraussetzungen. Zunächst muss auf das Protokoll *http* gesetzt werden und der übergebene Wert muss der Standardnotation „application/x-www-form-urlencoded“ unterliegen. So ist man nun in der Lage, POST-Parameter mit der URL an den Zielsystem zu übergeben. Einen Sonderfall stellen GET-Parameter dar. Diese werden zwar in der URL mit an den Zielsystem übergeben, die externe Anwendung wird aber nicht voll funktionsfähig sein, wenn sich Parameternamen mit Zope-eigenen Variablennamen überlagern. Um der Notationsvorgabe nachzukommen, kann man die von der Bibliothek mitgebrachte Funktion *urlencode* nutzen. Diese setzt Tupel zu einem String zusammen, um diesen der Funktion *urlopen* zu übergeben. Das Ergebnis entspricht einer Reihe von variable=wert-Paaren, welche durch ein „&“-Zeichen voneinander getrennt werden. Das System der TU Dresden kann so den Nutzernamen, die ausgewählte Spracheinstel-

lung und den aktuellen Pfad mit an den Zielsystem übergeben. Nachdem die funktionalen Grundlagen geklärt sind, stellt sich nun die Frage nach der Implementierung. Auch hier gibt es verschiedene mehr oder minder aufwendige Möglichkeiten. Naheliegender ist es, einen neuen Objekttyp zu generieren, der einzig für diese Art der Einbindung genutzt wird. Dies ist die wohl eleganteste Wahl, erfordert aber, eigene Content Types mitzuentwickeln. An der TU Dresden hat man sich dafür entschieden, kein eigenes Produkt zu entwickeln, sondern den Weg der Extension zu wählen. Dazu wurden zunächst zwei globale Variablen eingeführt (siehe Listing 1). Danach wurden die entsprechenden TAL-Blöcke in das Template eingefügt (siehe Listing 2). So ist es möglich, vor und nach dem eigentlichen Objekthinhalte eine externe URL einzubinden. Vorteil dieser Methode ist, dass so ein einleitender Text bzw. ein Hinweis auf einen externen Dienst wie gewohnt im CMS eingetragen werden kann und bei Nichtverfügbarkeit der externen Anwendung der Nutzer nicht einfach auf eine leere Seite kommt oder gar eine Fehlermeldung sieht. Bei der Verwendung dieser Einbindung sind einige Hinweise bezüglich der PHP-Anwendung zu beachten. Die Zeichenkodierung der PHP-Anwendung muss der Plone-Instanz entsprechen, da es sonst zu Darstellungsfehlern kommen kann. Ebenso müssen eingebundene Bilder etc. absolut verlinkt sein, da sie sonst vom anfragenden Client nicht geholt werden können. Für die Funktionalität der PHP-Anwendung ist es ebenfalls entscheidend, dass die aufgerufene Datei dem Zope-Objekt als Property bekannt ist und etwaige Parameter ohne expliziten Dateiaufruf übergeben werden. Codebeispiel:

```
http://meine.plone.seite/mein/plone/objekt?meine=php&parameter=1
```

Nachteilig wirkt sich diese Umsetzung beim Caching aus. Der über „include“ in die Seite einbezogene Inhalt ist so nicht von einfachen Dokumenten zu unterscheiden. Um sicherzustellen, dass dem Nutzer auch aktuelle und korrekte und keine veralteten Daten übertragen werden, muss die Cachezeit auf Null gesetzt werden (siehe Listing 3). Um den dadurch

entstehenden Leistungsverlust zu minimieren, wird diese Einstellung nur in den Unterverzeichnissen angewandt, in denen „includes“ vorkommen.

Zusammenfassend kann man sagen, dass der Einsatz der TUD-Helpers in vier Schritten erfolgt:

1. Erstellen der eigentlichen Funktionalität in einem Python-Skript
2. Einbindung des Skriptes über eine externe Methode
3. Beachtung und Einarbeitung der Caching-Problematik
4. Anpassung der Ausnahmen und Sonderregeln in der PHP-Anwendung

## Fazit

Die von der TU Dresden entwickelte Methode stellt nicht das Ende der Entwicklung dar. Bisher kann die URL nur mit Zugriff auf das ZMI eingetragen werden. Dieses Prozedere wurde gewählt, um die externe Anwendung vor der Einbindung zu begutachten und mögliche Risiken für das Gesamtsystem im Vorfeld zu erkennen. Bevor das Ziel, jedem Redakteur die Einbindung zu ermöglichen, umgesetzt werden kann, müssen Qualitätsparameter (quality of service) definiert werden. Diese müssen sicherstellen, dass das Zope-System bei Fehlverhalten oder Ausfall der externen Anwendung unberührt bleibt. Schwachstelle der Methode ist auch die Einbindung in Single-Sign-On-Szenarien. Eine wirkliche bidirektionale Verbindung, in der Nutzerdaten ausgetauscht werden, wird nicht aufgebaut. Hier eine den Sicherheitsanforderungen gerecht werdende Erweiterung zu programmieren, wird Teil der zukünftigen Weiterentwicklungen sein. Einer der wesentlichen Vorteile ist die Unabhängigkeit von der externen Zielanwendung.

Ob PHP oder ASP oder ein ganz anderes System – unabhängig von der Technik kann mit diesem Weg externer Inhalt eingebunden werden. Eine Lösung für die Caching-Problematik könnte die Einführung von entity-Tags sein. Eine andere Möglichkeit wäre die Auswertung der include-property, mit der für jedes Objekt separat ein Cache-Pragma gesetzt wird.

## Die Autoren

Tobias Miosga ist seit 2007 Leiter der Abteilung Medien und Informationssysteme des Medienzentrums der TU Dresden. Er hat an der TU Dresden bis 2006 Medieninformatik studiert. Seit seinem Abschluss ist er im Umfeld des Zope-basierten Webportals der TU Dresden beschäftigt.

Christian Jungstand ist seit 2004 am Medienzentrum tätig. Zunächst als studentische Hilfskraft eingestellt, übernahm er bald auf Honorarbasis die Systemadministration des zentralen Web-Clusters. Als Freiberufler obliegt ihm die Geschäftsführung seiner Firma dd-webdesign. Im Moment arbeitet er am Versionsupgrade des zentralen Zope-Systems.

## Literatur

- [1] JABLONSKI, MAIK: *PHPParser – a PHP-Parser for ZOPE*. <http://www.zope.org/Members/mjablonski/PHPParser>
- [2] HE, WEI: *PHPParser/PHPGateway*. <http://www.zope.org/Members/he-wei/PHPParser>
- [3] COMAN, IOAN: *PHP Document*. <http://www.zope.org/Members/loan/PHPObject>
- [4] PYTHON SOFTWARE FOUNDATION: *Python Library Reference*. <http://docs.python.org/lib/module-urllib.html>

## Listings

Listing 1: Variablendefinition in der document\_view

```
1 <metal :main fill-slot="main"
2 tal:define="text here/Text | here/text ;
3 global include_before here/include_before | nothing ;
4 global include_after here/include_after | nothing ;">
```

Listing 2: Include-Eintrag in der document\_view

```
1 <tal:block tal:condition="include_before">
2 <div tal:content="structure
3     python:here.getUrl2Html(include_before,request) ">
4 renderhere
5 </div>
6 </tal:block>
```

Listing 3: Cache-Parameter im Page Template

```
1 <metal:cacheheaders define-macro="cacheheaders">
2 <metal:block tal:define="dummy
3     python:request.RESPONSE.setHeader(Pragma', 'no-cache')"/>
4 </metal:cacheheaders>
```