Using Aspect-Oriented PHP to Implement Crosscutting Concerns in a Collaborative Web System[*]

Otávio Augusto Lazzarini Lemos,
Daniel Carnio Junqueira,
Marco Aurélio Graciotto Silva and
Renata Pontin de Mattos Fortes
Depto. de Computação
ICMC/USP - São Carlos
Caixa Postal 668
São Carlos, SP, Brasil 13560-970
[oall,danielcj,magsilva,renata]@icmc.usp.br

John Stamey
Department of Computer Science
Coastal Carolina University
P.O. Box 261954
Conway, SC, USA 29528-6054
jwstamey@acm.org

ABSTRACT

Aspect-Oriented Programming (AOP) is a new technology that was proposed to improve separation of concerns in software development. AOP's main focus is to untangle and gather the crosscutting concerns throughout the system and to implement them in individual aspect modules. Much research has been focused on AOP's application to traditional software development, but little has been done towards its application to Web development. Aspect-oriented PHP (aoPHP) is an addition to PHP that allows the use of AOP in the Web development context. In this paper we describe an application of AOP to Web development using aoPHP. In particular, we have implemented two crosscutting concerns in a collaborative Web system named CoTeia: the access control and the version control functionalities. Furthermore, we discuss how AOP can enhance the design of Web applications by analyzing the refactored system.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.3.2 [**Programming Languages**]: Language Constructs and Features

Keywords

aspect-oriented programming, Web engineering, PHP

1. INTRODUCTION

Aspect-Oriented Programming (AOP) is a new technology proposed to improve the practice of separation of concerns in software development. The main idea is that while object-oriented, procedural and other programming techniques by themselves help separating out the different concerns implemented in a software system, there are still some concerns that cannot be clearly mapped to isolated units of implementation. Examples of those concerns include mechanisms to persist objects in relational databases, access control, quality of services that require fine tuning of system properties, synchronization policies and logging. These are often called *crosscutting* concerns, because they tend to cut across multiple elements of the system instead of being localized within specific structural pieces [4]. AOP supports the implementation of such concerns in separate modules called aspects, that have the ability to define behavior on multiple other modules of the system. From such mechanism, it has been claimed that, among other features, AOP increases understandability and eases the maintenance burden, because modules tend to be more cohesive and less coupled.

Although a lot of research on applying AOP to traditional software can be found, little has been done towards applying it to Web development. However, evidences show that the use of AOP can also be effective in such context, since Web applications are similar to traditional software in many senses. For instance, supporting good separation of concerns is essencial for Web systems as well [1, 10, 18, 16, 17].

CoTeia is a collaborative educative Web system based on CoWeb [5]. CoTeia is implemented in PHP, a widely used open source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML [?]. This paper presents how two crosscutting concerns in CoTeia can be implemented using AOP through aoPHP – an extension of PHP to support AOP. The concerns chosen were the version and access control functionalities, due to their crosscutting nature. We explain how AOP helps to produce a more cohesive and less coupled set of modules. Moreover we also discuss how other concerns in the system could also be considered to be implemented using aoPHP.

The remainder of this paper is structured as follows. Section 2 presents the CoTeia Web system and how content and metadata is effectively implemented on it. Section 3 presents AOP basic concepts and how they are supported by the aoPHP language. Section 4 shows the implementation

[*]This work is partially supported by FAPESP, SP – Brazil.

of the version and access control concerns on CoTeia using aoPHP and Section 5 draws some conclusions and discusses future work.

2. THE COTEIA WEB SYSTEM

CoTeia is a Web system designed for collaborative content development. It is a wiki based system mainly used for academic purposes in a learning environment, although it could also be used in any environment where a collaborative content development tool is needed.

To provide an idea of CoTeia's size, Table 1 shows a summary of the amount of code in LOCs (lines of code). Each column refers to a directory of the system.

Directory	LOC
root	2229
admin	1051
doc	155
install	238
plugins	6309
Total	9982

Table 1: Summary of the size in LOC of the CoTeia Web system.

CoTeia's main functionality is the hyperdocument authoring. The implementation complexity of such feature can be seen by the root's amount of code. Additional functionalities are provided by extensions located at the plugins directory. These extensions have a great impact in CoTeia's complexity. An example of such extension is the metadata support [12], that augments CoTeia with a complex metadata subsystem in order to capture the learning objects' attributes from the swikis: it comprises 4104 LOC (comparing to the 2229 LOC of CoTeia's core).

In this section we present the CoTeia contents (swikis) and metadata subsystems, as well as how the version and access control mechanisms were implemented prior to using aoPHP.

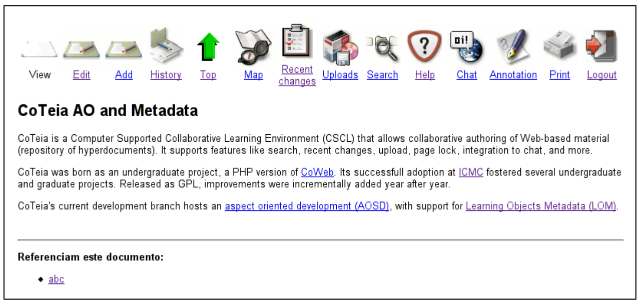
2.1 Content in CoTeia

CoTeia is a collaborative tool based on CoWeb (*Collaborative Website*) from Georgia Institute of Technology [5]. In the original CoWeb system, Squeak – an open source implementation of Smalltalk – was the programming language used, and swikis were used to organize data.

Swiki is a Web application that allows users to add content, as on an Internet forum, but also allows anyone to edit the content. The name swiki comes from *wiki* written in the *S*queak language, but the name was inherited by CoTeia, though it is implemented in PHP.

Figure 1 shows a screenshot of a swiki in CoTeia. The toolbar on the top of the page supports the following operations:

- **view:** used to show the content of the swiki.
- **edit:** provides access to the editing form (this form is shown in Figure 2).
- **add:** allows users to add text content to the swiki.
- **history:** shows the interface, as in Figure 3, allowing users to see older revisions of the content.
- **top:** goes to the top page (usually the swiki's parent page).
- **map:** shows the map of the CoTeia's swikis.
- **changes:** shows all the changes made in the CoTeia system using information stored in the CoTeia's database – only the time and date when changes occurred are shown, but not the changes themselves.
- **files:** provides access to file upload and download interface. Files can be uploaded and links to them can be created in the swikis.
- **search:** redirects user to the search page, which allows users to search swikis by title, contents or keyword.
- **help:** simple help.
- **chat:** provides access to a chat.
- **annotations:** an annotations system was implemented and integrated in the CoTeia tool, and this system can be accessed through this button.
- **print:** formats a swiki for a printer-friendly version.
- **logout:** logs out from CoTeia.



CoTeia AO and Metadata

CoTeia is a Computer Supported Collaborative Learning Environment (CSCL) that allows collaborative authoring of Web-based material (repository of hyperdocuments). It supports features like search, recent changes, upload, page lock, integration to chat, and more.

CoTeia was born as an undergraduate project, a PHP version of CoWeb. Its successfull adoption at ICMC fostered several undergraduate and graduate projects. Released as GPL, improvements were incrementally added year after year.

CoTeia's current development branch hosts an aspect oriented development (AOSD), with support for Learning Objects Metadata (LOM).

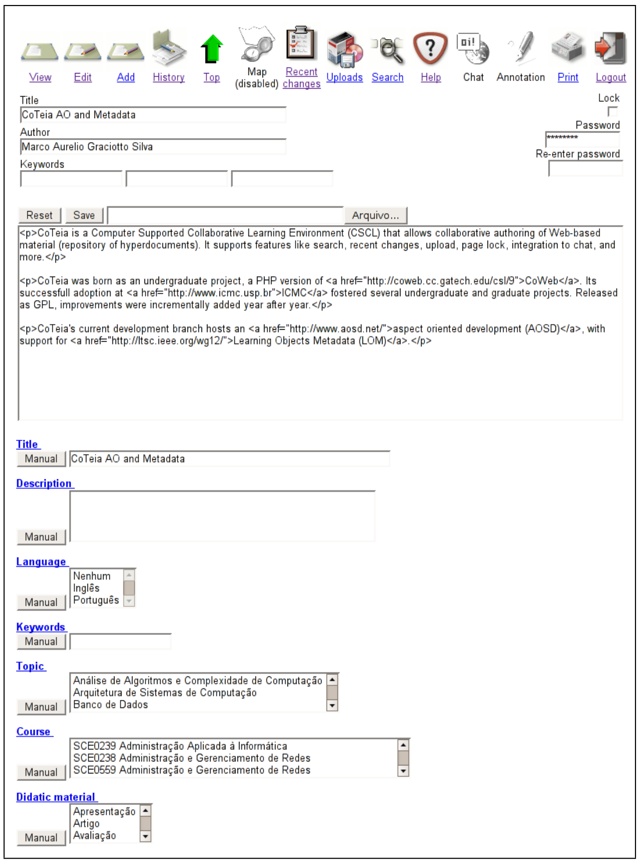
Referenciam este documento:

- abc

Figure 1: A CoTeia swiki.

The swikis' content is defined using a simple language based on well-formed HTML, *i.e.* XHTML. The following elements can be present in the content of a swiki:

- **Headings:** HTML tags <H1>, <H2> and <H3> can be used to create headings.
- **Text:** simple text without any tag.
- **Internal links:** links to other swikis. These elements must be inside <LNK> tags. Identification of the swiki is done by the name specified for this swiki (an unique ID). If the swiki does not exist yet, a link for creation is made available.
- **Paragraphs:** simple text paragraphs using <P> tags.
- **External links:** links to external pages using <A> tags.
- **Bold, italic and underline text:** bold text using <B> tags, italic using <I> tags and underline text using <U> tags.



Title: CoTeia AO and Metadata

Author: Marco Aurelio Graciotto Silva

Keywords

Lock

Password: *********

Re-enter password

```
<p>CoTeia is a Computer Supported Collaborative Learning Environment (CSCL) that allows collaborative authoring of Web-based
material (repository of hyperdocuments). It supports features like search, recent changes, upload, page lock, integration to chat, and
more.</p>

<p>CoTeia was born as an undergraduate project, a PHP version of <a href="http://coweb.cc.gatech.edu/csl/9">CoWeb</a>. Its
successfull adoption at <a href="http://www.icmc.usp.br">ICMC</a> fostered several undergraduate and graduate projects. Released
as GPL, improvements were incrementally added year after year.</p>

<p>CoTeia's current development branch hosts an <a href="http://www.aosd.net/">aspect oriented development (AOSD)</a>, with
support for <a href="http://ltsc.ieee.org/wg12/">Learning Objects Metadata (LOM)</a>.</p>
```

Title: CoTeia AO and Metadata

Description

Language: Nenhum / Inglês / Português

Keywords

Topic: Análise de Algoritmos e Complexidade de Computação / Arquitetura de Sistemas de Computação / Banco de Dados

Course: SCE0239 Administração Aplicada à Informática / SCE0238 Administração e Gerenciamento de Redes / SCE0559 Administração e Gerenciamento de Redes

Didatic material: Apresentação / Artigo / Avaliação

Figure 2: Web form for editing data and metadata in CoTeia.

- **Ordered and unordered lists:** simple lists of elements can be created with <OL> (ordered) and <UL> (unordered); items of both ordered and unordered lists must be inside <LI> tags.

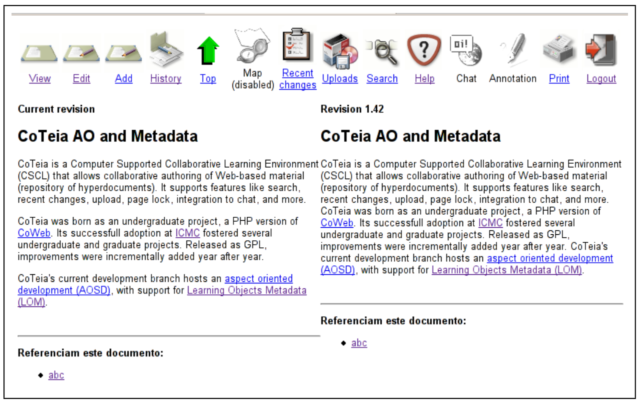
When a new swiki is created or an existing swiki has its contents changed, its content is saved to an XML file, validated and, if it is right, a new HTML file is generated (using XSLT) and stored. All these editing and creation operations should trigger the version control tasks to be performed.

2.2 Metadata in CoTeia

The CoTeia's swikis have metadata associated with them. Originally, CoTeia had only a few metadata, such as author's name, date of the last modification and keywords.

Recently, a work to improve CoTeia metadata mechanism was done, and the system can now capture, store and restore much more metadata, mainly related to the learning environment [12]. Such initiative aims to contribute for gathering more precise results during information retrieval and to provide more meaningful data on the edited contents. The model used to implement improved metadata was LOM, which stands for Learning Object Metadata. As defined by the Learning Technology Standards Comitee of IEEE[1], LOM is a set of attributes required to fully/adequately describe a Learning Object, and Learning Objects are defined in this case as any entity, digital or non-digital, which can

[1]`http://ltsc.ieee.org/wg12`



Current revision

CoTeia AO and Metadata

CoTeia is a Computer Supported Collaborative Learning Environment (CSCL) that allows collaborative authoring of Web-based material (repository of hyperdocuments). It supports features like search, recent changes, upload, page lock, integration to chat, and more.

CoTeia was born as an undergraduate project, a PHP version of CoWeb. Its successfull adoption at ICMC fostered several undergraduate and graduate projects. Released as GPL, improvements were incrementally added year after year.

CoTeia's current development branch hosts an aspect oriented development (AOSD), with support for Learning Objects Metadata (LOM).

Referenciam este documento:

- abc

Revision 1.42

CoTeia AO and Metadata

CoTeia is a Computer Supported Collaborative Learning Environment (CSCL) that allows collaborative authoring of Web-based material (repository of hyperdocuments). It supports features like search, recent changes, upload, page lock, integration to chat, and more. CoTeia was born as an undergraduate project, a PHP version of CoWeb. Its successfull adoption at ICMC fostered several undergraduate and graduate projects. Released as GPL, improvements were incrementally added year after year. CoTeia's current development branch hosts an aspect oriented development (AOSD), with support for Learning Objects Metadata (LOM).

Referenciam este documento:

- abc

Figure 3: History interface adapted to show contents and metadata history.

be used, re-used or referenced during technology supported learning.

LOM defines the Base schema that defines a hierarchy of data elements for learning objects metadata. At the top level of the hierarchy are nine categories, which are described using the clear definitions from the LOM specification itself [11]:

1. The **General** category groups the general information that describes the learning object as a whole.

2. The **Lifecycle** category groups the features related to the history and current state of this learning object and those who have affected this learning object during its evolution.

3. The **Meta-Metadata** category groups information about the metadata instance itself (rather than the learning object that the metadata instance describes).

4. The **Technical** category groups the technical requirements and technical characteristics of the learning object.

5. The **Educational** category groups the educational and pedagogic characteristics of the learning object.

6. The **Rights** category groups the intellectual property rights and conditions of use for the learning object.

7. The **Relation** category groups features that define the relationship between the learning object and other related learning objects.

8. The **Annotation** category provides comments on the educational use of the learning object and provides information on when and by whom the comments were created.

9. The **Classification** category describes this learning object in relation to a particular classification system.

CoTeia uses a web form to capture metadata as shown in Figure 2. When a user edits a page, this form is shown and both content and metadata can be edited. As described in the previous section, content is saved in XML, validated and then converted to HTML files. Metadata, on the other hand, is just saved in XML format.

The CoTeia metadata interface is able to collect the following data, that are organized in the nine categories presented before:

- **General category:** *identifier*, which is automatically captured as the URI of the page; *title*, the title of the page which can be edited by users; *keyword*, which can consist in one or more keywords related to the respective swiki; and *language*, the language of the swiki;
- **Lifecycle category:** *version*, which indicates current version of the swiki, and one *contribute* element for each time the swiki is edited – this element stores the role of the person who edited the swiki, in this case it is author in almost all the cases, and when this edition has occurred. These data are automatically captured.
- **Meta-metadata category:** *identifier*, *contribute* and *metadata schema*, all automatically captured data.
- **Technical category:** *format*, *size*, *location*, *requirement*, automatically captured data which technically describe the document.
- **Educational category:** this category stores the *intendedEndUserRole*, *difficulty* and *language.*
- **Rights category:** data on this category is about *costs*, *copyrightAndOtherRestrictions* and *description.*
- **Annotation category:** contains *entity*, *date* and *description.*
- **Classification category:** stores *keywords*, *description* and *taxonPath.*

In the original implementation of the metadata mechanism, every time any metadata changed, the old data was discarded and replaced by the new data. To provide metadata persistence over time, it was implemented a version control mechanism of content and metadata of CoTeia, described in details in the following section.

2.3 Version Control in CoTeia using the CVS System

The version control concern involves managing large amounts of information, assuring that changes on information files are registered and controlled. Version control tools control a repository of configuration items which content can not be changed. To work on a configuration item, it must be moved from the repository (*checked out*) to a working directory and, when the work is done, the item is inserted back (*checked in*) into the repository, creating a new version automatically [15].

The version control of CoTeia is done using CVS, an open source control version system that maintains a history of a source tree, in terms of a series of changes. CVS stamps each change with the time it was made and the name of the person who made it [3].

For the purposes of CoTeia, concurrency has not been addressed so far, since only one user at a time can check in the content/metadata of a swiki. Thus, version control using CVS in CoTeia is used only to manage versions of content/metadata for each swiki.

Some specific operations are used in CoTeia system to deal with version control. Events of CoTeia are handled and, when an particular event is captured, one of the following operations is executed in the CVS repository:

- **add:** this operation is run when a new swiki is added to the repository. Both swiki's content and metadata are added to the repository, in distinct places;
- **checkout:** performed when a user requests a stored revision (history) and after editing a swiki, before committing the changes;
- **commit:** commit changes to the repository. This operation is performed when a swiki's content or metadata changes; and
- **log:** log is used to generate revisions history list.

There are some basic PHP functions in CoTeia to implement the version control mechanism. The functions are:

- **cvs_add_file:** used to add a file to the repository. This function is called when a new file (metadata or contents file) is created;
- **cvs_update_file:** updates a file in the repository, generating a new revision of a file;
- **cvs_checkout_file:** gets the latest revision of a file; and
- **cvs_checkout_file_revision:** gets a specific revision of a file.

The content and metadata version control uses the same functions. Their data is stored in files with a naming pattern that allows the CoTeia system to identify them automatically.

2.4 Access Control in CoTeia

The access control functionality is concerned with restricting access to parts of the system to authorized users. It commonly includes authentication, authorization and audit. In this paper we focus on the authentication mechanism.

In CoTeia, the access control concern is implemented by two authentication mechanisms: one for contents and other for metadata. When the access control to a swiki is enabled, an user name and password are required to gain access to its content. With respect to the metadata, an authentication is also required before editing a swiki.

3. ASPECT-ORIENTED PROGRAMMING AND THE AOPHP LANGUAGE

Aspect-oriented programming (AOP) has been proposed as a mechanism to support the modular implementation of crosscutting concerns [8, 9]. The key mechanism in AOP is enabling the implementation of isolated modules – the aspects – which have the ability to collaborate with the implementation of several other modules in a *crosscutting* way. This mechanism which allows an aspect to interfere in several points of the system generally supports the development of programs whose structure more closely corresponds to their designs [19].

AOP extends traditional programming techniques with several concepts, including: join points (JPs), pointcut descriptors (PCDs), introductions or inter-type declarations

(ITDs), advice, and aspects. These concepts are implemented differently depending on the language, however, most of them follow the implementation of AspectJ [7] – the most prominent AO language.

JPs are, in general, defined as points in the execution of a program which are subject to *advising*. Pieces of advice extend or override the action before, after or around the JPs, this last having the possibility to *proceed* with the execution of the original join point. A PCD is a declarative expression that specify sets of JPs with a common property where advice should run. AspectJ, for instance, supports a rich set of PCDs: based on method executions and calls, field accesses and modifications, exception handler executions, etc. Because a PCD cuts new interfaces through the modules of a system, an advice can have effects that cut across the whole program structure [19, 9].

ITDs are declarations used to introduce members (methods, fields and interfaces) into other modules. Advice, PCDs, ITDs and also ordinary data members and methods are grouped into class-like modules called aspects. Most AO languages are based on object-oriented programming languages (*e.g.* AspectJ), however, AOP has also been implemented based on procedural languages (*e.g.* AspectC [2]).

aoPHP is an extension of PHP that supports AOP [18] in the Web development context. Currently, functions are the first class of JPs supported in aoPHP, so the PCDs are defined based on function calls. Before, after and around advice are supported and must be declared inside aspect files with the `.aophp` extension. These aspect files must be located at the same folder where the `.php` files are. The PHP files that are going to have their behavior affected by aspects must define which `.aophp` files are going to affect them. This is done by using the `<?aophp ?>` tag instead of the `<?php ?>` one. Figure 4 shows a simple example of a *log in* PHP file (`login.php`) that is affected by a *logging* aspect PHP file (`login_a.aoPHP`).

The `a_login.aophp` aspect module contains three pieces of advice. The first before advice acts on calls to `checkPW()` and logs the time and IP address of the user which is attempting to log in. This is done through the `exec(checkPW($p))` PCD which matches the calls to `checkPW($p)`. Note that the advice can access contextual information at the JP, in this case the password (through the `$p` parameter). The other pieces of advice are responsible for logging the succesful and unseccesful loging in of a user, respectively. For simplicity, the log info is printed in the browser but in reality it could be registered in a database or XML file, for instance.

```
<!-- login.php -->
<?aoPHP filename="login_a.aoPHP"

function invalidLogin($p){
  echo "<b>Can Not Login,
           $p Invalid Password</b><br>";
}

function redirectMember($p){
  echo "<a href=\"member.php\">
        Click to Enter</a><br>";
}

function checkPW($p){
    if($p != "test"){
        invalidLogin($p);
    } else {
        redirectMember($p);
    }
}


$f = $_REQUEST['f']; switch($f){
    case "check":
      $pw = $_POST['password'];
      checkPW($pw);
      break;
    default:
      echo "<form method=\"POST\"
         action=\"login.php?f=check\">
         <p>Enter Password to Login</p>
         <p><input type=\"text\"
         name=\"password\" size=\"20\">
         </p>
         <p><input type=\"submit\"
         value=\"login\"
         name=\"button1\"></p>
         </form>";
      break;
    }
?>

<!-- login_a.aophp -->
before(): exec(checkPW($p)){
  $ip = $_SERVER['REMOTE_ADDR'];
  //This is Where you Log to
  //File or Database
  echo "<i>$ip attempting login @ " .
    time() . "</i><br>";
}
after(): exec(redirectMember($p)){
    $ip = $_SERVER['REMOTE_ADDR'];
    //This is Where you Log to
    //File or Database
    echo "<i>$ip logged in @ " .
      time() . "</i><br>";
}
after(): exec(invalidLogin($p)){
    $ip = $_SERVER['REMOTE_ADDR'];
    //This is Where you Log to
    //File or Database
    echo "<i>$ip failed @ " .
      time() . "</i><br>";
}
```

Figure 4: A simple aoPHP example.

4. IMPLEMENTING TWO CROSSCUTTING CONCERNS IN COTEIA USING AOPHP

The version control concern implemented in CoTeia can be considered a crosscutting concern because it affects different JPs of the application and, if implemented using only functions, the base code would be tangled with the version control concern. For instance, every time a swiki is altered, the previous version of content and metadata related to it must be saved in the CVS repository. If the code for editing content/metadata of the swiki calls a function to update the CVS repository every time the content/metadata changes, it will contain code related to more than one concern, *i.e*, editing of content/metadata and also version control. This tangling further complicates the evolution of the system and also its comprehension.

Therefore it would be interesting if everything related to the version control concern could be implemented in a single separate module so that the places where it affects could be oblivious of it. Also, when an evolution task related to such concern takes place, the developer would know exactly where to go.

To implement the version control concern in CoTeia using aspects, we created an aspect file called `cvs.aophp`. This aspect contains two pieces of advice which define behavior in two distinct JPs of the system: one related to content and another related to metadata of the swikis. We identified these JPs in the `edit.php` file which is the module responsible for the edition of swikis (both content and metadata). These JPs are shown in Figure 5.

```
...

 writeMetadataToRDFFile($file, $metadata);
 // metadata editing join point

...

 $format = $DEFAULT_OUTPUT_FORMAT;
 $result = update_wikipage( $wikipage_id, $format );
     // content editing join point

 if ( $result !== true ) {
   show_error( _( "An error has been found " .
   "in this wikipage. Please, contact " .
   "the system administrator." ) );
 }

 session_write_close();
 header("Location: show.php?wikipage_id=$wikipage_id");

...
```

Figure 5: Join points identified in `edit.php`.

The first join point identified is a call to the `update_wikipage` function, responsible for generating the final `HTML` containing the swiki content. The second join point is a call to the `writeMetadataToRDFFile` function, which is responsible for writing all the metadata related to the swiki to an `RDF` file. These two JPs were identified as the places where the CVS system must act. It creates a new version of content and metadata if the files are already in the repository, or adds them to the repository.

As these control version actions must take place after the changes are made to the files, we created two pieces of advice to act on those JPs. The first one must be an around advice because we must first check whether the `update_wikipage` returned successfully prior to calling the CVS update or add functions. This is done using the `proceed` function which is used to proceed with the execution of the join point. In this case, since the join point is a call to `update_wikipage`, the proceed function will return true if the final `HTML` containing the swiki content was successfully generated, and false otherwise.

The second advice does not need to check any conditions so it can be an after advice. Figure 6 shows the two pieces of advice which are part of the `cvs.aophp` aspect.

After all the modules are put together to run, every time a swiki is altered the CVS saves a new version of both content and metadata or adds the file to the repository through the aspect. This implementation keeps the code much simpler because the `edit.php` module does not contain any code related to CVS, *i.e.*, all CVS add and update functions are called inside the CVS aspect.

A similar situation arises for the access control concern. Since the authentication has to take place every time a swiki is accessed or edited, the access control concern crosscuts the code related to such actions. We then created two aspects to implement this concern: one for the content and other for the metadata.

```
around() : execr(update_wikipage($wikipage_id, $format)) {
        global $CVS_WIKIPAGE_MODULE;

        $result = proceed($wikipage_id, $format);

        if ( $result === true ) {
          $filename =
            cvs_wikipage2workcopy( $wikipage_id );
          $result = cvs_update_file($CVS_WIKIPAGE_MODULE,
            $filename);
          // If we cannot update the file, try to add it.
          if ( $result === false ) {
            result = cvs_add_file( $CVS_WIKIPAGE_MODULE,
              $filename );
          }
        }
        return $result;
}


after() : execr(writeMetadataToRDFFile($file, $metadata)) {
        global $CVS_WIKIPAGE_MODULE,
          $PATH_COWEB, $XML_DIR, $wikipage_id;

        $filename = cvs_wikipage2workcopy( 'metadata' .
          $wikipage_id );
        $filename = $PATH_COWEB . '/' . $XML_DIR . '/' .
          basename($filename, '.html') . '.rdf';
        $result = cvs_update_file( $CVS_WIKIPAGE_MODULE,
          $filename );
        // If we cannot update the file, try to add it.
        if ( $result === false ) {
          $result = cvs_add_file( $CVS_WIKIPAGE_MODULE,
            $filename );
        }
}
```

Figure 6: `cvs.aophp` aspect file.

The contents' access control occurs in many places of the code. In general, it happens before loading any data from the database. Thus we created an aspect file called `swiki_authentication.aophp` to affect these points. This aspect contains an after advice which defines behavior in five JPs of the system: each call to the `db_connect` function. We identified these JPs in the `checkout.php`, `edit.php`, `list.php`, `repository.php` and `show.php` files, which are the modules responsible for: downloading files attached to swikis, editing swikis, showing swikis' indexes, managing files attached to swikis and rendering swikis. Figure 7 shows one of these JPs in the `show.php` module. The aspect is not shown here for space reasons.

```
      ...
// Find the swiki the wikipage belongs to
db_connect();
 // content access join point

// Create wikipage if it doesn't exist yet.
$query = "select id_pag from gets where id_sw='$swiki_id' and
id_pag='$wikipage_id'"; $result = mysql_query( $query ); if (
mysql_num_rows( $result ) == 0 ) {

...
```

Figure 7: Content's access control join point identified in `show.php`.

With respect to the metadata's access control, it occurs while editing a swiki using the `edit.php` module. The authentication must take place before calling the

```
...
$metadata = metadata_initialize($wikipage_id);
  // metadata access control join point
$feedback =
  metadata_feedback_initialize($wikipage_id);

// Apply user templates
$metadata =
  getMetadataFromTemplate($wikipage_id,
  $metadata, $feedback);

...
```

(a) The metadata's access control join point.

```
before(): exec(metadata_initialize($wikipage_id)) {
  session_start();
  if ( $_SESSION == NULL || $_SESSION['vcard'] == NULL
      || ! isset($_SESSION['vcard']) ) {
    $url = "login_user.php";
    $url .= "?dest=" . $_SERVER["REQUEST_URI"];
    session_write_close();
    header( "Location: $url" );
    exit();
  }
}
```

(b) The `user_authentication.aophp` aspect.

Figure 8: Code related to the metadata access control.

`metadata_initialize` function which is responsible for loading the swiki's metadata. We thus created an aspect file called `user_authentication.aophp` with a before advice that is responsible for the authentication. Figure 8 shows the JP in the `edit.php` module and the `user_authentication.aophp` aspect.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have shown a simple example of how AOP can help in the development of Web applications. The version and access control concerns implemented in the CoTeia system were factored out into aspects. This type of separation of concerns further facilitates the evolution of the system since different concerns are implemented in different modules. Moreover the comprehension of the application is also facilitated.

Although we have refactored the previous version of the system, the actual goal was to show that AOP can also be applied to Web development. There is a field of research that investigates the identification of aspects in active and legacy object-oriented code bases and the subsequent refactoring of such systems in aspect-oriented systems [14, 6, 13]. Thus, it could also be an interesting direction to investigate their application to Web systems.

To further assess the results of using AOP in the Web development, more case studies must be conducted. In the future we plan to implement other crosscutting concerns in CoTeia to help in such assessment.

6. REFERENCES

[1] S. Casteleyn, Z. Fiala, G.-J. Houben, and K. van der Sluijs. From adaptation engineering to aspect-oriented context-dependency. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 897–898, New York, NY, USA, 2006. ACM Press.

[2] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using aspectC to improve the modularity of path-specific customization in operating system code. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 88–98, New York, NY, USA, 2001. ACM Press.

[3] CVS Community. The Concurrent Versions System Home, 2005. available at: `https://www.cvshome.org/` accessed on: 07/29/2005.

[4] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, 2001.

[5] M. Guzdial, J. Rick, and C. Kehoe. Beyond adoption to invention: Teacher-created collaborative activities in higher education. *Journal of the Learning Sciences*, 10(3):265–279, 2001.

[6] J. Hannemann, G. C. Murphy, and G. Kiczales. Role-based refactoring of crosscutting concerns. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 135–146, New York, NY, USA, 2005. ACM Press.

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *ECOOP '01: Proceedings of the* 15th *European Conference on Object-Oriented Programming*, pages 327–353, 2001.

[8] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer-Verlag.

[9] G. Kiczales and M. Mezini. Aspect-oriented programming and modular reasoning. In *Proceedings of the* 27th *International Conference on Software Engineering (ICSE'2005)*, pages 49–58. ACM Press, 2005.

[10] S. Kojarski and D. H. Lorenz. Domain driven web development with webjinn. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 53–65, New York, NY, USA, 2003. ACM Press.

[11] U. Ogbuji. Thinking XML: Learning Objects Metadata. accessed on 07/29/2005. available at `http://www-128.ibm.com/developerworks/xml/library/x-think21.html`.

[12] L. T. E. Pansanato and R. P. M. Fortes. Strategies for automatic lom metadata generating in a web-based cscl tool. In *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, pages 1–8, New York, NY, USA, 2005. ACM Press.

[13] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proceedings of*

the 24th International Conference on Software Engineering, pages 406–416, New York, NY, USA, 2002. ACM Press.

[14] D. Shepherd, J. Palm, L. Pollock, and M. Chu-Carroll. Timna: a framework for automatically combining aspect mining analyses. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 184–193, New York, NY, USA, 2005. ACM Press.

[15] I. Sommerville. *Software Engineering.* Addison Wesley, 8 edition, 2003.

[16] J. W. Stamey and B. T. Saunders. Aspect-oriented documentation. In *SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication*, pages 1–2, New York, NY, USA, 2005. ACM Press.

[17] J. W. Stamey and B. T. Saunders. Implementing database solutions on the web with aspect-oriented programming. *Journal of Computing Sciences in Colleges*, 20(4):249–255, 2005.

[18] J. W. Stamey, B. T. Saunders, and M. Cameron. Introducing aoPHP. *International PHP Magazine*, June 2005.

[19] K. Sullivan, W. G. Griswold, Y. Song, Y. Cai, M. Shonle, N. Tewari, and H. Rajan. Information hiding interfaces for aspect-oriented design. In *ESEC/FSE-13: Proceedings of the* 10^{th} *European software engineering conference held jointly with* 13^{th} *ACM SIGSOFT international symposium on Foundations of software engineering*, pages 166–175, New York, NY, USA, 2005. ACM Press.