AOP Support for Data Mining

John W. Stamey Jr.[1], Simon Blanchard[2], Bryan T. Saunders[1], Otávio A. L. Lemos[3]

[1] Coastal Carolina University, P.O. Box 261954, Conway, SC, USA

[2]HEC Montreal, Montreal, Canada

[3] Instituto de Ciências Matemáticas e de Computação - USP
Av. Trabalhador São-carlense, 400 - Centro
Caixa Postal: 668 - CEP: 13560-970 - São Carlos - SP

{jwstamey,btsaunde}@coastal.edu, simon.blanchard@hec.ca, oall@icmc.usp.br

Abstract. *Aspect-Oriented Programming (AOP) is found to provide a convenient way to produce flat files that are used in data mining. Data from the web would traditionally be processed and stored in a data warehouse, ultimately being used to create a data mart to facilitate analysis. We examine a prototype system that has been developed to collect data from an Ecommerce application, and write out a flat file for analysis in real-time. The web application is written in PHP and Aspect-Oriented PHP (aoPHP). The Aspect-oriented real-time delivery of flat files for data mining is found to be quite cost effective for smaller, web-based applications.*

1. Introduction

Aspect-Oriented Programming (AOP) has been applied to a number of application areas. The use of AOP in middleware for banking applications is widely known in the AOP community [Colyer and Johnson 2005]. We have also seen how existing programming languages exhibit AOP-like behavior; in particular, the designers of COBOL [Limmel and De Schutter 2005] and HTML/XHTML with Cascading StyleSheets [Stamey et al. 2005] seemed to have notions such as advice and separation of concerns in mind when these two widely-used programming platforms were developed.

This paper examines AOP as a tool to help in the collection of data, generated from users of websites, for the purposes of data mining. Data mining of information obtained from the web is generally called *web mining*. Many data mining tools currently available require flat files, also known as CVS files, to facilitate the analysis of data. Unlike data stored in relational databases, the flat files feature redundant data and therefore need special processing for their creation.

2. Support for Data Mining

Data Warehouses are, by structure, very different from relational databases. For instance, if one system has a client information table (called dimensions table) and a transactions table (called a fact table), the client information needs to be added for every purchase [Jagadish et al. 1999]. Traditional integrity questions do not apply to data warehouses as the content is typically historical data [Chaudhuri 1997].

To solve specific problems, the data is extracted and merged into a single table. It then needs to be sampled or partitioned, cleaned and transformed before data mining tools can be used to discover potentially interesting information in the data. As far as the actual mining goes, *data mining* refers to algorithmic methods by which patterns or structures, not explicitly visible in the data, are extracted for evaluation. Classic families of data mining problems include predictive modeling, clustering, associations mining and data summarization [Bradley et al. 1999].

Gathering the data needed for data mining applications from an operational database can be very tedious [Kohavi 2001]. AOP, used in conjunction with traditional systems, can facilitate the creation of that data by logging transactions to flat files and automatically integrating master table information as needed. Especially for systems where the transactions volume will not make this log-process cumbersome for the web servers, such applications can greatly encourage as these files will be ready for data analysts to process with the other steps needed for data mining applications.

The traditional method for aggregating and summarizing data for mining is data warehousing. In order to solve problems related to specific business process areas, data marts are created [Chenowith et al. 2003]. Tools use for data mining frequently requires that flat files be created with information pulled together from multiple and different tables.

An example of this procedure would be the creation of a flat file (CSV file) to analyze customer purchasing patterns might include the product and quantity of each purchased along with demographic information about the purchaser such as their location (zip code) and the URL from which they were referred to the site. The process of combining information from multiple tables into one table (or, flat file) with potentially redundant data is similar to the creation of non-relational user views of data [Pernul 1987, Sanders and Shin 2001].

We present an approach avoids the potentially costly process of extracting information to create a flat file from information from the data warehouse. Using AOP, we create advice that creates a flat file, specifically for data mining, in real time as the user is accessing the website. The implementation, featuring aoPHP, bypasses the traditional intermediate step storing the data in the warehouse, and extracting the data into a flat file.

3. An illustrative example

Retailers are always looking for the best combinations of products in order to maximize cross-selling. While vendors in a bricks-and-sticks retail store traditionally rely on their experience when suggesting a purchase of socks with that new pair of shoes, Ecommerce systems typically rely on association mining algorithms. Initially proposed by Agrawal et al. (1993), association mining algorithms find rules in an e-commerce context to provide information about which products are frequently bought together. An example of such a rule would be: if a consumer buys a TV and a DVD player, there is an X% chance that the consumer will also buy blank DVDs. In the case of large and robust Ecommerce retailers, association mining algorithms are implemented in the shopping carts.

Effective cross-selling of products is not always based on a high-occurrence of mutual purchases. The most frequent combinations of products purchased at the same time might not be the most profitable, or might simply not be aligned with the company’s image [Brijs et al. 1999]. For these reasons, good association rules are developed for strategic reasons. In many instances, these rules are developed manually. Creating a flat file in real-time that analysts can use with association mining applications can facilitate the process of identifying candidate association rules.

We present a solution to creating flat files from form-based input that can be generalized to more complex systems of this type. Figure 1 shows a form written in PHP that can be used to collect information for a typical Ecommerce application. Three products can be ordered by changing the quantity in the SELECT boxes. Name and address information is collected from the customer. The goal of this small test system is to write out a flat file in real-time for the purposes of data mining at the time the order is processed.

```
<!-- order.php -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Data Mining Example</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
</head>
<body>
<form name="form1" method="post"
action="process.php">
<p>Name
    <input name="name" type="text" id="name">
    <br>
    Address
    <input name="addr" type="text" id="addr">
    <br>
    City
    <input name="city" type="text" id="city">
    <br>
    State
    <input name="state" type="text" id="state">
    <br>
    Zip
    <input name="zip" type="text" id="zip" size="8">
    <br>
  Email
  <input name="email" type="text" id="email">
  <br>
</p>
```

```
<table width="200" border="0">
  <tr>
    <td><strong>Product</strong></td>
    <td><strong>Price</strong></td>
    <td><strong>Qty</strong></td>
  </tr>
  <tr>
    <td>Doo Dad </td>
    <td>$5.00</td>
    <td>
      <input name="q1" type="text" id="q1" value="0"
size="6">    </td>
  </tr>
  <tr>
    <td>Wing Ding </td>
    <td>$6.00</td>
    <td><input name="q2" type="text" id="q2" value="0"
size="6"></td>
  </tr>
  <tr>
    <td>Widget</td>
    <td>$4.00</td>
    <td><input name="q3" type="text" id="q3" value="0"
size="6"></td>
  </tr>
</table>
<br>
<input type="hidden" name="items" value="3">
<input type="submit" name="b1" value="Order">
</form>
</body>
</html>
```

Figure 1. order.php.

The rows of the flat file would include the email address and zip code, normally stored in a Customer table, coupled with each product and quantity ordered that would normally be found in a Product table.

The Around-advice, found in Figure 2, written in aoPHP (Aspect-Oriented PHP) [Stamey et al. 2005, aoPHP Development Team 2005] is executed based on the return value of the PHP function `addProductData()` that adds product information to the database. As *field read and write joinpoints* (set and get) pointcuts are not implemented in aoPHP V2.0, the repetition of the function is the only way the advice can know if add to the Product table succeeded or failed. Once the field read and field write joinpoints become available in aoPHP V3.1 and above, this repeated function call will not be needed. If no error occurs in the call to `addProductData()` found in `proc_adc.aophp`, then a row of the flat file is written for that particular product that contains the zip code and the email address of the purchaser. Figure 3 shows the PHP code found on the Post page, resulting from the code in Figure 1. In Figure 3, customer information is added to the table, and all of the products are added to the product table. A random number generator was used to simulate success or failure of the database transaction.

```
<!-- proc_adc.aophp -->
around(): execr(addProductData($custID,$prodNo,$qty)){
    $z = addProductData($custID,$prodNo,$qty);
    if($z>0){
// Query the database for Customer
// information
// Product information is passed to the
// aspect
// Example values from the Customer table
// have been hard-coded for illustration
        $zip = "29526";
        $email = "aoPHP@coastal.edu";

        $t = time(); // Unix Timestamp
        $fd = fopen("data.txt","a") or die("Cant Open file");
        $fout =
fwrite($fd,"$zip,$email,$t,$prodNo,$qty\\r\\n");
        fclose($fd);
    }
    return $z;
}
```

Figure 2. proc_adc.aophp.

```
<!-- process.php -->
<?aophp filename="proc_adv.aophp" debuf="off"

function addClientData(){
    // Adds client information to database
    // For testing purposes, we return a random
number 0-5 to simulate Failure or Success of the
record being inserted
    // 0 Indicates Failure, 1 Indicates Success
    $x = rand(0,5);
    return $x;
}

function addProductData($custID,$prodNo,$qty){
    echo "Order: $prodNo x $qty by $custID <br>";
    // Adds product information to database
    // For testing purposes, we return a random
number 0-5 to simulate Failure or Success of the
record being inserted
    // 0 Indicates Failure, 1 Indicates Success
    $x = rand(0,1);
    return 1;
}
$clientID = addClientData();
if($clientID==0){
    echo "Client Add Failed, Error With Billing
    Information<br>";
    exit;
}else{
    // Client Add was OK
    $cnt = $_POST['items'];
    for($i=1;$i<=$cnt;$i++){
        $t = "q" . $i;
        $q = $_POST[$t];
        if($q!=0){
  // Below is the joinpoint which triggers
            // the Around-advice
            $r = addProductData($clientID,$i,$q);
        }
    }
}
?>
```

Figure 3. process.php.

Creation of the flat file is a system requirement that goes above and beyond the core concerns of collecting and processing the information for each Ecommerce transaction. It is one of (potentially) a number of crosscutting concerns that produce flat files for analysis through data mining.

One other particularly useful flat file is one that provides a trace of pages visited by the consumer. We would also include the referring URL in the page trace. The referring URL is important for a company to identify the amount of traffic they get from paid search engine placement and advertisements (such as Google AdWords).

Ideally, the page-trace flat file can be coupled with information about the completed Ecommerce transaction to help data miners answer determine such things as the characteristics of a website visit that typically lead to a purchase.

The code to manage writing of two different flat files, such as those described above, would involve code tangling (constructing parts of the files in different locations of the website code) and code scattering (duplicate code found in different locations of the website code). For this reason, the early use of aspects in the design of a system (such as when web mining is a consideration) is necessary to avoid code scattering and code tangling.

4. Conclusions

We have presented a web-based solution to creating flat files for data mining using AOP. It is straightforward for this type of solution to be expanded to capture and produce flat files for the purpose of analyzing URL referrer and page traces in conjunction with customer information and purchasing habits. This solution is well suited for smaller Ecommerce applications where the resources of a full data warehouse have not been provided, or are not practical. Extrapolating, we can see that a similar solution can be applied to the harvesting of any clickstream data for systems that either do or do not feed into a data warehouse. PHP used in combination with aoPHP is found to provide an extensible and reliable solution to creating flat files for web mining. We are currently expanding our test system to collect and analyze clickstream data (including Ecommerce transactions) for use in a high-traffic website. Collection of this data from real-world applications will provide more insight to developing associations for small but robust Ecommerce applications.

5. References

Colyer, A. & Johnson, R. (2005) AOP in Spring. Invited Industry Talk (Abstract available in *AOSD.05 Conference Programme*). March 17, Toledo Room, InterContinental Hotel, Chicago, IL.

Limmel, R. & De Schutter, K. (2005) What does Aspect-Oriented Programming mean in COBOL? *Proceedings of AOSD 2005,* pp. 99-110.

Stamey, J.W., Saunders, B.T., & Blachard, S. (2005) The Aspect-Oriented Web. *Proceedings of ACM SIGDOC 2005 (to appear).*

Jagadish H.V., Lakshmanan, L.V.S. & Srivastava, D. (1999) What can hierarchies do for data warehouses? *Proceedings of the 25th international conference on Very Large Data Bases (VLDB),* 530-541.

Chaudhuri S., U. Dayal. (1997) An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record,* 26 (1), 65-74, March.

Bradley, P. S., Fayyad, U. M. & Mangasarian, O. L. (1999) Mathematical programming for data mining: formulations and challenges. *INFORMS Journal on Computing*, 11(3):217-238.

Kohavi, R. (2001) Mining e-commerce data: the good, the bad, and the ugly. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining,* pp. 8-13.

Chenowith, T., Schuff, D. & St. Louis, R. (2003) A method for developing dimensional data marts. *Communications of the ACM*, Volume 46, Issue 12, December. Pages: 93 – 98.

Pernul, G., (1987) An Unnormalized Relational Data Model Based on User Views. *ACM SIGMOD Record*, Vol. 16, No,. 2, September.

Sanders, G.L., Shin, S. (2001) Denormalization Effects on Performance of RDBMS. *Proceedings of the 34th IEEE Annual Hawaii International Conference on Systems Sciences (ICSSS-34).*

Agrawal R., Imielinski T., Swami A. (1993) Mining Associations between Sets of Items in Massive Databases, *Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data*, Washington D.C., May, 207-216.

Brijs T., G. Swinnen, K. Vanhoof, G. Wets (1999) Using association rules for product assortment decisions: a case study. *Proceedings of the fifth ACM SIGKDD*, 254-260.

Stamey, J., Saunders, B.T., & Cameron, M. (2005) Introducing aoPHP. International PHP Magazine, June.

aoPHP Development Team (2005) aoPHP.net, Available at: http:/www.aoPHP.net/, accessed on 7/2/2005.