



Bebop-ExtJS und Zope3

Uwe Oestermeier



Überblick

Der Einsatz von Zope3 im Institut für Wissensmedien

Die JavaScript-Bibliothek ExtJS

Bebop & ExtJS: Ein paar Beispiele

LivePages

Server-seitige Eventverarbeitung

Client-seitige Eventverarbeitung

Generierung von ExtJS

Probleme



Hintergrund: Das Institut für Wissensmedien

Forschungsinstitut der Leibniz-Gemeinschaft

Ca. 80 Mitarbeiter

Medien-bezogene Lehr-Lernforschung

Starke experimental-psychologische Ausrichtung
(Laborforschung)

Prototypische Lehrszenarien an Schulen und Hochschulen
(Feldforschung)

Evaluations- und Drittmittelprojekte





Zope3 & Bebop als Entwicklungsplattform

2001: Zope2 wird erstmals im Intranet des IWM eingesetzt

2004: Umstieg auf Zope3

- Bebop als Baukasten für experimentelle Web-Applikationen
- Blog & Wiki-Implementationen für die Lehre
- Experimente mit Zope3 & wxPython

2007: Redesign des Intranets

- Beginn der Arbeiten am ExtJS-Client
- Umstellung der Webseiten auf Zope3

2008: Zope2 wird abgeschaltet



Die ExtJS JavaScript-Bibliothek

Beruhet zum Teil auf der YUI-Bibliothek
Kombinierbar mit anderen Bibliotheken

Vorteile

- Überschaubarer Umfang bei großer Funktionalität
- Rege Community
- Fits my brain

Nachteile

- Umstrittene Lizenzpolitik (LGPL vs. GPL)
- Vorschnelle Releases



Grundidee: JavaScript aus Python generieren

ExtJS-Konfigurationsoptionen sehen schon fast wie Python aus

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [ {
        region: 'west',
        minSize: 100,
        width: 200,
        margins: '5 0 0 0',
        items: [Bebop.filetreepanel.view]
    },
    {
        region: 'center'
        collapsible: false,
        border: false,,
        margins: '5 5 5 5',
        items: [Bebop.mainContent]
    },
    {
        region: 'east',
        collapsible: false,
        border: false,
        width: 200,
        el: Ext.get('info-column')
    }
    ]
});
```



Anwendungsbeispiele

- Redaktionsansicht für Institutswebseiten
- Mitarbeiterverwaltung
- Bibliotheks- und Berichtswesen
- Kontaktdaten & Massenmails

Inspiriert durch Twisted, Nevow & Athena

<http://divmod.org/trac/wiki/DivmodNevow>

Neuere Entwicklungen: cometd, Bayeux-Protocol

Grundidee: MVC-Web-Applikationen

- Änderungen werden „live“ als JSON-Objekte übertragen
- Event-Broadcasting (polling, server-side-push, Flash-Sockets, ...)
- „long polling“ per Ajax-Requests als Fallback

Locking als Beispiel



Servervoraussetzungen

Hauptproblem: Langfristig offene Verbindungen

Jeder Request wird in einem eigenen Thread abgearbeitet

Defaulteinstellungen zope.conf

- pool-size = 7
- threads = 4

Trade-off zwischen Anzahl der Threads und Größe des Objekt-Caches



Lösung 1: Zwei Twisted Server

Standard HTTP Server für Zope3

- läuft auf Standardport
- schreibt in die ZODB und bestückt globale Event-Queue

Eigener LivePage Server

- läuft auf einem zusätzlichem Port
- läuft in einem eigenen Thread
- liest die globale Event-Queue

Vorteile: Läßt sich auch über Flash XMLSockets ansprechen

Nachteile: zusätzlicher Port, ca. 800 Zeilen Code, Twisted



Lösung 2: Apache + mod_wsgi

Ein Server für alles

Statische Dateien werden von Apache ausgeliefert

Zope3 als WSGI-Applikation

- läuft in einem eigenen Prozess
- schreibt Events ins Dateisystem

LivePage-Requests

- laufen in einem weiteren Prozess
- lesen Events aus dem Dateisystem

Paster als Entwicklungsserver



Apache mod_wsgi Konfiguration

```
WSGIDaemonProcess zope threads=10
WSGIDaemonProcess live processes=3 threads=128
WSGIDaemonProcess cache threads=10
```

```
<Directory /Users/uo/buildouts/iwmsite>
    WSGIApplicationGroup %{GLOBAL}
    Order allow,deny
    Allow from all
</Directory>
```

```
Alias /@@output /Users/uo/buildouts/iwmsite/src/bebop/apache/live.wsgi
<Location /@@output>
    SetEnv LIVE_CLIENTS
/Users/uo/buildouts/iwmsite/var/livedir/clients/
    SetHandler wsgi-script
    Options +ExecCGI
    WSGIProcessGroup live
</Location>
```



```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type',
                               'application/json, charset=utf-8')])
    uri = environ['REQUEST_URI']
    if '?' in uri: uri = uri.split('?')[0]
    guid = uri.split('/@@output/')[1]
    path = os.path.join(environ['LIVE_CLIENTS'], guid, 'events')
    if os.path.exists(path):
        os.utime(path, None)
        for i in range(16):
            for filename in sorted(os.listdir(path)):
                if filename.startswith('.'):
                    continue
                file = os.path.join(path, filename)
                event = open(file).read()
                os.unlink(file)
                yield event; return
            time.sleep(0.5)
            os.utime(path, None)
    else:
        yield ReloadEvent; return
    yield IdleEvent
```



Server-seitige Eventverarbeitung

Für jede LivePage wird ein temporäres Verzeichnis erzeugt

Der Benutzer löst einen Ajax-Request aus:

- Server-seitig wird die Transaktion durchgeführt und alle Zope3-Events werden gesammelt
- Bevor die Transaktion abgeschlossen wird, erhalten alle registrierten Client-Komponenten die Gelegenheit, die Zope-Events in JSON-Events zu übersetzen
- Die JSON-Events werden im afterCommitHook ins Dateisystem geschrieben
- Alle Clients bekommen in ihrem temporären Verzeichnis einen Symlink auf die sie betreffenden Events

Nach dem Lesen des Events wird der Symlink gelöscht



Probleme

- Wie repräsentiert man den Client auf der Server-Seite über die verschiedenen Requests hinweg?
- Wie repräsentiert man unterschiedliche Konfigurationen von rechteabhängigen Views und Plugins?
- Wie repräsentiert man die Sammlung von Events als transaktionale Einheit?
- Wie informiert man dem Benutzer bei lang anhaltenden Aktionen (z.B. Importfunktionen) über den Verarbeitungsfortschritt?



Client-seitige Eventverarbeitung

Nach dem Aufbau der LivePage beginnt das “long polling”

Hereinkommende JSON-Events werden deserialisiert

ExtJS-Komponenten reagieren auf die Events



Komplikationen

Jede ExtJS-Komponente benötigt spezifische Event-Informationen

Beispiel: Änderungen an einem Dokumente

- Titel: Navigationtree, Breadcrumb, Dokumenten-Tab und Inhaltsverzeichnisse benötigen neuen Titel
- Inhalt: Der Dokumenten-Tab benötigt neue Darstellung, Inhaltsverzeichnisse benötigen ggf. Modifikationsdatum als Sortierkriterium, Navigationtree und Breadcrumb benötigen keine Informationen



Zope3-Komponentenarchitektur als Lösung

- ExtJS-Objekte werden aus Zope-Komponenten generiert
- Die Zope-Komponenten übersetzen die Zope-Events in JSON-Events und annotieren sie ggf. mit Komponenten-spezifischen Informationen
- Der server-seitig registrierte LivePageManager fasst die JSON-Events pro Transaktion zusammen
- Die client-seitige LivePage informiert alle generierten ExtJS-Komponenten über Events der Transaktion
- Jede involvierte ExtJS-Komponente liest die ihr zugedachten Informationen aus



Die Generierung von ExtJS-Komponenten

- Jede ExtJS-Komponente benötigt in der Regel eigene JavaScript- und CSS Dateien
- Die JavaScript-Dateien werden teils aus String-Templates, teils aus korrespondierenden Python-Objekten erzeugt
- Das Verhalten von ExtJS-Widgets und -Views wird weitgehend über Konfigurationsoptionen gesteuert, die in der Regel sehr einfach aus Python generiert werden können
- Formulare werden vollständig über die zope.formlib aus Zope-Schemata generiert und ggf. über Konfigurationsoptionen modifiziert
- Listenansichten können weitgehend auf eine Schemadefinition für die Spalten reduziert werden



JavaScript-Generierung mittels Python-Klassen

```
class DateWidget(zope.app.form.browser.textwidgets.DateWidget):  
  
    def __call__(self):  
        value = self._getFormValue()  
        if value is None or value == self.context.missing_value:  
            value = ''  
        return JSClass('Ext.form.DateField',  
                       value=value,  
                       format='Y.m.d')
```



JavaScript-Generierung mittels String-Templates

```

${name} = new Bebop.GridPanel({
  id: '${ref}-state',
  ref: '${ref}',
  document: '${document}',
  breadcrumb: ${breadcrumb},
  classname: 'Plugin',
  title: '${title}',
  iconCls: '${css}-tab',
  loaded: false,
  exportDialog: null,
  idProperty: 'ref',
  pluginCmd: '${name}',
  autoDestroy: false,
  autoShow: true,

  listeners: {
    activate: function(tab) {
      if (tab.ref) {
        Bebop.show(tab.ref);
      }
    }
  },
  ...

```



Entwicklungsziele

Generierung von AddViews, EditViews, DisplayViews aus Schemata

JavaScript-Basisklassen für Dokumente, Containerviews, Inhaltsverzeichnisse, etc. inkl. Event-Handling

Offline-Modus (Google-Gears, Adobe-Air)



Probleme

Tests für JavaScript

Skalierung der Server-Architektur unklar

Sicherheitsaspekte bislang vernachlässigt

Plattform-Unterschiede bestehen nach wie vor

Große Performanzunterschiede zwischen Browsern



- svn.iwm-kmrc.de