Design Patterns in Web Programming

by Nathan Wallace
http://www.e-gineer.com/articles/design-patterns-in-web-programming.phtml

1 Introduction

I was hacking code and searching through the PHP mailing list archives late one night when I stumbled across the message calling for Apachecon speakers. It was 5am and the deadline for submissions about one hour away. Suddenly the idea came to me – "Wouldn't it be interesting to try and identify patterns in web programming". So I wrote the ten line proposal and collapsed into bed.

It's not until after the talk had been accepted (and I'd had some sleep) that I started to really think the idea through. Design patterns are usually the result of years of experience, collaboration and refinement. But web programming as we know it is really only a few years old and this presentation is a solo effort.

So, think of this as the first step in an ongoing process.

Design patterns generally arise through solving problems. Over time the best techniques to solve each problem are discovered and refined. We will identify, examine and attempt to classify a number of the problems commonly encountered by web programmers.

Patterns are usually obvious to experienced programmers. There is a sense of déjà vu when solving similar problems. More difficult is the task of describing these patterns in a way that makes them accessible and understandable to less experienced developers. We will identify the important aspects of web programming patterns and construct a framework for their description.

Finally we will construct some formal web programming patterns. This will be the first step towards creating a catalogue of patterns for use by web developers.

2 Introduction to Design Patterns

According to Christopher Alexander:

> *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"*

So a design pattern is a formal description of a problem and its solution. Design patterns must have a simple, descriptive name that can be readily used when referring to the pattern. A pattern should document the problem, its solution and the consequences of using it.

Design patterns can be used to assist us in solving related problems. If we can learn to recognize the existence of known patterns in new problems we can apply the same techniques to solve them.

The importance of design patterns is that they formally document problems and their solutions. This gives us a means to recognize and refer to known problems. We also have a means for comparing alternative solutions with full awareness of the consequences of each alternative.

By formally describing design patterns for web programming we can capture the knowledge of experts into a form that can be used by any developer. Well-named design patterns will also improve the ability of web developers to communicate their ideas and intended solutions.

3 Common Web Programming Problems

3.1.1 Admitting our Infancy

The first thing we should do when documenting common web programming problems is admit that we are relatively new to this game. Web sites and applications are rapidly evolving and grow more complex everyday.

While the web is growing at a fantastic pace there are definite standards and techniques that are emerging. For example, the basic design of portal sites such as Yahoo has settled into an established pattern.

3.1.2 Web Programming Application Levels

One of the most exciting and difficult aspects of web programming is that it requires expertise on many different levels. Web gurus need skills in user interface design, human machine interaction, information design, scripting, code library development, database design and database queries. These can be organized and categorized in this diagram:

User Interface
Human Computer Interaction
Information Design
Scripting
Code Library Development
Database Queries
Database Design

Web applications and the problems commonly faced by developers extend across many or all of these levels.

Any framework for describing patterns in web programming will need to be capable of explaining the design considerations across any or all of these levels.

3.1.3 Common Problems and Techniques

3.1.3.1 Form Processing

Form processing is used to capture the problem of getting and verifying input from a user. It is at the core of all web applications and there are many different approaches for solving this problem available to the web developer.

The basic method and constructs for prompting the user remain the same regardless of the actual content being entered. All forms must:

- Display an empty form to the user
- Verify the data entered is valid and display an error if it is incorrect
- Perform the required action using the data

Forms can be implemented using either a single or multiple page design. The single page technique utilizes a self-referring script to display and process the form. Multi-page forms take the user through a series of pages to enter, confirm and submit the data.

The other important area to consider when thinking about form design is the reporting mechanism for invalid data. Some people like to display the form with the input highlighting any errors. This makes it simple for the user to find and fix problems in the data. Others prefer to keep the page and application design as simple as possible by reporting errors on a separate page. The user then needs to either go back to the original form using the back button on their browser or they may be presented with a link to a page which will contain their entered data.

3.1.3.2 Navigation

Clean, structured navigation and information design is one of the most important aspects of web design. There needs to be a consistent look and feel throughout the site. Users need to be able to immediately recognize their location in the site as a search engine referral can throw them anywhere.

There are three major navigation designs employed on the web today. Single level navigation has a list of top level areas within the site.

Multi-level navigation breaks the site into a hierarchy. The user can then drill down through these levels to find the desired content. All levels of the hierarchy are displayed as part of the navigation at all times. As such the hierarchy is usually only two levels deep to keep the display simple.

Dynamic multi-level navigation also uses a hierarchy except here the next level of navigation is not shown to the user until they have selected the parent.

3.1.3.3 Database Operations

How to best access a database is a fundamental problem for web developers. In fact, it's so common that we could easily forget to consider it as being a problem we need to solve. Most people access databases through specific functions or an abstracted database wrapper.

Many web applications and forms revolve around letting the user insert new information into the database and edit existing entries. The data being used is irrelevant to the solutions employed making this a good candidate for a pattern.

If we examine data entry at a higher level many objects / items we use in web programming are at their core captured by a row in the database. Creating a set of useful functions and information based on these items is a common problem for web developers.

3.1.3.4 Authentication

Any request to a web site is authenticated at one of the following levels:

- None – we do nothing to record or track the request
- Session – we track this anonymous person through their current visit only
- Visitor – we track this anonymous person across numerous sessions at the site
- User – we have information about the person and require them to authenticate

These levels of authentication are related and have similar implementations. They should be considered together as we need to know the authentication level of any request to the site.

3.1.3.5 Error Handling

Building graceful degradation into a web site is a great challenge. Catching errors before they confuse users is vital. Logging and notifying the maintainers of these errors is equally important to prevent them happening again.

Another good error handling problem to consider is keeping the site alive even if components are off-line. For example, if the database is being backed up and thus read-only can we continue to serve database requests without trying to write. Or if the database is offline completely can we continue to show pages that do not really need to use the database.

3.1.3.6 E-commerce

E-commerce is an interesting problem because it requires the form and authentication problems above to be solved. The challenge is to build a pattern for e-commerce that is not dependent on a particular implementation of forms or authentication.

3.1.4 Categorizing the Application Levels affected by each Problem

3.1.4.1 Introduction

Placing problems into their levels will help us to understand which problems are specific to web programming. For example, problems that fit solely into the library code layer are probably best described using an existing design pattern language.

3.1.4.2 Form Processing

All form processing is done through a user interface interaction with the user. This results in calls to the high-level functions implemented in the code libraries. As such, form processing is limited to all of the front-end levels.

3.1.4.3 Navigation

Single and multi-level navigation is limited to the user interface and human computer interaction levels.

Dynamic multi-level navigation requires knowledge of the users current location and the ability to generate navigational menus. Backend libraries will also be required to generate these menus. Whether these should be part of a pattern for multi-level navigation or just listed as a collaborator is open to debate.

3.1.4.4 Database Operations

Database wrapper code lives in the code library and database query levels. It actually only needs to know about simple database queries since most are just passed through the wrapper class.

Formalizing the process of inserting and editing data into the database requires the use of all levels. We need to design the database to facilitate this type of data entry. The user interface must be developed to make the process clear.

Trying to encapsulate database rows as objects or items is limited to the backend levels. The front-end has access to this data through the API developed for the items.

3.1.4.5 Authentication

User level authentication requires interaction with the person through the front-end. It also requires backend functions to verify and process the information arising from this interaction. It uses all levels of web programming.

All other forms of authentication require no interaction with the user. They are completely contained in the backend levels.

3.1.4.6 Error Handling

Errors must be reported to the user and detected at all levels of the application. Error handling code uses all levels.

3.1.4.7 E-commerce

E-commerce requires interaction with the user and backend processing of the data. It requires all levels.

4 Design Patterns in the Context of Web Programming

4.1.1 Choosing an Appropriate Level of Abstraction

I spent a lot of time thinking about what would be some example web programming design patterns. I knew that these patterns should describe the techniques I'd learnt over time but I also felt that the patterns should be as generic and abstract as possible.

My confusion arose because I did not choose an appropriate level of abstraction. Web programming is built on top of the principles of programming in general. As I tried to abstract the patterns out they ended up becoming patterns appropriate for generic programming, they were less obvious and useful when applied to the web programming domain.

Web programming is a very specific area. Our aim is to develop patterns that are useful in the context of that domain. Patterns should describe the high level concepts that are used in the chosen domain. Thus, in our case the developed patterns will also be fairly specific.

Let's consider an example to illustrate this point. Object oriented programming can be used in any domain. The patterns developed need to be a level of abstraction away from this method of programming. So, these patterns describe ways to solve problems in object oriented programming. Our patterns need to describe ways to solve problems in web development.

4.1.2 Application Levels

I originally felt that there was no need to consider design patterns for problems that fell into areas with existing documented patterns. For example, a database abstraction class really falls into the object-oriented realm.

While it's important to think about application levels when describing patterns, we should ignore them when choosing patterns to describe. We are looking for patterns relevant to web programming, whatever their application level.

The question then remains as to whether we should describe patterns that fall into certain application levels. Should the database abstraction class above be described using the object oriented design pattern language or our web programming design pattern language?

Again the answer lies in our viewpoint. We are describing patterns from the viewpoint of a web developer. Our patterns need a suitable level of abstraction from that starting point. The object-oriented design pattern language was created to help people describe abstract concepts about classes, methods and their interaction. Our database abstraction pattern needs to provide more specifics than this language may allow.

Our web programming design pattern language should allow us to describe patterns that fall across any of the application levels.

5 A Design Pattern Language for Web Programming

5.1.1 Basis

This language is a modified version of that presented for object oriented programming on page 6 of Design Patterns: Elements of Reusable Object-Oriented Software by Gamma et al.

In the interests of reuse, I have reproduced their clear descriptions of the pattern language elements where appropriate below.

5.1.2 Definition

5.1.2.1 Pattern Name and Classification

The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of your design vocabulary. The pattern's classification reflects the scheme introduced in the next section.

5.1.2.2 Intent

A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?

5.1.2.3 Also Known As

Other well-known names for the pattern, if any.

5.1.2.4 Motivation

An example describing the function and use of the pattern.

5.1.2.5 Applicability

What are the situations where this design pattern can be applied? How can you recognize these situations?

5.1.2.6 Sequence of Events

A step by step description of how the pattern executes. This should describe the high level steps, not implementation specific details. Take particular care to document the sequence of HTTP requests.

5.1.2.7 Participants

The patterns and / or classes participating in the design pattern and their responsibilities.

5.1.2.8 Collaborations

How the participants collaborate to carry out their responsibilities.

5.1.2.9 Consequences

How does the pattern support its objectives? What are the trade-offs and results of using the pattern?

5.1.2.10 Implementation

What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?

5.1.2.11 Sample Code

Code fragments that illustrate how you might implement the pattern in PHP, Java or your favorite web programming language.

5.1.2.12 Related Patterns

What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

6 Web Programming Design Pattern Space

As the catalogue of design patterns grows we need a way to organize them. This helps us improve our understanding of patterns and their effects.

Behavioral patterns describe the way a web application proceeds to perform a given task.

Structural patterns describe an event sequence for web programs.

7 Example Patterns

7.1.1 Filter

7.1.1.1 Classification

Behavioural

7.1.1.2 Intent

Provide a mechanism for conditionally executing code at the start or end of each page request.

7.1.1.3 Motivation

Having the ability to conditionally run functions as part of a web page is a very powerful tool for web programmers. It gives us the flexibility to add and configure logging functions, authentication checks and so on. Filters control the registration and execution of these functions.

7.1.1.4 Applicability

Filters can be used to execute functions at the beginning or end of a script based on certain conditions. These conditions may include things like:

- The requested URL matching a given regular expression
- A random chance factor (eg: run the function 50% of the time)

A good example of using filters for logging would be to have some sample of requests being logged. This is achieved by setting an appropriate random chance factor.

Another model use of a filter is to add code allowing feedback to all pages matching a given regular expression. For example, you might want to give users the ability to add comments to your editorials, but not your articles. Just add a filter with the appropriate URL regular expression.

Need to add a footer to every page on your site? Just use a filter.

Filters can be named. Named filters are useful when you want to have cascading filter permissions. For example, if you want to have authentication levels setup on your site. Imagine placing a User authentication filter to be executed for all pages. Sections of the site that are to be visible to anyone now need to be explicitly declared that way by registering functions. We could add a Visitor authentication filter that returns filter_break_named on the /public folder for example. On the public folder both filters will be executed. But the Visitor authentication filter is registered first so it will run and then the filter_break_named will prevent anymore filters with that name being run so the User authentication filter will be ignored.

7.1.1.5 Sequence of Events

1. Client requests page
2. Register filters
3. Check conditions on all post_auth filters and run if met
4. Run page script
5. Check conditions on all trace filters and run if met

7.1.1.6 Participants

None

7.1.1.7 Collaborations

The developer may register and run a number of post_auth and / or trace filters. The return values of these filter functions influence the continuing execution of the filters. See the Implementation section below for more information.

7.1.1.8 Consequences

1. Configuring conditional code execution is easy. Adding a new logging function or authentication filter is just a filter registration function call away.

2. Be careful of filter side effects. Filters can stop the rest of a page from completing execution or stop other filters from running. You need to pay special attention to the order that filters are registered and their likely return values.

7.1.1.9 Implementation

Filters are run before the page script (post_auth) or after it has completed (trace).

Developers should be able to register any function to be triggered by a filter. The registration should specify the conditions that must be met (or not met) for the filter to be triggered. Registration should give developers the ability to pass variables to the registered function at the time it is run.

The filter function must return one of three values:
- filter_ok
- filter_break
- filter_break_named
- filter_return

The meanings of these values in a postauth filter are:
- filter_ok - continue as normal, including the processing of other matching postauth filters.
- filter_break - do not process any more postauth filters, but continue processing the request
- filter_break_named - do not process any more filters with the same name as the current filter
- filter_return - do not process any more postauth filters, do not process the request. trace filters are run. It is assumed that the filter has returned a proper response to the client.

The meanings of these values in a trace filter are:
- filter_ok - continue as normal, including the processing of other matching trace filters.
- filter_break - do not process any more trace filters
- filter_break_named - do not process any more filters with the same name as the current filter
- filter_return - do not process any more trace filters

7.1.1.10 Sample Code

```
ss_register_filter ($when, $method, $url_pattern, $run_on_match, $function,
        $arg1 = ss_ignore_this_arg,
        $arg2 = ss_ignore_this_arg,
        $arg3 = ss_ignore_this_arg,
        $arg4 = ss_ignore_this_arg,
        $arg5 = ss_ignore_this_arg)
```

Registers a filter to be run when the parameters are matched.

$when can be postauth or trace. A postauth filter is run before the code to generate the page is executed. postauth refers to the fact that this filter is being run after the user has been authenticated by the web server (not by the system). A trace filter is run after the page has been generated. Ideally, it would be after the page has been returned to the user, but Apache/PHP can't do that.

$method is GET, POST or ANY. This allows filters to be applied to only certain types of requests.

If $run_on_match is true then the filter is run only when the request uri matches $url_pattern. If $run_on_match is false then the filter is run when the request uri does not match any of the reg exp's in $url_pattern.

Note that $url_pattern can be any regular expression. Multiple filters can be run for the same request if there are multiple matches. You may also specify a set for $url_pattern. This should be a set of strings which are all the reg exps that this filter is to be used for. If you specify a set of reg exps for the filter then you should be careful that

they do not overlap. For example if you give Set('.*', 'foo*') the filter will be run twice for any url that matches foo*. One easy way to avoid this is to give the filter a name and use filter_break_named as the return value from the filter function.

If this filter is triggered then $function is run and sent the list of arguments. Any argument that is set to ss_ignore_this_arg is not sent to the function. This allows up to five arguments to be defined.

7.1.1.11 Related Patterns

None.

7.1.2 Other Behavioral Patterns

7.1.2.1 Data Entry

This pattern is used to describe a form sequence that facilitates the entry of new data into the database. The data is checked for validity before being entered.

7.1.2.2 Data Edit

Data Edit is a pattern describing the form sequence for editing data that already exists in the database.

7.1.2.3 Data Maintenance

We can combine the ideas of Data Entry and Data Edit into a single pattern Data Maintenance which shares forms and checking code to allow both insertions and editing.

7.1.2.4 Authentication

This pattern describes an interface that abstracts the authentication process from its caller. This allows us to easily switch between authentication schemes or change the site security at any time.

7.1.2.5 Error Handling

The Error Handling pattern describes a process for dealing with errors. Users should be given a suitable message and the site maintainer notified.

7.1.3 Structural Patterns

7.1.3.1 Self-Referring Page

This pattern describes the layout and execution process for a form that has the display elements, validation code and final operations all contained in a single script. The action element for the form refers back to itself.

7.1.3.2 Multiple Page Form

The Mulitple Page Form pattern describes the optimal layout for form display, validation code and final operations across a sequence of pages.

8 Future Work

This catalogue of design patterns is by no means complete or even well written. Perfecting the description of these patterns is an ongoing process that will take a lot of time and collaboration to complete.

No doubt as the catalogue expands so will our understanding of what is required to adequately describe a web programming design pattern. I would expect the design pattern language to evolve and mature over time.