The Aspect-Oriented Web

John Stamey
Department of Computer Science
Coastal Carolina University
Conway, SC
843-349-2552
jwstamey@coastal.edu

Bryan Saunders
Department of Computer Science
Coastal Carolina University
Conway, SC
843-349-2552
btsaunde@coastal.edu

Simon Blanchard
Department of Management Science
HEC Montréal and GERAD
Montréal, Canada
514-340-6053 ext. 6052
simon.blanchard@hec.ca

ABSTRACT
We examine Aspect-Oriented Programming (AOP) as it applies to web development. XHTML was designed to separate form from content, a fundamental principle of separation of concerns in AOP. Cascading Stylesheets and Javascript naturally provide support for AOP. The release of AOPHP (Aspect-Oriented PHP) provides a more traditional way to implement AOP in the LAMP (Linux, Apache, MySQL, PHP) web development environment, weaving advice code into PHP source code prior to the PHP pre-processing step.

Categories and Subject Descriptors
D.2.10 **[Software Engineering]** Design; D.3.2 **[Programming Languages]** Languages

General Terms
Documentation, Design, Languages, Standardization, Theory.

Keywords
Aspect-Oriented Programming, AOPHP, Web Development, Separation of Concerns, AOP-Like Behavior, LAMP

1. INTRODUCTION
Lammel and De Schutter (2005) have show how COBOL provided support for aspect-oriented programming constructs since the 1960s, with statements such as DECLARATIVES, USE FOR DEBUGGIN, and USE BEFORE REPORTING. This early support for aspects is done directly by the COBOL compiler, and does not rely on the need for weavers and development environments. HTML and XHTML have shown support for elements of Aspect-Oriented Programming (AOP) since the development of Cascading Stylesheets (CSS) in the early 1990s. Similar to COBOL, Aspect-Oriented constructs on the web are implemented directly by the web server.

This paper will discuss programming constructs and ideas that are well-known to object-oriented programmers in C-styled languages (C, C++, Java and PHP). These constructs include separation of concerns and identification of aspects, joinpoints, pointcuts, and advice. The identification of Aspect-Oriented in HTML/XHTML provides a more complete view of native support for AOP. We then discuss how AOP may be integrated into web development through Aspect-Oriented PHP (AOPHP). The use of AOP in web development is intended promote coding that is more efficient and easier to maintain, as well as programming architectures that are better organized and more extensible. These code improvements will ultimately lead to improved implementation of the design of communication on the web.

2. THE ASPECT-ORIENTED PARADIGM
2.1 About The Paradigm
Aspect-oriented programming centers around the identification of concerns, reminiscent of modularization, that are found in various parts of a programming system, and the effective management and reuse of the associated code. Breaking programs in modules is present in some form in most if not all programming languages. Parnas[1] identified benefits of modularization, including the reduction of development time because of the divide and conquer approach, as well as increasing software flexibility and understanding. Our discussion in this section describes concepts in AOP from the world of C-style languages (Java, C, C++, PHP). Subsequent sections deal with concepts from AOP as they relate to web programming.

A concern is simply any system-level requirements [9]. The separation of concerns paradigm is an approach to software development with a goal of separating main program code from special purpose concerns [10]. Special purpose concerns include ancillary tasks that are necessary for a program to perform its required tasks, yet are not really part of the algorithmic process.

Tasks such as logging or synchronization are usually found in multiple locations within a system, and are given the name crosscutting concerns. Encapsulation and modularization of crosscutting concerns has found to provide a number of advantages, including optimization of memory usage, ease of developing and maintaining code, and facilitating propagation of changes throughout the system when code is executed [2]. Tasks that are directly related to the algorithm, or data manipulation, are called core concerns. The goal of separation of concerns is to encapsulate the crosscutting concerns from the core concerns.

Two implementation problems surface when crosscutting concerns are not encapsulated, or separated, from core concerns. Code scattering occurs when a crosscutting concern (such as logging) is implemented in more than one location. Such duplication leads to duplicate code throughout a system that becomes difficult and time-consuming to maintain. Code tangling occurs when related tasks have multiple, or slightly different, implementations. Related tasks ideally should be implemented in the same module.

The code to implement crosscutting concerns is called advice. The execution of advice code is triggered by points in code such as the execution of functions/methods, instantiation of objects, or assignment/access of variables. These points of code that trigger the execution of advice are called joinpoints. A collection of logically related joinpoints is called a pointcut.

Advice code is executed based on the existence of a joinpoint. It resides in a file separate from the code in which it "invades," or changes, at the time of execution. Before-advice is, as its name indicates, code that is executed before a joinpoint. After-advice is code that is executed after a joinpoint. Around-advice checks the state of input parameters (the set of which is collectively called context) and makes decisions to execute or not execute advice based on the conditions of the variables.

2.2 AOP-Like Behavior

Aspect-Oriented programming languages have several characteristics that separate them from other languages. First, there is really no such thing as an aspect-oriented programming language. Some languages, such as Java and C++, have add-ons to their development environments that weave advice code into the source (target) code. An example of this is the AspectJ plugin for Java. *Five characteristics of AOP languages and AOP-Like Behavior* include [17]:

1. Advice code resides in files separate from the source code
2. Advice code is integrated into (woven into) code at runtime
3. Weaving is done at either the source-code level, or the byte-code level
4. When advice is changed, the behavior of the entire system changes
5. Aspects are a way to implement cross-cutting concerns, and are not to be confused with modularization of core concerns

This paper describes a number of behaviors that have been observed in a programming environment that is not traditionally thought of as an AOP environment. It is interesting that the behavior of website programming languages have always exhibited a number of Aspect-Oriented properties. COBOL was the first programming language that included elements of AOP. [12] The identification of such behaviors expands our understanding of an existing development paradigm, and will hopefully lead to more exploration of properties found in web development. We also introduce AOPHP, the first AOP environment developed for web programming.

3. SEPARATION OF CONCERNS

3.1 Separating Form and Content

The philosophy of separation of concerns has long been a part of web programming. Early on, the separation of document structure from the document layout had been a goal of HTML from its inception in 1990 [6]. Later on, the main goal of XML was to provide a clear separation of data from the presentation of the data [5]. Such a separation creates truly portable textual data. A frequently used phrase that embodies this paradigm is the separation of form and content [4].

Before the use of stylesheets, web developers focused on static, or absolute, placement of content. Markup code to implement such placement was too often written with content tangled together with the presentation mechanisms. Tables, originally meant to represent tabular data, were used to create complete formats for webpage layout. Blockquotes were used to provide global page formatting, rather than to set apart and highlight smaller sections of text.

In addition to tangling of presentation markup code with content, repeated and almost identical markup code was used in different pages in a website. Such scattering of similar code segments increased difficulty for maintaining websites in the face of sometimes rapid change. Errors would occur when many pages had to be frequently edited and changed; this lead to potential errors and sometimes lengthy maintenance tasks.

3.2 Development of Dynamic Content

Server-Side Includes, or SSIs, are statements in webpage code that are expanded by the server before the file is returned to a browser. SSIs can also be thought of as macros that are expanded by the web server[1]. SSIs can be deployed in HTML and XHTML, as well as through the use of middleware such as CGI, PHP, ASP and ColdFusion. SSIs originated the mid '90s with dynamic webpages, when the content of a page was framed by the header, navigation, and footer, which remained constant. A typical example of code to create dynamic content is seen in the PHP code in Figure 1:

```
<?php
// PHP Code to Include Content
$file = $_REQUEST['nextpage'];
    // $file is the URL variable nextpage
 if($file == "" || $file == null){
    // however, if $file is not defined
 $file = "home";
    // then assign it the string home
 }
 $file .= ".php";
    // concatenate .php to the
    // end of the filename string
    // to create the name of a valid PHP
    // file such as home.php
 include($file);
    // open the specified file and
    // include that code in the webpage
 ?>
```

Figure 1

SSIs are certainly one way to implement aspects in web programming. However, we must be careful to differentiate between modularization and aspect-oriented programming. A typical example of modularization occurs when an input form, used in several different source pages, is placed in a SSI; calling the body of a webpage, seen in the example above, is another example of modularization. It has long been known that modularization can decrease errors that can occur when changing code that repeats throughout a website [1].

While modularization is typically associated with core concerns, AOP is a collective term for the practice of modularizing crosscutting concerns. A typical example of this may be seen in the AOPHP (Aspect-Oriented PHP) code segment shown in Figure 2 that checks for division by zero [9]:

```
  around(): execr(div($a,$b)){
    if($b == 0){
      echo "Cant Divide by 0, Returning -1<br>";
      return -1;
    } else {
      echo "Division Allowed, Proceeding<br>";
      proceed($a,$b);
    }
  }
```

Figure 2

Javascript is known for providing client-side functionality in webpages. Cascading Style Sheets (CSS, singular; CSSs, plural) are used to provide uniform formatting throughout a website [14]. Both will be seen to provide elements of AOP in web development.

4. JOINPOINTS/POINTCUTS & WEAVING

In Java with the AspectJ plug-in ("Java+AspectJ"), *joinpoints* are identifiable execution points in a system at which advice code is executed. Joinpoints trigger execution of advice code, either before their execution, after their execution, or based on the state of the variables at the time of execution. Common Java+AspectJ joinpoints include

- At method call, or at the beginning of method execution
- At constructor call, or at the beginning of a constructor call
- When a variable is changed through assignment
- When a variable is used (read)
- When an object is instantiated

Six categories of joinpoints we have identified in W3 CSS2 selectors specification [11], including those seen in Figure 3. These joinpoints correspond to tags in the source code. XHTML joinpoints may be coded either in-line in the XHTML or in the CSS. A pointcut would be a logical collection of these joinpoints. An example of an XHTML pointcut is the following two font type selectors, with associated class selectors:

```
font.headline {...}
font.bodyCopy {...}
```

CSS allows additional extensibility, whereby pointcuts may be combined.

Selectors in CSS
Type – types corresponds to XHTML tags; an example is the `font` tag
Type Selector - the name of a document language element, such as `h1`;
Attribute Selector – specifies values of parameters for a type selector; an example is: `h1 { font-family: sans-serif };`
Class Selector – assigns style to the selectors to which it is appended; an example is `font.bodyCopy` , where bodyCopy is defined as `bodyCopy { font-family: sans-serif };`
Universal Selector – is available as a class to any type selector; an example is `*.bodyCopy { font-family: sans-serif };`, where * is optional
ID selectors – adds a specific match to a type associated with an ID, such as div=page1; an example is `h1#page1 { color: red };`
Pseudo-classes – permit formatting based on information that lies outside of the document tree; an example is `a:link {color:"blue"};`

Figure 3

A *pointcut* define a set of related joinpoints. In Java+AspectJ, `*.foo( )` is a pointcut of all methods named foo() in any class. In web development, we can say a pointcut is any logically related set of joinpoints. One example would be the set of tags using Javascript function `onSubmit()`.

Weaving is the process of combining advice code with source code to produce either a source-code or a byte-level code version of the code that includes both source code and advice code. For AOP implementations of Java and C++, the actual woven source code is virtual, in as much as there is no way to see the final woven code. This is not the case with web programming. The *view source* option in the browser allows woven code of HTML/XHTML to be seen, establishing a connection with Characteristic 3 of AOP-Like Behavior.

5. ADVICE

5.1 Implementation

CSS syntax is composed of three parts, the selector, the property and the value [15]. The general form of the syntax is:

```
selector {property: value}
```

Selectors represent HTML/XHTML tags; properties represent the attributes of the XHTML tag that are to be changed; and, properties are modified by a set of values at runtime. Such modification at runtime establishes a connection with Characteristic 2 of AOP-Like Behavior. Two simple examples include:

```
body {color: "silver"}
ul {list-style-image:url('arrow.gif')}
```

Selectors are specified inside a CSS, and are responsible for setting styles and formatting when associated tags are found in the XHTML source. The placement of this control information separate from the tags allows tags to be oblivious to the specific formatting instructions found in the CSS This placement establish a connection between CSS selectors and Characteristic 1 of AOP-Like Behavior [12]. As both of these selectors determine formatting prior to the display of text and images in the body of the page, and prior to display of the unordered list, they may be classified as Before-advice. The body selector provides universal quantification of the background color over the entire page, and the unordered list selector provides point-wise quantification of the style of the bullets wherever an unordered list appears on the page [13]. Any change in the selector, property, or value immediately change the behavior of the entire website, establishing a connection with Characteristic 4 of AOP-Like Behavior.

Java+AspectJ pointcuts may be composed through the operations or ("||"), and ("&&") and not ("!"). Similarly, pointcuts in CSS may be composed using the .class operator:

```
.       redColor { color : red }
        font.prettyBig { font-size:18 px; }
```

may be combined as follows (using an in-line example):

```
        <span class="font.prettyBig.redColor">
```

In examining the ANCHOR tag, there are certain characteristics of the display that are based on the state of certain variables. A set of Boolean variables contain the following information:

- Was the link visited previously?
- Is the mouse pointer on top of the hyperlink?
- Is the mouse currently being clicked to enable the hyperlink?

These three conditions must be identified by the browser, and have the state seen in Figure 4:

Condition	"On" State	"Off" State
link_visited	visited	not visited
mouse_location	over hyperlink	not over hyperlink
mouse_action	clicked	not clicked

Figure 4

The three Boolean variables listed above that contain this information are collectively called the *context* of the advice.

Below, we see an example of the ANCHOR selectors, as they are used to control the links on a webpage:

```
        a:link {color:"blue";
                text-decoration:none;
                font-family:verdana; }
        a:visited {color:"blue";
                text-decoration:none;
                font-family:verdana; }
        a:active {color:"navy";
                text-decoration:none;
                font-family:verdana; }
        a:hover {color:"navy";
                text-decoration:none;
                font-family:verdana; }
```

The basic algorithm to determine the proper action is seen in Figure 5. This algorithm implements Around-advice, based on completion of a jump to the URL specified by the hyperlink. More traditional Before-advice and After-advice may be implemented in CSS with before and after pseudo-elements. In an example from http://www.aopworkshop.org/, the text "This website courtesy of www.aophp.net" is displayed after the `div` theBody (Figure 6).

```
IF (mouse_location over hyperlink)
    IF (mouse_action clicked)
        RETURN (Use color from a:active)
        PROCEED (GO TO URL
               REPRESENTED By HYPERLINK)
    ELSE
        RETURN (Use color from a:hover)
ELSE  // mouse_location not over hyperlink
    IF (link_visit visited)
        RETURN (use color from a:visited)
    ELSE
        RETURN (use color from a:link)
```

Figure 5

```
   #theBody:after {
        content: "This website courtesy
               of www.aophp.net";
        display: block;
        background: #ffc;
        width: 100%;
        margin-top: 2em;
        text-align: center;
        }
```

Figure 6

CSSs allows the option for a page to display according to the media type of the reader. The following CSS code changes the font based on the page being printed or read on a screen, another example of Before-advice:

```
        @media screen
        { body {font-family:verdana,sans-serif;
               font-size:14px} }
        @media print
        { body {font-family:times,serif;
               font-size:10px} }
```

5.2 Javascript

No discussion of implementing advice in XTHML would be complete without examining Javascript functions and the way they may be invoked in an aspect-oriented manner. In Figure 7, we see code to implement Around-advice. A class is selected based on the value of the field size that is chosen in the select box. The selection is made in the Javascript function `setClass()`, that simulates Around-advice. In the head section of a webpage, we find the following tag, indicating the location of the external JavaScript file:

```
        <SCRIPT LANGUAGE="JavaScript"
        SRC="validate.js"></SCRIPT>
```

and the FORM tag, which acts like a joinpoint for the `OnSubmit()` function. The Javascript is contained in a separate file, just as Java+AspectJ would do. Then, any modification in `validate.js` (above) would affect the code (below) when it is executed. The resulting action is Around-advice, either allowing the Posting action to occur, or showing a Javascript Alert Box if there is not an ampersand in the email string. In all, the effect of Around-advice can be achieved from Javascript events, with the advice code contained in the .js file. Weaving takes place at runtime.

```
<head>
<title>Example</title>
<script language="JavaScript">
function setClass()
{
if(document.ex1.size.value=="1"){
        document.all.hw.className =
        'text10';
}
else if (document.ex1.size.value== "2"){
        document.all.hw.className = 'text12';
        }
else if (document.ex1.size.value == "3"){
                document.all.hw.className =
'text14';
        }
}
</script>
<style type="text/css">
 .text10       { font-size: 10pt; }
 .text12       { font-size: 12pt; }
 .text14       { font-size: 14pt; }
</style>
<body>
<form method="post" action="bob.php" name="ex1">
<select size="1" name="size"
onClick="setClass()">
        <option value=1>1</option>
        <option value=2 selected>2</option>
        <option value=3>3</option>
</select>
</form>

<div class="text12" id="hw">
Hello World
</div>
</body>
</html>
```

Figure 7

```
// validate for @ in an email address
function validate()
{
x=document.myForm
at=x.email.value.indexOf("@")
if (at == -1)
        {
        alert("Not a valid e-mail")
        return false
        }
}
```

Figure 8

Figure 8 shows a JavaScript function (from W3 Schools, [15]) to validate one characteristic of an email address – the presence of an ampersand. Such a task may be needed for any input form that gathers visitor information in a website. For that reason, email validation is a crosscutting concern (characteristic 5 of AOP-Like Behavior). This code may be placed in an external file, `validate.js`, to create a more traditional arrangement of the aspect code in a separate file. The joinpoint is then established by the `onSubmit()` function found in the form tag in Figure 9.

```
<html>
<head>
<SCRIPT LANGUAGE="JavaScript"
SRC="validate.js"></SCRIPT>
</head>
<body>
<form name="myForm" action="tryjs_submitpage.htm"
     onSubmit="return validate()">
Enter your E-mail:
<input type="text" name="email" size="20">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Figure 9

6. ASPECT-ORIENTED PHP

AOP has been introduced into PHP through the AOPHP project [18]. An AOPHP preprocessor identifies pointcuts and inserts code at the proper location, accomplishing source-code level weaving of aspects. Advice is coded and placed in separate .aophp files. At the top of each PHP file, the name of the aspect file to weave with that particular PHP file is specified. An example of the code is:

<?aophp file="Filename" debugging="on" ?>

Tasks may be handled server-side with AOPHP that might formerly have been done client-side in Javascript. In addition, crosscutting concerns can now have separate implementations and enjoy benefits of encapsulation; some typical concerns include form data validation, error handling, debugging, logging, and concurrency control (a task that is not handled by web databases such as MySQL).

AOPHP V3.0 will allow seven different ways in which advice can execute upon a function call. These are listed in Figure 10.

Application of Advice
Before advice only
Around advice only
After advice only
Before advice followed by around advice
Before advice followed by after advice
Around advice followed by after advice
Before advice followed by around advice followed by after advice

Figure 10

At runtime, all incoming script requests are processed through Apache's mod_rewrite module. If the script requested is of type PHP, then the following three steps occur:

- Specified .aophp files and the requested PHP script are fed into the AOPHP pre-processor.
- The output file, expanded to include the aspects, is saved as a temporary file
- The temporary file is then used as input to PHP to perform the requested actions, expanded to include the aspects.

The three types of joinpoints will ultimately be implemented in AOPHP V3.0 and higher: exec (a hybrid of execution and call from AspectJ), set and get. The syntax may be seen in Figure 11.

Join Point Category	Pointcut Syntax
Function Call	exec(FunctionSignature) with BEFORE, AROUND and AFTER advice
Field Read Access	AOPHP V 4.0 (future)
Field Write Access	AOPHP V 4.0 (future)

Figure 11

Generic examples of syntax for BEFORE and AFTER advice may be seen in Figure 12.

```
    AOPHP_before(FunctionSignature)
    // PHP function statements
    #
    AOPHP_after(FunctionSignature)
    // PHP function statements
    #
```

Figure 12

AROUND advice has the following syntax, and also allows the use of the keyword `proceed()`. Context is captured by the advice through the `ParameterList` and passed to statements for execution. This is depicted in Figure 13.

```
    AOPHP_around(FunctionSignature(
          ParameterList))
    // PHP function statements
    // ...
    // PHP function statements
          {
          // PHP function statements
          proceed(ParameterList)
          // PHP function statements
          }
         #
```

Figure 13

We will select an example that is used in http://www.aopworkshop.org/, the website for the Brazilian Aspect-Oriented Programming workshop. The website has a form page (`form1.php`) that lets interested parties give their name, email address, country and postal code in exchange for periodic emails about the workshop. The ACTION part of the form is a simple MySQL database insert, coded in Figure 14. The Around-advic,e found in file `mail.aophp`, may be seen in Figure 15:

7. IN CONCLUSION

We have seen that web development in LAMP can include AOP from two different perspectives. HTML/XHTML have a number of Aspect-Like features that have been built into the languages from their beginning. Cascading Stylesheets and Javascript are responsible for a majority of this behavior. AOPHP provides true aspect-oriented programming to implement crosscutting concerns such as validation and logging. Traditional AOP is found in languages such as Java and C++. It is hoped that our discussion of the Aspect-Oriented Web will help programmers understand the thought process behind AOP as it relates to web development.

```
function addUser($fname,$lname,$email,
        $country,$areacode){
    $sql = "INSERT INTO list
    VALUES('','$fname','$lname','$email',
        '$country','$areacode')";
    $results = mysql_query($sql) or
        die("Could Not Add Email.<br>");
    ("Could Not Add Email.<br>");
    echo "<b>Email Added</b><br><br>";
    echo "<b>Added:</b> $fname $lname
        $email<br>";
    echo "<br><br><a href=index.php>
        Return to Home Page</a>";
}
```

Figure 14

```
around(): exec(addUser($fname,$lname,$email,
     $country,$areacode)){
 $correct = 0;
 if(empty($country)){
  $correct = 1;
 }else if(empty($areacode)){
  $correct = 1;
 }else if((strrpos($email,'@')===false)||
     (strrpos($email,'.')===false)){
  $correct = 2;
 }
 if($correct == 0){

proceed($fname,$lname,$email,$country,$areacode);
 }else if($correct == 1){
  echo "Invalid Country or Postal Code<br>";
 }else if($correct == 2){
  echo "Invalid Email Address<br>";
 }
}
```

Figure 15

8. ACKNOWLEDGEMENTS

The authors would like to thank Jean-Louis Lassez for his support and encouragement, and Gregor Kiczales for his advice and support.

9. REFERENCES

[1] Parnas D.L., On the Criteria To Be Used in Decomposing Systems into Modules, *Communications of the ACM 15*, 5 (December 1972), 1053-1058.

[2] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M., and Irwin, J., Aspect-Oriented Programming, In *Proceedings of ECOOP 1997*, (Finland, June 1997). 25pp.

[3] Lerner, R.M, At the Forge: Server-Side Includes, Linux Journal, June 1998. Archived at http://www.linuxjournal.com/article/2919

[4] Scheuhammer J., Accessibility, and Separating Form from Content, Adaptive Technology Resource Centre, University of Toronto, July 2000. Archived at http://www.utoronto.ca/atrc/reference/staff/scheuhammer/musings/FormContent/AccessFormContent.html

[5] Extensible Markup Language, W3C Working Draft, November 1996. Archived at http://www.w3.org/TR/WD-xml-961114.html

[6] Lie H.W., Bos B., Cascading Style Sheets – Designing for the Web 2nd edition, Addison Wesley, 1999.

[7] Modularization of XHTML™ 1.0, W3C Working Draft, February 2004. Archived at http://www.w3.org/TR/2004/WD-xhtml-modularization-20040218/

[8] Colyer A., Clement A., Aspect Librairies of Pre-Built Components in Large-Scale AOSD for Middleware, In *Proceedings of the 3rd International Conference on Aspect-Oriented Software Development*, March 2004. ACM Press New York, NY, USA.

[9] Stamey J.W., Saunders B., Cameron M., Introducing AOPHP, International PHP Magazine. To appear June 2005.

[10] Hursch W.L., and Lopes C.V., *Separation of concerns*, Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA, February 1995.

[11] Selectors, W3C Candidate Recommendation, November 2001. Archived at http://www.w3.org/TR/2001/CR-css3-selectors-20011113/

[12] Laemmel R., De Schutter, K., What Does Aspect-Oriented Mean to Cobol ?, In *Proceedings of Aspect Oriented Software Development 2005, March 2005, ACM Press* New York, NY, USA.

[13] Filman R.E., Friedman D.P., Aspect-Oriented Programming is quantification and obliviousness. In M. Aksit, S. Clarke, T. Elrad and R.E. Filman, editors, *Aspect-Oriented Software Development*. Addison-Wesley, Reading, MA, September 2004.

[14] Badros G.J., Borning A., Marriott K., Stuckey P., Constraint cascading style sheets for the web. In the *Proceedings of the ACM Symposium on User Interface Software and Technology*, November 7-10, Asheville, USA, 1999.

[15] W3School. Visited on June 9th at http://www.w3schools.com/

[16] Javascript DOM examples, Archived at: http://www.w3schools.com/js/js_examples_3.asp

[17] Kiczales, G. et. al., Aspect-Oriented Programming, ACM Computing Surveys 28-4, December 26, 1996.

[18] Stamey J.W., Saunders B., Aspect-Oriented PHP website. Visited on June 9th 2005 at: http://www.aophp.net