

# UPGRADING THE WEB: A MODEL FOR TRANSITIONING WEB APPLICATIONS TO THE WEB 2.0 SERVICE ORIENTED ARCHITECTURE

**Scott A. Wymer**  
Morehead State University  
s.wymer@moreheadstate.edu

## Abstract

*This paper presents a model for transitioning existing web applications based upon a standard scripting architecture to Web 2.0 applications that rely more upon a Service Oriented Architecture, SOA. An overview of what is considered standard web application scripting practice and architecture is presented, and this is contrasted to the current architecture model for Web 2.0 applications based upon the AJAX "standards". The benefits of moving to the new architecture are presented and explained in terms of the motivating factors for developing this model. The model is laid out as a methodology for making the transition and is explained as a staged process. Each stage is designed to produce a working prototype of the application.*

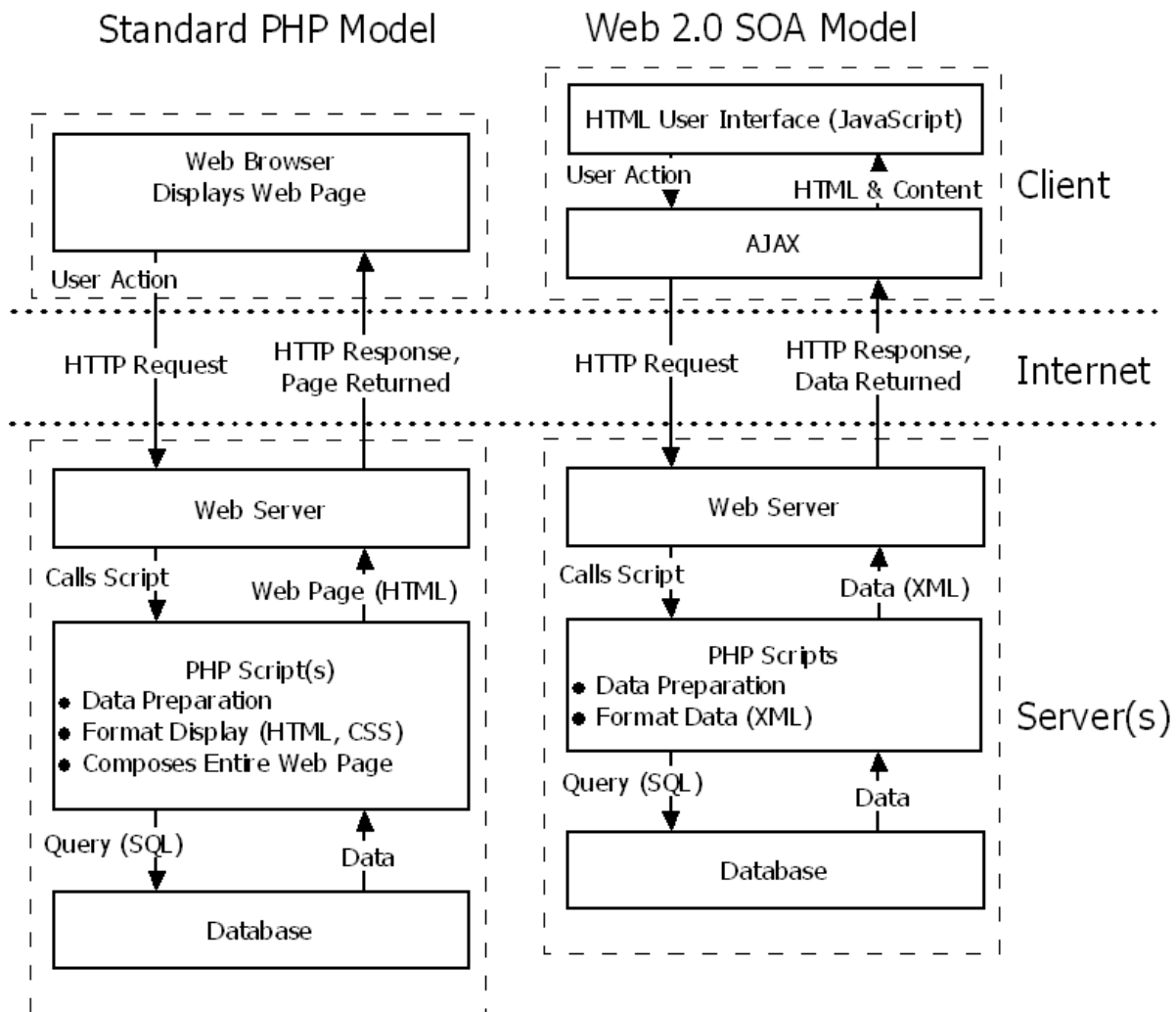
**Keywords:** Web Application Development, PHP Programming, AJAX, Web 2.0, Service Oriented Architecture, JavaScript Programming

## Introduction

There is a great deal of excitement today about the current and potential use development of what is being called the "Web 2.0". This term was coined in part by the O'Reilly Media publishing company, and "there's still a huge amount of disagreement about just what Web 2.0 means, with some people decrying it as a meaningless marketing buzzword, and others accepting it as the new conventional wisdom" (O'Reilly, 2005). From the article published by O'Reilly defining the term, this paper focuses on the defining aspects of the "Web 2.0" that deal with the technical development of a web application, namely: an application that is based upon a "light-weight", service oriented architecture, SOA; and uses AJAX (Asynchronous JavaScript and XML) (Garrett, 2005, Mahemoff, 2006). AJAX is used to access the services that compose the application, and to provide a richer user interface (O'Reilly, 2005). Despite all the hype and emotion surrounding the term Web 2.0, there are many aspects of it that derive from fundamental principles of good application and user-interface design, and thus there is value in considering moving applications that currently run under older web scripting frameworks to the new architecture. Though there are many books and articles written explaining what the new architecture is and how to program it there is no discussion of a systematic way to transition a legacy application in the standard framework into the new Web 2.0 framework. This paper presents such a model that can be used to convert an existing "standard" web application, i.e. one that is built on the pre-Web 2.0 (sometimes referred to today as "Web 1.0") model, to take advantage of elements of the new application architecture.

The first step is to examine the differences between the standard web application model and the Web 2.0 model (Figure 1). This paper focuses on Web applications based upon standard practice using scripting languages such as PHP and Perl. The model presented would vary greatly if one was using more object-oriented programming languages such as Java or Ruby. The main difference in using more procedural languages, such as PHP, is the need for a separate step to encapsulate code to be built into a service. Scripting languages such as PHP have the ability to perform in-line execution of small subprograms that are "included" at any point in a script (*php.net*, 2006). This "included" code runs as part of the main code at the location in which it is included, and all of the variables and functions in this code have the same scope as the surrounding lines of code in the main application at the include point. The use of this ability to include small subprograms will be the key factor in our model for transitioning to service oriented architecture. While both PHP and Perl are object-oriented languages they can both be used, and

most commonly are, utilizing few or no object-oriented programming methods. That is they are most commonly used as procedural programming languages (Knight and Dai, 2002).



**Figure 1. Comparison of architecture models and client, server responsibilities in the standard PHP web programming model, and the Web 2.0 Service Oriented Architecture, SOA, model**

Most standard PHP web applications do a poor job of separating data preparation and display logic, and most applications are a mixture of programming languages (PHP and JavaScript, often with embedded SQL queries as well), and display markup (HTML and CSS). This common web application development methodology, involving mixing data preparation and display logic, violates the standard design pattern which calls for the separation of data preparation, and display logic. This is one of the major problems in web application development and in supporting web applications (Kerer and Kirda, 2001). Moving an existing web application to the Web 2.0 service oriented architecture allows for a natural redesign of the application to meet this design pattern. In the Web 2.0 architecture, the small server-side scripts handle only data preparation logic and deliver data in a standardized format which is then parsed and formatted for display by the client/browser.

This distribution of responsibilities between the software on the server, i.e. the PHP scripts, and software running on the client machine, i.e. in the browser via JavaScript, is one of the major differences in the new Web 2.0 architecture. In standard web applications, it is the server's responsibility to deliver complete and finished web pages (Figure 1). So the server scripts must handle all processing to deliver these pages, including: data preparation (deciding what data is needed, and getting data from the database), and then formatting that data for display by generating the final HTML markup around the data. In the standard architecture the client contributes very little to the overall application, and simply acts as the interface to accept data and tell the browser when to refresh the page. JavaScript may add some dynamic elements in the user interface, but only with pre-loaded data from the last page refresh.

In standard and Web 2.0 architectures the scripts on the server-side are very different. While in the standard architecture the script(s) are designed to act as a single “master script” which accesses the required data and delivers a complete formatted webpage. In the Web 2.0 architecture model there is still one master script which will be the script called in the URL when a page loads. In addition, there are smaller scripts which are part of the SOA, each of which works independently to deliver the required data for a small section of a webpage. When the master page is loaded the smaller “child scripts” will be called upon to deliver their data for each of the appropriate sections of the master page as it is displayed in the browser. After the master page has loaded further user interaction with the page takes place mainly in the child scripts and calls to the master page, which require a complete page refresh, are avoided. This is in contrast to the standard PHP architecture where any display of new data requires the full master page to be refreshed and individual sections cannot refresh independently.

Also the client role is significantly different in the Web 2.0 architecture. Here JavaScript on the client-side, using AJAX, can dynamically call to small scripts on the server side to update small dynamic sections of the page. Thus the client-side portion of the application can now dynamically respond to user actions to update only the data that is required, and the user does not have to wait for a complete page refresh to see the resulting new data. The client JavaScript code can dynamically update whichever portions of the page are required to show the newly fetched data. Also, in the Web 2.0 model the JavaScript running in the browser takes over some of the responsibility for formatting the data for display. AJAX, on the client side, can automatically parse XML data, which can then be inserted into the HTML of the web page and formatted with CSS (Cascading Style Sheets) or XSL (eXtensible Stylesheet Language) (Figure 2) (Mahemoff, 2006).

### ***Motivation for the Developed Methodology for moving PHP Applications to Web 2.0***

Clearly there are distinct advantages for web applications using the Web 2.0 architecture over the standard PHP architecture. By developing a methodology for transitioning standard web applications to this new architecture we can benefit from these architectural improvements without having to totally rewrite the existing application. In addition, the Web 2.0 service oriented architecture is not just a better theoretical design; it also provides significant run-time efficiencies for the Web application. Avoiding page refreshes, by allowing AJAX calls to only refresh data for small sections of a webpage, can show significant savings in both server processing and network bandwidth usage since only the data that is being changed is called over the network, and the rest of the data on the page remains static and is not refreshed. This also avoids extraneous calls that would be necessary under the standard architecture, to the database to refresh data that has not changed in the displayed page. For the user the service oriented architecture provides pages which can be much more responsive and can load requested changes much faster. In terms of responsiveness, it should be noted that calls using the AJAX protocol can be asynchronous and a user could perform a second action requesting data for a different section of the page while waiting for the data refresh from a previous action. Such asynchronous behavior is not possible in the standard architecture where a complete page refresh is required between every action.

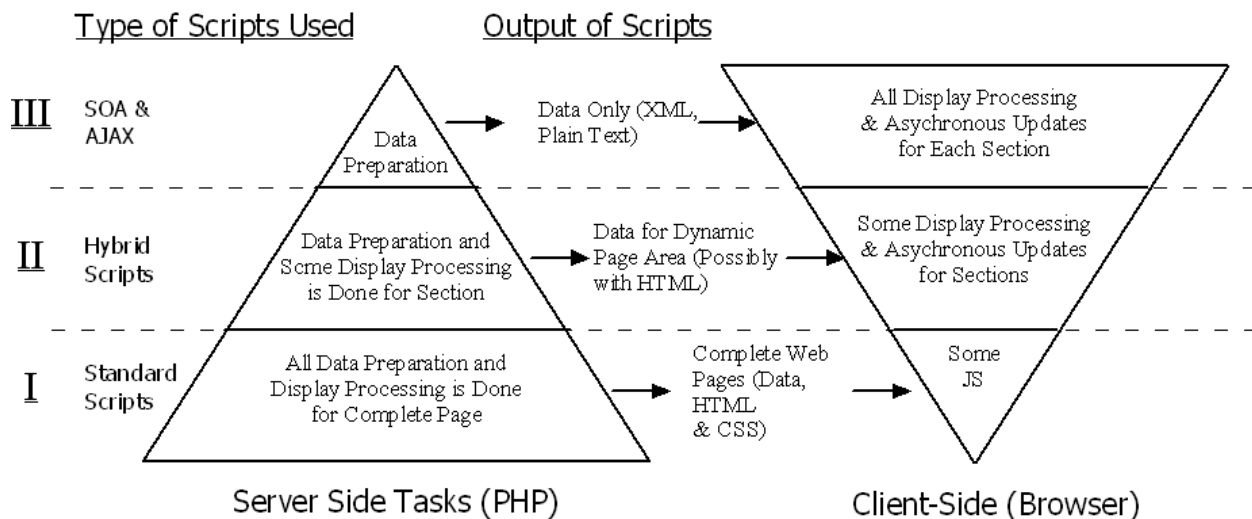
The advantages of using the new Web 2.0 architecture are clear; the question becomes how can a developer transition legacy Web applications in the most efficient manner to benefit from the new architecture as simply and quickly as possible. Any methodology for transitioning scripts developed in this article was required to meet the following criteria:

1. It should allow transition of the scripts incrementally without a complete rewrite of the application
2. the methodology was to be composed of incremental steps that build upon one another
3. the output of each step should be a working prototype of the application which is some kind of hybrid between a standard architecture and Web 2.0 architecture
4. the output hybrid code should allow possible parallel paths for further development in either the standard architecture, or the new Web 2.0 architecture
5. the methodology should allow the developers doing transition, who might be familiar only with the standard architecture, to adopt a slower learning curve in understanding the Web 2.0 architecture.

### **Methodology for Transitioning Standard PHP Application to Web 2.0**

The goal of the methodology developed in this paper is not to produce a final application that meets all of the aspects of the Web 2.0 architecture. In particular the Web applications are not being rewritten to output XML data. While XML data is not strictly necessary for the Web 2.0/Ajax architecture, it has significant advantages in allowing

the child scripts that are developed to be used as general Web services for other applications. However rewriting the script to use XML output and to parse and display this as HTML, violates the criteria four and five above: XML output would not allow continued programming in the standard PHP framework, and it would require developers who have no experience with XML to learn a whole new framework and paradigm. Figure 2 shows how the proposed hybrid architecture that this methodology would develop (Level II in the figure) will fit between the standard architecture, Level I, and a new Web 2.0/AJAX architecture, Level III.



**Figure 2. Model of proposed Hybrid Architecture (Level II). Level I shows the standard PHP Scripting Architecture where almost all processing is done on the server, Level III shows the Web 2.0 architecture composed of SOA on the server side and AJAX on the client side. Level II is a hybrid, transitional architecture that facilitates conversion**

The methodology to be followed, that both meets the guidelines specified above, and will produce the desired hybrid architecture, has four steps:

1. Isolate the dynamic sections of the web page into separate include files
2. Give include files more independence
3. "AJAX'ify" the scripts
4. Move to full version of SOA.

To convert a web application the steps are applied to each web page that comprises part of the interface for the application. Each step can be done independently and each step will produce a working prototype version of the web application. The focus of this method is, for each web page of an application, to take the one monolithic PHP script and break it up smaller independent child scripts that each provides a simple service of delivering the data necessary for a given dynamic section of the webpage being viewed. Then rebuild the master script so that it properly integrates the child scripts.

In step 1 the goal is to identify and isolate all the sections of a page that depend upon dynamic data and could or would change each time the page is loaded. Once the dynamic sections of the Web page are identified the code for that section is cut from the master script and inserted into a new child file. At the point in the master file where the code had previously occupied an include statement is inserted to call the child script and include its source code at that point. At the end of Step 1 the original monolithic master script now holds only: formatting for the layout of the entire page, the static page contents that will not change, and an include statement for each of the dynamic sections. In order to facilitate manipulation of the dynamic sections by JavaScript, each should be wrapped in a uniquely identified HTML "div" element.

Moving toward the next step, each of the individual child files produced should be as self-contained as possible. Therefore any unique variables used by the sections of code in each child file should be moved inside the files. Each child file will also handle its own database calls and all the code necessary to process and format the returned data. In this hybrid model, each child script will output finished HTML markup for displaying the data it handles

for the section of the master page. To test each child file a generic, test wrapper script can be written to provide a generic page template. This wrapper script would contain an include call to the single child file, and if the files have been configured correctly the wrapper script will display the page section from the child with the dynamic data formatted properly.

The goal of Step 2 is to produce child scripts which are truly independent and could function without having to be called as part of a master script. The child scripts should handle all of the data that they might require being passed from browser forms via HTTP. The main issue here is the need to make sure that the child files can handle any session or security logic, and contain all of the variables they'll need to run fully independent of the master script. Including session handling and security logic into each child file can lead to some conflicts when they are built as includes into the master file, and care must be taken. At this stage in order to build a simpler working prototype where the child files are still incorporated as includes into the master file, i.e. before moving on to step three, simple conditional logic can be put into each child to avoid conflicts with possible session and security logic already established from the master file, or a previous child include. In Step 2 no editing of the master file should be required. At this point, the prototype produced will still act as a standard PHP script in that all changes to any dynamic content will require a complete page refresh. In the next steps, JavaScript code will be used to make AJAX calls to our child files, and we start using a true service oriented architecture.

Steps 3 and 4 involve adding new JavaScript code to allow AJAX calls from the browser to the server to pull the dynamic content of the child files. For step three, the child scripts are edited so that any user action which would have previously caused the page form data to be submitted for updating is now handled by appropriate JavaScript event handlers embedded in the markup delivered by the child script. The JavaScript functions associated with these event handlers will then make AJAX calls to the child scripts to fetch updated section contents. The JavaScript will also force the uniquely identified section that the child script is associated with to update its contents when the AJAX call returns the new data and markup. At this point we have a totally different model for user-interactivity. Instead of requiring a whole form or page's data to be submitted, and waiting for the entire page to refresh, we can provide multiple possible ways for the users to interact with a given section that would simply refresh the contents of that one section. This new model for interactivity and responsiveness should initiate some rethinking and redesign of how the user can interact with the web page. Discussions of how AJAX changes design models for user interactivity though is beyond the scope of this paper. At this point, since the child scripts are being called independent of a refresh of the master script it becomes necessary for each child script to be able to act independently as was proscribed in Step 2.

Now, at the end of Step 3 the child scripts can be tested in our simple wrapper script and the dynamic contents should work and update properly. The master page though at this point still calls the child scripts through PHP-based include statements. It is the goal of Step 4 to replace the PHP-based includes file calls with JavaScript-based AJAX calls to acquire the contents of the child files in the browser when the page initially loads. In this step the master file will be edited to replace all the includes and the child files can be edited to remove any extraneous conditional blocks that were required to handle issues of possible conflicts with multiple independent includes.

## Conclusions

With this process a legacy PHP application can be safely transitioned to the new SOA architecture slowly and carefully with a working prototype at each step. The methodology proposed also has the clear advantage that it provides a gradual transition, allowing developers to more slowly and carefully move their applications further into the new architecture as their knowledge and experience grows. The transitional working prototypes assure that as the process happens the whole application can be properly tested and even implemented at each of the transitional steps.

The final step is not strictly necessary to produce the improved hybrid scripts that will be able to show the type of interactivity available in Web 2.0 applications. However, it is a first step down the path to producing a fully service oriented architecture that will offers great flexibility in terms of component reuse. From the output of the final step four, with knowledge of XML, it would be an easy step to modify the child scripts to act as agents in a web services architecture to deliver their data in a generic format that could be usable in a wide variety of applications. This shows the last great benefit of this new architecture. Because we would end up with independent child scripts we can use them in other applications. Assuming these child scripts put out data in a more generalized format (XML, comma separated value, etc.) they could easily be used, or made available, as service agents for any type of client software that could use the data they deliver.

## References

- Garrett, J. J. (2005). Ajax: A new approach to web applications. Retrieved 12/02/2006, 2006, from <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- Kerer, C., & Kirda, E. (2001). Layout, content and logic separation in web engineering. In G. Goos, J. Hartmanis & J. vanLeeuwen (Eds.), *Web engineering: Software engineering and web application development* (Vol. 2016, pp. 135). Berlin: Springer Verlag.
- Knight, A., & Dai, N. (2002). Objects and the web. *IEEE Software*, 19(2), 51-59.
- Mahemoff, M. (2006). *Ajax design patterns* (1st ed.). Sebastopol, CA: O'Reilly Media.
- O'Reilly, T. (2005, 09/30/2005). What is web 2.0 - design patterns and business models for the next generation of software. Retrieved 1/4/2007, 2006, from <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- php.net (2006). Include. Retrieved 12/15/2006 from <http://us3.php.net/manual/en/function.include.php>