Enterprise PHP Architecture through Design Patterns and Modularization

by Aaron Saray

Why trust this guy?

- Programming for over 2 decades
- Web Development Manager
- Author of WROX Professional PHP Design Patterns
- MKEPUG CoFounder

Why are we here?

- Focus on Enterprise Development
- Understand decoupling
- See real reasons for doing things to pattern

Why is this important?

- PHP is everywhere
- PHP runs organizations
- IT changes faster than any other area
- Long term investment
  - Do things the right way, and be valuable - longer
    - quality
    - speed
    - maintainability

Lets get into it

Disclaimer:

- There are many ways to refer to the concepts discussed next
- Don't get stuck on names
  - Get stuck on the ideas

What are the main topics?

- Data Models
- Domain Logic Models
- Mappers
- Services

Data Model

- This class represents a real entity in the organization

- Examples of entities:
  - Visitor of the website
  - Product for sale: computer
  - Mail message
  - Blog entry, Comment

Data Model (con't)

- Data models contain properties that are unique/descriptive to that entity
  - Harddrive for sale: has property of:
    - size
    - suggested sale price
    - speed
  - Does **not** have property of:
    - favorite song
- Data models contain methods that understand their properties
  - User has methods:
    - fullname() - brings together properties

Data Model (con't)

- Very dumb
- Harddrive example:
  - has a property called suggested sale price
  - has a property called sold price
  - What happens if the sold price is less than the suggested sale price?
    - This model does not care. It's a dumb data store

Sneak peek: Domain Logic Model

- Validation of the sale price would be done in the domain model

- What about string length?
  - Understand that your limits should be on your domain product
    - Is it true that your limit for a product is 255 chars - just cuz its a varchar(255) column?
      - Your data provider should handle your requirements, not the other way around

To Summarize Data Model

- Represents some sort of entity
- Has properties of that entity
- Has methods that understand its own properties only
- Dumb
- Could be really messed up if it doesn't have a relationship with a Domain Object

Domain Logic Model

- Represents the logical steps and decisions that make your app (business) unique
- Handles simple data validation
- Handles complex business rules
- The gatekeeper of the Data Model

Domain Logic Model (con't)

- Has relationships with Data Models
  - Can be 1-1, 1-N, N-1, N-N, depending on your choices and architecture. Probably is a hybrid
- Has methods that comprehend and understand Data Models
- Has properties or constants from the business configuration
  - Maximum amount of discount on sale

Domain Logic Model (con't)

- Going back to harddrive example
  - Harddrive data object has suggested price of $40
  - Input has the sale price coming in at $30
  - Domain Logic Model to the rescue!

- Data Logic Model validates the sale
  - Property says maximum discount is 20%
  - Input discount is 25%
  - Rejects input, does not allow Data Model to accept $30 as the sale price

  - Perhaps maximum discount is 0% & must be exact
  - Rejects input if input is not exactly $40

Domain Logic Model (con't)

- Another example
  - Blog Entry Model has a title
  - Business says Blog entry titles should not be longer than 100 characters
  - Domain Logic Model will validate that the input ready for this Blog Entry model is <= 100

Why is the separation important?

- Two reasons
  - Reuse
    - 100 different items for sale, but they don't have a common ancestor (ie, you can't extend a class. In this example, we choose not to use traits)
    - Each one must sell for the exact sale price
    - Only ONE object is required to validate that each time - instead of duplicating that code in each model
    - ** Design patterns - there are more options than just extending forever!!

Why is the separation important? (con't)

- Two reasons
  - Data models should just represent an entity, not the logic behind it
    - Your car has a maximum load. The steel doesn't know that
    - Harddrive example!
      - Harddrive is for sale at main warehouse. It is the same harddrive for 10 resellers.
      - Each reseller has different properties that determine the final sale price
      - The harddrive ITSELF never changes - so that class should be exactly the same for all retailers

Summarize Domain Logic Models

- Understands the business logic
- Comprehends data models
- The Gatekeeper of valid data

Data Mapper Object

- Data mappers map data from a Data Access Object to Data Models

- Data Access Object
  - not covered too much in this talk
  - provides access to a data source like a database, salesforce, as400, REST, via a predictable interface

- Concerned with retrieving or persisting a single Data Model (or collection)

Data Mapper Object (con't)

- Not particularly smart
  - will work to retrieve data based on parameters
  - will populate a Data Model generally without validation
    - ** caveat: sometimes a hybrid is created when working with remote sources to introduce Domain Logic Models here... but that's another talk
- Works directly with the Data Model
  - Creates new instances
  - Understands properties
  - Determines what to persist

Data Mapper Object (con't)

- Harddrive example
  - Can take a Harddrive Data Model and save it in the proper format to MySQL
    - Remember, the object has the data, but perhaps the database is normalized
  - Has a method to find a harddrive by serial number
    - Internally, determines what db field is the serial number, runs the query, and then populates the model with the full properties
  - Build collection
    - findAll() - parameter where size is > 250gb

Why is Data Mapper Important?

- Data Models are dumb
  - Don't care what their data is or where it came from
- Domain Logic Models only care about decisions
  - They don't care about the source
- Something needs to retrieve and persist
- Data Mapper is still not your "public" interface
  - (that's a service) - can swap out Data Mappers

Summarize Data Mapper Model

- Works directly with Data Model and Data Access Objects
- Retrieves and Persists Data Models
- Understands properties of Data Models
  - Can map data source to properties
  - Can determine retrieval method based on properties

Services

- A lot of confusion with this term
  - Sometimes they're the public interface of your app
  - Other times they reflect connections to third parties, like Amazon or Salesforce
  - Doesn't matter - let's just move to the concept
- Services are the public face to your controllers
  - These classes allow for calling methods that create, persist, modify, and retrieve data
  - Used to translate and process information from user input or other helper classes

Services (con't)

- Services are responsible for providing the Data Model or collection to the consumer
- Services act as the layer to work with your external consumer or framework
- Services do processing that is the same no matter what Data Model, Data Mapper, or other Service is involved
  - retrieving data from a helper model, filtering search term and city/state, etc.

Services (con't)

- Harddrive example!
  - Service::save() would accept a harddrive model, grab the proper mapper (perhaps the harddrive mapper associated with mysql), and introduce those two. It would then reintroduce the saved model

  - Service::findBySerialNumber() would accept a serial number parameter, translate that into a parameter for the mapper, get the mapper, retrieve the collection, and pass that back to the consumer

Services (con't)

- Harddrive example (con't)
  - Service::validateUpdate() - would take user input in, validate that it is valid and could match up with the Data Model, and return whether the input was acceptable or not
  - Service::populateModel() - takes that validated user input and the model, calls the mapper with this information to populate it (mapper knows fields, not services), and returns it

Why are Services Important?

- Services are the public interfaces to the rest of the classes in your app
- Things that you allow 'the consumer' to do are only defined in the service layer
- If data sources, mappers, models change, your service stays the same, and so does the consumer side of your app

Summarize Services

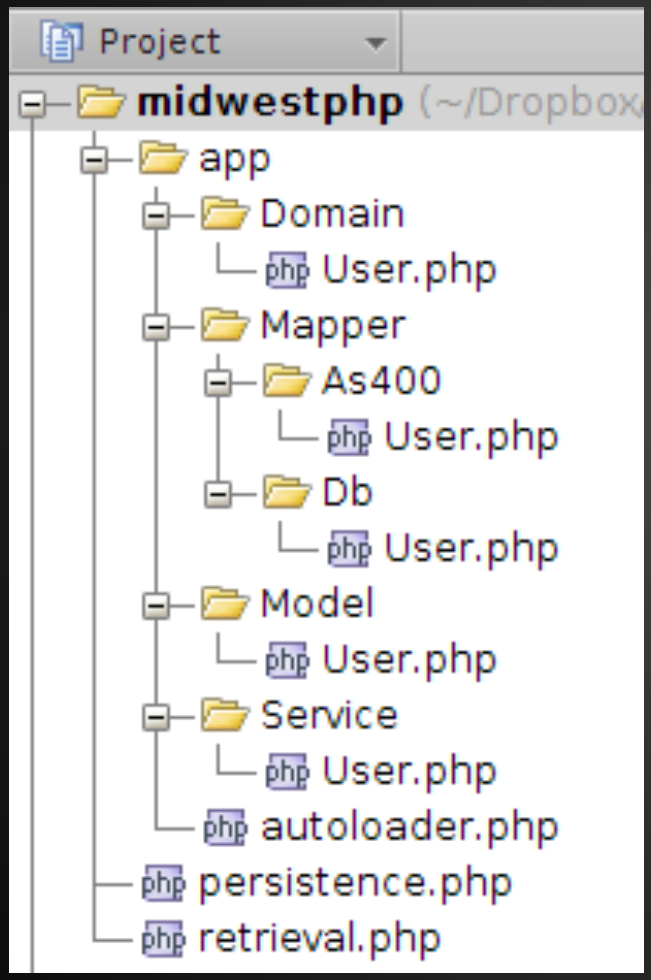
- Public interface to the rest of your code
- Understand Data Models, Domain Models, Mappers
- Execute processing that is always the same, never differs no matter what objects are in use

Example Code

- Let's see this in action. Things to remember:
- This is not a whole app - don't treat it as one
- It is example code
- This is not a whole app
- Don't forget, this is not a completed app

Just trying to demonstrate the concepts with the least amount of distraction!

Layout - Overview



Main layout of files

We will cover each code section

Retrieval Options (A Controller)

retrieval.php

```php
<?php
require 'app/autoloader.php';

$service = new Service_User();

//one id 3
$user = $service->findOneById(3);

// find only 5 first users
$users = $service->findAll(array('limit'=>5));

//users created before today's date
$legacyUsers = $service->findAllCreatedBeforeToday();
```

Retrieve one by ID

User.php

\Service_User

```php
<?php
class Service_User
{
    public function findOneById($id)
    {
        $mapper = $this->_getMapper();
        $params = array('id'=>$id);
        return $mapper->findOne($params);
    }

    protected function _getMapper()
    {
        return new Mapper_Db_User();
    }

    //snippy
```

Other retrieval code

User.php

\Service_User

```php
<?php
class Service_User
{
    public function findAll($params = array())
    {
        $mapper = $this->_getMapper();
        return $mapper->findAll($params);
    }

    public function findAllCreatedBeforeToday()
    {
        $mapper = $this->_getMapper();
        $params = array('createdBefore'=>date('Y-m-d'));
        return $mapper->findAll($params);
    }
```

The Mapper

User.php

\Mapper_Db_User

```php
<?php
class Mapper_Db_User
{
    protected static $pdo = null;

    protected function _getDao()
    {
        if (self::$pdo == null) {
            // obviously remove credentials and don't use root
            self::$pdo = new PDO('mysql:host=localhost;dbname=test', 'root', 'password');
        }
        return self::$pdo;
    }

    // and then more code ...

```

The Mapper (con't)

```php
    public function findAll($params = array())
    {
        $whereStrings = $whereParams = array();

        if (isset($params['id'])) {
            $whereStrings[] = 'id = ?';
            $whereParams[] = $params['id'];
        }
        if (isset($params['firstname'])) {
            $whereStrings[] = 'firstname = ?';
            $whereParams[] = $params['firstname'];
        }
        if (isset($params['createdBefore'])) {
            $whereStrings[] = 'datecreated < ?';
            $whereParams[] = $params['createdBefore'];
        }

        $sql = "select * from user";
        if (!empty($whereStrings)) {
            $sql .= " where " . implode(' AND ', $whereStrings);
        }
        if (isset($params['limit'])) {
            $sql .= " limit " . intval($params['limit']);
        }

        $statement = $this->_getDao()->prepare($sql);
        $statement->execute($whereParams);

        $results = $statement->fetchAll();

        return $this->_populateCollection($results);
    }
```

More Mapper

```
39      public function findOne($params = array())
40      {
41          $collection = $this->findAll($params);
42
43          if (count($collection) > 1) {
44              throw new Exception("More than one result found");
45          }
46
47          $returnOne = null;
48          if (!empty($collection)) {
49              $returnOne = array_shift($collection);
50          }
51
52          return $returnOne;
53      }
```

Mapper (populating / mapping)

```php
    protected function _populateCollection($results)
    {
        $return = array();

        foreach ($results as $result) {
            $return[] = $this->mapFromArray($result);
        }

        return $return;
    }

    public function mapFromArray($array, Model_User $user = null)
    {
        if (is_null($user)) $user = new Model_User();
        if (isset($array['firstname'])) $user->firstname = $array['firstname'];
        if (isset($array['lastname'])) $user->lastname = $array['lastname'];
        if (isset($array['email'])) $user->email = $array['email'];
        if (isset($array['id'])) $user->id = $array['id'];
        if (isset($array['datecreated'])) $user->datecreated = $array['datecreated'];
        return $user;
    }
```

Data Model

User.php

```php
<?php
Class Model_User
{
    public $firstname;
    public $lastname;
    public $email;
    public $id;
    public $datecreated;

    public function getFullName()
    {
        return trim($this->firstname . ' ' . $this->lastname);
    }
}
```

Next Up...

Retrieval is done...

Let's look at persistence

Persistence Controller (1 of 2)

persistence.php

```php
<?php
require 'app/autoloader.php';

$service = new Service_User();

// save new user
$inputFromUser = array('firstname'=>'aaron', 'lastname'=>'saray', 'email'=>'me@me.com');
try {
    $user = $service->createFromUserInput($inputFromUser);
    $service->save($user);
}
catch (Exception $e) {
    print $e->getMessage();
}
```

Persistence Controller (2 of 2)

```
16      // save new user, invalid data
17      $inputFromUser = array('firstname'=>'aaron', 'lastname'=>'', 'email'=>'me@me.com');
18      try {
19          $user = $service->createFromUserInput($inputFromUser);
20          $service->save($user);
21      }
22      catch (Exception $e) {
23          // would go to exception, deal with it how you'd like
24          print $e->getMessage();
25      }
26
27      //update existing user with data
28      $user = $service->findOneById(1);
29      $inputFromUser = array('firstname'=>'Joe');
30      try {
31          $service->applyUpdateFromUserInput($user, $inputFromUser);
32          $service->save($user);
33      }
34      catch (Exception $e) {
35          print $e->getMessage();
36      }
```

Creating a User

User.php

\Service_User

```php
<?php
class Service_User
{
    public function createFromUserInput($params = array())
    {
        $domainValidation = $this->_getDomainValidation();

        if (!$domainValidation->validateFirstname($params)) {
            throw new Exception("First name did not validate");
        }
        if (!$domainValidation->validateLastname($params)) {
            throw new Exception("Last name did not validate");
        }

        $mapper = $this->_getMapper();
        $user = $mapper->mapFromArray($params);

        return $user;
    }

    // more code below
```

Creating User (con't)

```
58      protected function _getDomainValidation()
59      {
60          return new Domain_User();
61      }
62
63      public function save(Model_User $user)
64      {
65          $this->_getMapper()->save($user);
66          $this->_audit("User has been saved");
67      }
68
69      protected function _audit($text)
70      {
71          // can log text here
72      }
73
74      protected function _getMapper()
75      {
76          return new Mapper_Db_User();
77      }
```

Domain User Model

User.php

```php
<?php
class Domain_User
{
    public function validateFirstname($values)
    {
        return !empty($values['firstname']) && strlen($values['firstname']) < 25;
    }

    public function validateLastname($values)
    {
        return !empty($values['lastname']) && strlen($values['lastname']) < 45;
    }
}
```

Mapper DB - Save

User.php

\Mapper_Db_User

```php
<?php
class Mapper_Db_User
{
    public function save(Model_User $user)
    {
        if ($user->id) {
            $sql = "update user set firstname = ?, lastname = ?, email = ? where id = ?";
            $params = array($user->firstname, $user->lastname, $user->email, $user->id);
        }
        else {
            $sql = "insert into user (firstname, lastname, email) values (?, ?, ?)";
            $params = array($user->firstname, $user->lastname, $user->email);
        }

        $statement = $this->_getDao()->prepare($sql);
        $statement->execute($params);
    }
```

Apply Update From User

```
41      public function applyUpdateFromUserInput(Model_User $user, $params = array())
42      {
43          $domainValidation = $this->_getDomainValidation();
44
45          if (isset($params['firstname']) && !$domainValidation->validateFirstname($params)) {
46              throw new Exception("First name did not validate");
47          }
48          if (isset($params['lastname']) && !$domainValidation->validateLastname($params)) {
49              throw new Exception("Last name did not validate");
50          }
51
52          $mapper = $this->_getMapper();
53          $user = $mapper->mapFromArray($params, $user);
54
55          return $user;
56      }
```

Persistence Done

Yay

Now - let's introduce the value!

Users must be sync'd to AS400

- Previously was from mysql db
- Need to change as little code as possible
- The only thing changing is how we map data
  - to / from the data source
  - "map"... mapper
- Lets substitute in a different mapper

Modify User Service

```
64      protected function _getMapper()
65      {
66          //return new Mapper_Db_User();
67          return new Mapper_As400_User();
68      }
```

This is the ONLY change to your code.

All else is NEW code.

What does this mapper look like?

User.php

\Mapper_As400_User

```php
<?php
class Mapper_As400_User
{
    protected static $pdo = null;

    public function findAll($params = array())
    {
        $in = array();

        $id = isset($params['id']) ? $params['id'] : 0;
        $in['id'] = str_pad($id, 10, '0', STR_PAD_LEFT);

        $firstname = isset($params['firstname']) ? $params['firstname'] : '';
        $in['firstname'] = $firstname;

        if (isset($params['createdBefore'])) {
            $createdBefore = date('Ymd', strtotime($params['createdBefore']));
        }
        else {
            $createdBefore = '00000000';
        }
        $in['createdBefore'] = $createdBefore;

        $out = array(
            'id'=>'',
            'firstname'=>'',
            'lastname'=>'',
            'email'=>'',
            'id'=>'',
            'datecreated'=>''
        );
```

AS400 Mapper Continued

```
33          $query = "CALL MYLIB.GETUSER(?, ?,?,?,?,?,?,?,?)";
34          $stmt = $this->_getDao()->prepare($query);
35          $stmt->bindParam(1, $in['id']);
36          $stmt->bindParam(2, $in['firstname']);
37          $stmt->bindParam(3, $in['createdBefore']);
38          $stmt->bindParam(4, $out['id'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
39          $stmt->bindParam(5, $out['firstname'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
40          $stmt->bindParam(6, $out['lastname'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
41          $stmt->bindParam(7, $out['email'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
42          $stmt->bindParam(8, $out['id'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
43          $stmt->bindParam(9, $out['datecreated'], PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
44
45          $results = array();
46
47          do {
48              $stmt->execute();
49              $results[] = $out;
50          } while ($stmt->nextRowset());
51
52          return $this->_populateCollection($results);
53      }
54
55      public function findOne($params = array())
56      {
57          $collection = $this->findAll($params);
58
59          if (count($collection) > 1) {
60              throw new Exception("More than one result found");
61          }
62
63          $returnOne = null;
64          if (!empty($collection)) {
65              $returnOne = array_shift($collection);
66          }
67
68          return $returnOne;
69      }
```

AS400 Mapper Continued

```
71      public function mapFromArray($array, Model_User $user = null)
72      {
73          if (is_null($user)) $user = new Model_User();
74          $user->firstname = trim($array['firstname']);
75          $user->lastname = trim($array['lastname']);
76          $user->email = trim($array['email']);
77          $user->id = intval($array['id']);
78          $user->datecreated = date('Y-m-d', strtotime($array['datecreated']));
79          $return[] = $user;
80
81          return $user;
82      }
```

AS400 Mapper Continued

```
84          protected function _populateCollection($results)
85          {
86              $return = array();
87
88              foreach ($results as $result) {
89                  $return[] = $this->mapFromArray($result);
90              }
91
92              return $return;
93          }
94
95          protected function _getDao()
96          {
97              if (self::$pdo == null) {
98                  self::$pdo = new PDO('odbc:iSeriesDSN', 'MYUSER', 'MYPASS');
99              }
100             return self::$pdo;
101         }
102     }
```

Whew! Done!

Check it out at

http://aaronsaray.com/midwestphp

Tweet at @aaronsaray