IMPLEMENTATION OF MODEL VIEW CONTROLLER (MVC) ARCHITECTURE ON BUILDING WEB-BASED INFORMATION SYSTEM

Shofwatul 'Uyun[1], Muhammad Rifqi Ma'arif[2]
[1,2]Universitas Islam Negeri Sunan Kalijaga
Jl. Marsda Adisucipto No. 1 Yogyakarta 55281
Telp. (0274) 519739, Fax (0274) 540971
E-mail: shofwa_uyun@yahoo.com, muhammad.rifqi@gmail.com

ABSTRAKS
The purpose of this paper is to introduce the use of MVC architecture in web-based information systems development. MVC (Model-View-Controller) architecture is a way to decompose the application into three parts: model, view and controller. Originally applied to the graphical user interaction model of input, processing and output. Expected to use the MVC architecture, applications can be built maintenance of more modular, rusable, and easy and migrate. We have developed a management system of school fees at the high school. The developed system uses the MVC (Model-View-Controller Architecture) to provide a description of how the MVC works and its benefits.

Keywords: web-based information system, MVC, modularity, reusability.

1. INTRODUCTION

In order to be usefull, software, of course has to interact with something. Sometimes it interacts with other machine, and more often with people. And to make the interactions more efficient, it needs some interfaces. Several web-based applications need more resources to build an effective and modular interface. The needed resources above is better than the process logic of the application's resources.

Conventional web-based system, still mixing the codes between process logic and interfaces. The interface in the conventional web-based system can be used in only one process logic. This will reduce the modularity of applications and make the system more difficult to maintenance. It is make the interfaces hard to modified in order to use to the other applications too (Deacon, 2009). In the technical code, the server script (PHP, JSP, ASP, etc.) are still mixed with the presentation (HTML, WML, JavaScript, etc.).

The effects above have rise the idea to separate application logic with the interface. So the built application can easily replace their interface any time. In 70's. Inventor of The smaltak's, Trygve Reenskaug, defined an architecture to resolve this problems, it called the model-view-controller architecture. Since that time the MVC design has become commonplace. The main idea of the architecture is to separate business logic and presentation applications

2. MVC (MODEL-VIEW-CONTROLLER) ARCHITECTURE

Model-View-Controller is a concept introduced by the inventor of Smalltalk's (Trygve Reenskaug) to encapsulate data along with its processing (model) and isolate it from the process of manipulation (controller) and presentation (look) to be represented on the User Interface (Deacon, 2009). MVC follows the most common approach of Layering. Layering is nothing but a logical split up of our code in to functions in different classes. This approach is well known and most accepted approach. The main advantage in this approach is re-usability of code (Satish, 2004).

The technical definition of MVC architecture are divided into three layer (Burbeck, 1992)

a. The model is used to manage information and notify observers when there was a change of information. Only models containing data and functions associated with processing the data. A model summarizing more than just data and functions that operate on it. A model approach is used to model computer or abstraction of some real-world process. This not only captures the state of process or system, but how the system works. For example, the programmer can define a model that bridges the back-end computing with front-end GUI. In this scenario, the wraps and abstract model of computation engine functions and act as a liaison who request services from the system model.
b. The view is responsible for mapping graphics onto a device. A view typically has a one to one correspondence with a display surface and knows how to render to it. A view attaches to a model and renders its contents to the display surface. In addition, when the model changes, the view automatically redraws the affected part of the display to reflect those changes. There can be multiple view onto the same model

and each of these view can render the contents of the model to a different display surface.

A controller accepts input from the user and instructs the model and view to perform actions based on that input. So that, the controller is responsible for mapping end-user action to application's response. For example, when the user clicks the button or chooses a menu item, the controller is responsible to determine how the application should response.

The model, view and controller are intimately related and in constant contact. Therefore, they must reference each other. The picture below illustrates the basic Model-View-Controller relationship:

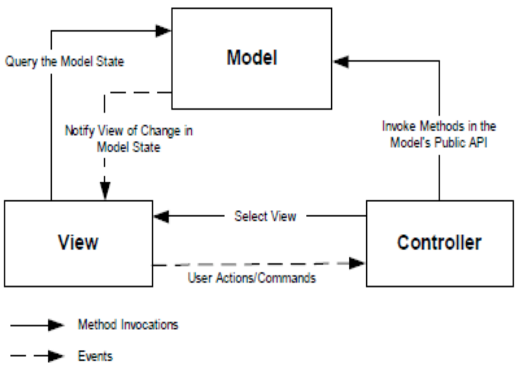

Figure 1. Relation between model, view, and controller (Gulzar, 2002)

The MVC architecture has the following benefits (Balani, 2002). The separation between model and view allows multiple view to use the same model. Consequently,an application's model components are easier to implement, test, and maintain, since all access to the model goes through these components.

3. SYSTEM DESIGN

To give an idea of how MVC works and its benefits, we have developed a web-based information systems to manage the payment of school fees at a vocational high school. This system is designed using UML (Unified Modeling Leanguage). We have described the data model into class diagram. Class diagram of the developed system is illustrated in Figure 2.

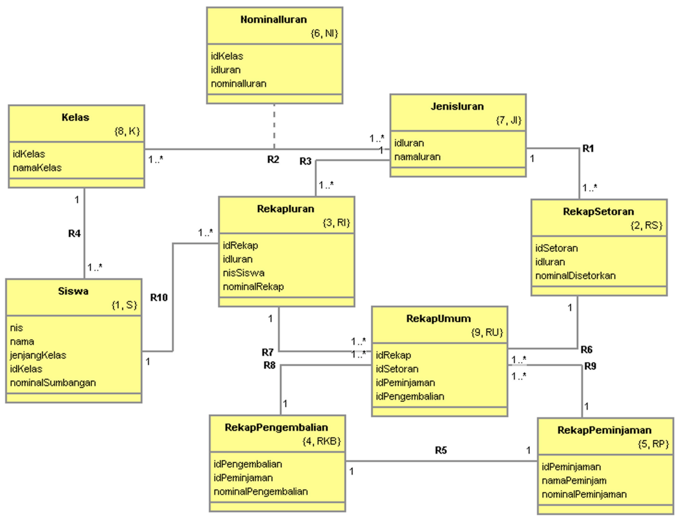

Figure 2. The data model

Class diagram is a diagram that describes a set of classes, interfaces, collaborations, and the relationships between them. Class diagram is a special type of diagrams and shares the same general nature as all the diagrams, the names and graphical content which is the projection to the model (Booch et al, 1998). From the class diagram above, we can describe the relationship between the objects that composed the system. From these relations, we can get the main structure of the database that will be used. In MVC architecture, database access is handled by the model.

User requests and feedback are illustrated by figure 3. In this figure, the arrow indicates the user's request and the ellipse shows the system response to the request. Users make a request through the graphical interfaces (buttons, links, etc.). The response is mapped to a graphical interface. User requests and responses are handled by the controller. The view are responsible for providing a graphical interface.

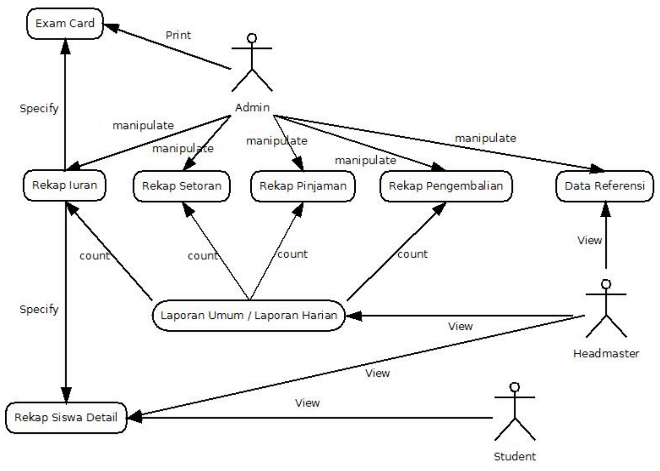

Figure 3. The user requests and its response

4. MVC ARCHITECTURE IMPLEMENTATION

In the developed system, the MVC architecture is implemented in every application module. Each module has one model, one controller (divided into three types of files), and some view. The model consists of (at least) one file which is responsible for accessing the data into the database. This file contains one or more SQL statements. Below is a sample source model:

```
$sql['get_detail_transaksi']="
    SELECT
       nisSiswaRekap AS NIS,
       namaSiswa AS NAMA,
       tanggal AS TANGGAL,
       idIuranRekap AS ID_TRANS,
       nominalRekap AS NOMINAL
     FROM
       sis_rekap_iuran
      JOIN
      sis_siswa ON
      nisSiswa=nisSiswaRekap
      WHERE
       idRekapSiswa = '$idTr'"
";

$sql['update_transaksi']="
    UPDATE
       sis_rekap_iuran
      SET
       nisSiswaRekap='%1',
       nominalRekap='%2'
      WHERE
       idRekapSiswa='%3'
```

Figure 4. The model's code

SQL query above is used for accessing data from the database, then the query is called in the controller. To connect between the user interface and business logic in the data model it needs a controller. A controller consists of three types of files. The first file for handling user responses, the seccond file to handle feedback and the other file to connect with the model.

```
$nis=$_POST['nis'];
$idTr=$_POST['idTr'];
$nominal=$_POST['nominal'];
$idR=$_POST['idRekap'];

$transaksi=new Transaksi;
$action=$transaksi->update_transaksi
($nis,$nominal,$idR);

if($action==true){
    echo"<b>Insert Data Succes</b>"; }
else{
    echo"Insert Data Failed";
```

Figure 5. The first controller's code

The source code above illustrates a controller that is responsible for handling user requests through a graphical user interface (buttons, links, etc.). Then the file will create a new object from a class described in another controller that has the responsibility to acces the model, source code below is an example of this controller

```
class Transaksi extends mysql_db{
var $queries;

function Transaksi(){
require_once
      ("rekapTransaksi.sql.php");
            $this->queries = $sql;
      }

function get_detail_transaksi(){
    $this->execute($this-
    >queries['get_customer'],
    array());

      $row=0;
    while($result=$this->get_array()){
      $number = $row+1;
      $data[$row]=array(
      'NO'=>$number,
      'NIS'=>$result['NIS'],
      'NAMA'=>$result['NAMA'],
      'ID_TRAN'=>$result['ID_TRAN'],
      'NOMINAL'=>$result['NOMINAL']);
      $row++;
    }
    return $data;
}

function update_transaksi($nis, $nom,
$idTr){
      $query=$this->execute($this-
>queries['update_transaksi'],
      array($nis, $nom, $idTr));
      if($query){
            return true;
      }else{
            return false;
      }
    }
}
```

Figure 6. The seccond controller's code

After the controller receives feedback from the model, then the controller will select the appropriate view to display information. The source code to perform this task are:

```
$tmpl-
>setBasedir("module/transaksi/template
s");
$tmpl-
>readTemplatesFromFile("transaksi.html
");
$tmpl->addVar("article",    "HEADLINE",
"Form Transaksi");
$tmpl->addVar("article",     "URL_XLS",
"module/transaksi/process_rekap_xls.ph
p");
$transaksi = new Transaksi;
if(!empty($code['idUniv'])){
      $data        =        $transaksi-
>get_buyer($code['idUniv']);
      foreach ($data as $display){
      $tmpl->addVars("data",
$display);
      $tmpl-
>parseTemplate("data","a");
}
```

Figure 7 . The Controller's Code

A View is responsible for displaying the information into a graphical interface. One view must have one controller that handles requests from users and feedback. The source below is an example of the view :

```
<!--  patTemplate:tmpl  name="article"
-->
<h3>{HEADLINE}</h3>
<div class="toolbar">
<a href="{URL_XLS}">Excel</a>
<table class="table-common">
    <tr>
      <th rowspan=2>No.</th>
      <th rowspan=2>Tanggal</th>
      <th colspan=2>Pemasukan</th>
      <th colspan=2>Pengeluaran</th>
      <th rowspan=2>Saldo</th>
    </tr>
    <tr>
      <th>IURAN</th>
      <th>PENGEMBALIAN</th>
      <th>SETORAN</th>
      <th>PEMINJAMAN</th>
    </tr>
    <!--  patTemplate:tmpl  name="data"
-->
    <tr class={CLASS_NAME}>
      <td>{NO}</td>
      <td>{TANGGAL}</td>
      <td>{TOTAL_IURAN}</td>
      <td>{TOTAL_PENGEMBALIAN}</td>
      <td>{TOTAL_SETORAN}</td>
      <td>{TOTAL_PEMINJAMAN}</td>
      <td>{SALDO}</td>
    </tr>
    <!-- /patTemplate:tmpl -->
</table>
<!-- /patTemplate:tmpl -->
```

Figure 8. The view's code

5. CONCLUSION

From the developed system, we can conclude that using the MVC architecture to develop web-based information system can improve modularity and reusability of the application. It is possible because the source code become neater and the separation between business logic and user interface are more explicit. Finally, by using this architecture, the complexity of the code in the software can be significantly reduced. Thereby, increasing the flexibility and modularity of software systems.

6. REFFERENCES

Balani, Naveen. (2002). *Web services architecture using MVC style.* Retrieved on March 30, 2010 from Http://www.webifysolutions.com?subject=Web services architecture using MVC style

Booch, Graddy cs. (1998). *The UML User Guide*. USA : Addison Wesley.

Burbeck, Steven. (1992). *Application Programmings in Smaltalk's 80 ™ : How To Use MVC*. Retrieved on March 14, 2010 from http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Deacon, John. (2009). *Model-View-Controller Architecture*. Retrieved on March 13, 2010 from http://www.jdl.co.uk/briefings/index.html/#mvc .

Gulzar, Nadir. (2002). *Fast track to struts : what it does and how.* Retrieved on March 30, 2010 from http://media.techtarget.com/tss/static/articles/content/StrutsFastTrack/StrutsFastTrack.pdf

Satish. (2004). *Model View Controller (MVC) Architecture*. Retrieved on March 24, 2010 from http://www.dotnetspider.com/resources/316-Model-View-Controller-MVC-architecture.aspx