Design Patterns applied to Web Programming in PHP

Paul A.J. Braam
Vrije Universiteit van Amsterdam
Faculty of Sciences
Department of Computer Science
stud.nr. 1123270
pajbraam@cs.vu.nl

14th March 2004

Abstract

Design is important and can be very usefull in web programming in PHP. I got interested in solving programming problems by applying patterns to PHP. Applying general OO-patterns to PHP is not always as easy as it seems to be. I adapted three such patterns for PHP: Singleton, Stragegy and Model View Component(MVC). I also introduce another pattern that I wrote specially for PHP: SessionObject.

Contents

1 Introduction

A few years ago I did a lot of work in web programming (building web applications like web shops as http://www.casspijkers.nl) in PHP. While programming I faced a lot of problems. For all these problems I came up with my own solution. Of course it was crazy to find solutions to the problems myself because most web programmers will face the same problems and we don't want to reinvent the wheel again and again and yes...again.

The solution to this problem is patterns. While looking on the web I figured out that there is much work to do on defining patterns for web programming especially for PHP. There are some sites providing patterns for PHP, but most of them are of a bad quality. I found a lot of design and implementation errors in those patterns.

Before I give a short introduction to design patterns I will make clear why design in general is important for web programming. After the introduction to design patterns I want to state some major web programming problems (or: challenges). Related to those problems I will give some examples of how design patterns can be used to overcome those problems.

2 About PHP

PHP stands for: PHP: Hypertext Preprocessor. It is a simple open source server-side scripting language that is widely used on web servers. The current version (PHP4) offers limited implementation of object-oriented programming. It supports class definitions, member functions, and inheritance. But it doesnt support multiple inheritance and real encapsulation (because member attributes cant be privately defined). Also the support for polymorphism is in its infancy. With the upcoming introduction of version 5 most of these problems will be solved.

3 PHP and Design

Before talking about design patterns we should have a look at what a good design can offer you while programming web applications. As I stated before, PHP is a simple scripting language. Some people think that men should never use objects because functions and simple pages will do. For some situations this is certainly true (for a lot of OO criticism, see [15]), but in others you might want a higher level of abstraction. Here are some features that a good design can offer you. A good design can increase the following properties:

- readability
- reusability
- simplicity
- modularity
- portability
- adaptability
- maintainability
- extendibility

As you can imagine most of these features are very important for web applications bigger than a few pages.

4 Introduction to Design Patterns

This essay is not about explaining what design patterns are, therefore this introduction is rather short. For a better understanding of design patterns see [1].

Design patterns are written descriptions of well-known solutions to common design problems. Design patterns are a form of reuse: if one person writes it down, many people can use the same ideas in their own designs. A design pattern consists of the following elements:

Name used to identify the pattern

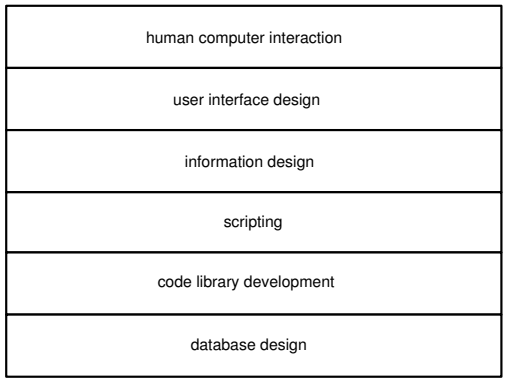
Problem what problem is attacked with this pattern?

Context in what context should this pattern be applied?

Solution & Consequences the solution for the stated problem and its consequences

5 Layers in Web Programming

Webprogramming is concerned with many layers. These are: human computer interaction, interface design, information design, scripting, code library development, database design. We could envision them in the following diagram:



A design pattern for web programming exists in one of these layers. It is good to know at which layer the design pattern has to be applied. With this layering you get insight to what extend the design patterns is language or domain specific. Design patterns residing in the scripting layer will of course be more language specific than one residing on the user interface design layer.

6 Common web programming problems

Multiple Presentations: Most of the web applications are accessible via the internet. That means that people from all over the world can access these applications. Therefore its often important to offer multiple presentations of the same application. Think for example about an international website that needs support for different languages or needs support for different currencies.

Navigation: One of the most important aspects in web design is navigation and information design. Users need always to be able to recognise their position within the website. In a site without a clear and structured navigation it is hard to find what you are looking for.

There exist a lot of navigation structures nowadays. Choosing what kind of navigation best fits your site is not always straightforward.

Database operations: Nearly every large website gets its content from a separate database. It is very important to think about how you are going to make use of that database; how you are going to access

it. For example: putting your SQL-queries directly in the PHP-page where you need them is not a good way of using the database. If the database structure changes, you have to manually update all the separate queries in the different PHP-pages.

Authentication, sessions, etc.: User- or session tracking is done to provide personalized websites for the user. In earlier times, webprogrammers had to do the sessionhandling manually but since the introduction of PHP4 there are several standard functions for automatic authentication and session tracking. Therefore it is not really a big problem anymore.

7 Design Patterns for Web Programming in PHP

7.1 Singleton pattern

Name: Singleton

Problem: You need exactly one global instance of a class and a global point of access to it.

Context: Any

Solution & Consequences: Make an encapsulated static variable holding the single instance of a class. Provide a get-method that:

- has a static variable for holding the one and only instance
- instantiates the instance if it does not exist already
- returns the instance.

This method returns the one and only instance of the class. So from the performance point of view you gain a lot. The singleton pattern prevents the application for making unnecessary many objects, which results in a performance gain.

Implementation: There are some problems with implementing the singleton pattern in PHP:

- In PHP4 it is not possible to define static variables in a class. But we definitely need a static variable to store the instance. Happily we *can* define static variables in a function.
- PHP did not support static methods but the latest version of PHP4 you can just call a method like this: class::method(). This is used in the implementation example.

Now, lets have a look at how an implementation would look like:

```
//
// This is the class of which you want only one instance
//
class yourClassName {

...

  //
  // This is your classcontructor
  //
  function yourClassName() {
    ...
  }

  //
  // This is your helperfunction for retrieving the instance
  //
  function &getInstance() {
    static $instance;
    if (!isSet($instance)) {
        $instance = new singleton();
    }
    return $instance;
  }

  function someMemberFunction() {
    ...
  }

  ...

}
```

The helperfunction getInstance is the global point for accessing your instance. Within your application you can now use your class like this:

```
$instance =& yourClassName::getInstance();
$instance->someMeberFunction();
```

Notice the ampersand (&) used in the assignment of $instance, this sign is very important because it makes sure you get the object itself and not a copy of it. Just the same story goes for the ampersand preceding the name of the member function getInstance() in the declaration of the class.

While looking on the web I found several implementations of the singleton pattern for PHP that are not working, because they do not make use of the ampersand. So be careful with implementing this pattern and make sure that you test your implementation.

7.2 Strategy pattern

Name: Strategy

Problem: You need several different stategies that solve the same problem to be easily interchangeable with each other. For example, a website with support for multiple languages and currencies.

Context: Any

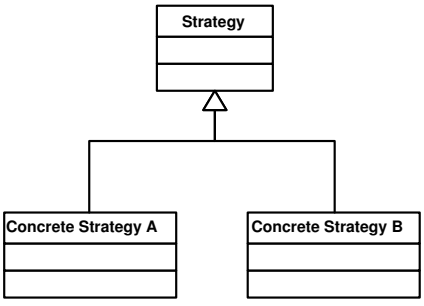
Solution & Consequences: Solution:

- Create a super class (Strategy) that is used as a shared interface for the concrete strategy classes.
- Define a subclass (ConcreteStragegy) for each strategy needed, that overrides the member functions of the super class.
- Create a reference in your PHP application to one of the subclasses (the concrete strategy that you need) and use this reference throughout your program.

The consequence is that in the PHP-application you can freely change the reference from one subclass to another (from one concrete strategy to another) because they now have the same interface.

PHP does not support interfaces, so you are not forced to stick to the interface of the super class. It is your own responsibility to check if you stick to the interface.

UML diagram:



Implementation: In this example implementation we want to print a list of names. We want two strategies: one for printing the list lexicographically ascending, and one for printing the list lexicographically descending.

Here is a sketch of the implementation:

```
//
// Superclass (the interface class)
//
class printListInterface {

  function execute($namelist) {
  }

}

//
// The subclass (concrete strategy) for printing the list ascending
//
class printListAscending extends printListInterface {

  function execute($namelist) {
    // Sort \$namelist in lex. ascending order and
    // print the list
    ...
  }

}

//
// The subclass (concrete strategy) for printing the list descending
//
class printListDescending extends printListInterface {

  function execute($namelist) {
    // Sort $namelist in lex. descending order and
    // print the list
    ...
  }

}
```

Now you can make a reference in you PHP application to the concrete strategy of your choice:

```
$printList = new printListAscending();
```

and use the reference:

```
$names[0] = 'Ralf';
$names[1] = 'Paul';
$names[2] = 'Frank';

$printList->execute($names);
```

7.3 Model View Controller pattern

Name: Model View Controller

Problem: You want to be able to change the appearance(UI) of your web application from time to time, or you want to offer multiple UIs for the same application in an easy manner.

Context: Changing or multiple UIs for the same application

Solution & Consequences: Devide your application into 3 parts:

The Model (Procesing): the model is a representation of the data and the possible operations on it.

The View (Output): the view renders the contents of the model. It is a view on the model.

The Controller (Input): the controller translates interactions with the view into actions to be performed by the model.

Implementation: This is a sketch of an implementation of a MVC-pattern applied to a web shop. In this example the user can look up a product, or add a product to the system.

The Model Component:

This class is like a database wrapper or database adaptor

```
class ShoppingModel {

  //
  // Retrieve the product from the database
  //
  function getProduct($id) {
    ...
  }

  //
  // Store a product in the database
  //
  function addProduct($id, $name, $description, $price) {
    ...
  }

}
```

The View Component:

The view class is responsible for generating HTML(the view on the model)

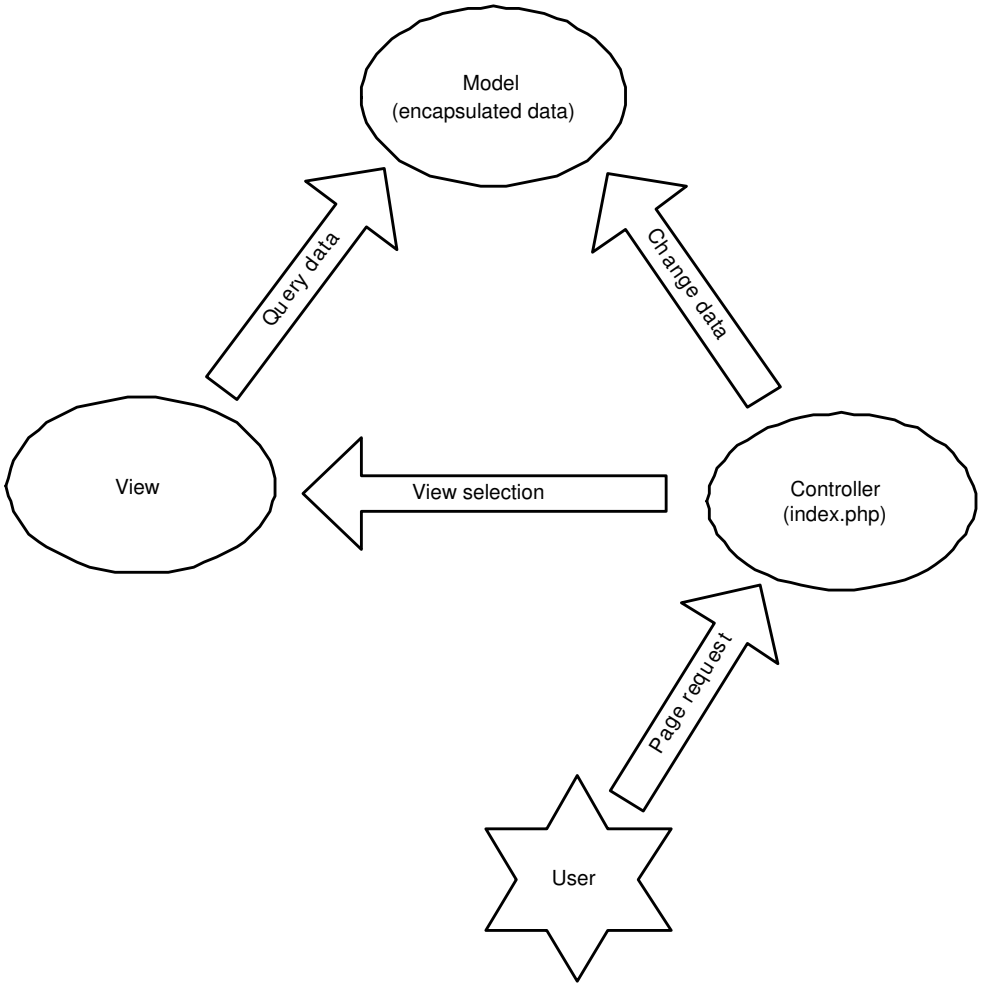
```
class ShoppingView {
  var $model;

  //
  // Class contructor that stores the model
  //
  function ShoppingView($model) {
    $this->$model = $model;
  }

  //
  // Print HTML-header
  //
  function printHeader {
    ...
  }

  //
  // Print HTML-footer
  //
  function printFooter {
    ...
  }

  //
  // Print HTML-representation of a product
  //
  function printProduct($id) {
    $view->printHeader();
    $product = $this->$model->getProduct($id);
    // Give the HTML representation of the product item.
    // By printing a some HTML tags and properties of the product
    ...
    $view->printFooter();
  }

  //
  // Print HTML-representation of a message
  //
  function printMessage($message) {
    $this->printHeader();
    print($message);
    $this->printFooter();
  }

}
```

Let us have a look at the controller component. I found some implementations of the MVC-pattern for PHP (like the MVC-pattern at [11])

that have a separate class for the controller part. That class is used in the PHP-file the user requested. This suggests that the controller class is the controller component of the MVC-pattern. This is not a nice way to apply this pattern to PHP.

In this way the controller class is just a subset of the functionality of the controller component. The controller class and PHP file together are the controller component in the MVC-pattern. Because the functionality done in the PHP file, also belongs to the controller, it is the part that interacts with the user.

A better way of applying the MVC-pattern is to look at the PHP-file as the controller component, so the controller component is the PHP-file itself:



Continuing the implementation example:

The Controller Component(index.php):

```
// initialize the model
$model = new ShoppingModel();
// initialize the view
$view = new ShoppingView($model);

// Check what action is defined in the URL
switch ($_GET['action']) {
  // Show a product
  case 'show_product':
    $view->printProduct($_GET[id]);
    break;
  // Add a product
  case 'add_product':
    $model->addProduct($_GET[id],$_GET[name],$_GET[desc],$_GET[price]);
    $view->printMessage('Product added!')
    break;
  default:
    $view->printMessage('No action found!')
    break;
}
```

So possible calls to this PHP-page:

- Show product with id 5:

  ```
  index.php?action=show_product&id=5
  ```

- Add the candy bar to the products:

  ```
  index.php?action=add_product&id=10&name=Candy%20bar&descr=...
  ```

These calls are normally made by the same (if you extend this controller) or other controllers(read: PHP-pages) with the same model and view.

With this implementation we have a separation of the model, view and the controller. As you can see now, it is rather easy to plug-in a new View class resulting in a new UI for your web application.

7.4 SessionObject pattern

All the previous patterns are projections of existing pattern onto PHP. I mean, they are general (and mostly Object Oriented) patterns that are adapted to be used in PHP. The now following pattern is a new pattern that I would like to introduce. It is a solution that I have used very often in my history as a web programmer.

Name: SessionObject

Problem: You are programming a web application that makes use of sessions and you want to remember the information about the session (throughout the session) in an organised and structured way. E.g. keep the shopping basket content available throughout the session in a web shop.

Context: Web applications with sessions. Session information has to be available throughout the session.

Solution & Consequences: Create a class definition that represents the information that needs to be available throughout the session. At the top of each PHP-page:

- Start or continue a session.
- Register a session variable(for holding the session object)
- Check if a session object exists, if not: create an instance of the class(defined earlier) and assign it to the session variable.

Now you can always access the session information in an organised way(via an object) and that object will be available during the whole session.

Normally objects do not live very long in php, because they only exist during the time of processing the PHP-page (mostly just a fraction of a second). When you call a page, the objects are created and they are used for the processing of the page. After the result has been sent back to the user, all variables, including the objects, will be lost. With this pattern you register an object as a session variable. That means that, after the processing of the page, the object is stored in a textfile together with the rest of the sessioninformation. At the time you that you load the session again(in another pagerequest), the object will be loaded again and will be available in your application.

Implementation: Let us have a look at an example of an implementation. This is a stripped version of a ShoppingBasket object that I used in web shops strore the products chosen / selected by the user. In this example I make use of a well known ShoppingBasket pattern.

```
//
// This class stores the selected products
//
class ShoppingBasket {

var $productArray;
var $numberOfProducts;
...

  //
  // Adds a product to the productArray
  //
  function addProduct($id) {
    // Add product with id=$id to the productArray
    ...
  }

  //
  // Returns the array with the selected products
  //
  function getProductArray() {
    return $productArray;
  }

  ...

}
```

Your PHP-pages have to begin with this(because the function session_start() must be called before any output is written):

```
// If a session already exist: load session variables or create a new session if none exists.
session_start();

// Register the ShoppingBasket object as a sessionvariable
session_register(basket);

// If the basket-object does not exist, create one
if (!IsSet($basket)) {
  $basket = new ShoppingBasket();
}

// Use the ShoppingBasket
$basket->addProduct(5);
$basket->addProduct(7);
$products = $basket->getProducArray();
```

If you have more than a few pages you could make a procedure (e.g. my_session_start()) containing these lines. Then you just have to call my_session_start() at the beginning of each page.

I tried to mix this pattern with the singleton pattern. My idea was to make the ShoppingBasket-object a singleton, so you do not have to check

for the existence of an instance in the PHP-pages. But unfortunately this does not work like that in PHP. This because of the fact that the static instance variable in the loaded class (the session class) is different to the static instance variable of our class definition in the PHP-file.

8 Problem and Pattern matching

In the beginning of this essay I stated some common problems. Afterwards I introduced a few PHP-patterns. In this section I want to make a matching of problems to a few important patterns.

Multiple presentations MVC pattern see 7.3

Navigation this is not a typically PHP problem, for navigational patterns see [6] and [9]

Database operations Adaptor pattern (GoF) see [1] (convert the database interface to an easy to use interface by making use of an adaptor class) not discussed in this essay, but the Adaptor pattern is easy to project on PHP.

Authentication and sessions SessionObject pattern see 7.4

These are some matchings that I find very usefull. This is not inteded to be an exhausting list.

9 Conclusion

At first sight projecting known patterns to PHP looks quite easy. But if you take a closer look and try to implement such patterns in PHP you will probably figure out that there are some more problems to solve than you thought of in the beginning. That is why there are a lot of PHP patterns floating over the internet with poor quality.

These problems are mainly due to the fact that PHP is not a real object oriented language and that the most patterns are object oriented. So we can conclude that patterns depend on language expressiveness. In the future it will be easier to map these(OO-) patterns to PHP because with the introduction of newer versions the OO support in PHP will become more mature.

It is important to know that patterns do not have to be object oriented. Because the godfathers of the patterns, the Group of Four (GoF) [1], wrote their patterns for OO, and mostly all patterns you find nowadays apply to an OO-environment, some people think that patterns are always related to OO. But this is not true; see for example: patterns for functional strategic programming, see [14]

The first three patterns I described in this essay (the singleton, strategy and the mvc) are existing patterns (for the first two see [1]) that I have adapted for PHP. The implementation is made specific for PHP, but the concept of the patterns are generally applicable. This is in contrast to the last pattern (the SessionObject pattern see Section 7.4) that I wrote specially for PHP (so it is no adaptation of an existing pattern).

I think that these four patterns are very useful to PHP-programmers. Years ago, I did a lot of PHP-programming myself, but never saw some of these patterns. If I had seen these patterns in that time, I would have gained a lot of time with figuring out what the best solution was to a given problem.

References

[1] Erich Gamma, Richard Helm,Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley (1994).

[2] G. Rossi, D. Schwabe and A. Garrido. *Design Reuse in Hypermedia Applications Development*, Proceedings of ACM International Conference on Hypertext (Hypertext97), ACM Press (1997).

[3] Michael J. Flynn. *Introduction to "Influence of Programming Techniques on the Design of Computer"*, Proceedings of the IEEE, VOL. 85, NO. 3, March 1997.

[4] A. Garrido, G. Rossi, and D. Schwabe. *Patterns Systems for Hypermedia*, Proceedings of PloP97 (1997), Available at:
http://st-www.cs.uiuc.edu/users/hanmer/PLoP-97.html

[5] F. Lyardet, Gustavo Rossi and D. Schwabe. *Patterns for Dynamic Websites*, Proceedings of PloP98 (1998), Available at:
http://jerry.cs.uiuc.edu/ plop/plop98/

[6] G. Rossi D. Schwabe F. Lyardet. *Improving Web Information Systems with Navigational Patterns*, Available at:
http://www8.org/w8-papers/5b-hypertext-media/improving/improving.html

[7] J. Conallen. *Building Web Applications with UML*, Addison-Wesley (2003)

[8] J. Lam. *Introduction to Design Patterns Using PHP*, (2003), Available at:
http://www.devarticles.com/c/a/PHP/Introduction-to-Design-Patterns-Using-PHP/

[9] *The Hypermedia Design Patterns Repository Website*, Available at: http://www.designpattern.lu.unisi.ch/

[10] L. atkinson. *Applying patterns to PHP*, (2001), Available at: http://www.zend.com/zend/trick/tricks-app-patt-php.php

[11] *The phpPAtterns Website*, Available at: http://www.phppatterns.com/

[12] *The official PHP Website*, Available at: http://www.php.net/

[13] T. Converse, J. Park. *PHP Bible 2nd Adition*, John Wiley & Sons (2002)

[14] R. Lämmel, J. Visser. *Design Patterns for Functional Strategic Programming* Proc. of Third ACM SIGPLAN Workshop on Rule-Based Programming RULE'02, ACM Press (2002), Available on: http://www.cs.vu.nl/Strafunski/dp-sf/

[15] *Object Oriented Programming Oversold!*, Available at: http://www.geocities.com/tablizer/oopbad.htm