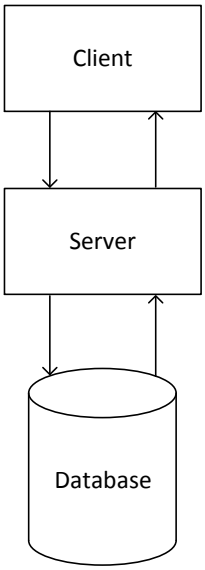
Web Engineering: Miniproject

Jesper Riemer Andersen
Nicklas Andersen
Simon Reedtz Olesen

November 20, 2013

1 Block 1: Introduction

We have simple 3-tier architecture which allows us to communicate with the server and extract data from our database. This protects the database from unwanted queries from the client.



1. The client opens his web browser and sends a HTTP request to the server.

2. The server runs the PHP script which queries data from the database.

3. The database returns the result according to the query.

4. Then the server returns the HTTP response to the client.

5. The response is then output as HTML.

```
1  <?php
2  $connection = mysql_connect("localhost", "root", "");
3  mysql_select_db("webengi", $connection);
4
5  $query = mysql_query("SELECT testtext FROM hello", $connection);
6
7  while ($row = mysql_fetch_assoc($query)) {
8      echo $row['testtext']."</br>";
9  }
10 ?>
```

This code shows the steps explained above. We chose to run the server on localhost using WampServer, which also comes bundled with a MySQL database.

We start by connecting to the localhost with the username *root* and empty password. Then we select the database called *webengi*, and then query the database. For each row that is returned by the query, we echo the content of the column called *testtext*.

2 Block 2: Data I: XML etc.

We found an XML document containing a small set of computers for sale on eBay, which we will display a subset of. We will be using the following:

- An XML file containing data on computers for sale.
- An XSL file with our XSL transformations (XSLT).
- A HTML file which contains only a little javascript to load the other files and display the result.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL
       /Transform">
3
4  <xsl:template match="/">
5    <h2>Computers for sale on eBay</h2>
```

```
6     <table border="1">
7       <tr bgcolor="#9acd32">
8         <th>Current bid</th>
9         <th>Memory</th>
10        <th>Hard drive</th>
11        <th>CPU</th>
12      </tr>
13      <xsl:for-each select="root/listing">
14      <tr>
15        <td><xsl:value-of select="auction_info/current_bid"/></td>
16        <td><xsl:value-of select="item_info/memory"/></td>
17        <td><xsl:value-of select="item_info/hard_drive"/></td>
18        <td><xsl:value-of select="item_info/cpu"/></td>
19      </tr>
20      </xsl:for-each>
21    </table>
22  </xsl:template>
23  </xsl:stylesheet>
```

This is the XSL file with our transformation. The following box is copied from w3schools.com [**?**].

> Since an XSL style sheet is an XML document, it always begins with the XML declaration: `<?xml version="1.0"encoding="ISO-8859-1"?>`.
>
> The next element, `<xsl:stylesheet>`, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).
>
> The `<xsl:template>` element defines a template. The `match="/"` attribute associates the template with the root of the XML source document.
>
> The content inside the `<xsl:template>` element defines some HTML to write to the output.

2.1 XML vs XSLT

XQuery and XSLT are two languages used for quering XML. We have chosen to experience a bit with both to find out how they worked.

We spent a lot of time trying to make XQuery work with PHP Zorba. Zorba is a sort of virtual machine for query processing, however we could not make it work with the newest version of PHP for some reason, so we ended up using an Eclipse plugin called "Sausalito Tools" for the Xquery,

allowing us to read an XML document and extract information but we did not connect it to our database.

It allowed us to get a general understanding of XQuery and XPath and how to extract information from an XML file using these queries.

XSLT is primarily conceived as a stylesheet language used to render XML on screen where as XQuery is conceived as a database query language and in our case our main focus is to query an XML file in our database and not styling our website.

XSLT is stronger when it comes to making small changes to a document, because it can make use of a coding pattern that involves a template that copies all nodes unchanged, modified by specific templates that modify selected nodes.

XQuery does not support dynamic binding or polymorphism, but this ability is more noticeable when writing large applications, which we do not expect to be doing.

2.2 Block 4: Architecture

For the architecture we used a Model-View-Controller.

This allows us to divide our application into three kinds of components;

- A model is a representation of some data in a domain. It is an object containing all data and behaviour, other than that used for the UI.
- The view represents the display of the model in the UI. The view is only about displaying the information, any changes that might be made to the data is handled by the controller.
- The controller takes user input and manipulates the model and the view is updated appropriately.

Model

View

We chose to use the Template View because it is easy to implement and for our assignment we only needed to display a static HTML page because our data was also static.

If we had dynamic data, it might have been more appealing to use a Transform View where we would have used the data as input and "transformed" it into HTML, thus giving us a more dynamic HTML view for different sorts of data.

If we had had more than one web page, a two step view could have been appealing to use. It allows you to have a consistent look and organization across your website.

Due to the simplicity of our dataset, a template view was more than sufficient with its easy implementation it was very appealing for us.

Controller

Advantages and disadvantages of MVC

Advantages

- It is possible to substitute and re-use views and controllers for the same model.
- Many views for the same model
- Clear separation between presentation logic and business logic
- Easy to maintain code for future improvements
- All views are synchronized and reflect the current state of the model
- It is easy to test the core of the application

Disadvantages

- Increased complexity because an application can use other patterns at the same time as MVC (for example in an active model using an Observer pattern)
- Changes to the model require changes in the view and might require changes to the controller as well.
- It is very difficult to have a strict separation between view and controller.