Rapid web development using AJAX and Python

A. Dolgert, L. Gibbons, V. Kuznetsov

Cornell University, Ithaca, NY 14853, USA

E-mail: `vkuznet@gmail.com`

Abstract. We discuss the rapid development of a large scale data discovery service for the CMS experiment using modern AJAX techniques and the Python language. To implement a flexible interface capable of accommodating several different versions of the DBS database, we used a “stack” approach. Asynchronous JavaScript and XML (AJAX) together with an SQL abstraction layer, template engine, code generation tool and dynamic queries provide powerful tools for constructing interactive interfaces to large amounts of data. We show how the use of these tools, with rapid development in a modern scripting language, improved the scalability and usability of the the search interface for different user communities.

1. Introduction

Some of the most visible and successful information retrieval systems are available through web browsers. Recently Asynchronous JavaScript and XML, or AJAX, significantly improved how users interact with web interfaces. The Google maps service is an excellent example of how AJAX improves interaction between users and large amounts of data.

The Data Bookkeeping System (DBS) [1] has been developed to catalog all CMS data, including Monte Carlo and detector sources. Simulations and the detector will create about 5 PB a year, all of which should be accessible to physicists. The DBS represents a service whose primary responsibility to answer questions about data existence, their characteristics, description and location in CMS. We developed a web service, called data discovery [2] which provides a front-end interface for physicists to communicate with DBS. It serves as a specialized search engine to lookup CMS data. Here we present details of its implementation based on a stack of software components written in python and decorated with AJAX technology to improve usability of the front-end interface.

2. The search task

The CMS Event Data Model (EDM) [3] represents a hierarchy of data objects optimized for processing, transfer and storage. At the highest level of the EDM, the Primary Dataset is a classification for the trigger stream or Monte Carlo type that is the source of individual Datasets. Each Dataset contains Blocks of similar files, where Blocks are the primary entity for data transfer. Copies of Blocks on different sites have the same Logical File Name. When a Block is created by a step of the processing chain, from RAW to DIGI, SIM, and RECO, it is assigned a Data Tier of the same name. Each of these files is linked to the detector conditions during its creation, the version of the application that created it, the configuration parameters for its processing, and more. The distributed nature of data management and complexity of data flow introduce a barrier for average users. Such complexity motivated us to start developing a single data discovery service.

We identified three main roles of users to access the service, Physicists, Production Managers, and Run Managers [4]. A Physicist might ask, "I would like to find Monte Carlo data produced with a certain software release for the $t\bar{t}$ sample." A Production Manager would see datasets in the DBS as traces of a successful workflow. These mental models, of physics collisions and algorithms or of distributed computing, are very specific and very different from the Event Data Model.

The presentation model of the search interface mediates between these two. The implementation of this interface was the most complex task during the development cycle. We explored a variety of layouts to show the enormous amount of information in the DBS. The search interface presents different search strategies to each role, in both a novice and advanced form. There is one format that is tailored precisely to the main use case of each role, such as dataset summaries for Physicists containing the location, number of events, number of files, and logical filenames. Production Managers see, in addition, the person who made the dataset with a link to contact information and a time stamp of when the dataset was created or modified. There is also a less directed form of search which forms more arbitrary queries of the DBS.

On the one hand, we have the complexity of the EDM and ongoing changes in its semantics. On the other, we have a user group whose geographical distance makes it difficult to do direct testing of the interface. After initial focus groups and a few interviews, most user experience data has to come from log analysis and helpful emails. Lack of certainty and the complexity of the search task put pressure on the development environment.

3. Python and AJAX

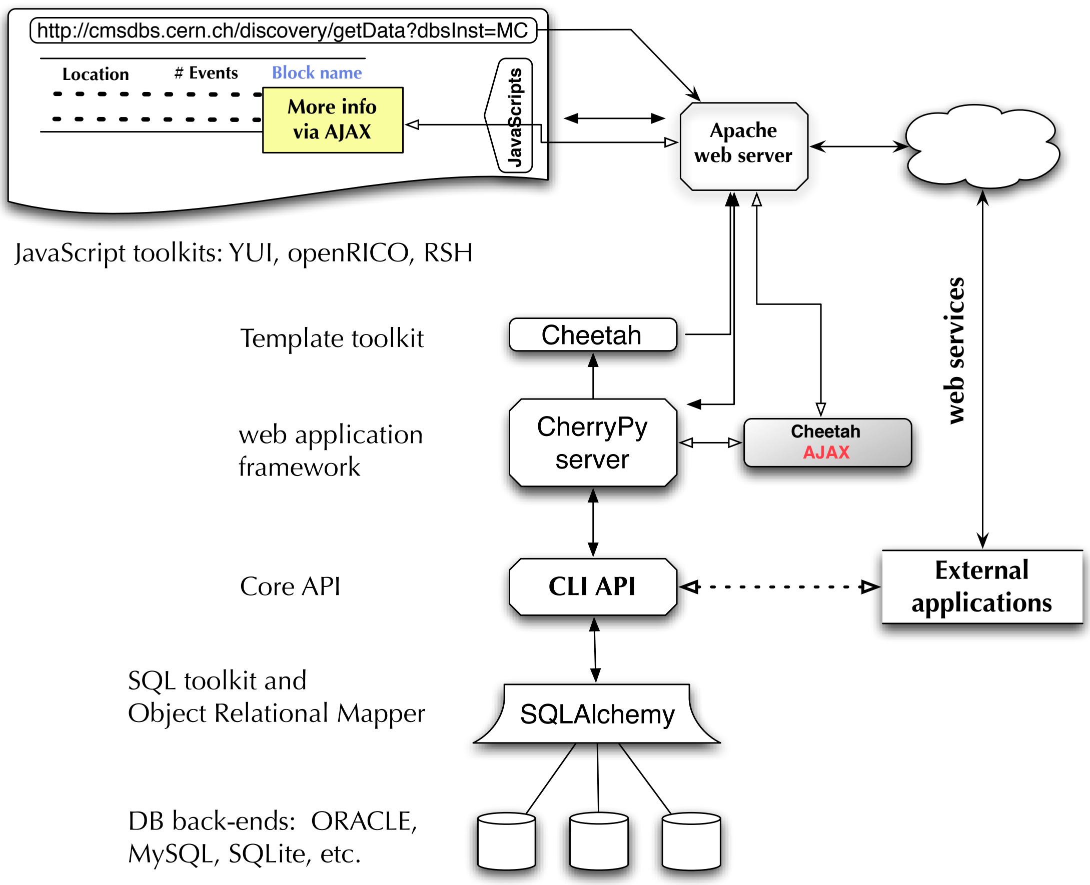
The idea of rapid web development using python and AJAX was driven by two main factors, a fast turn around with changes, including user interface, use cases, DB schema, etc., and support of user interaction with the DBS. Python had a great set of tools to accommodate our tasks. Among a large number of possibilities we choose the following components, see Fig. 1:

- SQLAlchemy [5], the python SQL toolkit and Object Relational Mapper, as SQL abstraction layer, which provides us transparent access to different DB back-ends (ORACLE, MySQL, SQLite);[1]
- CherryPy [6] the python web application framework;
- Open RICO framework [7], a JavaScript libraries which served all AJAX communications between web server and DB back-end;
- Really Simple History (RSH) JavaScript library framework [8], for session based history of JavaScripts user calls;
- Cheetah template framework [9], written in python for presentation layer of web pages.

It worthwhile to mention that those components were already used in other open source python web development frameworks [10, 11, 12]. Those web application frameworks were designed to accomotate all user needs, including data management. In our case the data model was designed and developed independently and data discovery was designed to be a read-only service. When it was feasible to adopt one or another web application framework we leaned towards re-using individual software components.

This stack of software components fits very well in the model-view-controller (MVC) architecture where we separated our data model from presentation layer. As we mentioned already, we used a SQL abstraction layer. Even though, depending on implementaion, it can be

[1] We woule like to note here that SQLAlchemy provides a full Object Relation Mapper (ORM) capability but it was not worthwhile in our case since the schema and DBS server were developed and deployed independently.



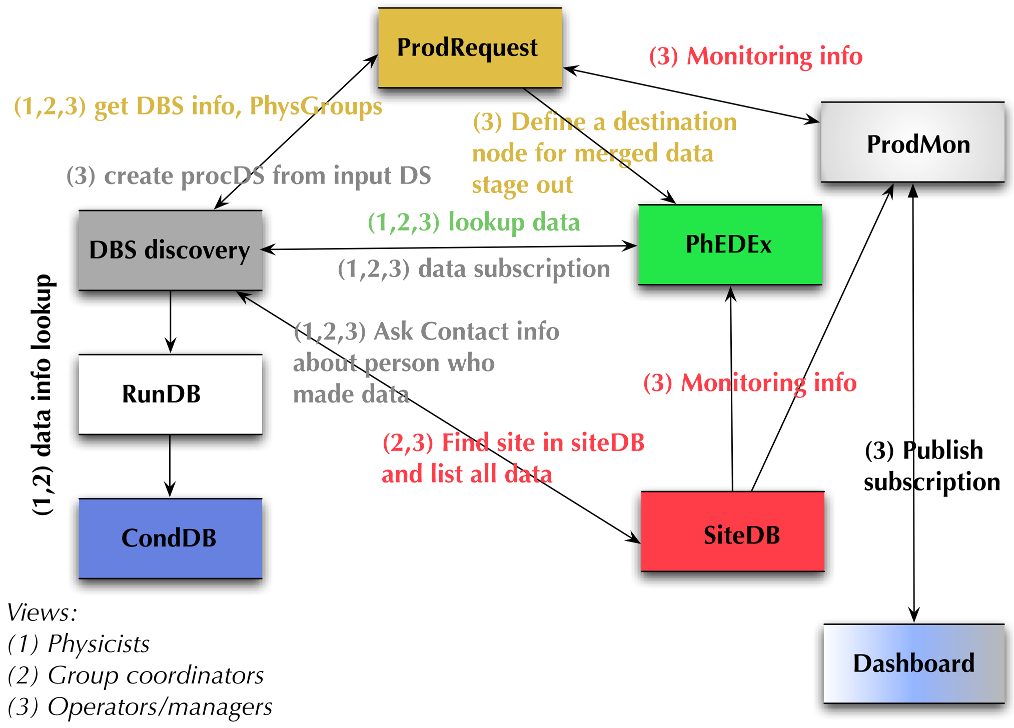
http://cmsdbs.cern.ch/discovery/getData?dbsInst=MC
Location
# Events
Block name
More info via AJAX
JavaScripts
JavaScript toolkits: YUI, openRICO, RSH
Apache web server
web services
Template toolkit
Cheetah
web application framework
CherryPy server
Cheetah AJAX
Core API
CLI API
External applications
SQL toolkit and Object Relational Mapper
SQLAlchemy
DB back-ends: ORACLE, MySQL, SQLite, etc.

Figure 1. Data discovery stack of software components.

slow with respect to plain DB access, our access pattern was not too high. If we take the size and distributed nature of CMS collaboration we would expect not more then a few hundreds access calls during a working day per user, with ∼100 users per time zone. The SQLAlchemy also took care of pooling database connections. Even though we did not yet come up with a query cache it can be done very easily. We also auto-load the DB schema from back-end DB which not only simplifies our life with frequent DB schema changes, but allows us to create a dynamic advanced query interface, discussed below.

At presentation layer the usage of template engine was a big plus due to fast changes in user requirements and use cases. We were able to redesign the entire page layout on a daily basis. Plus python-based tools provide a whole language semantics support to this layer. For example, we were able to pass from the core API dictionaries and lists and manipulate them in templates as if it were normal python code. Such abilities were used to create AJAX pages on a fly which constructed other pages with other AJAX calls. In the future, we would like to allow each user to customize both the search and data presentation pages [13].

In the context of the data discovery tool we were able to integrate different CMS web services via AJAX calls, see Fig. 2 For instance, the data presentation page uses AJAX to load asynchronously the status of related production requests. Sometimes, however, extensive use of AJAX technology leads to great confusion for the user. For example, originally we used AJAX to implement all calls to the DBS. While this successfully constructed pages with working menus and links, the “Back” button in the browser toolbar no longer worked. Because users are



ProdRequest
(3) Monitoring info
(1,2,3) get DBS info, PhysGroups
(3) Define a destination node for merged data stage out
ProdMon
(3) create procDS from input DS
DBS discovery
(1,2,3) lookup data
PhEDEx
(1,2,3) data subscription
(1,2) data info lookup
(1,2,3) Ask Contact info about person who made data
RunDB
(3) Monitoring info
(2,3) Find site in siteDB and list all data
(3) Publish subscription
SiteDB
CondDB
Views:
(1) Physicists
(2) Group coordinators
(3) Operators/managers
Dashboard

Figure 2. Integration of different CMS web services.

accustomed to the browser button, we were forced to redesign how we implemented the calls.

For a large part of the development cycle, we kept users happy by careful application of AJAX calls. One sucessfull example was keeping a session history of JavaScript calls on a page using the Really Simple History (RSH) tool [8]. Once the user found datasets, we recorded their interactions, such as viewing dataset details, on a web page. Another example is the ability to see "summary" and "details" views for the same dataset. A compact view, "summary", represents a snapshot of information about dataset short enough to help users to understand if it represents some interest and, if so, a "detailed" view was just one click away (thanks to AJAX).

3.1. Data discovery services

The CMS Data discovery service [2] was developed based on a well-known use cases, such as the ability to lookup datasets by selecting software releases, or lookup data for a given site. Even though those predefined queries cover a majority of use cases, an ability to place arbitrary query was desired to explore such a complex data bookkeeping system. To accommodate both scenarios we developed several search interfaces. In total we end up with five separate interfaces: "Navigator" a menu-driven interface, "Finder" an arbitrary query interface, "Analysis" an analysis-centric interface, "Sites" a site-specific interface and "Runs" a run-specific interface. Here we will discuss the benefits of using python and AJAX in those interfaces.

The first one, called "Navigator" represents a menu-driven approach where a web page contains a set of hierarchical menus. In our case it was: *Physics group, Data tier, Software release, Data types, Trigger Stream, Sites.* We chose this order based on use case analysis of how physicists search for data. A simple hierarchy of menus does not work well if it does not narrow possibilities based on user choice. By using AJAX calls we provided this ability in menus. For instance all menus had “Any” as a default choice. Once the user makes a selection, underlying menus were updated on a fly, narrowing the results in submenus.

The “Finder” represents an opposite search interface. Instead of guiding users through a set of selections we expose a relevant part of the DBS schema to them and provided an interface to make a data selection. To accomplish this task we used an abstract SQL layer (via SQLAlchemy [5] written in python) by auto-loading the underlying DB schema in our application. This approach has several advantages, such as being independent from schema changes and being transparent to DB back-end. ORACLE and MySQL were used at Tier0,1 and Tier 1,2, respectively. We should stress here that we have not had any performance issue due to the relatively low rate of queries. Using AJAX technology we push the auto-loaded schema to the web page via a treeview control and allow users to construct arbitrary queries, by selecting table columns and placing query constrains, without knowing SQL language and schema relationships. Under the hood, we used foreign keys of auto-loaded schema to establish a shortest path between selected table columns from a user request. Once the query was constructed and executed, the resulting page was presented to the user in a table format with the ability to place another query, such as lookup a processed and analysis dataset of out the results page.

The “Analysis” interface is the most uncertain one and still under development. A fast turnaround with user requirements was necessary in development cycle. Therefore the benefit of using python and template engine was clear.

The “Runs” interface represents a mix of data retrieval from different CMS web services. Originally we showed users the information stored in DBS, but with time it became clear that it was not enough to satisfy users needs, especially for Run managers. This group of users required a monitoring capability on the discovery page where information from DBS, Run summary DB and PhEDEx CMS system could be presented simulteneously. The Run summary and PhEDEx information were retrieved by constructing a single AJAX call to their web services about runs retrieved from DBS. Both responses were treated independently and placed into appropriate HTML tags on a page.

The user’s query was also saved in a separate DB for later usage, and an interface to retrieve user’s queries was presented on the page as well. In addition, we added a command-line tool with the same functionality where users created an input XML or text file specifying what they want to lookup in DBS. Using an XML-RPC call, such a request went to the same web service and results were returned back in an XML form which were parsed back for users. This allows users to use this technique in command-line and other python applications.

The advantage of AJAX techniques allows us to build a query interface and interactive results pages which were able to interpret query results and place another query upon user request.

3.2. Web services integration

We were pleased to see that with success of data discovery the other CMS web tools were migrated to use the same model [14]. An additional step forward was done to create a framework, where individual services, such as data discovery or Site DB, were able to register within a single server. It is also apparent that personalization of web pages becomes feasible by using AJAX technology and under discussion. The ability to auto-load schema and the ability of our algorithm to construct arbitrary queries on demand were perceived with great interest by other web developers in CMS collaboration and can be applied to any DB and schema.

As can be seen from Fig. 2 the Data Discovery service was very well integrated with the rest of CMS web services, among them the Run summary DB, PhEDEx, Production request system, and SiteDB. Some information was automatically requested upon page load, for instance retrieval of transfer status and Run summary information such as run type, trigger information and the magnetic field value used during this run. And other information was only requested upon user action. For example, on the dataset output page we provide a link to the retrieval status of a dataset in the Production request system. The status fields were requested per single dataset or for all datasets shown on a page. In this case either single or multiple AJAX requests were sent to the Production request system. At the same time Data Discovery service also serves other CMS applications. For example, the Production request system asks Data Discovery which software releases are available, and AJAX helps to establish a communication channel between two subsystems. It is worth mentioning that the Production request system, Data Discovery and SiteDB were unified under a single mini-framework which is discussed elsewhere [14].

As we saw from the previous discussion, the usage of python and AJAX helped us to keep up with changes in user interfaces, our understanding of user roles, schema and changes and user interaction with a system. It also makes significant improvements in the data discovery search interface. The data discovery search interface is quite different from casual web page searches due to a clear understanding of user roles and the user's model of the data, as well as common search patterns. Given a predictable set of choices by the user, AJAX helps funnel a large set of information into a manageable size on a web page.

4. Conclusions

We discussed rapid web development using python and AJAX technology. It has proven to be a very productive tandem for the CMS data discovery service. The simplicity, flexibility and power of python allowed us to concentrate on the usability of the service, by re-writing different stack components in our software model. The modern AJAX technology gave our users an application-like behaviour of web interface and the ability to integrate multiple CMS web services, such as Production Request system, Site DB, run quality and summary DB, Condition DB and PhEDEx. Our final goal was to satisfy CMS users and provide them with the ability to quickly locate their desired data. The data discovery service provides menu-driven, application specific, arbitrary searches.

5. Acknowledgements

We would like to thank Chris Jones and Dan Riley for extensive discussions of various use cases involved in the CMS data model and data management. We also thank Lee Lueking, Anzar Ajaq, Vijay Sekhri for valuable discussions about data discovery requirements.

This work was supported by National Science Foundation and Department Of Energy.

References

[1] A. Afaq, et. al. "The CMS Dataset Bookkeeping Service", see proceeding of this conference.
[2] http://cmsdbs.cern.ch/DBS2_discovery/
[3] https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideFrameWork
[4] A. Dolgert, L. Gibbons, V. Kuznetsov, C. D. Jones, D. Riley "A multi-dimentional view on data retrieval of CMS data", see proceeding of this conference.
[5] http://www.sqlalchemy.org/
[6] http://www.cherrypy.org/
[7] http://www.openrico.org/
[8] http://www.onjava.com/pub/a/onjava/2005/10/26/ajax-handling-bookmarks-and-back-button.html
[9] http://www.cheetahtemplate.org/
[10] http://www.turbogears.com
[11] http://pylonshq.com/
[12] http://www.djangoproject.com/
[13] F. Yang, J. Shanmugasundaram, M. Riedewald, J. Gehrke, A. Demers, "Hilda: A High-Level Language for Data-Driven Web Applications", ICDE Conference, April 2006
[14] M. Simon et. al., "CMS Offline Web Tools", see proceeding of this conference.