Search for CMS data:
rapid web development using python and AJAX

Valentin Kuznetsov
Cornell University
Introduction

LHC experiments set a new scale for data management

Requirements:

- Easy data access
- Dynamic content and navigation
- Intelligent data query
- Integration with other applications
Choice

Requirements

• Fast turn around with changes, schema/tools evolution
• Rapid web development
• different views for data discovery (production, physicists, etc.)

• Java too heavy, Ruby too new
• Python is well adopted, used almost everywhere in CMS
  • easy to use, very flexible, tons of tools, ideal for prototyping
  • AJAX (Asynchronous JavaScript and XML) is a tool to provide dynamic web content
Tools

- Java Script toolkits:
  - Really Simple History (RSH) framework makes it easy for AJAX applications to incorporate bookmarking and back and button support.
  - Yahoo UI (YUI) as a cross-browser JavaScript library
  - OpenRICO as AJAX framework
  - Cheetah for web page templates
  - CherryPy as a reach web server
  - SQLAlchemy for transparent DB access (SQL abstraction layer/Object Relational Mapper)

Stackable approach: model-view-controller architecture
Workflow

SQL toolkit and Object Relational Mapper

DB back-ends: ORACLE, MySQL, SQLite, etc.

More info via AJAX

Template toolkit

web application framework

Javascript

CLI API

External applications

webservice

Apache web server

CherryPy server

Cheetah

SQLAlchemy

Location # Events Block name

http://cmsdbs.cern.ch/discovery/getData?dbsInst=MC

EVents BlocK name
Example

CherryPy server

```python
import cherrypy, MyTemplate
from Cheetah.Template import Template

class Main(object):
    def ajaxResponse(self, data, **kwargs):
        cherrypy.response.headerMap['Content-Type'] = "text/xml"
        return """<ajax-response>
        <response type="element" id="dataHolder">
        <div>We got data: '%s'</div>
        </response></ajax-response>
```

```python
ajaxDo.exposed=True

def index(self):
    t = Template(MyTemplate, searchList=[{'iList':[1,2,3]}])
    return str(t)
index.exposed = True

cherrypy.root = Main()
cherrypy.server.start()
```

JavaScript (AJAX callback)

```javascript
function getData(data) {
    ajaxEngine.sendRequest('getData', 'data=' + data);
}
function registerAjaxCalls() {
    ajaxEngine.registerRequest('getData', 'ajaxResponse');
    ajaxEngine.registerAjaxElement('dataHolder');
}
```

MyTemplate

```html
<div id="dataHolder"></div>
# for item in $iList
<a href="#" onclick="getData('$item')">Ajax call $item</a>
# end for
```

Fast, simple, elegant
Data discovery

- **Physicists & Production views** provide different level of details over your data lookup
- **Navigator** is a menu driven approach
- **Finder** constructs your own queries
- **Site search** explores data on your site
- **RSS feeds** to stay tune with your data

Regardless which way you go your data is just one click away
Finder as an example

Explore MetaData

Place your condition

Query builds dynamically

To use Finder follow these three steps:
- Use DBS tables treeview on your left to choose a column you wish to look at
- Click on column name to add it to Apply condition input area
- Fill your condition by placing appropriate operator and value

Condition has a form:
Table.Column <operator> '<value>'

Leave spaces between Table.Column operator value

Supported operators: >, <, >=, <=, like

To match a pattern use %, e.g. TEST% will match TEST with everything else

Several conditions can be grouped together using AND, OR and brackets
  e.g., condition1 AND (condition2 OR condition3)
Finder under the hood

- Auto-load schema from DB using SQLAlchemy
  - store it into the cache
  - send them via AJAX to the web form
- Establish a short path between Table A to Table D using foreign key relationships
  - Build query, validate applied condition (where clause)
  - send back results with a form to lookup dataset out of your query results
- iterate if necessary
Summary

- Web development using python & AJAX is simple as 1,2,3
- Python: a lot of tools around
- AJAX: variety of toolkits is available, pick right one for your needs
- We used stackable design and Model-View-Controller architecture
- concentrate on physicists needs