Documenting Aspect-Oriented PHP (AOPHP)

John W. Stamey, Jr.
Department of Computer Science
Coastal Carolina University
jwstamey@coastal.edu

Bryan T. Saunders
Department of Computer Science
Coastal Carolina University
btsaunde@coastal.edu

ABSTRACT

This paper describes the process for documenting programs using Aspect-Oriented PHP through AOPHPdoc. We discuss some of the problems involved in documenting Aspect-Oriented programs, solutions to these problems, and the creation of documentation with AOPHPdoc.

Categories and Subject Descriptors

D2.10 [**Software Engineering**]: Design
D.3.2 [**Programming Languages**]:Language Constructs and Features

General Terms

Documentation, Languages

Keywords

Documentation, Aspect-Oriented Programming, Aspect-Oriented Documentation, AJdoc, PHP, Aspect-Oriented PHP, AOPHP

1. INTRODUCTION

The AOPHPdoc tool is the standard API documentation generation program for Aspect-Oriented PHP (AOPHP, found at http://www.aophp.net/). It performs two important functions. First, it generates webpages from documentation comments. Second, it generates color-coded XHTML that represents the final source-code level woven code. Both of these are important for documentation and debugging of systems written in AOPHP.

Friend [1] and Kramer [2] have underscored the importance of generating documentation from comments in source code. At a more fundamental level, Lopes, et. al [3] note the difficulty of existing descriptive mechanisms in conveying relevant information about software systems in general.

A fundamental question in the development of documentation systems is "What can be reasonably developed and provided to assist developers with the platform at hand?" This question is of enormous importance when dealing with AOP, wherein the final source-code or byte-level code is never actually available to the developer for inspection.

2. DOCUMENTING ASPECT-ORIENTED PROGRAMS

Aspect-Oriented Programming takes lines of code from a file (the advice file) and inserts these lines (through a process called weaving) into the target source file. Weaving is accomplished at either the byte-code level (in Java) or at the source-code level (in PHP with AOPHP). [4] A major problem from the standpoint of documentation (and debugging) is that the final code to be executed is not available for inspection by the developer. The absence of such final code leads to a problem in testing, debugging, and documentation. [5]

The primary tool used to document Aspect-Oriented code in Java is AJdoc.[6] It is available as plugin for the Eclipse Development Environment, and may be found at http://www.eclipse.org/. AJdoc is implemented in the style of JAVAdoc, creating a hyperlinked API for use by the developer.

3. PHP DOCUMENTATION

Five primary documentation tools are currently in use. There are two basic architectures. The primary approach to documentation features output that has the look and feel of the hypertext-based JavaDOC architecture, originally proposed by Friendly [1] and developed by Kramer [2]. A secondary approach to documentation is the generation of comments. These include:

1. *phpdoc.org* PHPDocumentor, written in PHP, is considered the "standard" for PHP documentation creates the same type of hyptertext-based documentation as JavaDoc.
2. *phpdoc.de* Similar in look and feel to PHPdocumenter, the open source PHPDoc is another documentation tool written in PHP.
3. *sourceforge.net/projects/phpautodoc* PHPautodoc is another clone of PHPdocumentor.
4. *sourceforge.net/projects/phpdoctor* PHPDoctor is a JavaDoc style comment parser for PHP code, written in PHP using the PHP tokenizer extension.
5. *phpscribe.sourceforge.net* phpScribe is a PHP documentation tool that parses comments to generate documentation.

4. AOPHP DOCUMENTATION

A different documentation strategy, other than the JavaDoc style or comment-generation, is needed to deal with code

resulting from Aspect-Oriented Programming. The main problem is that code segments (called *advice*) are inserted, or *woven*, into PHP code. Given segments of advice code, they can be woven into code based on the presence of specified user-defined functions.

The PHP engine preprocesses existing code. AOPHP operates as an additional front-end preprocessor, inserting and changing code as necessary. Code segments can execute BEFORE named functions, AFTER named functions, or based on a CONDITION present in proximity of named functions. These three types of code insertions are called, respectively, *before advice*, *after advice* and *around advice*.

The following steps are used to generate the woven code with AOPHPdoc. A byproduct of the weaving is an HTML file containing the original PHP code (in black) and the woven advice code (in red). The documentation from the woven code can be generated from a command line in Linux. The command to do so is:

$*aophp <input filename> <output filename> -d*

When this command is invoked, the AOPHP preprocessor then performs the following steps:

1. OPEN FILES: Open and read the PHP file into which aspects are to be woven, stripping off comments and extra white space

2. READ ASPECTS: Read aspect file names given in the aophp_init() function

3. OPEN ASPECT FILES:

a. Parse out individual aspects from the aspect files (there may be more than one aspect in the file);
b. Parse the aspects to identify the named pointcuts and advice; and,
c. Load names of advice into a table.

4. GENERATE HELPER FUNCTIONS: Helper functions are functions that contain the advice to be woven. An important implementation issue with helper functions is that the names of these PHP functions must be unique. As the helper functions are generated by AOPHP, the PHP function names serving as pointcuts are generated using the MD5 Checksum, [7] giving the very low chance (1:10 billion) of name duplication. Generated function names are of the form:

ao_*CODE_MD5HASH_NAME*

where

- *CODE* represents the type of advice - bf(before), ar(around), af(after), fi(final);
- *MD5HASH* is the MD5 Checksum based on original function signature; and,
- *NAME* is the actual name of original function.

5. WEAVE CODE: Here, helper functions are inserted into the PHP code. This version of the code will serve as the actual input to the PHP preprocessor. At the joinpoints, the calls to the original functions are replaced with calls to the helper functions.

6. DOCUMENTATION OPTION: If the *-d* flag is set, then the color-coded HTML file is written.

Three segments of example code are found in Figures 1, 2 and 3. Figure 1 is an example of PHP source code. The simple logic adds two numbers and then divides two numbers. In advance, we know that division by zero will be problematic. The *aophp_init( )* function indicates the name of the file containing the aspect code (advice) will be *aspect.php*.

The advice file, Figure 2, contains two things worthy of mention. First, we see a named pointcut, *add( )* as well as some before and after advice.

In Figure 3, we see a sample of the HTML output from the AOPHP preprocessor, with woven code in red. The helper function, *ao_fi_MD5HASH9346_add($a,$b)* actually controls the calling of the before advice, *ao_bf_MD5HASH9346_add($a,$b)*.

```
<?php
function add($x,$y){
        return $x+$y;
}
function div($a,$b){
        return $a/$b;
}
aophp_init("aspect.aophp");
$a = 4;
$b = 5;
$c = add($a,$b);
echo $c;
$d = div($c,5);
echo $d;
?>
```

Figure 1. PHP source file

5. FUTURE WORK AND DEVELOPMENT

The future of Aspect-Oriented Programming will be closely tied to the associated tools to provide clear and readable system documentation. HTML documentation available from AOPHPdoc is an important first tool for PHP programmers choosing to use the power of Aspect-Oriented Programming in their development. Currently, work is underway for the *–d* flag to produce documentation similar to JAVAdoc and AJdoc.

```
aspect Math {
        pointcut add = execr(math.add($x,$y));
        before($p1,$p2): add {
                echo "Adding $p1 and $p2";
        }
        around($p1,$p2): execr(math.div($a,$b)) {
                if($p2 == 0){
                        return -1;
                }else{
                        proceed($p1,$p2);
                }
        }
}
```

Figure 2. Aspect file

```
<?php
function add($x,$y){
        return $x+$y;
}
function div($a,$b){
        return $a/$b;
}
$a = 4;
$b = 5;
$c = ao_fi_MD5HASH9346_add($a,$b);
echo $c;
$d = ao_fi_MD5HASH2847_div($c,5);
echo $d;
function ao_bf_MD5HASH9346_add($x,$y){
        echo "Adding $x and $y";
}
function ao_ar_MD5HASH2847_div($a,$b){
        if($b == 0){
                return -1;
        }else{
                div($a,$b);
        }
}
function ao_fi_MD5HASH9346_add($x,$y){
        ao_bf_MD5HASH9346_add($x,$y);
        $temp = add($x,$y);
        return $temp;
}
function ao_fi_MD5HASH2847_div($a,$b){
        $temp = ao_ar_MD5HASH2847_div($a,$b);
        return $temp;
}
?>
```

Figure 3. Woven Code

6. REFERENCES

[1] Friendly, A. The Design of Distributed Hyperlinked Protramming Documentation, Presented at the International Workshop on Hypermedia Design ’95.

[2] Kramer, D., API documentation from source code comments: a case study of Javadoc. In *Proceedings of the 17th annual international conference on Computer documentation*, SIGDOC '99.

[3] Lopes, C.V., Dournish, P., Lorenx, D.H. & Lienerherr, K. Beyond AOP: Toward Naturalistic Programming. In *Proceedings of OOPSLA ’03*, Anaheim, CA.

[4] Laddad, R.V. (2003) *AspectJ in Action.* Greenwich, CT: Manning Press.

[5] Stamey, J.W. & Saunders, B.T. Unit Testing and Debugging with Aspects. In P*roceedings of ACM CCSC-NE* , 2005.

[6] *ajdoc: the AspectJ documentation tool.* Found at: http://www.eclipse.org/aspectj/doc/released/devguide/ajdoc-ref.html/.

[7] Rivest, R. 1992 The Md5 Message-Digest Algorithm. RFC. RFC Editor.