WormBase Reimplementation

23 January 2008

What We Have Now

Hardware Platform

- ✓ 24/7/365 uptime (about)
- ✓ Sophisticated caching/load balancing architecture
- ✓ Horizontal scaling to accommodate demand

Software Platform (I)

- ✓ Monolithic CGIs in Perl
  - ✓ Fast development, few strictures
  - ✓ Many devs, many styles
    - ✓ Code degradation (eg inline evals)
    - ✓ Many hacks == many bugs

Software Platform (II)

- ✓ Intermixed application / display logic
  - ✓ Difficult to change logic w/o breaking display
  - ✓ Locked in to one display
- ✓ Lots of wheel reinvention
- ✓ Doesn't scale well: performance or team

What We Need

✓ Lean on Open Source; extend our resources

✓ Flexible / Extensible / Maintainable

✓ Facilitate usability and design research

✓ **We need a web framework**

Web Frameworks

- ✓ Make common web tasks easy:
  - ✓ forms, sessions, authentication, url mapping
- ✓ Enforce project structure and coding style
- ✓ Should be popular and have active community

Due Diligence

- ✓ Ruby on Rails
- ✓ CGI::Application
- ✓ Maypole
- ✓ Catalyst
- ✓ Reaction

Catalyst

- ✓ Model-View-Controller separation
- ✓ Common web tasks way easy
- ✓ Intuitive directory layout
- ✓ Built in test server, code profiling
- ✓ Breaks URL -> script paradigm; URLs are actions

Catalyst Drawbacks

- ✓ Documentation
  - ✓ New book, mailing lists, IRC
- ✓ *Very* flexible
  - ✓ which templating system
  - ✓ which configuration format
  - ✓ which action structure

Catalyst Advantages

- ✓ Common tasks already solved
- ✓ Plugin architecture
- ✓ Code split by design
- ✓ Scales well with multiple developers

Introduction to MVC

✓**Controllers** contain application logic and handle interactions with users

✓**Models** interact with data stores

✓**Views** contain display logic

Crash Course: Controllers

```
# Controllers define actions

package WB::Controller::Gene;

use base 'Catalyst::Controller';

# URL: /gene/name

sub name : Local {

    my ($self,$c) = @_;

    $c->stash->{name} = $c->req->params();

 }
```

Crash Course: Models

```
# Models interact with datastores

package WB::Model::Gene;

use base 'Catalyst::Model';

# Fetch the name of the object

sub name {

    my ($self,$c) = @_;

    my $object = $c->fetch_object($c->req->params);

    return $object;

 }
```

Crash Course: Views

Gene Summary

<h2>[% name %]</h2>

EASY

- ✓ Common web tasks
  - ✓ sessions, authentication, authorization, url mapping
- ✓ Ajax integration
- ✓ Flexible layouts (TT, Mason, etc), multiple formats (PDF, XML, HTML)

First Steps

✓ Keeping the baby

✓ Current site structure as guide

✓ Sections eq **Widgets**

✓ Subsections eq **Fields**

Widgets

- ✓ Widgets correspond to sections
- ✓ Widgets defined in configuration file
- ✓ User-based customization for every element
- ✓ Conditional statements in TT control display

Conversion steps

1. Add configuration for widgets and contents
2. ~~Write simple Controller actions (for now)~~
3. Strip logic and move to Model::*
4. Write templates (optional)

View Granularity

- ✓ Each **page** is a template
- ✓ Each **widget** is a template
- ✓ Each **field** is a template
- ✓ All wrapped when rendered (minimal buffering; Ajax)

View Features

✓ Any **widget/field** is URL-accessible

✓ Dynamic / Lazy loading

✓ Multiple Formats: PDF, XML, HTML

✓ Web Services: XML-RPC, SOAP, REST

e.g.: Configuration

```
gene => {

       widget_order => [ qw/identification location expression/ ],

        widgets      => [

                    identification => [

                              qw/description

                                 ncbi_kogs

                                  species

                                   other_sequences

                                   ncbi

                                   gene_models

                                   cloned_by/],
```

For free

actions: /gene/*/identification

views: per field, widget, page

Going Forward

End of February:

- [ ] Define stash structure
- [ ] CGIs->WormBase::Model

End of March

- [ ] Refined Controller/View logic

Future Calls

- [ ] Documentation
- [ ] Configuration basics
- [ ] Models: structure, stash
- [ ] Controllers: dynamic actions, root actions
- [ ] Views: design decisions, flexibility