2008

Utilizing AJAX Calls to Separate the Presentation & Application Layers in Web Applications

Kevin Holleran
Grand Valley State University

Utilizing AJAX Calls to Separate the Presentation & Application Layers in Web Applications

Health Tracker: An Example

Kevin Holleran
10/9/2008

Over the past ten years, the Internet and the World Wide Web has brought a change to the way the world, consumer and business alike, communicates. These changes have been sweeping over all walks of life, changing the way we do business, the way we find information, the way we learn, and the way we live. More information, businesses, and groups become involved in the Internet every day as technology changes as rapidly as businesses can adapt. Traditional development methodologies still contain merit and applications developed in traditional languages such as C++, C, or COBOL are not going to go away any time soon. Legacy code such as COBOL as well as needs for high end graphics and large, processor intensive applications will continue to demand these types of technologies. However, with the continued growth of the Internet, so to has the need for distributed applications grown tremendously. From social networking sites to the utilization of web services to peer to peer file sharing and instant message applications, the need for distributed applications and *flexible* applications has come to a forefront. The three tier model has been relatively easy in traditional languages but the mixing of markup languages and server side scripting in Web programming has mixed the Presentation and Application layers, thus preventing the modularity that allows web applications to be flexible and adapt to changes in technology. How can Web Developers adhere to the three tier architecture for web applications to fully utilize the benefits of the separation? Utilizing AJAX calls from the Presentation Layer to the Application Layer will separate these layers and solve this issue.

The Three Tiered Architecture

The three tier model emerged in the 1990's. This model separates the Presentation Layer from the Application Layer from the Data Layer. By separating these different modules out, the system becomes flexible and changes can be made to one layer without affecting how another layer is structured and how it operates. The model provides greater flexibility, maintainability, scalability, and re-usability than a two tiered approach. The three tier model is made of three modules. The Presentation Layer deals with all interaction with the user. The user interface presentation, formatting of the data, and handling of user events (i.e. mouse clicks, key presses, etc) all occur here. The Presentation Layer than communicates requests to the Application Layer. The Application Layer (sometimes referred to as the Logic Layer or Business Logic Layer) handles all of the application logic involved in the system. This is the "brains" of the system. This layer communicates with the Data Storage Layer in some fashion. The Data Storage Layer stores the data. This often is a database of some sort but certainly does not have to be.

The separation in these layers means that if the business logic changes (say, a per hour rate is changed), the data in the Data Storage Layer does not have to change, and the front end user interface does not have to change; only the programming logic that computes a total needs to be changed. This enhances re-usability and maintainability of code. Furthermore, let's look at the following piece of code (assume the database connection is established). This piece of code queries the database and echoes the results to the screen (written in PHP5 and HTML):

```
$sqlString = "SELECT Name, Email, DateJoined FROM Users WHERE
DateJoined > '2008-7-30' ORDER BY DateJoined DESC";
$resultSet = mysql_query_db($sqlString) or die("SQL Error");
$numResults = mysql_num_rows($resultSet);
echo "<table border = 1 width =
'100%'><tr><th>Name</th><th>Email</th><th>DateJoined</th></tr
>";
for ($i=0; $i<$numResults; $i++) {
    $row = mysql_fetch_array($resultSet, MYSQL_ASSOC);
    echo "<tr><td>" . $row['Name'] . "</td><td>" . $row['Email'] .
"</td><td>" . $row['DateJoined'] . "</td></tr>";
}
echo "</table>
```

Figure 1: Example of Mixed Application Layer and Presentation Layer Code

This piece of code does a very common job. It simply queries the database, grabs the results, and displays the results to the screen in a table. However, the presentation (HTML building the table and displaying the data) is embedded within the logic of the application (the PHP code querying the database). This does not seem like a big deal but what happens if the rules of the business logic change and a new column needs to be added to the results? We have to change this logic and add the column to the SQL query. However, we ALSO have to change the presentation layer code to reflect this change. Now, we are changing not only the application logic but the presentation code as well, and the change had only to do with the logic. Though it may not seem like much to add to the echo line to make it output another column, imagine if this took place over an application with thousands of lines of code. We may have to change hundreds of lines of presentation code when the only change we made was to the logic of the application! However, if we broke up the layers and the application layer simply returned the data, the presentation layer could iterate through that data and display it to the screen, thus the only change taking place at the Application Layer and no changes to the Presentation would be required. For example, if the data was returned formatted in XML, the presentation layer (in this case, a web browser) could use JavaScript to step through all elements in the XML tree, displaying the data appropriately. It would not matter if there was a three rows to display or a million, the presentation code would be the same.

Now imagine that same piece of code referenced above. Everyone in technology has seen the rapid growth of new technologies. These pop up faster than development can keep up. Will the code work on the "next big thing," whether it be a SmartPhone, PDA, or iPod? Will it work on something we do not even have a thought about yet, like something ubiquitous? The above logic code that queries the database would have to be rewritten along with new presentation code. Imagine a system that tracks exercise (perhaps similar to our example application later). Perhaps our watch will someday be able to communicate via bluetooth and update the system with the amount of time we perform an exercise. Can code similar to the above interface with something like that? Probably not. However, the *application logic* for inserting that time into the user database would not change. Therefore, we could reuse the application layer logic for the insert and only have to add a presentation layer for the new watch interface.

Utilizing AJAX to Achieve a Three Tiered Architecture

How can we separate these layers effectively? In an object oriented technology like Java, use of objects allows us to separate our user interface objects from the objects that we act upon and process. To achieve this in Web Development is much more difficult. Over time or without careful planning, code like the above can be created, creating a maintenance nightmare and greatly reducing re-usability and perhaps grinding scalability down to nothing. Web applications are extremely popular as the web shrinks the globe and faster connections become cheaper for the masses. Furthermore, with the wave of Web 2.0 applications and pushes for ideas such as Software as a Service (SaaS) web applications have come to the forefront. How do we prevent intermixed application and presentation code? As technologies grow and mature and new technologies open more doors of communication, how do we make sure our product is scalable to be able to quickly come to market with new technologies that we

do not even know exist yet? Utilizing AJAX calls answers these questions and ensures that our product will be scalable and flexible.

Asynchronous JavaScript and XML (AJAX) is a technology that has been around for a while but recently has gained in popularity due to Google's utilization of it and the fact that XML is being utilized increasingly often. Though very powerful, it is also relatively simple. AJAX allows asynchronous calls to the server without page reloads. This technique can allow us to make requests to the application layer while keeping it separated from our presentation layer code. These calls allow the JavaScript to communicate with the server, receive data back, and then act upon that data when it returns. This data does not *have* to be in the form of XML; the script could return formatted HTML and display this, but that would not result in our goal of separation of the Application and Presentation Layers. However, if only the data is returned (in XML) from the call, the Presentation Layer can handle formatting and displaying that data. Changing the font style or moving the table will not result in the need for a change to the server side script or the AJAX call and a change to how the server side script computes a value will require no change to the code in the presentation layer. Utilizing AJAX calls accomplishes our goal of the separation but still requires discipline to return only the data.

HealthTracker: An Example

The HealthTracker system puts these principles into action to demonstrate the usability of this ideal. This system is a tracker for users to input their food and drink intake as well as their exercise. The system then tracks across many categories, including calories, fat, vitamins, etc. It also tracks the calories burned by the exercise. The system allows intake of standard items or user created custom items. The user can keep track of his/her favorite items and set goals for weight and daily intake of specific categories. The system will require a user to sign up for an account and will generate a random forty character string. The user will then be sent an email with a unique address in it for verification to activate the account. Accounts must be activated within one day of sign up and a scheduled cron job will remove old URLStrings from the database if they are not activated. All of this functionality is implemented at the Application Layer, written in PHP5. The Storage Layer consists MySQL database (please see the E/R Diagram and Relational Models in Appendix A & B respectively).

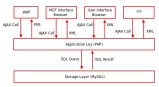


Figure 2: High Level Diagram of HealthTracker

The high level look illustrates the tiered approach. There are several Presentation Layers utilizing the same application layer functions. The current system has a presentation for the user interface and a management interface that exist in a standard web browser such as Mozilla Firefox. This contains formatting data such as Cascading Style Sheets (CSS) and HTML for the presentation of the data and utilizes AJAX calls to get data from the Application Layer. The WAP interface, a browser for cell phones/smart phones, also uses AJAX calls to receive and format data, however, the presentation of the data is not as in depth as the standard browser and displays the data in a different way. The WAP browser has a completely different presentation layer than the standard browser due to its intended use (limited screen size but highly mobile) and yet it can use the same application layer as the standard browser. This could be extended to provide an interface for a Lynx browser (a text based browser) or for a web service. Furthermore, this could support other front ends from Java based GUIs or perhaps our watch example earlier. This is flexible in that as future devices and technologies develop, these can also call the application layer and receive the XML formatted data back.

Here we examine how the User Interface Presentation Layer interacts with the Application Layer by looking at an example of the interaction. The example illustrates the AJAX calls and returns from a PHP script for returning a CommonBasicItem to the Presentation Layer and how it parses this.

The first two elements to be examined is the initiation of the AJAX object and the JavaScript function

that is called to receive the CommonBasicItem. For the sake of brevity, only the JavaScript function is called and the specific ID value for the CommonBasicItem has already been identified and is being passed into the JavaScript function.

```
var xmlHttp;

function initializeAJAX() {
    try  {
        // Firefox, Opera 8.0+, Safari
        xmlHttp=new XMLHttpRequest();
    } catch (e)  {
        // Internet Explorer
        try {
            xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
                    xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {
                alert("Your browser does not support AJAX!");
                return false;
            }
        }
    }
}
```

Figure 3: AJAX Set up

The first item is a function that sets up an XMLHttpRequest object. Unfortunately, since this is a client side script, the set up must take into account how various browsers handle the code set up. That leaves us with a confusing block on the surface. Fortunately, it is easily explained. First, the script attempts to set up the object for Firefox, Opera, and Safari browsers. If this attempt fails, the script attempts to set up the object for Internet Explorer 6 and later. If this also fails, the last attempt is to set the object up for Internet Explorer 5.5. If all of these attempts fail, the browser does not support AJAX calls and the user is alerted. Assuming that this does not fail for our example, the AJAX Object is now loaded into the variable xmlHttp.

Next, the Presentation Layer for the AJAX call is presented.

```
function getItem(itemID) {

        xmlHttp.onreadystatechange=function() {
            if(xmlHttp.readyState==4) {
                var xmlDoc = xmlHttp.responseXML;
                var root = xmlDoc.getElementsByTagName('XMLData').item(0);
                var items = new Array(root.childNodes.length);
                for (var iNode = 0; iNode < root.childNodes.length; iNode++) {
                        var node = root.childNodes.item(iNode);
                        items[root.childNodes.item(iNode).nodeName] = new
Array(node.childNodes.length);
                        for (i = 0; i < node.childNodes.length; i++) {
                            var childNode = node.childNodes.item(i);
                            for (j = 0; j < childNode.childNodes.length; j++) {

items[root.childNodes.item(iNode).nodeName][node.childNodes.item(i).nodeName] =
childNode.childNodes.item(j).nodeValue;
                            }
                        }
                }
                output(items);
            }
        }
        xmlHttp.open("GET","app/returnCommonBasicItems.php?itemID=" + itemID,true);
        xmlHttp.send(null);
```

Figure 4: Presentation Layer Code for AJAX Call to Application Layer

This function accepts the itemID to return as a parameter in the function. The first line to focus on is at the bottom of this function; the xmlHttp.open line calls the PHP script called returnCommonBasicItems.php and passes in the itemID as a query string variable. The AJAX call is initiated with the xmlHttp.send() function call and the system awaits a return from the server. Remember, AJAX is an acronym for **Asychronous** JavaScript and XML. This means that since the call is asynchronous, the system is not held up while it is waiting. The system goes about its business and still interacts with the user while it awaits the return. The opposite of this is SJAX (Synchronous JavaScript and XML), sometimes called synchronous AJAX. This holds the system up until the call is returned from the browser. This is useful as well if you are need access to the data before your application can move on. The status of the call is accessed by calling the onreadystatechange and when this returns a readyState of four (indicating that the call is fully completed), the data is loaded into a variable. In this case, the responseXML property is called since the script is returning XML. If the javascript called the reponseText property, the data returned would not be considered an XML object and thus, the properties and functions in javascript that are used to walk the XML tree could not be called. The XML object is loaded into the variable xmlDoc. The root variable is loaded with the root element in the XML, called XMLData. The script then uses a series of for loops to walk the XML tree, instantiating a two dimensional associative array and loading it with the data in the format **items[ItemID, [ColumnName, Value]].** When this is completed, the script passes the array into another function that handles how the data will be displayed to the user. While the AJAX call is awaiting a response, what is happening under the hood?

The PHP script is called and processes the parameters, performs its logic and returns its output.

```
<?php

    session_start();
    header('Content-Type: text/xml');

    include("ConnectDB.php");

    $itemID = $_GET['itemID'];

    echo returnCommonBasicItems($itemID);
```

Figure 5: Application Layer Code for PHP Script to Perform Application Logic

The PHP script is called, initiates the users session and connects to the database. The script then loads the query string variable into a PHP variable $itemID. This $itemID is passed into the returnCommonBasicItems() function.

The PHP script interacts with the Data Storage Layer with the following SQL query.

```
SELECT CommonBasicItems.BID, CommonBasicItems.ItemName,
CommonBasicItems.ItemDesc, CommonBasicItems.ImagePath,
BasicItemsCategories.CategoryName, Units2.Abbrev, ItemsTracked.ItemName,
Units1.Abbrev, CommonItemsValues.Value
FROM CommonBasicItems, CommonItemsValues, BasicItemsCategories, ItemsTracked,
Units Units1, Units Units2
WHERE CommonBasicItems.BID = CommonItemsValues.BID AND
CommonItemsValues.TID = ItemsTracked.TID AND Units1.ID = ItemsTracked.UnitsID AND
CommonBasicItems.AmtID = Units2.ID AND  CommonBasicItems.Category =
BasicItemsCategories.CatID AND CommonBasicItems.BID = " . $itemID;
```

Figure 6: SQL Call From PHP Script Interacting with Data Layer

The SQL query for this is complicated as it queries across several tables. This is not a dicussion on SQL queries and, thus, will not be discussed in detail here. The SQL query returns the ID of the item, the name, the description, the path to an image if one exists, the units for the item, the items tracked and the values and units on each of those items. It returns something like:

```
mysql> SELECT CommonBasicItems.BID, CommonBasicItems.ItemName, CommonBasicItems.ItemDesc, CommonBasicItems.ImagePath, BasicIt
emsCategories.CategoryName, Units2.Abbrev, ItemsTracked.ItemName, Units1.Abbrev, CommonItemsValues.Value FROM CommonBasicItem
s, CommonItemsValues, BasicItemsCategories, ItemsTracked, Units Units1, Units Units2 WHERE CommonBasicItems.BID = CommonItems
Values.BID AND CommonItemsValues.TID = ItemsTracked.TID AND Units1.ID = ItemsTracked.UnitsID AND CommonBasicItems.AmtID = Uni
ts2.ID AND  CommonBasicItems.Category = BasicItemsCategories.CatID AND CommonBasicItems.BID = 2;
+-----+----------------+-------------+-----------+--------------+--------+-----------------------+--------+---------+
| BID | ItemName       | ItemDesc    | ImagePath | CategoryName | Abbrev | ItemName              | Abbrev | Value   |
+-----+----------------+-------------+-----------+--------------+--------+-----------------------+--------+---------+
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Calories              | cal    |   11.60 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Total Fat             | g      |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Saturated Fat         | g      |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Trans Fat             | g      |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Cholesterol           | mg     |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Sodium                | mg     |   21.60 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Total Carbohydrates   | g      |    2.60 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Sugar                 | g      |    1.60 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Fiber                 | g      |    0.60 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Protein               | g      |    0.30 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Calcium               | mg     |    6.70 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Potassium             | mg     |   90.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Folic Acid            | mcg    |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin A             | IU     | 2000.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin B1 (Thiamin)  | g      |    8.70 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin B2 (Riboflavin) | mcg  |   10.30 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin B6            | mg     |    0.03 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin B12           | mcg    |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin C             | mg     |    2.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin D             | IU     |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin E             | mg     |    0.00 |
|   2 | Baby Carrots-Raw | Baby Carrots |           | Vegetable    | ou.    | Vitamin K             | mcg    |    2.60 |
+-----+----------------+-------------+-----------+--------------+--------+-----------------------+--------+---------+
22 rows in set (0.00 sec)

mysql>
```

Figure 7: Data Storage Layer Query Results

These results are returned to the PHP script in the form of a result set. The rest of the PHP script is below:

```
/* Return CommonBasicItem with ID of $itemID - w/ its values */
function returnCommonBasicItems($itemID) {
    $sqlString = "SELECT CommonBasicItems.BID, CommonBasicItems.ItemName,
CommonBasicItems.ItemDesc, CommonBasicItems.ImagePath,
BasicItemsCategories.CategoryName, Units2.Abbrev, ItemsTracked.ItemName,
Units1.Abbrev, CommonItemsValues.Value FROM CommonBasicItems,
CommonItemsValues, BasicItemsCategories, ItemsTracked, Units Units1, Units Units2
WHERE CommonBasicItems.BID = CommonItemsValues.BID AND
CommonItemsValues.TID = ItemsTracked.TID AND Units1.ID = ItemsTracked.UnitsID AND
CommonBasicItems.AmtID = Units2.ID AND  CommonBasicItems.Category =
BasicItemsCategories.CatID AND CommonBasicItems.BID = " . $itemID;
    $resultSet = mysql_query($sqlString);
    $numResults = mysql_num_rows($resultSet);
    $XMLString = "<?xml version='1.0' encoding='utf-8'?><XMLData>";
    $itemID = "";
    for ($i=0, $j=0; $i<$numResults; $i++,$j++) {
        $row = mysql_fetch_array($resultSet, MYSQL_NUM);
        if ($row[0] != $itemID) {
            if ($i != 0) {
                $XMLString .= "</ID" . $itemID . ">";
            }
            $j = 0;
            $XMLString .= "<ID" . $row[0] . ">";
            $XMLString .= "<ItemName>" . $row[1] . "</ItemName>";
            $XMLString .= "<ItemDesc>" . $row[2] . "</ItemDesc>";
            $XMLString .= "<ImagePath>" . $row[3] . "</ImagePath>";
            $XMLString .= "<CategoryName>" . $row[4] . "</CategoryName>";
            $XMLString .= "<ItemUnit>" . $row[5] . "</ItemUnit>";
        }
        $itemID = $row[0];
        $XMLString .= "<TrackedItemName" . $j . ">" . $row[6] . "</TrackedItemName" . $j .
"><TrackedItemUnit" . $j . ">" . $row[7] . "</TrackedItemUnit" . $j . "><TrackedItemValue" .
$j . ">" . $row[8] ."</TrackedItemValue" . $j . ">";
    }
    $XMLString .= "</ID" . $itemID . ">";
    $XMLString .= "</XMLData>";
    return $XMLString;

}

?>
```

Figure 7: Application Layer Code for PHP Script to Perform Application Logic

The result set is loaded into a variable named $resultSet. The XMLString that will be returned is started with an XML header, defining the version of the XML document being built. It also opens the root layer element, XMLData. The number of rows returned in the query is stored in a variable $numResults and then the script then loops through a for loop processing the rows of the result set, each iteration grabbing the next row in an associative array by column name. Other logic is built into this function, such as monitoring when the itemID changes and closing that element due to this function being reused for other purposes, including return multiple CommonBasicItems. The first time through it adds in the elements common to all rows for the same itemID, such as item name, description, and category. After this, it adds each item tracked with its unit type and its value for the CommonBasicItem. When the loop exits, it closes the last itemID element and the root XMLData element tag. It returns the XMLString which is echoed back to the browser making the AJAX call. It returns an XML Document that

looks like this:

```
<XMLData>
  <ID2>
    <ItemName>Baby Carrots-Raw</ItemName>
    <ItemDesc>Baby Carrots</ItemDesc>
    <ImagePath/>
    <CategoryName>Vegetable</CategoryName>
    <ItemUnit>ou.</ItemUnit>
    <TrackedItemName0>Calories</TrackedItemName0>
    <TrackedItemUnit0>cal</TrackedItemUnit0>
    <TrackedItemValue0>11.60</TrackedItemValue0>
    <TrackedItemName1>Total Fat</TrackedItemName1>
    <TrackedItemUnit1>g</TrackedItemUnit1>
    <TrackedItemValue1>0.00</TrackedItemValue1>
    <TrackedItemName2>Saturated Fat</TrackedItemName2>
    <TrackedItemUnit2>g</TrackedItemUnit2>
    <TrackedItemValue2>0.00</TrackedItemValue2>

    .... Lines Cut for Brevity ....

    <TrackedItemName20>Vitamin E</TrackedItemName20>
    <TrackedItemUnit20>mg</TrackedItemUnit20>
    <TrackedItemValue20>0.00</TrackedItemValue20>
    <TrackedItemName21>Vitamin K</TrackedItemName21>
    <TrackedItemUnit21>mcg</TrackedItemUnit21>
    <TrackedItemValue21>2.60</TrackedItemValue21>
  </ID2>
</XMLData>
```

Figure 8: XML Document Returned

This document is returned to the JavaScript that made the call. The script then walks the XML Document and loads the values into an array as discussed earlier (items[ItemID, [ColumnName, Value]]). For example, items['ID2', ['ItemName', 'Baby Carrots-Raw']] is a value in the array, as is items['ID2', ['TrackedItemName0', 'Calories']] and items['ID2', ['TrackedItemValue0', 11.60]]. The flow chart of this process is below:

Figure 9: Application Flow Chart

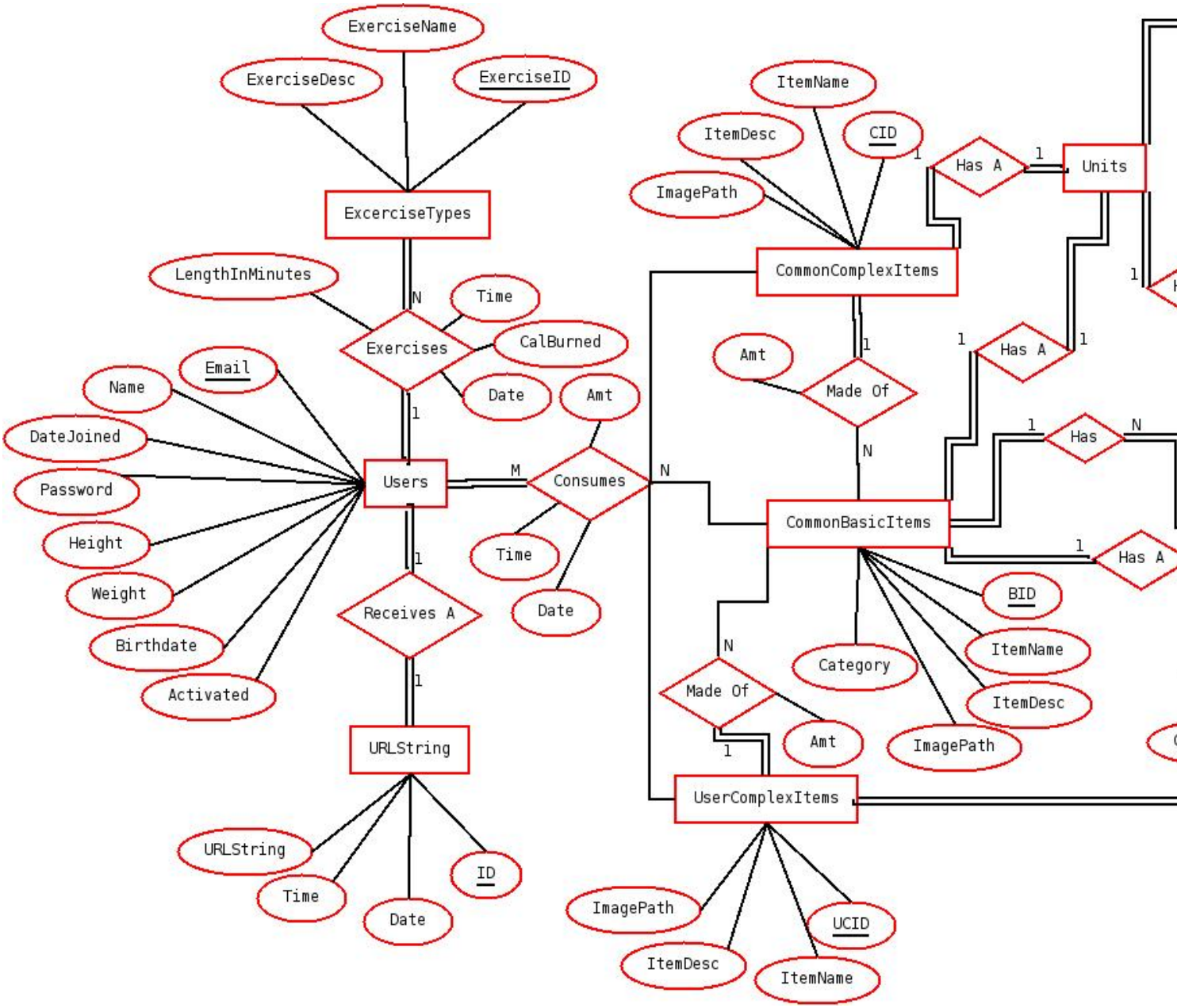
This flow chart illustrates the call flow for the above example. The main purpose for this modularity is to allow substution of the Presentation Layer. If we instead used a WAP browser, the call flow and XML returned would be identical, but the Presentation Layer would be the WAP, which would handle how it displayed that data completely different.

Conclusion

AJAX calls can be utilized to separate the Presentation Layer from the Application Layer in Web Applications. As the number of people and businesses connected to and leveraging the Internet continues to increase, distributed applications are becoming increasingly popular. Web technologies are quickly replacing traditional development technologies and methodologies and these must be able to

adapt to an ever changing technological front. To prevent an application from becoming obsolete, it must be able to adapt. With this need, the separation of the Presentation and Application Layers is a must. AJAX allows developers to accomplish this goal by standardizing calls and returning XML formatted documents. XML itself is inherently flexible and is becoming the norm for data transfer as it is platform independant. By leveraging this flexibility as communication between our Presentation and Application layers, future technologies will still be able to interact with our applications. This flexibility in our distributed applications will ensure the code written today will still be relevant tomorrow.

Appendix A: E/R Model



Appendix B: Relational Model

Users (Email, Name, DateJoined, Password, Height, Weight, BirthDate, Activated)

ItemsTracked (TID, ItemName, ItemDesc, UnitsID, DailyValue)

CommonBasicItems (BID, ItemName, ItemDesc, ImagePath, Category, AmtID)

CommonItemsValues (ID, BID, TID, Value)

BasicItemsCategories (CatID, CategoryName)

CommonComplexItems (CID, ItemName, ItemDesc, ImagePath)

CommonComplexItemsBasic (ID, CID, BID, Amt, AmtID)

UserComplexItems (UCID, Email, ItemName, ItemDesc, ImagePath)

UserComplexItemsBasic (ID, UCID, BID, Amt, AmtID)

Units (ID, Name, Abbrev)

Intake (ID, Email, Date, Time, ItemID, ItemType, Amt, AmtID)

Exercise (ID, Email, Date, Time, CalBurned, ExerciseID, LengthInMinutes)

ExerciseTypes (ExerciseID, ExerciseName, ExerciseDesc)

URLString (ID, Email, Date, Time, URLString)