Annales UMCS Informatica AI 3 (2005) 285-292

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

The design patterns in PHP language for the web documents aggregation model

Grzegorz Futa*

Department of Applied Computer Science, M. Curie-Sklodowska University,
pl. M.Curie Sklodowskiej1, 20-031 Lublin, Poland

Abstract

This paper considers the usage of design patterns in PHP language for the documents set model. The author discusses briefly to design patterns and one of the open web documents aggregation model. Some of new features on PHP and its consequences are also presented. The author states that usage of free, powerful development tools and standard problem solving in a new context gives the flexible and professional software building environment.

1. Introduction

1.1. Design patterns

In the seventies of the last century Alexander [1,2] wrote a couple of books dealing with patterns in civil engineering and architecture. The software community subsequently adopted the idea of patterns based on his work, though there has already been burgeoning interest in these ideas of the software community. Patterns in software were popularized by the book [3] by E. Gamma, R. Helm, R. Johnson and J. Vilssides (also known as the Gang of Four, or GoF). While the GoF's work resulted in patterns becoming a common discussion topic in the software development teams, the important point to remember is that the patterns they describe were not invented by the authors. Instead, having recognized recurring design in numerous projects, the authors identified and documented this collection.

Patterns are about communicating problems and solutions. They enable us to document a known recurring problem and its solution in a particular context. One of the key elements in the previous statement is the word recurring, since the goal of the pattern is to foster conceptual reuse over time. Also the documents set model can be designed with usage of some design patterns. The model is the specific application of object aggregation for the specific

* *E-mail address*: grzegorz.futa@umcs.lublin.pl

information system. According to Martin Flower "A pattern is an idea that has been useful in one practical context and will probably be useful in others", I will try to prove that the model and its implementation allow for rapid building of the web sites.

Many software pattern books have been published since the GoF's book. Those books cover patterns for various domains and purposes. In this paper I will try to discuss software design patterns for the document set model.

1.2. Document set model

The document set model was introduced in [4]. The model bases on the previous work on content management systems (CMS) [5]. This section brings up a short description and the most important assumptions of the document set aggregation model.

Let us define the document as any content that can be presented on web pages. It can contain hypertext, images, Flash animations, including Java Script scripts, etc. The document can not be treated as web page that the browser receives from the server. Web page can (but must not) contain the document. The document content is usually shown in the central part of the page. The aggregation model assumes that each of the documents has to belong to some set of documents.

Each of the documents belongs to at least one of the document sets. There is one special document set $\boldsymbol{W}$ that contains any document d_j and all other sets and subsets F_i (Eq. 1). We can say:

$$\bigvee_{W} d_j \in W, F_i \subset W \,. \tag{1}$$

To simplify the model, we assumed another condition (Eq. 2), that states that any document placed on the web does not belong "directly" to the subset $\boldsymbol{W}$.

$$\bigwedge_{j} d_j \in W \Rightarrow d_j \in F_i \,. \tag{2}$$

You can not put any document directly in $\boldsymbol{W}$. Each document has to be placed in some subset F_i. It can be understood, that documents have to be categorized.

Let us define the site structure as all the subsets F_i belonging to the set $\boldsymbol{W}$. We can say that the sets F_i and $\boldsymbol{W}$ are the site structure on $\boldsymbol{W}$. In a particular case, all the sets F_i can be empty sets (Eq. 3). It implies that $\boldsymbol{W}$ is an empty set. The site has only the structure if there exists at least one set F_i and any F_i contains no document:

$$\bigvee_{i} F_i \subset W, \bigwedge_{i} F_i = \{\varnothing\} \,. \tag{3}$$

The next assumption of the model limits documents placement in the sets. The document d_k belongs to the sets F_i and F_j only if the condition written below (Eq. 4) is satisfied:

$$d_k \in F_i \wedge d_k \in F_j \Rightarrow F_j \subset F_i \vee F_i \subset F_j . \quad (4)$$

This limit is the intentional assumption of the proposed model. It causes that a particular document can not be placed in several sets. The detailed discussion on this assumption can be found in [4].

1.3. New features in the PHP language

The PHP language succeeds in an older product, named PHP/FI. The PHP/FI was created by Rasmus Lerdorf in 1995, initially as a simple set of Perl scripts for tracking accesses to his online resume. He chose to release the source code for PHP/FI for everybody to see, so that anybody can use it, as well as fix bugs in it and improve the code. Till the version 3.0 was released (created by A. Gutmans and Z. Suraski) in 1997 the language was not popular. This version contained a lot of bugs that were fixed in the version 4.0+. This release of the server-side scripting language had introduced a simple way of the object-oriented programming. The simplicity of the model did not allow to treat the PHP language as full object-oriented technology.

The latest version of the PHP introduces several techniques that make the language more “objective”. The detailed description of the new features can be found in many publications (see f.e.: [6-9]). The author enumerates only a few the most important features:

- abstract classes,
- interfaces,
- object constructors and destructors,
- multiple class inheritance,
- class members and methods visibility,
- scope resolution operator and static members and object constants,
- class members and methods overloading,
- exception mechanisms,
- template method patterns,
- type hinting and objects cloning.

Many other functionalities of the new preprocessor engine has been omitted as functions that are not crucial in further discussion on the patterns for the documents aggregation model. The most important thing is the objectivity of the programming language. This allows to apply the design patterns for building the structures and components of the software.

2. The patterns

This section discusses some of the design patterns that can be applied in the document set model. The author concentrates only on two patterns, that is:

- Simple Factory and Abstract Factory,
- Decorator.

2.1. Simple Factory and Abstract Factory

In the object oriented programming, object creation is not difficult. The problem appears if object has to be created based on different conditions or context of call. Without applying the factory patterns (e.g. Factory Method, Abstract Factory or Simple Factory) developers spend a lot of hours on debugging and updating the software.

The document set model assumes that the sets can contain any type of document. The document is any data that can be described in the HTML language. Let us consider the situation where the system allows to put two types of document. Those documents are represented by two classes: `SimpleDocument` and `OtherDocument`. Each of the documents implements `Document` interface. There is also a factory class that creates any kind of document depends on type of document (Fig. 1).

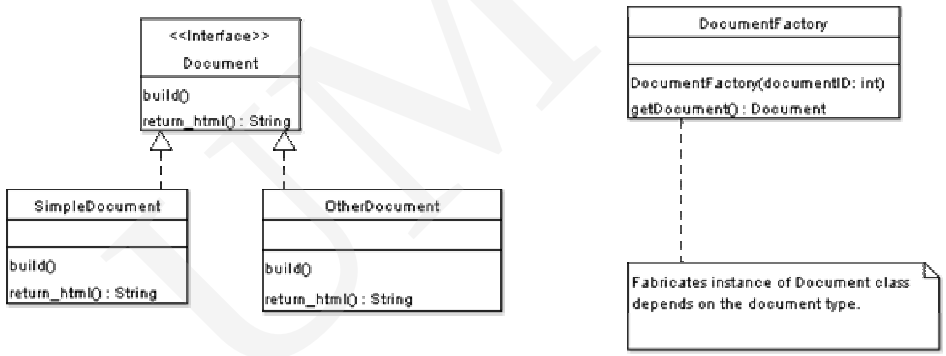

Fig. 1. DocumentFactory class and classes that implement Document interface

The classes related to each of documents are responsible for building the main content of the web page. It does not matter what kind of content they can return (method `return_html()`) or how to prepare the data to be displayed on the browser (method `build()`). The data preparation process depends on the type of document. In the simplest situation document preparation requires getting data stored in some database from one of the tables. More complex documents (e.g. document that represents discussion forum) require getting data form several tables or even connecting to another server. Also the method that generates HTML code returned to the client browser does it in its own way.

The class `DocumentFactory` is responsible for the fabrication of objects of any type of document. The objects are created upon the unique identifier of the document. In the proposed solution, the identifier is a type of integer number. The value of the identifier is passed to the requested page as one of parameters in Unique Resource Locator (URI). The disadvantage of this solution is necessity to define the type of the document upon its identifier. In the

implementation that was made by the author the additional SQL query is required. It does not impact the efficiency of the software. In the case of simple document retrieval, it is possible to receive complete information about the document. The `build()` method that is defined in classes implement `Document` interface does not have to make an additional query. The factory class returns objects of certain type that posses the whole inclusive information (method `getDocument()`).

The exemplary source code (Example 1) illustrate how to use Simple Factory pattern to create documents.

```
$documentFactory = new
DocumentFactory($_GET['docID']);
$someDocument = $documentFactory->getDocument();
$someDocument->build();
$doc_content = someDocument->return_html();
```

Example 1. Sample PHP code that using SimpleFactory design pattern

The variable `$doc_content` contains a sequence of characters that is the part of the page to be sent as a response to the web browser. The huge advantage of using this pattern is the possibility of code separation. The creation of the document content does not affect the main page script.

The document set architecture can be implemented without usage of the Simple Factory design pattern. The extension of the new document type would force changes in the main source code of the page. The design pattern usage allows to extend the system of the new types of documents that can be stored and rendered on the site. It does not require any changes to the skeleton of the system. This gives the easy way to add new functionalities and features.

The Simple Factory design pattern can be used with the Abstract Factory pattern together. The Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes. This pattern takes the abstraction to the next level by providing a common factory interface for a given family of objects. The code that actually uses the factory to create objects only expects an object that conforms to the interface of the abstract factory and does not know any details about concrete factory classes.

2.2. Decorator

The Decorator pattern is used for adding additional functionality to a particular object as opposed to a class of objects. It is easy to add functionality to an entire class of objects by subclassing an object, but it is impossible to extend a single object this way. With the Decorator pattern, you can add functionality to a single object and leave the others like it unmodified.

The documents set model allows to present content of some set as a *bricklet* [4]. The bricklet usually contains a list of documents or links placed in a given set. Depending on the renderer assigned to a certain set, the content of this set can be also rendered as an image with a name of document. If some options are set, the renderer also allows to display some additional metadata that describe the document. Nevertheless, the renderer is responsible only for the content visualization.

Fig. 2. The exemplary bricklets with different Decorator pattern applied

Figure 2 presents two different bricklets with a different content. The standard bricklet should be adapted to graphics style of the website and the corporate identity. Nevertheless, it is sometimes necessary to make one of the page elements visually distinguished. These are usually the navigation patterns or some promoted elements. On the other hand, the model in the sense of way in which the data are stored should use the same mechanisms and it should be independent of the way of visualization.

The diagram in Figure 3 presents the relations between the bricklets, its model and the objects responsible for its decoration.

The Decorator pattern applied in the documents set model allows to add new functionalities to any component that realizes the `VisualWebComponent` interface. This interface forces the `return_html()` method implementation in classes inherited by them. The decorator classes use this method to add different, appropriate for given style borders. The source code (Example 2) shows the pattern usage.

```
$best_bricklet = new Bricklet($someSet);
$best_bricklet->build();
$best_decor = new BestsellerDecorator($best_bricklet);
$content = $best_decor->return_html();
```

Example 2. The exemplary usage of the decorator pattern

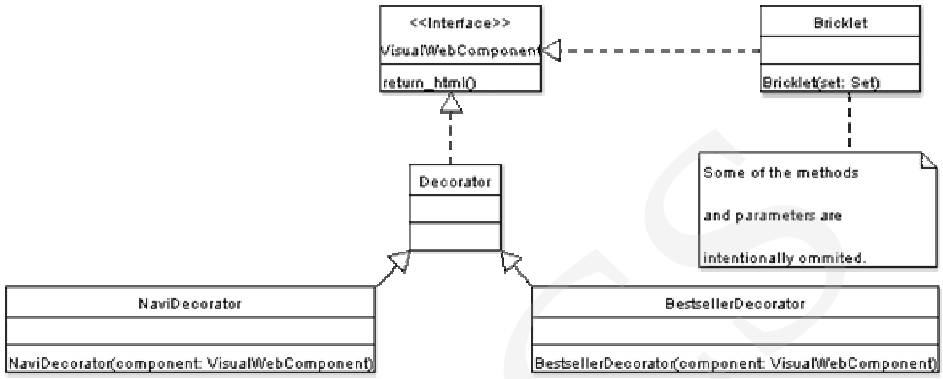

Fig. 3. The relationships between classes for Decorator pattern usage

The variable `$content` contains the HTML code to be displayed in the browser window. As shown in the example the object `$best_bricklet` contains only the data ready to be displayed with the `return_html()` method of `Bricklet` class. The decorator does not affect the content of the bricklet itself. This only adds a new functionality to the bricklet class. The new feature does not change the content, but only extends the possibility of custom visualization.

The clear separation of the presented data and its visualization gives the easy way out to decomposition of work among programmers. It also protects the data stored in the bricklet objects. The limited accessibility to some components can decrease the amount of bugs made by programmers. This can noticeably reduce the time of programming.

Another benefit of usage the decorator pattern is the objectivity. The typical way of decorator implementation in PHP is the *conditional including*. This technique of programming in interpreted languages causes many problems with code maintenance. The technique is very sensitive to the programmer and maintenance time errors. The usage of the decorator pattern eliminates those problems in the design time.

3. Conclusions

The two examples described below show that the idea of documents set model allows to use powerful techniques and methodologies of developing the software. The new features of PHP language, especially full objectivity support, give tools for rapid web sites building to the programmers.

The document set model is not only sophisticated, the theoretical model for document aggregation. The discussion above aimed at showing that it is the flexible and open framework. The Abstract Factory pattern allows to extend the designed web application with the new types of documents and keeps the

easiness of the document management. The decorator pattern presents a new way of graphics style management that was impossible to apply before PHP 5.0.

The standard way of solving problems can be discussed with other patterns proposed by GoF and their successors. The limited size of this publication does not allow to consider other important patterns (e.g. Builder or 2nd tier patterns). Nevertheless, the combination of the design patterns, proposed aggregation model and the extensions to the well known languages seems to give the flexible development framework and environment for building web applications.

References

[1] Alexander C., *The Timeless Way of Building*, Oxford University Press, New York, (1979).
[2] Alexander C., Ishikawa S., Silverstone M., Jacobson M., Fiksdahl-King I., *A Pattern Language*, Oxford University Press, New York, (1977).
[3] Gamma E., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, New York, (1998).
[4] Futa G., *Implementation of documents' set model for corporate miniportals*, Proceedings of the 5th International Conference "Multimedia in Business and Education", Technical University of Częstochowa, Częstochowa, (2005), (in Polish).
[5] Futa G., *The CMS systems for small and medium enterprises*, Proceedings of the Third KEI, PWSZ, Chełm, (2004) 37, (in Polish).
[6] Bergmann S., *The Template Method Pattern in PHP 5*, PHP 5 Feature Spotlight on http://www.zend.com/, Cupertino, July (2003).
[7] Bergmann S., *Introduction to Interceptors I: Implementing Delegation*, PHP 5 Feature Spotlight on http://www.zend.com/, Cupertino, August (2003).
[8] Bergmann S., *Introduction to Interceptors II: Implementing Lazy Initialization*, PHP 5 Feature Spotlight on http://www.zend.com/, Cupertino, August (2003).
[9] Hojtsy G., *PHP Manual*, PHP Documentation Group, (2005)