Building Applications with OpenInteract2

Chris Winters

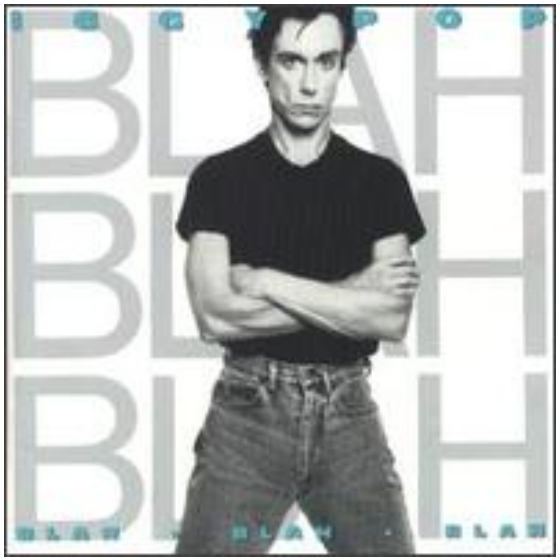
Optiron Corporation

June 17, 2004

Part 5: Advanced application features

- Add comments
- Add behavior to SPOPS object
- Add an observer

Simple commenting application in OI2



- We mean **simple**
- ...no threading
- ...no karma, meta-moderation

Any object can be commented on

- But here's something interesting!

Any object can be commented on

- But here's something interesting!
- Any SPOPS object can be a thread
- ...web pages, news items, beers, pubs...

Any object can be commented on

- But here's something interesting!
- Any SPOPS object can be a thread
- ...web pages, news items, beers, pubs...
- (threading == comments on comments? Madness!)

Comment display is a component

- Even cooler: adding comments is a one-liner

Component is just an action!

- Something new in our plugin: `OI.action_execute`
- Components are just actions
- ...and you can call any action from here

Component is just an action!

- Something new in our plugin: `OI.action_execute`
- Components are just actions
- ...and you can call any action from here
- Pretty handy that actions just generate content...

Add comments

Add comments to beer, pub, brewery display

Code to add

- People like to see what beers were added recently
- Add a method `get_recently_added()` to beer

I'm just a class

- It's really just a class
- Since you're subclassing you have access to everything
- ...and the outside world won't know the difference

But wait, something we forgot

- We don't have any date field on our 'beer' objects!
- So let's add one
- ...and also make it automatically treated as a `DateTime` object

SPOPS automatically finds fields

- Since `field_discover` is enabled we don't need to declare the field
- ...just a `convert_date_field` entry
- Note: this is another OI2 shortcut for an SPOPS rule

Add the method

Now the method is easy...

All actions are observable

- Observer/Observable a common and simple pattern
- ...just an message passing mechanism
- ...decoupling message broadcasters from consumers
- But broadcaster still knows it's broadcasting
- (...in some systems this is hidden)

Possible to declare relationships

- Since the two pieces do not have to know each other, we can link with a third piece
- ...

Observations are simple

- Just an inherited method call:

```
package MyApp::AddUser;
use base qw( Class::Observable );
...
sub add_user {
  my ( $self, $user ) = @_;
  Persistence->add_user( $user );
  $self->notify_observers( 'new user', $user );
  return $user;
}
```

Observers almost as simple

- Common: define class with `update()` method
- ...can be sub or object as well
- Observation == type + parameters
- ... observation is contract between the observable and observers

A simple observer

```
package App::UserActions;

sub update {
  my ( $class, $type, $user ) = @_;
  if ( $type eq 'new user' ) {
    create_radius_account( $user );
    enable_user_website( $user );
    create_user_email( $user );
  }
}

App::AddUser->add_observer( __PACKAGE__ );
```

A simple use: catching management task messages

- Management tasks generate lots of status messages
- Do we want to display them as we run? All at the end?
- ...cannot foretell, therefore tasks are observable
- ...fire two types of observations 'progress' and 'status'
- (look at the code)

Subscribe to new beers

- We want to let people get notifications of new beers
- ...who wants to know this?

Subscribe to new beers

- We want to let people get notifications of new beers
- ...who wants to know this?
- ...who doesn't want to know this?

Simple subscription application

- To facilitate this there's a `simple_subscription` app

Simple subscription application

- To facilitate this there's a `simple_subscription` app
- ...made exclusively for YAPC

Just an email tracker...

- You can define list types (e.g., 'new beers')
- ...users can sign up for the lists
- That's it.

Creating the observer

The observer is pretty simple:

- Every time it gets a 'new beer' observation

Creating the observer

The observer is pretty simple:

- Every time it gets a 'new beer' observation
- fetch the subscribers

Creating the observer

The observer is pretty simple:

- Every time it gets a 'new beer' observation
- fetch the subscribers
- send them each an email

Attach observer to action

Just add them to `filter.ini`

Attach observer to action

Just add them to `filter.ini`

- ...this will probably get renamed to `observers.ini`

Attach observer to action

Just add them to `filter.ini`

- ...this will probably get renamed to `observers.ini`
- The file gets read in at startup:
- ...include the observer (filter)
- ...assocate with action

For more information

OpenInteract Home Page
http://www.openinteract.org/

Current docs
http://www.openinteract.org/docs/oi2/

This presentation
http://www.openinteract.org/yapc_2004/

Chris Winters
Optiron Corporation
chris@cwinters.com
http://www.cwinters.com/