PSR-7 and Action-Domain-Responder

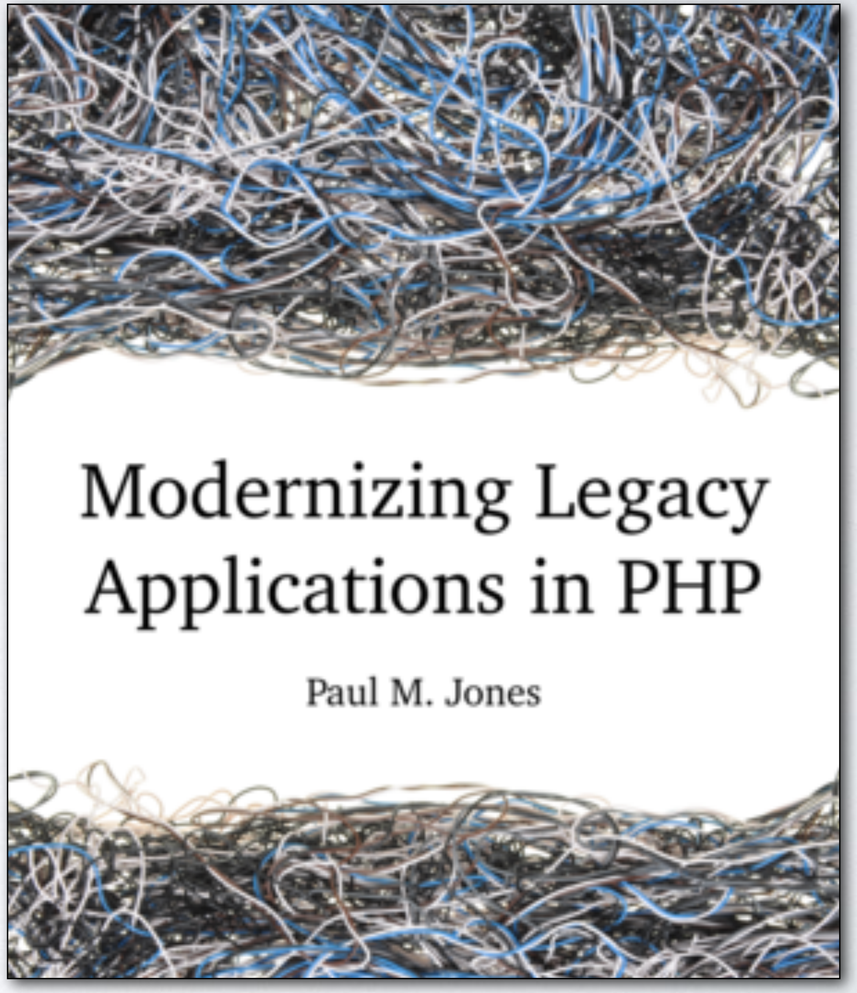
@pmjones on gab.com

http://paul-m-jones.com

pmjones.io/adr

About Me

- 8 years USAF Intelligence
- BASIC in 1983, PHP since 1999
- Jr. Developer, VP Engineering
- Aura, Zend FW, Relay, Atlas
- PHP-FIG Founding Member:
  PSR-1, PSR-2, PSR-4
- MLAPHP (http://mlaphp.com)



Overview

- PSR-7 "HTTP Messages" history and implementation
- (Re-)introduction to Action-Domain-Responder
- Using PSR-7 in ADR scenarios

PSR-7

Origins and Initial Draft

- Different projects in FIG have different HTTP objects
- Find common use cases, codify a standards recommendation
- Initial draft in Jan 2014 as client Request/Response interfaces
- Michael Dowling of Guzzle; referenced Buzz and Requests
- https://github.com/php-fig/fig-standards/pull/244/files

Initial Description

- Model of HTTP messages, with Request and Response interfaces
- Client, not server: PHP sends request and gets back response
- Set and get headers; PHP stream as message body; no URI interface
- Fully mutable: "immutable messages ... would not reflect what is currently being used by a majority of PHP projects."

Initial Drafter Departs

- Dowling steps down Aug 2014 (8 months)
- Lack of time and motivation
- Unconvinced of "one right way" for an interoperable interface

New Drafter Volunteers

- MWOP of Zend Framework adopts the proposal Sep 2014
- Has one right way in mind: Sencha Connect (later Expresss):
- "The reason I wanted to port Connect is this: an application consists of middleware. Each middleware is a callback that accepts a request, response, and a callback called next."
- `function ($request, $response, $next)`

Revised Draft

- Sep 2014 to May 2015
- Expands to include ServerRequest: $_GET, $_POST, $_ENV, etc.
- Solves long-standing omission in PHP
- Requires "immutability"
- Entirely unlike any member project implementations

Psr\Http\Message

- MessageInterface
- RequestInterface
- ResponseInterface
- ServerRequestInterface
- StreamInterface
- UploadedFileInterface
- UriInterface
- http://www.php-fig.org/psr/psr-7/

Immutability

- Cannot change the values inside an object
- Can get back a new instance of the object with changed values
- Isolates state, prevents "spooky action at a distance"

```
// mutable
$object->setFoo("new value");

// immutable
$newObject = $object->withFoo("new value");
```

ServerRequestInterface

```
/**
 * Return an instance with the specified body parameters.
 *
 * This method MUST be implemented in such a way as to retain the
 * immutability of the message, and MUST return an instance that has the
 * updated body parameters.
 *
 * @param null|array|object $data The deserialized body data. This will
 *     typically be in an array or object.
 * @return self
 * @throws \InvalidArgumentException if an unsupported argument type is
 *     provided.
 */
public function withParsedBody($data);
```

Guzzle & Zend Diactoros Implementations

```
public function withParsedBody($data)
{
    $new = clone $this;
    $new->parsedBody = $data;
    return $new;
}
```

Subverting "Immutability"

```
function one($request, $response, callable $next = 'two')
{
    // given `{"foo": "one"}`
    $request = $request->withParsedBody(json_decode($request->getBody()));
    echo $request->getParsedBody()->foo; // 'one'

    // invoke, and return from, next middleware
    $next($request, $response);

    // value has changed on same request!
    echo $request->getParsedBody()->foo; // 'two'
}


function two($request, $response, $next = null)
{
    $request->getParsedBody()->foo = 'two';
}
```

Quasi-Immutable

- You as the user must be careful to pass only immutable values
- Scalars and nulls; immutable objects; arrays with only immutables
- Cannot depend on enforcement of immutability elsewhere
- Even if fixed, message body Streams are still mutable
- http://paul-m-jones.com/archives/6400

Is PSR-7 Fatally Flawed?

- "It depends." Doesn't deliver on a core promise, but do you care?
- psr/http-message has 13M installs (791 deps) ... 12M are Guzzle: https://packagist.org/providers/psr/http-message-implementation
- symfony/http-foundation: 22.7M installs, 1433 dependents (Jul 2011)
- Be aware of its imperfections. Pick your tradeoffs.

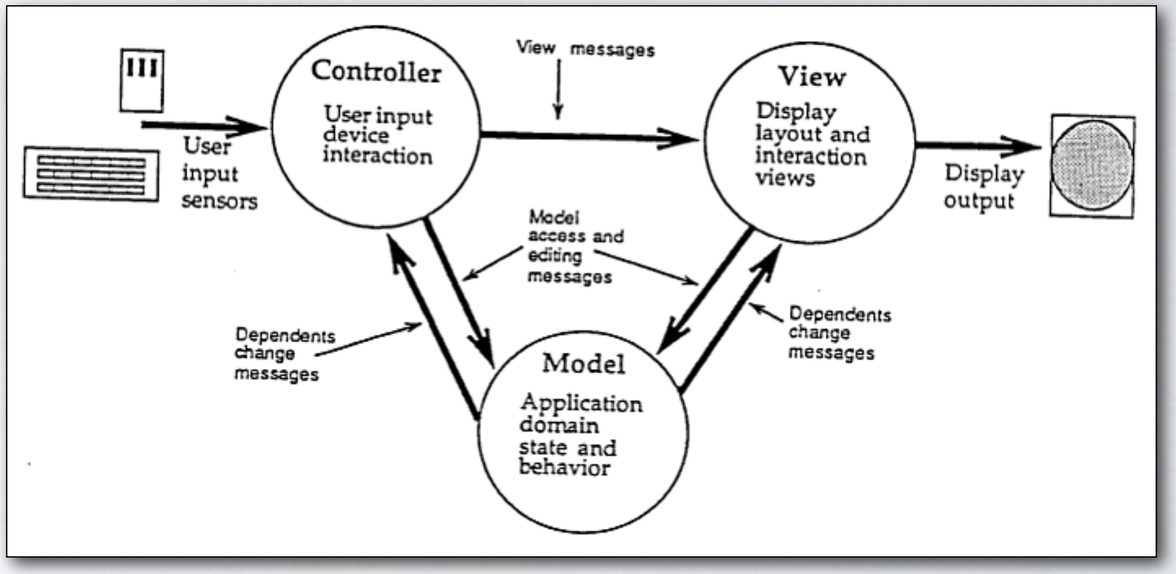
Action-Domain-Responder

A Brief Introduction

- "Model-View-Controller" has suffered from semantic diffusion
- A **user interface** pattern, not itself an application architecture
- Originated as in-memory, client-side, event-oriented
- Server-side "MVC" is over-the-network, request/response-oriented

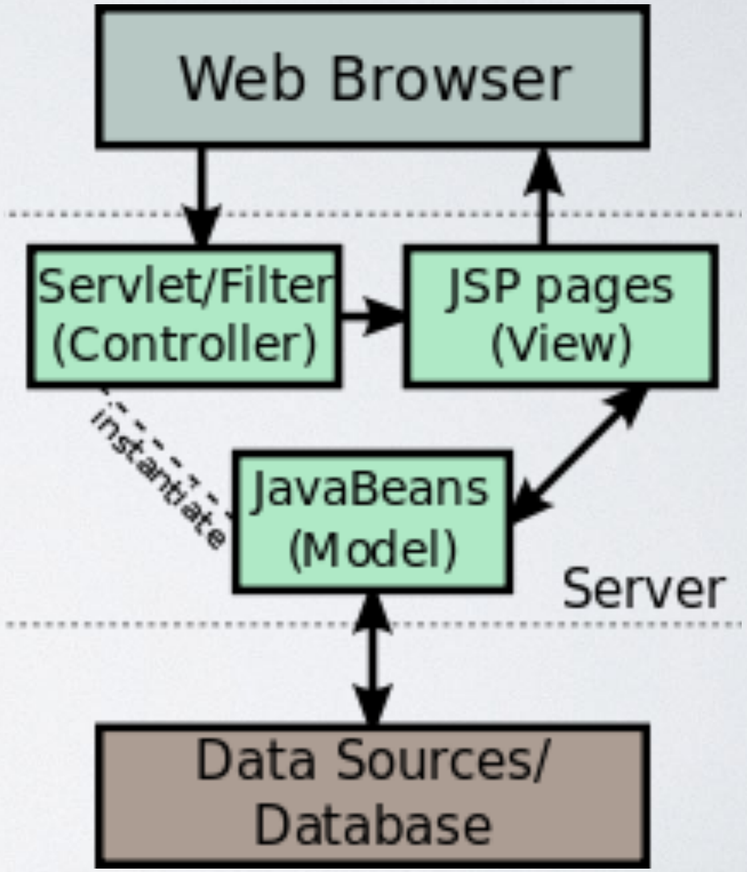
Smalltalk-80 MVC Collaborations

- Controller receives keyboard/mouse input events from User
- Controller notifies View and Model, respectively
- View and Model notify each other for updates
- View updates are rendered on the screen
- Hierarchical collection of interrelated MVC triads for each screen element (event system)



Sun Model 2 Collaborations

- From event-driven to request/response (pages)
- No more messaging interactions between triads
- One collected set of interactions delivered



Toward A Web-Specific UI Pattern

- Stop using in-memory desktop GUI patterns as server patterns
- Entirely new name to break the association with "MVC"
- Remember we are in a client/server (request/response) environment
- Use existing server-side "MVC" as a basis
- Refine the components and collaborations toward better practices

Refining the "Model" to "Domain"

- The "Domain" has essentially identical responsibilities
- Reminiscent of "Domain Logic": Transaction Script, Domain Model, Table Module, Service Layer
- Reminiscent of "Domain Driven Design": Repository, App Service

Refining the "View" to "Responder"

- Usually think of a View system as templates (screen elements)
- Client receives HTTP response of **both** body **and** headers
- This means the View in server-based MVC is **not** the template
- The View in server-based MVC is the **Response**

Intermingled Presentation Logic

- Template Views generally build HTTP **body** values
- Remaining Controller logic manipulates HTTP **header** values
- Presentation logic is mixed between Views and Controllers
- Need a layer that is completely in charge of building the Response

“Responder” For Presentation

- Responder layer handles setting headers, status, etc
- Additionally uses templates for setting body content
- Invoke a Responder for presentation of Response
- Remove Response presentation from all Controller action methods

Refining the “Controller” To “Action”

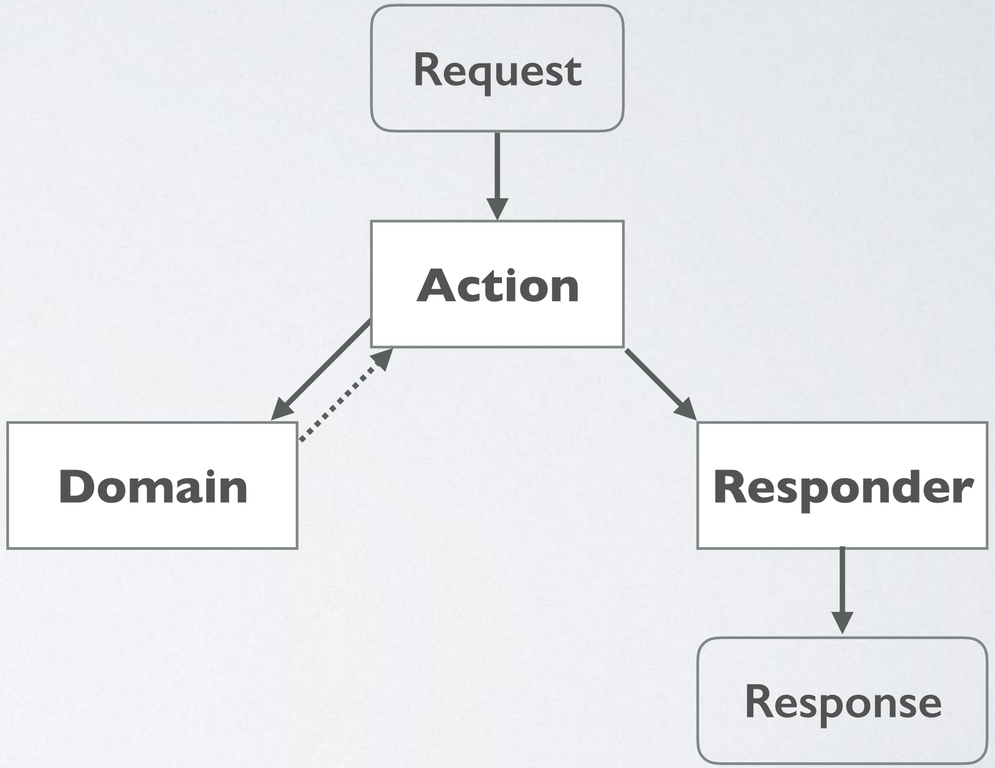
- Takes input, sends to domain, gets back payload, sends to Responder
- Move from Controller with index(), create(), read(), etc. …
- … toward one class per Action:
  IndexAction, CreateAction, ReadAction, etc.

Components

- **Domain** is the logic to manipulate the domain, session, application, and environment data, modifying state and persistence as needed.
- **Responder** is the logic to build an HTTP response or response description. It deals with body content, templates and views, headers and cookies, status codes, and so on.
- **Action** is the logic that connects the *Domain* and *Responder*. It uses the request input to interact with the *Domain*, and passes the *Domain* output to the *Responder*.

Collaborations

- Action feeds input from HTTP request to a Domain layer
- Action feeds payload from Domain layer to a Responder
- Responder builds the HTTP response headers and body



Moving Towards ADR

- ADR is a *refinement* of MVC, not a brand-new invention
- You're probably already *almost* doing ADR
- Change from Template to Responder, and move header work
- Move business logic from "Controller" (Action) to Domain
- Remaining Action code is minimalist, trivial

ADR Frameworks

- Radar (pmjones): <http://github.com/radarphp/Radar.Project>
  Aura.Di, Relay, Arbiter, Aura.Payload

- Equip (shadowhand): <https://github.com/equip/framework>
  Auryn, Relay, Equip Action, Equip Payload

- Adroit (shochdoerfer): <https://github.com/bitExpert/adroit> (PHP 7!)
  Container-Interop, Zend Stratigility, Adroit Action, Adroit Payload

ADR Considerations and PSR-7

Topics

- Middleware
- Actions
- Responders
- Content Negotiation
- Authentication/Authorization
- Sessions

"*Where does it go in 'MVC' ?* "

Middleware (1/2)

- Premise: Middleware is a user interface decoration system
- The UI is the HTTP request (input) and HTTP response (output)
- Middleware is not for your Domain work
- Middleware is a path in to, and out of, the core Domain

Middleware (2/2)

- Middleware might be an Action, or might be a Responder
- More likely that middleware invokes an Action to get a Response
- It should never be a Domain element; Domain is not UI.
- Interacting with storage or service? Not "user interface." Domain!

Actions

- Use PSR-7 ServerRequest as input element
- Marshal inputs into a non-HTTP structure and pass to Domain
- Validation? Not "user interface" -- Domain!
- Eventually, Actions end up very similar
- <https://github.com/arbiterphp/Arbiter.Arbiter>

Naive Generic Action as Middleware

```
function action($request, $response, callable $next = null)
{
    $input = array_replace(
        (array) $request->getQueryParams(),
        (array) $request->getParsedBody(),
        (array) $request->getUploadedFiles(),
        (array) $request->getCookieParams(),
        (array) $request->getAttributes()
    );

    $domainCallable = $request->getAttribute('adr:domain');
    $payload = $domainCallable($input);

    $responderCallable = $request->getAttribute('adr:responder');
    return $responderCallable($request, $response, $payload);
}
```

Responders

- Use PSR-7 Response as the output element
- Need helper for cookies:
  https://github.com/dflydev/dflydev-fig-cookies
- Need helpers for complex headers (Cache-Control)
- Lambda? Need factory for Response object (Stream)
  https://github.com/http-interop/http-factory (PSR-17)

Code Example:
Naive JSON Responder

Content Negotiation: Where Does It Go?

- Where does it go in ADR? (Where would it go in MVC?)
- Output formatting -- Responder!
- Parse $request->getHeader('Accept') and match to available types
- Build and send acceptable type, or send "406 Not Acceptable"

Content Negotiation: Routing Issues

- Inefficient to wait until Responder
- Examine "Accept" header in router to see if available type is present
- Proceed if q-value is non-zero, or respond early with "406"
- Aura.Router allows routing on Accept header: https://github.com/auraphp/Aura.Router/

Authentication: Where Does It Go?

- A little controversial
- Is Authentication a **user interface** task?
- If it interacts with storage, it is "Domain"
- Do authentication work in Domain, not Action or Responder
- Not a middleware task

Authentication: Routing Issues

- What about routing based on authenticated/anonymous?
- Examine expected header or body element (middleware or router)
- If present, **presume** authenticated and route appropriately
- Do real authentication and authorization checks in Domain

Sessions

- Very controversial
- What do session_start() et al. do?
- Reads/writes cookie direct to output, a la setcookie()
- Reads/writes to file, memcache, etc.

Sessions: Combined Concerns

- Reading cookie values is Action work
- Interacting with storage is Domain work
- Sending cookies is Responder work
- Cannot intercept or inspect cookies as part of Response

Option 1: Sessions in Middleware

- Middleware to start & commit session
- In Action, pass &$_SESSION as input to Domain
- Domain cannot do session work (login? logout? regenerate ID?)
- Still cannot see cookies in Response object

Option 2: Semi-Automatic Sessions

- Disable reading and writing of session cookies
- Read session cookie as input in Action, pass to Domain
- Use session_*() in the Domain, return session ID in payload
- Write session cookie in Responder

Config

```
ini_set('session.use_cookies', false);
ini_set('session.use_only_cookies', true);
ini_set('session.use_trans_sid', false);
```

Action

```
$cookies = $request->getCookieParams();
$input['sessionId'] = $cookies[session_name()] ?? null;
// ...
$payload = $domain($input);
return $responder($request, $response, $payload);
```

Domain

```
session_id($input['sessionId']);
session_start();

// ...

session_regenerate_id()
session_write_close();

// ...

$payload['sessionId'] = session_id();
return $payload;
```

Responder

```
$sessionName = session_name();
$newId = $payload->getOutput()['sessionId'];

$oldId = '';
$cookies = $request->getCookieParams();
if (! empty($cookies[$sessionName])) {
    $oldId = $cookies[$sessionName];
}

if ($newId !== $oldId) {
    // domain called session_start() or session_regenerate_id(),
    // so send a cookie with the new id
    $response = $response->withAddedHeader(
        'Set-Cookie',
        $this->getSessionCookie($sessionName, $newId) // builds cookie header
    );
}
```

Consequences

- ADR concerns are now separated
- Cookies available in Response
- session_cache_expire() & session_cache_limiter() ... ?
- PSR-7 and Session Cookies
  http://paul-m-jones.com/archives/6310

Conclusion

- PSR-7 and ADR
- PSR-7 in Actions and Responders
- Content Negotiation with ADR
- Authentication with ADR
- Sessions with PSR-7 and ADR

Thanks!

- pmjones.io/adr (ADR Paper)
- github.com/radarphp/Radar.Project (ADR "framework")
- paul-m-jones.com
- @pmjones on gab.ai