Web Applications & Technologies at LCLS

Igor Gaponenko

In this presentation

- **Revisiting the ideas which were proposed ~3 years ago:**
  - http://www.slac.stanford.edu/~gapon/TALKS/2011_Jan_PPA_ComputingMeeting/AdvancedWebTechPCDS.pdf
- **Major applications**
- **The deployment environment**
- **Technologies**
  - the main focus on the JavaScript ecosystem
- **Frameworks**
- **On-going developments, plans and trends**

APPLICATION AREAS

Managing Experiments

- **Experiment Registry**
  - Configuring experiments at LCLS for DAQ &OFFLINE
- **POSIX groups @LDAP**
  - Managing members of experiments
  - Tracking group management activities of users **new**
- **Authorization Database**
  - roles and privileges
  - Access to data (files) and metadata (via Web)
- **Experiment Switch** **redesigned**
  - Experiment activation service for DAQ & OFFLINE

Managing Data and Metadata

- **File Catalog (iRODS)**
- **HPSS operations**
- **HDF5 Translation services**
- **Electronic LogBook** **redesigned**
  - A collection of Web-based instruments
  - Web services for:
    - "Grabber" UI tool (PyQt4) @ instrument control rooms
    - Command line messages/images injection tools
- **Run Summary Tables** **new**
  - To compete w/ Google Spreadsheets
- …

Engineering Databases

- **PCDS Inventory for Electronic Equipment** new
  - 1000 items as of today
  - 11 active users
- **NeoCAPTAR** – cable management system new
  - Project-based approach (169 projects, 1123 cables as of today)
  - Workflow support: ordering, labeling, fabrication, deployment, maintenance, etc.
  - Special labels generation/printing
  - 17 active users

Hosted third-party applications

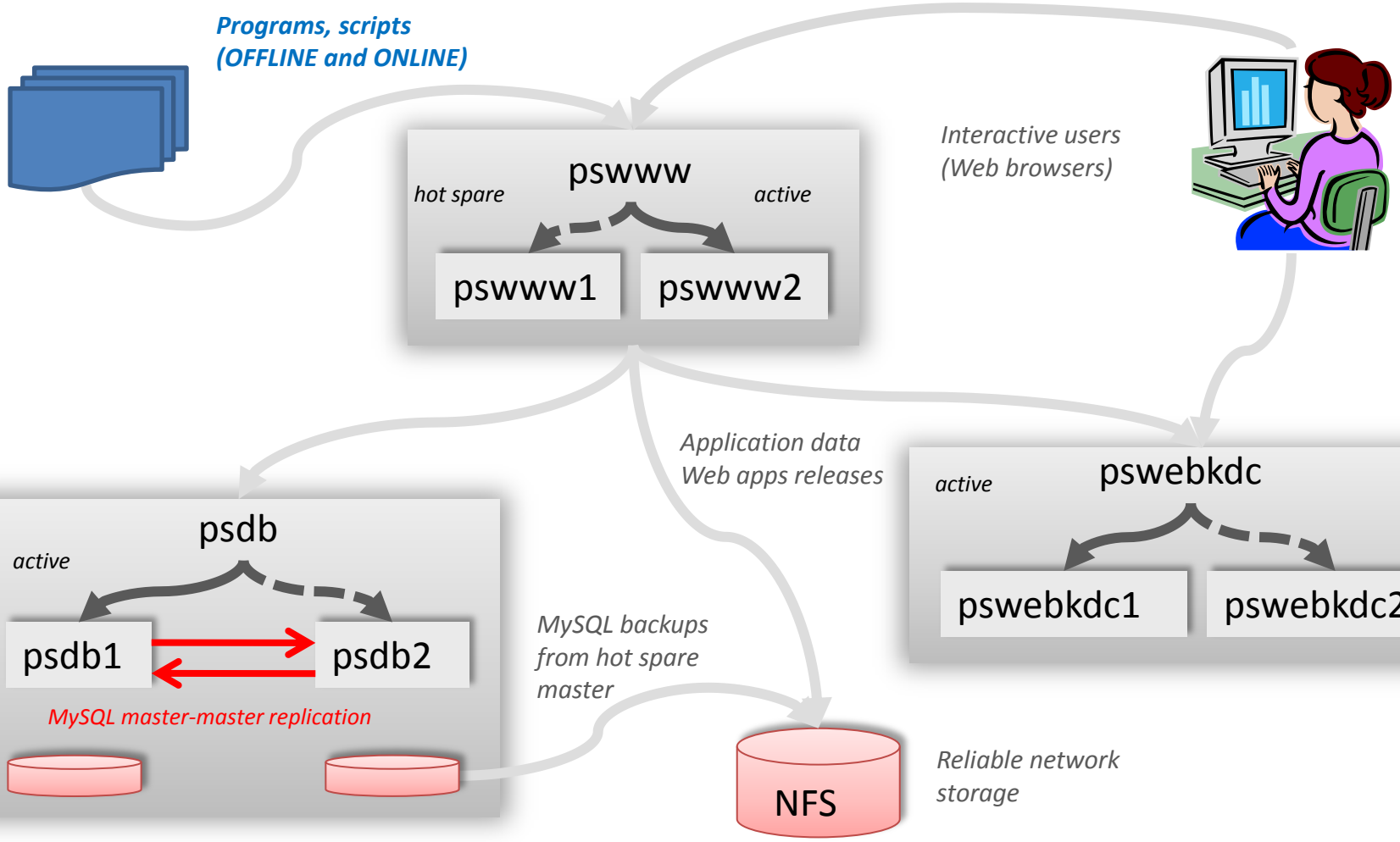
- **TRAC (Wiki)**
- **SVN (via Apache)**
- **Buildbot**
  - continuous code integration system
  - Release builds
- **Systems monitoring:**
  - Ganglia
  - M/Monit

new

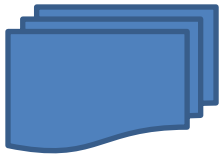
new

THE DEPLOYMENT ENVIRONMENT

24/7 (reliable infrastructure)



Security of Web apps



Programs, scripts
(OFFLINE and ONLINE)

Interactive users
(Web browsers)

AUTHENTICATION

Apache basic authentication

Kerberos authentication

WebAuth (single sign-on)

.htpasswd ← Apache → Kerberos

SLAC UNIX accounts

Web Applications | Web Services | Proxies

AUTHORIZATION

LDAP

LCLS POSIX groups

MySQL

Roles and privileges

TECHNOLOGIES

The Big Idea

The model has not fundamentally changed since 2011

- **A contemporary Web browser is a portable operating system**
  - It can run complex applications (JavaScript)
  - The JavaScript language supports asynchronous processes (via timers)
  - Applications communicate with Web services to load/save data
  - JavaScript renders data to a browser's window by modifying DOM (Document Object Model)
- **A computer on which a browser is run has plenty of resources to do more than just displaying a static HTML pages or images!**
  - **WebGL** (~=OpenGL)
  - **WebCL** (~=OpenCL)
- **Implementation:**
  - The markup is formed within the browser (NO HTML whatsoever generation on the server side)
  - Use Web server as a data source and a service provider
  - Single page (yet very complex) Web applications
- **Benefits:**
  - (Much) faster and more dynamic user interfaces
  - Separate data from presentation (towards the MVC paradigm)
  - Less data to transfer between the browser and the Web server

Why JavaScript

Use of the language has dramatically expanded since 2011

- **Billions of devices run JavaScript applications:**
  - All Web browsers (desktops, tablets, smartphones)
  - **Qt4**, **Adobe Acrobat**, **CAD** tools, etc.
  - Web servers and distributed systems (**Node.js**)

new

- **Naturally complemented by JSON (BJSON):**
  - THE data format for many **NoSQL** databases (**MongoDB**, **CoachDB**, etc.)
  - Web browser/ server communication
  - (much less) verbose compared with XML
- **Abundance of very fast interpretation engines:**
  - **Rhino**, **SpiderMonkey** (Mozilla), **V8** (Google), **JavaScriptOne** (Apple), **Chakra** (Microsoft), etc.

new

- **The language itself:**
  - Easy to learn
  - Yet powerful enough to support multiple programming paradigms:
    - Dynamic programming
    - Functional programming
    - Modular
    - OOP
- **An explosive growth of really useful libraries**

new

3rd party JavaScript libraries in use

- **Jquery** + plugins
- **Jquery UI** + plugins
- **UnderscoreJS**
  - for functional programming
- **RequireJS**
  - for modular programming
  - Provides a convenient mechanism for dynamic loading of modules

PHP

- **PHP 5.3**
  - Classes, interfaces, namespaces, functional programing
  - A huge collection of libraries for “everything”
  - Low threshold to learn and use
- **Used primarily to implement Web services:**
  - The middleware for accessing databases, LDAP, etc.
  - Object/Relational Mapping (mostly into JSON)
  - Implementing “Business” logic
  - Enforcing authorization
- **50k+ lines of code has been written (@LCLS)**
- **Issues/limitations:**
  - **Performance** (not as fast as Java)
  - (potentially) **Scalability** (haven’t tried the load balancing yet?)

Python

- **Pylons** (via Apache FCGI)
- The framework for developing Web services
- Provides routing for Web services URIs
- Limited use @LCLS
- Memory hungry
- So-so performance
- Questionable support status

FRAMEWORKS

What makes the development to scale

- **(Usually) The third Web application developed in a row triggers a search for a reasonable *framework***

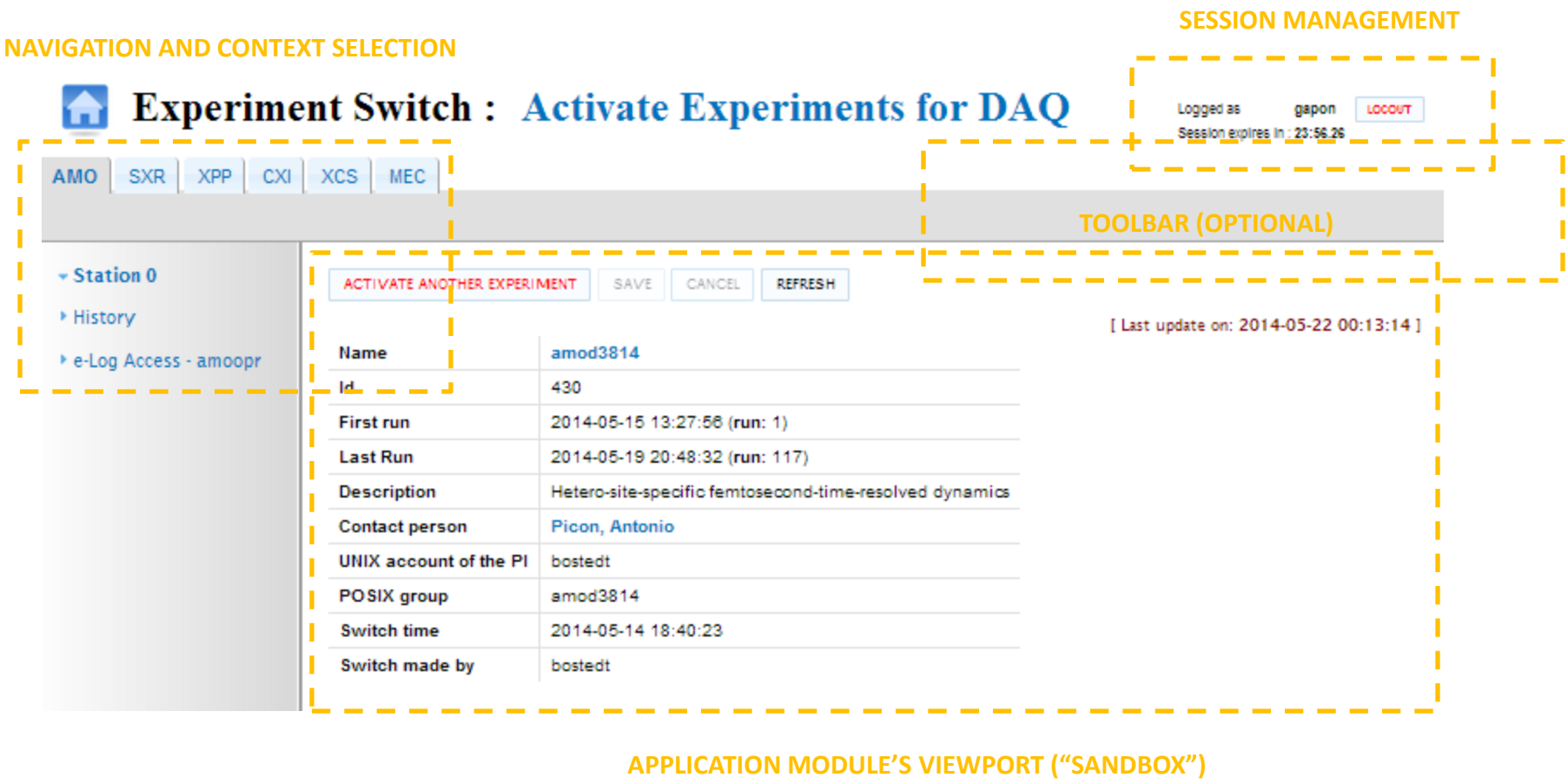
- **Two types of frameworks needed (in line with the “Big Idea”):**
  - For JavaScript applications
  - For Web services

- **Each addresses specific aspects of their domain**

Frontend Framework (JavaScript)

- **Developed in-house:**
  - Inspired by existing frameworks: **Yahoo UI 2.*** and **ExtJS**
  - Both were evaluated, neither seemed to be suitable (more discussion if needed?)
- **Defines a clean separation between an execution environment and application modules**
- **Provides managed viewports for the modules**
- **Takes care of the tedious tasks:**
  - Session management
  - General application layout management
  - Context management and navigation (tabs, menus) between modules
- **Provides services to the modules:**
  - Simplified integration with the backend Web service
  - Common error reporting
  - Communication mechanism between modules
  - Timer services (to implement processes and asynchronous activities within the Web browser)
  - Persistent configuration for modules (via Web services)

Visual Layout



Anatomy of the application module

```
function MyApplModule (p1, p2, p3) {

    // call the base class's c-tor
    FwkApplication.call(this);

    // @overload
    // the viewport is shown
    this.on_activate = function () { ... };

    // @overload
    // the viewport is hidden
    this.on_deactivate = function () { ... };

    // @overload
    // called on timer (1 Hz)
    this.on_update = function () { ... };

    this.render = function () {
        this.container.html( '<p>Show this in my viewport</p>' ) ;
    };
}
Class.define_class (MyModule , FwkApplication);
```

Configuring the Framework

```
Fwk.build({
  [ name: 'Tab 1',
      menu: [{
        name: 'Appl 1.1', application: new MyApplModule(…)},{
        name: 'Appl 1.2', application: new MyOtherApp()}
      ]],
  [ name: 'Tab 2',
      menu: ...
      ]]
});
```

Toward a library of widgets

- **The core idea:**
  - Making Web application to look & behave like the desktop ones!
  - Hence, no static HTML whatsoever
  - Replacing dynamically generated HTML with widgets
  - Keep using CSS to style the appearance and layout of widgets and applications
- **Why not to use an existing library?**
  - It just doesn't exist
  - **Jquery UI**: too low level and not so reach, bounds itself to DOM
  - **ExtJS**: is a great library which gave me a lot of inspiration. Sadly, it's too business-oriented (tabular representation of data is the main focus). I have personal issues with their styling (BTW: "It's About Time" is based on ExtJS).
- **Widgets wich has:**
  - **Widget** (the base class)
  - **Table**, **SmartTable**, **CheckTable**
  - **StackOfRows**
  - **RadioBox**
  - **Framework** itself has many widgets in its implementation
- **Demos to follow…**

Server-side framework for Web services (PHP)

```php
<?php

/**
 * A simple Web service locating an experiment
 * by its numeric identifier and returning
 * a JSON object.
 */
require_once 'dataportal/dataportal.inc.php' ;

use \DataPortal\ServiceJSON ;

ServiceJSON::run_handler ('GET', function ($SVC) {

    $id = $SVC->required_int ('id') ;

    $e = $SVC->regdb()->find_experiment_by_id ($id) ;
    if (!$e) $SVC->abort ('no such experiment') ;

    return array (
        'id'   => $e->id() ,
        'name' => $e->name()
    ) ;
}) ;
?>
```

The framework for PHP

- **Parsing and de-serializing input parameters**
- **Serializing results into JSON**
- **Standard error reporting mechanism**
- **Database connectors**
  - Automatic transaction management
- **What's missing:**
  - Routing for Web services URIs

ON-GOING DEVELOPMENTS, PLANS AND TRENDS...

On-going developments

- **Migrating older applications to the new Framework model and widgets**
- **Expanding a collection of Widgets**
- **Migrating applications to the dynamic module loading model (RequireJS)**
- **There are many requests to add more functionality to the Electronic Logbook & Run Summary tables**

Next steps

- **Three directions within JavaScript ecosystem:**
  - Finalizing the Model-View-Controller (**MVC**) separation
  - Adding support for mobile clients
  - Addressing the growing number of use cases for the **full-stack** JavaScript:
    - **NodeJS**
    - **MongoDB** or **CoachDB**
- **More sophisticated server-side framework for the PHP-based Web services:**
  - Dispatching and routing