Building Applications with OpenInteract2

Chris Winters

Optiron Corporation

June 17, 2004

Part 2: Add features to simple app

Some improvements to our simple application:

- another templating engine
- enable localization
- create a filter to externally modify the content

Add a templating engine

- OI2 supports multiple Content Generators
- The rule: Action shouldn't care what generator it uses
- Why? Actions only care about generating data
- They're the 'M' in MVC...
- ...so templating engines are externally configurable

Using HTML::Template

- Somehow, everyone does not adore the Template Toolkit
- Just for you: OI2 ships with `HTML::Template` support
- ...and `Text::Template` support

Using HTML::Template

- Somehow, everyone does not adore the Template Toolkit
- Just for you: OI2 ships with `HTML::Template` support
- ...and `Text::Template` support
- Also easy to add your own home-grown, half-assed system

Add new action

Now specify a new action but use a different content generator pointing to the same source

Associate template with action and task

- But we have a problem
- Method specifies tempate in `generate_content()` call
- So how will we move the data to our `HTML::Template`?

Get to know me!

- Easy: declare it in configuration
- Once you get to know OI2 you will see this again and again...

Modify the task and configuration

Move template name out of task into config

Restart and see the results

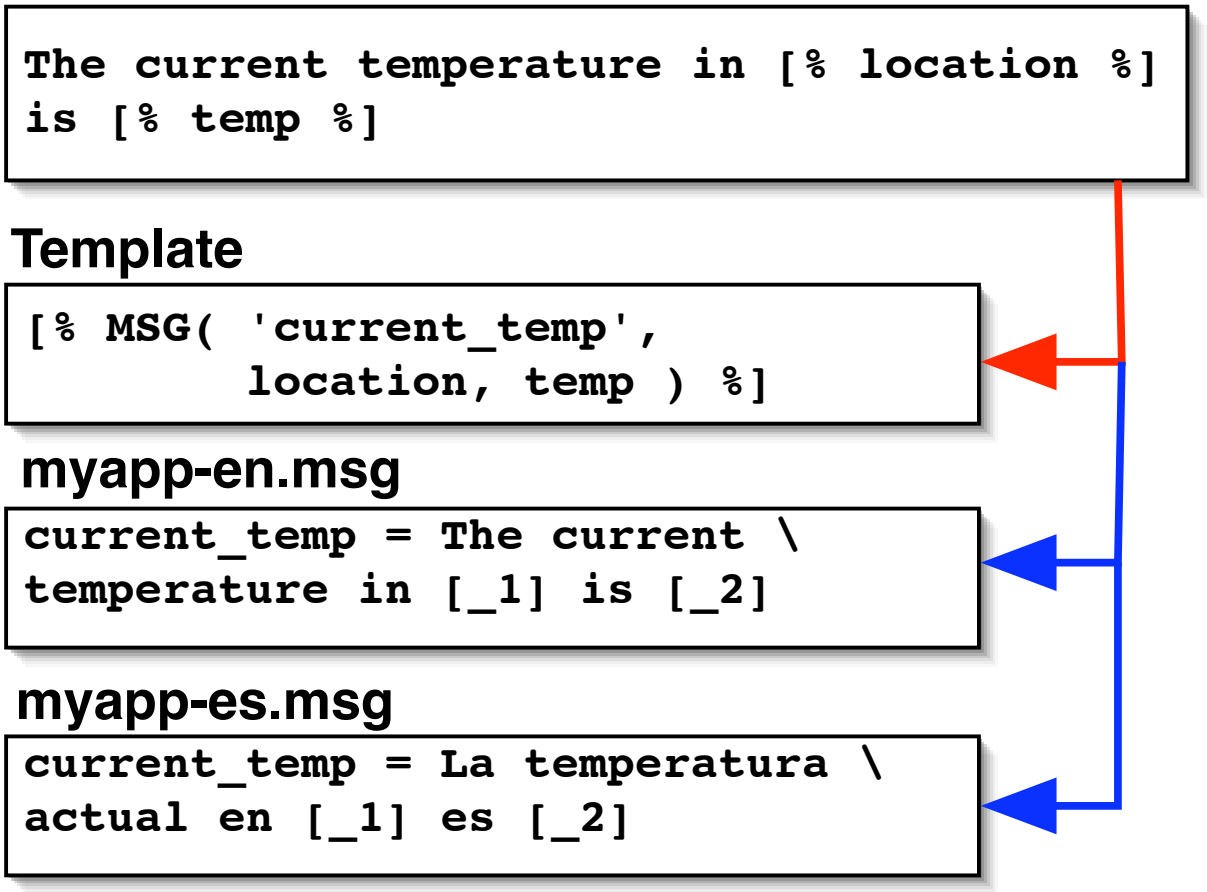
- Going to `http://localhost/mongers/`: same as before
- Going to `http://localhost/mongers_famous/`: same content but generated from `HTML::Template`

Localize your text

- Localization support through `Locale::Maketext`
- Every package defines its own set of keys
- ...but currently they're all in flat 'namespace'
- (Post 2.0 every package should have its own)

Replace text with tags

Each chunk of text gets a key and goes into message file



Reference tags from your template or code

- Support in Template Toolkit through `MSG` function
- Support in `HTML::Template`...?
- Code uses `Locale::Maketext` handle from request

Building a message file

- Each language has its own message file:
- `msg/mongers_en.msg`: English
- `msg/mongers_es.msg`: Spanish
- Language identified from browser (`Accept-Language`), user object or...

Customize the language reported

- Step in to set the language preference yourself
- ...makes it easy to implement 'Talk Like a Pirate Day'
- ...legitimate use: language by IP address

Generate the message files

Pull the text out of the template, generate message files

Restart and see the results

- Using browser with default language of 'English' shows english...
- ...and with 'Spanish' shows (mangled) spanish

Create a filter

- Action-generated content provides a hook for filters
- Step in after action generates content but before it's returned
- Do what you want...

Long live Kevin Smith!

- I want to modify all content

on my site

- (yes, Apache2 does this)
- Every 'Kevin' should link to

View Askew website

- Naturally: no code changes desired

What is a filter?

- Under the covers, they're just observers (`Class::Observable`; more later)
- The code is dead-simple: implement `update()`
- Next, map filter to action in server configuration
- Declare filter in package or in server configuration
- The action is none the wiser

Make the filter happen

Create the code and enter the mappings

Restart and see the results

- All instances of 'Kevin' in our listing should be linked

It's not that hard

- Most of OpenInteract is glue
- We use standard modules for:
- ...sessions, persistence, templating
- ...localization, cookies, MIME parsing
- ...exceptions, email, archiving, logging, caching

Gluing lots of pieces together

- OI2 is actually lots of fairly manageable pieces
- We touched on a few of them in our example:
- Actions
- Content generators
- Localization handles
- Object relational mapping
- Management tasks

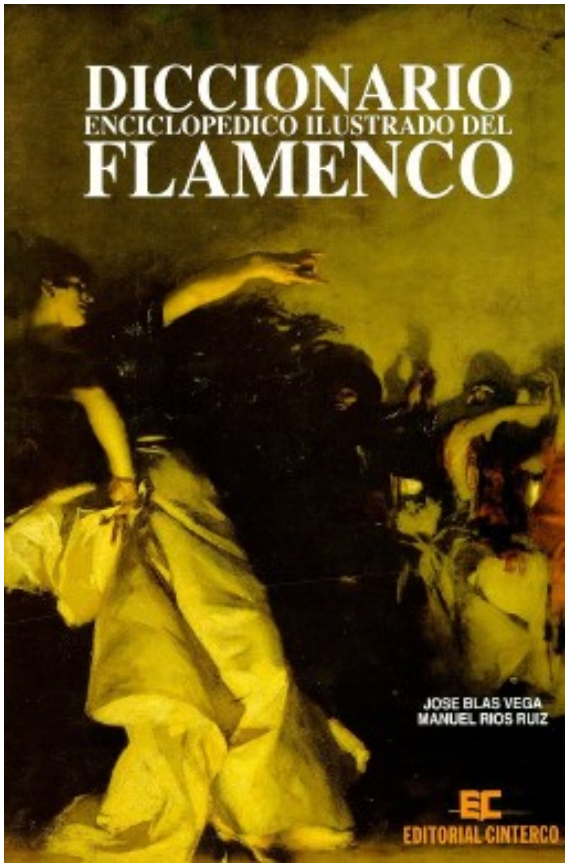
Every piece is configurable

- Some pieces can be configured by the package
- Some by the server
- Some by both - apps are no fun if you can't customize

Configuration is key!

- Java people have learned this fact well
- ...perhaps a little too well (EJBs, XML...)
- But don't throw the baby out with... well, you know

Quick architecture overview



This is not an architecture class, but we do need to define some terms.

Main pieces of the puzzle

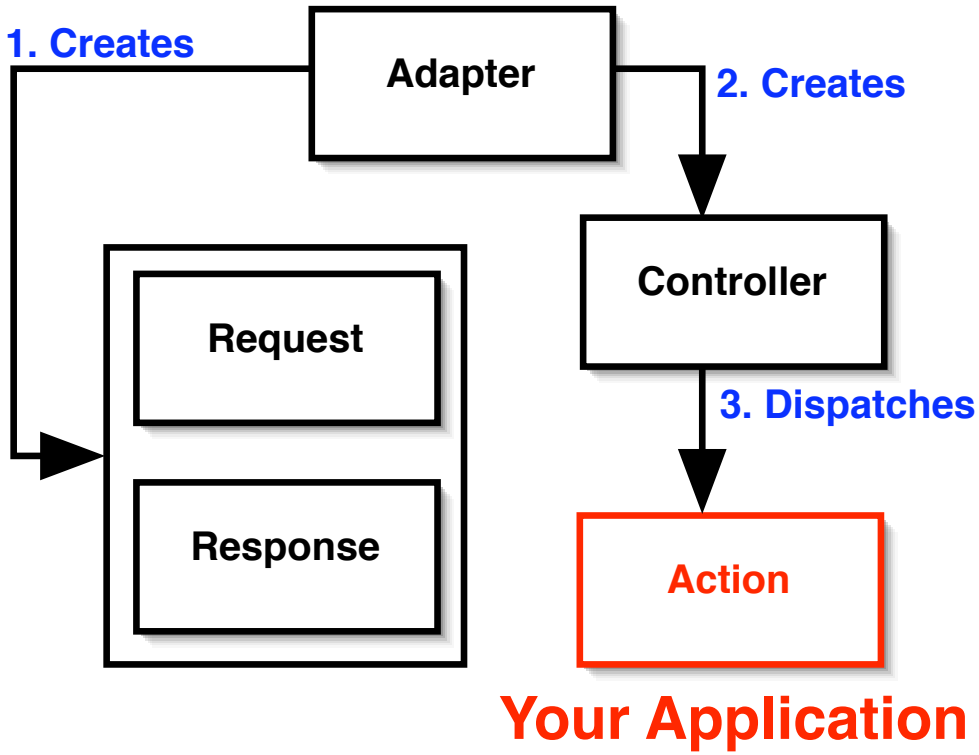
- Adapter: sits between server and OI2
- Controller: dispatches requests to actions
- Action: your application

How the pieces communicate

- Request: information from the user
- Response: data for the user including content
- Context: server configuration and gateways

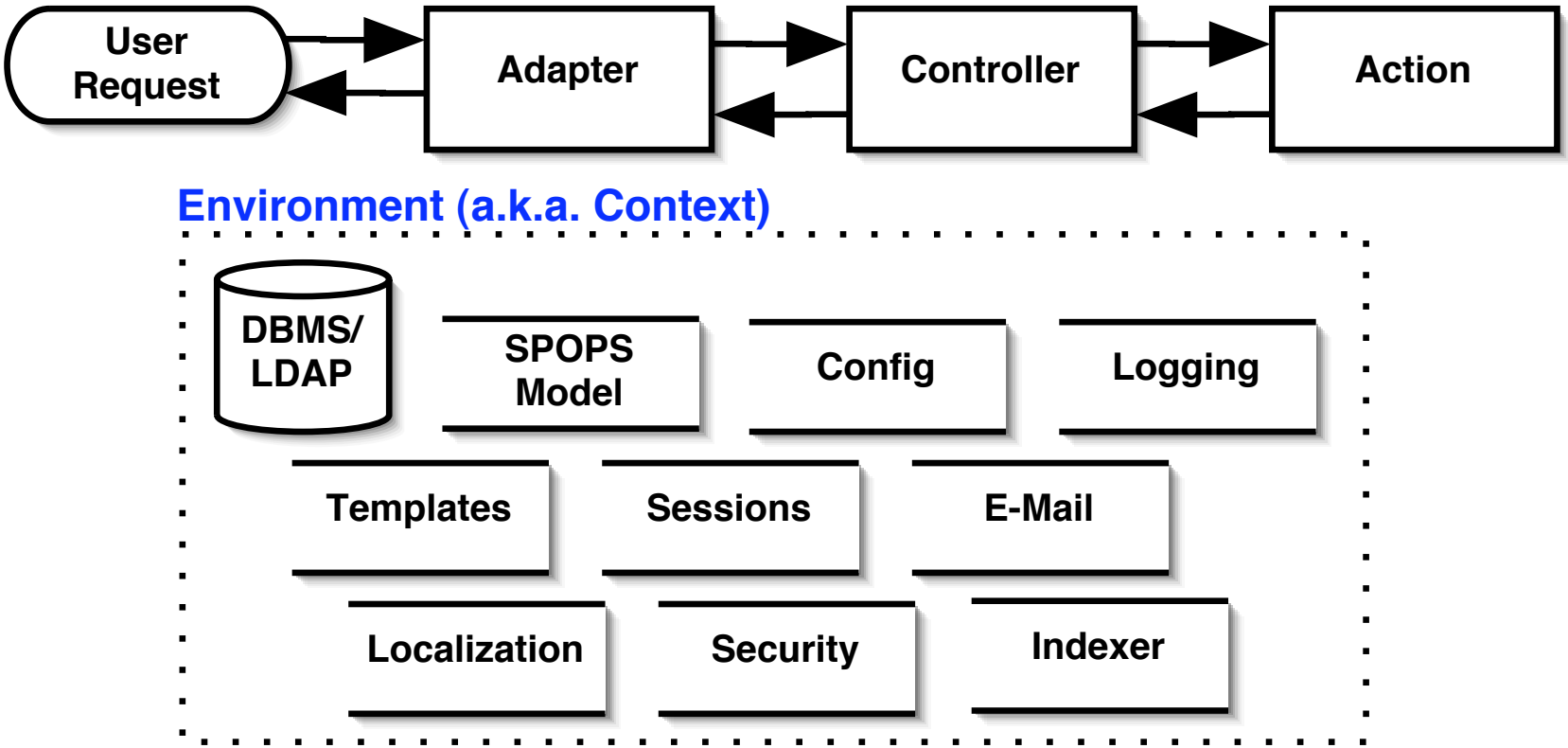
Creating a request

How the pieces are created per request



Another view with environment

Here are the pieces with the environment



Context is everywhere...

- The `OpenInteract2::Context` object is a singleton
- ...created at the beginning (server startup)
- ...mostly configuration + lookup methods
- ...available everywhere with imported `CTX` symbol

Some sample context usage

Large, but most methods variations on a theme

```
# SPOPS class lookup
my $mg_class = CTX->lookup_object( 'monger' );
my $monger = $mg_class->fetch_by_name( 'Kevin Lenzo' );

# Action lookup
my $action = CTX->lookup_action( 'news' );
my $latest_news = $action->execute({ task => 'latest' });

# Config lookup
my $ds_conf = CTX->lookup_datasource_config( 'main' );
print "Your DSN was:  ", $ds_conf->{dsn};
```

Adapter is simple...

- Everything begins with the adapter
- And the adapter is typically very simple
- Check out `Apache::OpenInteract2`...

...but it's deceptively simple

- Adapters are all about translating data
- ...data from the user (parameters, headers, uploads)
- ...data to the user (cookies, content, files)
- ...and all that hard work is done elsewhere

Anatomy of the request/response

- As a user you only deal with `OpenInteract2::Request`
- As a factory it creates new request objects...
- ...you tell it what type to create at server startup

Request hierarchy

How the pieces are created per request

At server startup:

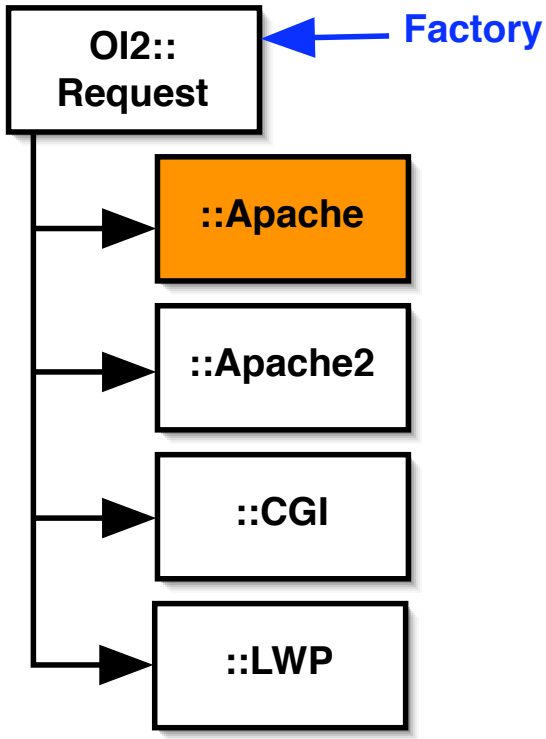
```
CTX->assign_request_type( 'apache' );
```

For each request:

```
my $request =
  OpenInteract2::Request->new( ... );
```

Retrieving in code:

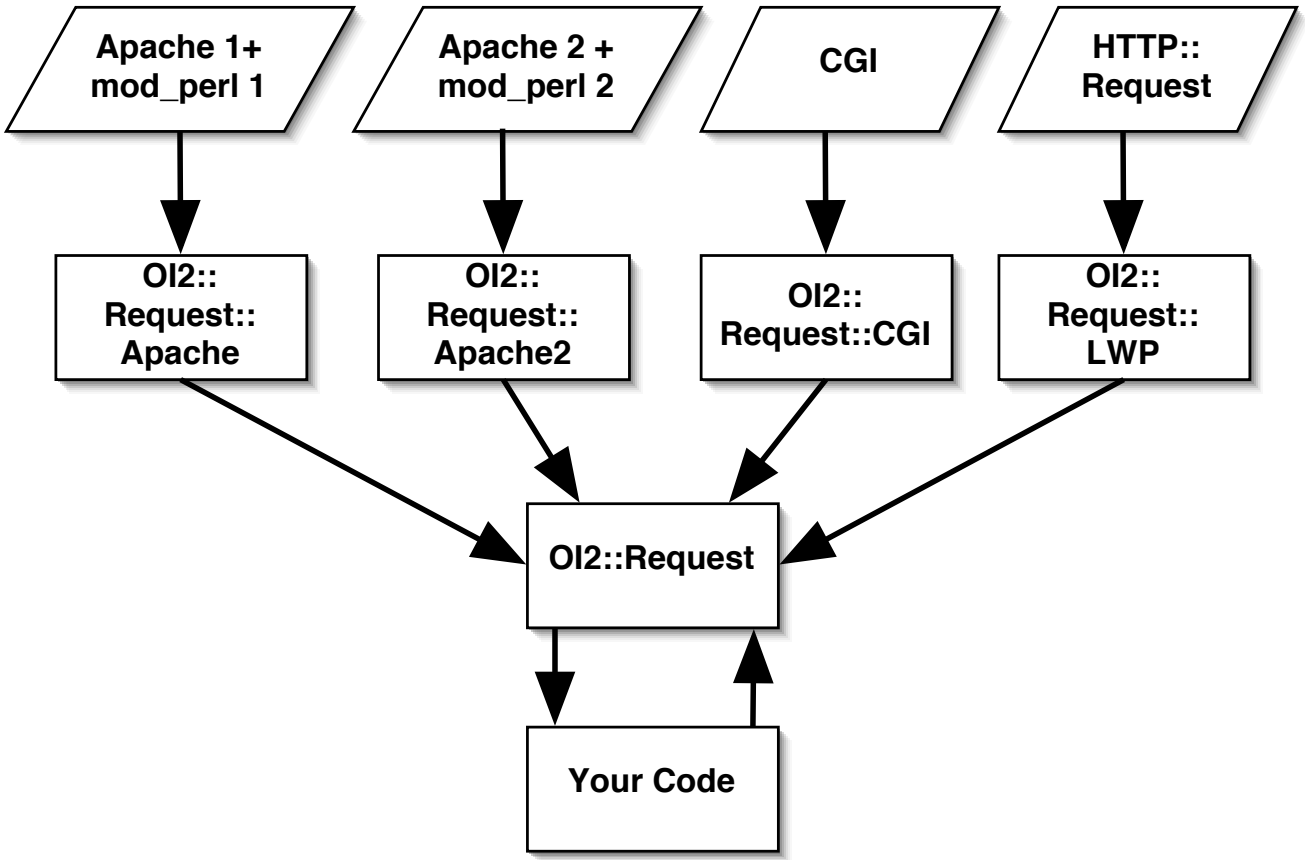
```
my $request = CTX->request;
```



Your code is insulated

- The request adapter insulates you from the platform
- ...you only deal with methods from `OpenInteract2::Request`
- ...no need to know how details are retrieved
- And your code should work everywhere!

Layers of insulation



Request contents

- Find out about the current user and groups
- ...parameters and uploads
- ...session data persisted between requests
- ...cookies, headers
- ...language preference + localization handle

Response hierarchy

- Hierarchy + setup is exactly the same
- ...except the direction of the arrows is reversed
- Response contains: headers, outbound cookies, status code
- ...oh yeah, and the content you worked so hard to generate

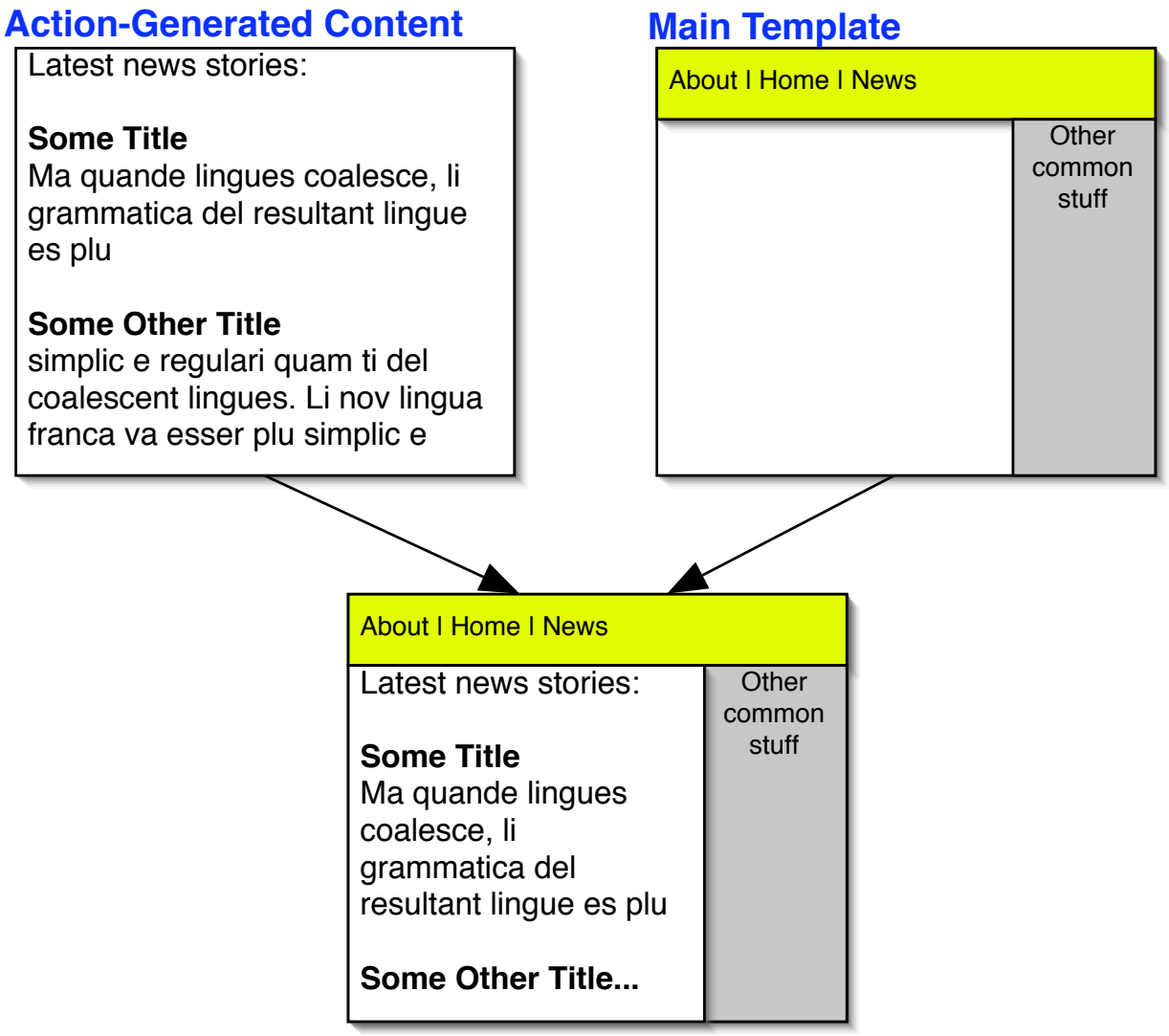
Controller dispatches to action

- After creating the request and response, the adapter creates a controller
- Controller inspects URL and finds corresponding action
- ...then executes the action which generates content
- ...the content is put into the response and control handed back to the adapter

Typical controller: main template

- This provides another place for us to step in
- ...between the action's content and the browser
- ...the perfect place for common elements like navigation

Merging content with the main template



Where the real action is...

- The action is where your application happens
- Actions only exist in packages
- ...and packages declare them in `conf/action.ini`

Name

```
[mongers]
class = OpenInteract2::Action::Mongers
task_default = list
is_secure = no
```

Where the real action is...

- The action is where your application happens
- Actions only exist in packages
- ...and packages declare them in `conf/action.ini`

Class to run

```
[mongers]
class = OpenInteract2::Action::Mongers
task_default = list
is_secure = no
```

Where the real action is...

- The action is where your application happens
- Actions only exist in packages
- ...and packages declare them in `conf/action.ini`

Properties

```
[mongers]
class = OpenInteract2::Action::Mongers
task_default = list
is_secure = no
```

Action = name + class + data

- Every action can be looked up by its name: `mongers`
- Typically this also maps to a URL: `http://.../mongers/`
- ...also: `http://.../Mongers/`
- ...also: `http://.../MONGERS/`
- But you can override the default behavior

Action + task = code

- An action name points to a class
- A task typically points to a subroutine
- `http://.../mongers/list/ == list()`
- `http://.../mongers/hello/ == hello()`
- ...but you can override the default behavior

Contract of an action

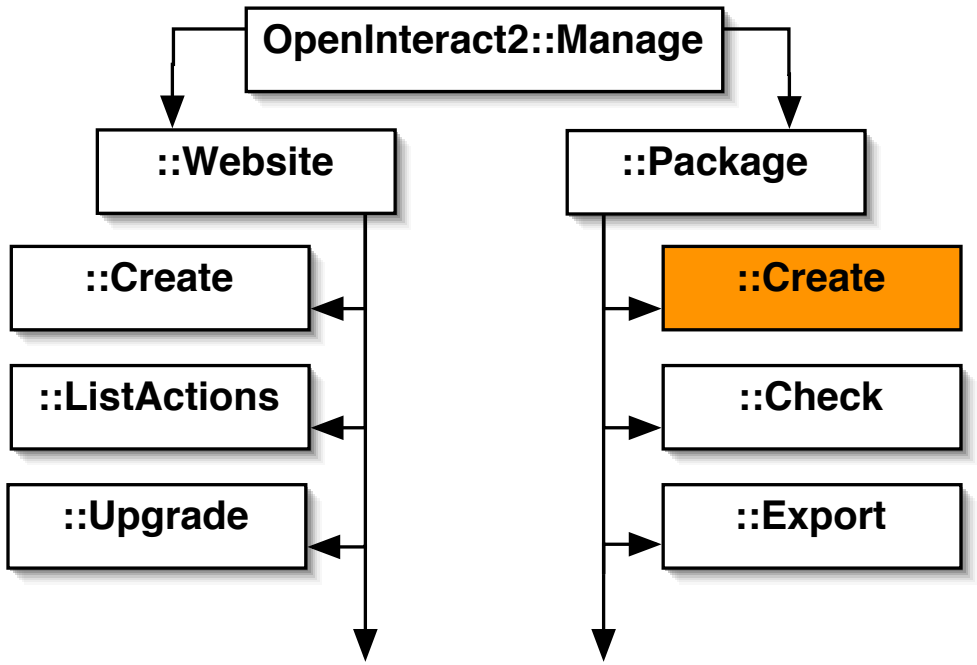
- Subclasses `OpenInteract2::Action`
- Return value is either content or an exception
- Default: task names == method names

A quick word about management

- OI2 comes with a framework for managing OI2
- Top-level and facade is `OpenInteract2::Manage`
- ...lots of tasks behind the facade, but standard way to run
- We were introduced to a few of these earlier...

Management hierarchy

The `oi2_manage` tool just wraps around these tasks:



Onto part three!

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/