# A Reengineering of the Architecture of WeBWorK

David Gage, Zhiyuan Wu, Claude Richard
University of Waterloo
{dgage, z9wu, clrichar}@uwaterloo.ca

## ABSTRACT

WeBWorK is service-based software that allows teachers to post mathematics homework online, and allows students to complete that homework online. The software is old, and its architecture has become difficult to work with. This paper presents an extracted software architecture of WeBWorK, which seems like a very workable architecture. We then talk about some problems in the current architecture that makes it difficult for developers to adapt to new web application standards. We then propose a conceptual architecture, based on the concrete architecture with additional incremental changes, which is intended to be used as a reference by the developers of WeBWorK so they can refactor the software to make it significantly more workable. We base the motivation for these changes on use-case scenarios. We conclude with suggestions on how to alter WeBWorK's code gradually to get from the current architecture to our proposed architecture.

**General Terms**
Software Architecture

**Keywords**
Software Architecture, Conceptual Architecture, Concrete Architecture

## 1. INTRODUCTION

WeBWorK is a project started in 1995 by Michael Gage and Arnold Pizer from the University of Rochester. Over the years it evolved from a single cgi script to one of the first full web apps. It is now managed by the Mathematical Association of America and is used by over 400 schools around the world to serve math homework to students on the web.

WeBWorK consists of two parts, both written in perl. The first part is a parser for a mark up language called PG ("problem generation") that is used to create the problems contained within WeBWorK. The language includes both perl and LaTex allowing for flexibility and symbol rendering. The second part is the application run by apache that serves courses, problem sets, and problems to users. This application keeps track of teachers, students, answers and other statistics. This will be the main focus of this paper. We will consider PG to be an outside library used by WeBWorK.

The current architecture for WeBWorK has been developed over the years and has worked well. However with the appearance of full fledged web apps like gmail users are starting to expect a more desktop style feel from their web apps. On the surface this means that WeBWorK's form based interface needs to be updated to a more fluid dynamic UI. This upgrade can be done with javascript and front end work without a major architectural change, however it is a good idea to refactor the server side and examine the current architecture to see where modifications can be made to make this transition and future development significantly easier.

Jacobson and Lindström has described similar cases of re-engineering in object oriented development. [2] This paper considers decisions to maintain, enhance, discard and re-engineer
a project based on its changeability and business value. Since WeBWorK has high business value and is tough to change, then according to Jacobson and Lindström analysis matrix, the appropriate action for WeBWorK is to re-engineer. In this case, re-engineering is composed of three steps:

1. Reverse engineering
2. Conduct changes
3. Forward engineering

This paper attempts to apply reverse engineering to WeBWorK. As such, the steps to reverse engineering are:
a) Extract a concrete architecture
b) Create an abstract (reference) architecture
c) Create a mapping between the two.

This paper will begin by introducing the components of WeBWorK through user scenarios in Section 2. Given these scenarios, we will show, in Section 3, how these scenarios are served by the current syste, in other words we will introduce the current concrete architecture. In Section 4, we will form a conceptual architectural model. In Section 5, we will describe possible plans to conduct changes. We will also describe how our proposed conceptual architecture allows for forward engineering of future systems.

## 2. USER SCENARIOS

This section describes WeBWorK's workflow from a user's perspective. There are 2 types of users: teachers and students. Each type of user has access to different functionality in WeBWorK. Some scenarios apply to both types of users, such as authentication and registration. We now describe some of the most important scenarios for each type of user.

### 2.1 Teachers

In this section we describe the functionality of WeBWorK which only teachers can use.

*2.2.1 Create a course*

Some teachers can create a course through the web interface. A teacher needs administrative privileges to be able to do this.

*2.2.3 Add students to a class*

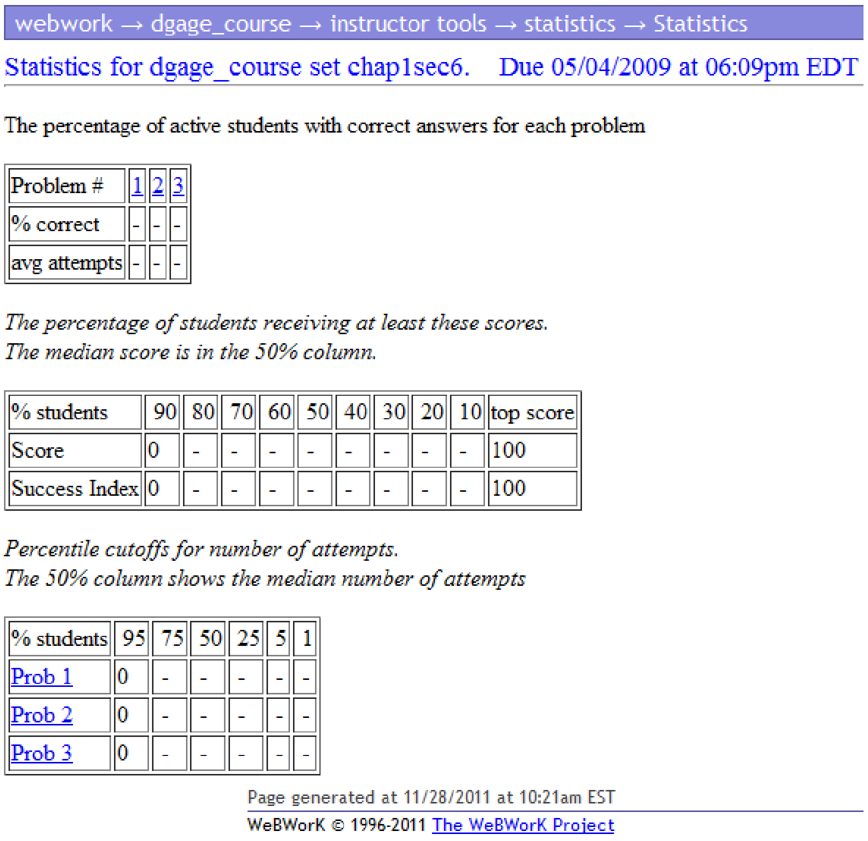
A teacher must add students to a class before assigning them to homework sets (does he?). He can do this in the web interface by either creating users or adding existing users to the course.

*2.2.2 Create/edit problems set*

A teacher can create problem sets and edit them. He then needs to assign students to the problem set so they can do their homework. The students may be a subset of the whole class so the teacher can assign customized homework for each student.

*2.2.4 View answers and statistics*

A teacher can view several statistics about the performance of his students on the problem set, including their progress before the deadline. He/she can click on “statistics” in the left-hand menu, then choose either a problem set or a user to view students' progress/performance.


webwork → dgage_course → instructor tools → statistics → Statistics

Statistics for dgage_course set chap1sec6. Due 05/04/2009 at 06:09pm EDT

The percentage of active students with correct answers for each problem

| Problem # | 1 | 2 | 3 |
|---|---|---|---|
| % correct | - | - | - |
| avg attempts | - | - | - |

*The percentage of students receiving at least these scores.*
*The median score is in the 50% column.*

| % students | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 10 | top score |
|---|---|---|---|---|---|---|---|---|---|---|
| Score | 0 | - | - | - | - | - | - | - | - | 100 |
| Success Index | 0 | - | - | - | - | - | - | - | - | 100 |

*Percentile cutoffs for number of attempts.*
*The 50% column shows the median number of attempts*

| % students | 95 | 75 | 50 | 25 | 5 | 1 |
|---|---|---|---|---|---|---|
| Prob 1 | 0 | - | - | - | - | - |
| Prob 2 | 0 | - | - | - | - | - |
| Prob 3 | 0 | - | - | - | - | - |

Page generated at 11/28/2011 at 10:21am EST

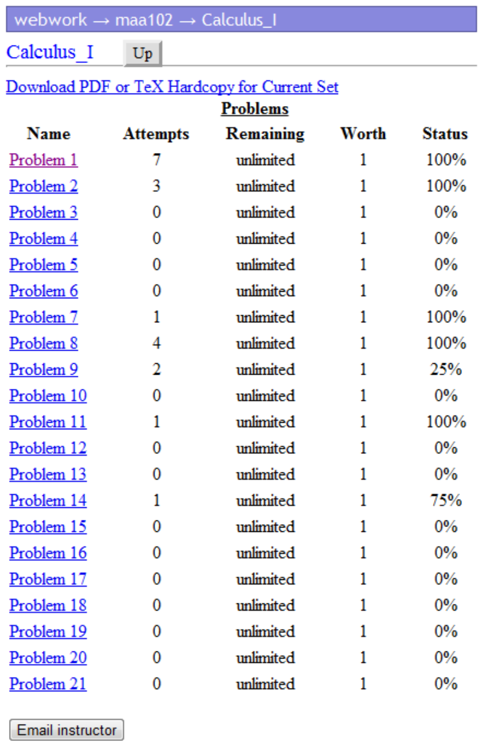
WeBWorK © 1996-2011 The WeBWorK Project

**Figure 2.1: Teacher views statistics for a specific problem set**

## 2.2 Students

A student can browse a homework set, work on problems in a homework set, and view his/her own statistics.

For each course that the student is assigned to, he/she can browse a homework set (or "problem set"). When the student clicks on a homework set, he/she is shown a list of the problems in that homework set. He/she is also shown some statistics on his/her current progress on the problems. When he/she clicks on a problem, a statement of the problem is displayed, along with some functionality (e.g. a textbox) for entering the answer.



webwork → maa102 → Calculus_I

Calculus_I Up

Download PDF or TeX Hardcopy for Current Set

**Problems**

| Name | Attempts | Remaining | Worth | Status |
|---|---|---|---|---|
| Problem 1 | 7 | unlimited | 1 | 100% |
| Problem 2 | 3 | unlimited | 1 | 100% |
| Problem 3 | 0 | unlimited | 1 | 0% |
| Problem 4 | 0 | unlimited | 1 | 0% |
| Problem 5 | 0 | unlimited | 1 | 0% |
| Problem 6 | 0 | unlimited | 1 | 0% |
| Problem 7 | 1 | unlimited | 1 | 100% |
| Problem 8 | 4 | unlimited | 1 | 100% |
| Problem 9 | 2 | unlimited | 1 | 25% |
| Problem 10 | 0 | unlimited | 1 | 0% |
| Problem 11 | 1 | unlimited | 1 | 100% |
| Problem 12 | 0 | unlimited | 1 | 0% |
| Problem 13 | 0 | unlimited | 1 | 0% |
| Problem 14 | 1 | unlimited | 1 | 75% |
| Problem 15 | 0 | unlimited | 1 | 0% |
| Problem 16 | 0 | unlimited | 1 | 0% |
| Problem 17 | 0 | unlimited | 1 | 0% |
| Problem 18 | 0 | unlimited | 1 | 0% |
| Problem 19 | 0 | unlimited | 1 | 0% |
| Problem 20 | 0 | unlimited | 1 | 0% |
| Problem 21 | 0 | unlimited | 1 | 0% |

Email instructor

**Figure 2.2: Student views a list of problems for a specific problem set**

## 2.3 Both Teachers and Students

In order to login to WeBWorK, a user currently needs to select a course, and log in to that course. In the future it would be desirable for users to login to a WeBWorK server before selecting a course.

Both students and teachers can download for example a pdf version of a homework set with the progress of a specific student. Some options are configurable, such as whether to include solutions in the pdf file, etc.

## 3. CONCRETE ARCHITECTURE

This section presents the current high-level architecture of WeBWorK. A high-level call graph is shown in figure 3.1.

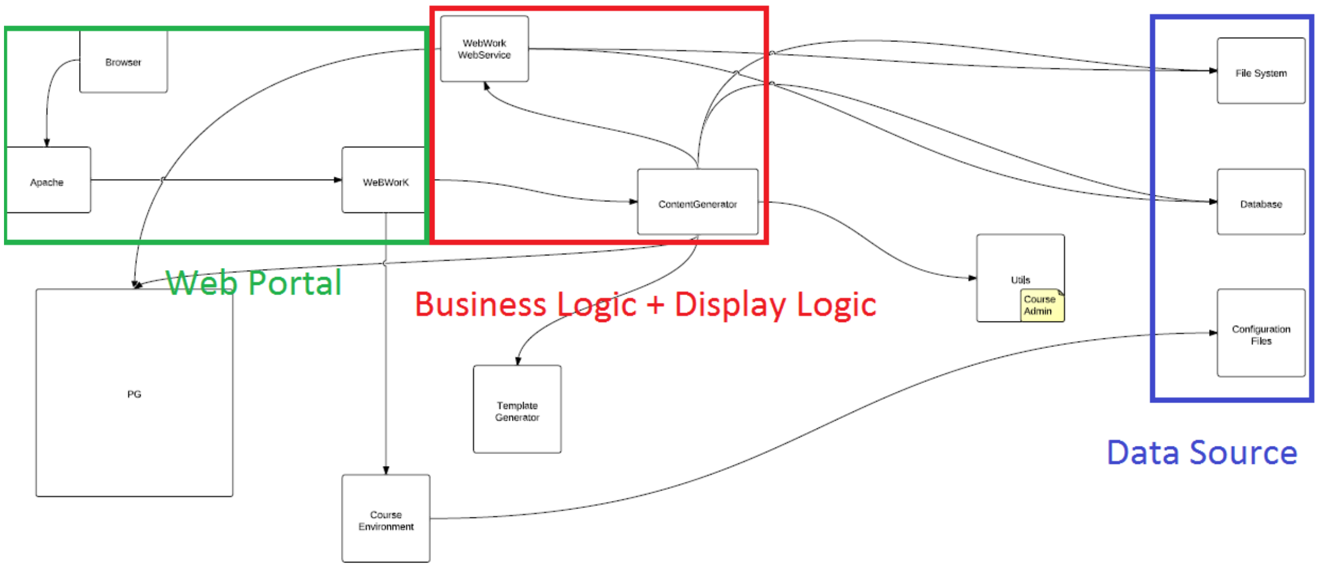


**Figure 3.1: High-level call graph of the concrete architecture of WeBWorK**

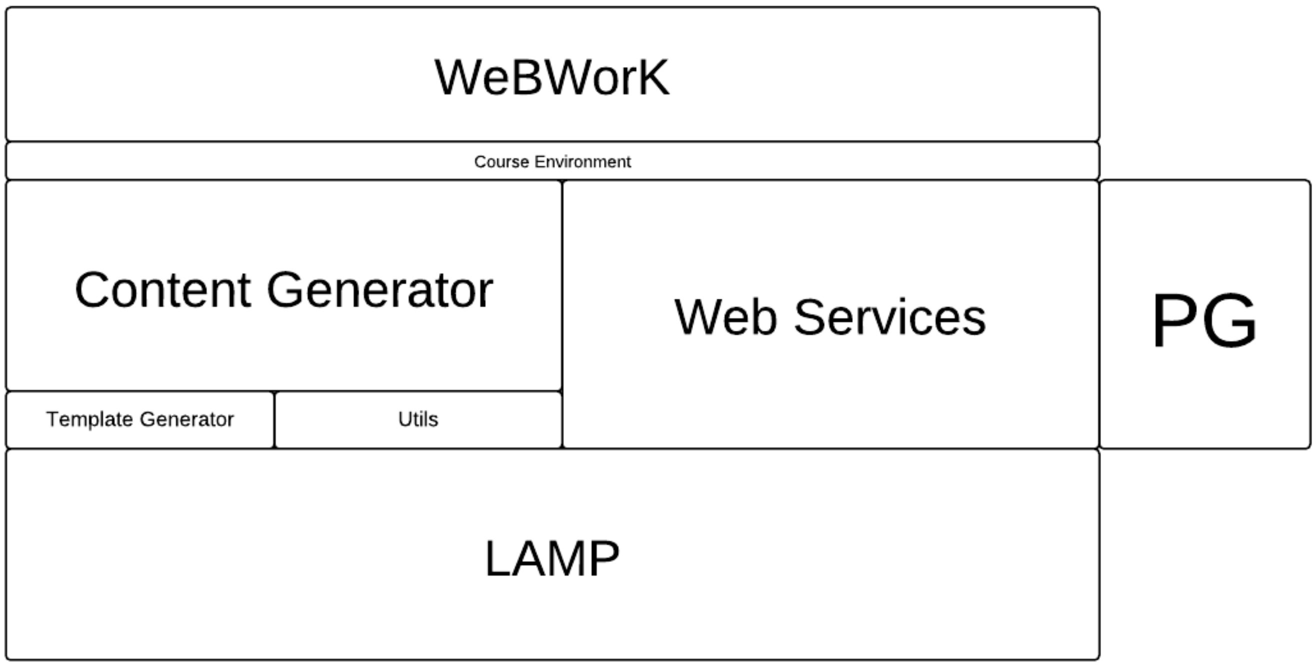


**Figure 3.2: Dependencies in the concrete architecture.**

## 3.1 WeBWorK and Course Environment

After being passed a request from Apache, WeBWorK takes over. It first builds a Course Environment using the information from the request, the .config files set up by the admin, and information from the database. The course environment will be used by any other components to figure out where files are located, what problem sets are available, and anything else they might need to know. WeBWorK also adds to the request, because Apache has limited fields in a request. WeBWorK implements its own version (Request.pm) adding on the fields necessary for the other components. Finally, the information is passed to the appropriate component of WeBWorK. The mapping between these components and the URL is found the URLPath.pm file.

## 3.2 Content Generator

After the WeBWorK module has done the set up, the Content Generator deals with finding the data and building the response. This module handles everything from course administration (through function calls to the Utils package) to displaying problem set statistics. We will not cover details for all of its components here, but essentially, any page a user can get to on a WeBWorK website is created within this package.

If the request involves rendering a problem the Content Generator finds the source code of the problem in the file system and sends it off to PG for rendering. PG is, in many ways, its own separate project, therefore it is not discussed in length in this paper. We treat it an external library for WeBWorK (which it essentially is).

There are two links in Content Generator which act as a bridge between the main content generation of WeBWorK and the web services. These are instructorXMLHandler.pm and renderViaXMLRPC.pm.

## 3.3 Web Service

The web service is currently a separate entity within WeBWorK. It is usually used by javascript ajax calls. However, they can be used by anything that might need XML or JSON responses. Some of the web services are also used by scripts written by users to make use of some of the functionality. Local problem rendering is a good example of this. There are two versions of web services at the moment: SOAP (WeBWorKSOAP), and XMLRPC (WeBWorKWebservice). Most recently, XMLRPC has been worked on to allow ajax calls between the server and the Library Browser 2 page. The content generation for these web services lives almost entirely in their directories, which usually results in duplicate code from the functions in Content Generator. One benefit of an architectural change will be a chance to consolidate these web services and Content Generator, so that code changes would only have to be made in one place.

## 3.4 The Problem

The concrete architecture is quite workable, assuming the user always wants the data requested to be rendered in the same way. A problem of the current architecture that the WeBWorK developers come across is when users may request the same data, but may want it to be rendered in a different way.

For example, a student logs in and clicks on a course in order to view a list of problem sets for that course. The content generator receives the request, queries the database to get the list of problem sets that the student requested, and returns an html representation of the list. In another view, let's say the student wants to subscribe to an rss feed of the problem set. In this case, the content generator needs to send the request to a completely different component that will do the same queries on the database, but render the resulting list differently.

ContentGenerator and WeBWorK Service are the most major examples of this kind of situation, where the ContentGenerator generates HTML and WeBWorK Service generates JSON. It makes changes difficult for the developers, since a change in the way the database is queried will result in several changes to the code in different places. The reason why the code is currently like this is because at the time of initial development of WeBWorK, HTML was the only way that users would want to render information, thus a separation between querying the database and rendering was unnecessary.

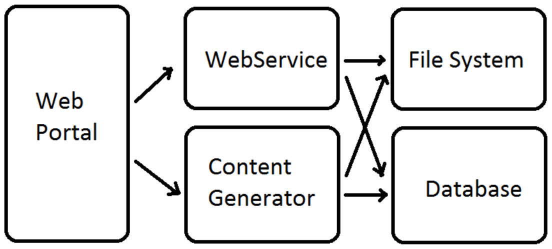
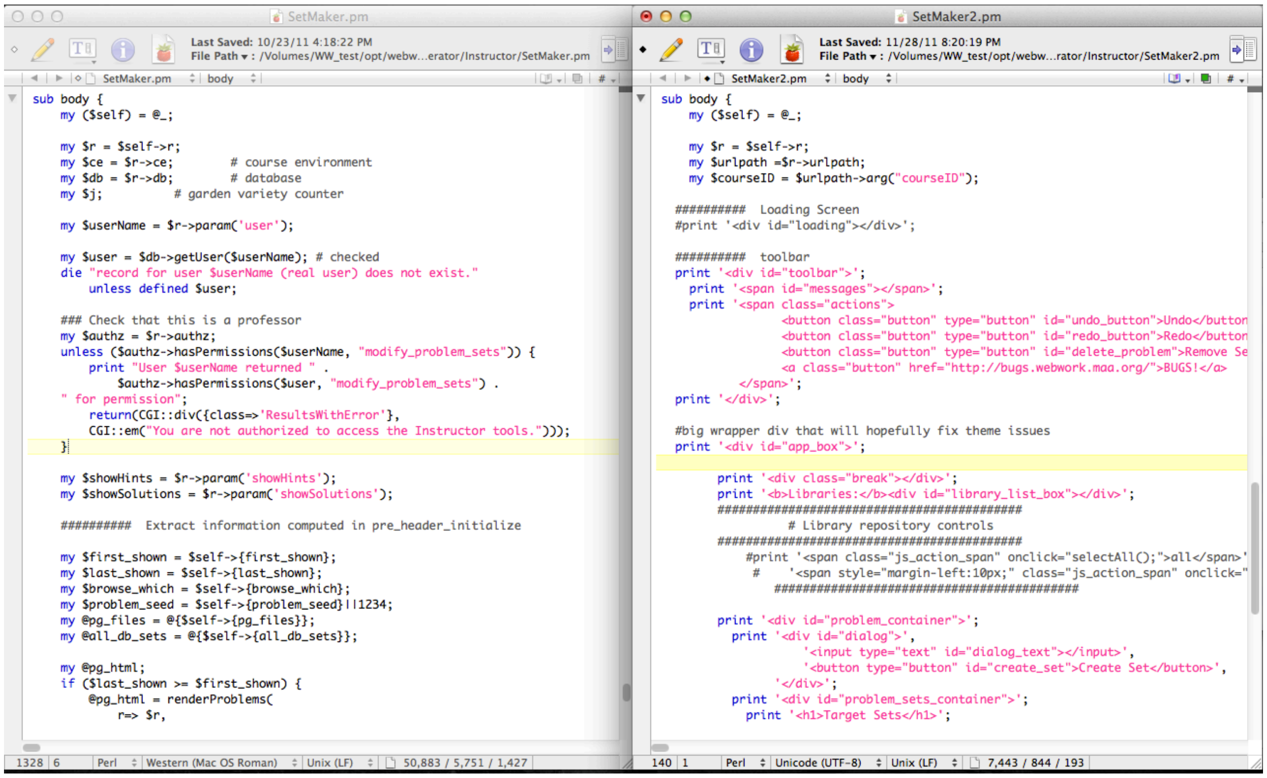


**Figure 3.3: WebService and Content Generator do the same queries but render the information differently**


SetMaker.pm
Last Saved: 10/23/11 4:18:22 PM
File Path ▾ : /Volumes/WW_test/opt/webw...erator/Instructor/SetMaker.pm
SetMaker.pm
body
1328 6
Perl
Western (Mac OS Roman)
Unix (LF)
50,883 / 5,751 / 1,427

```
sub body {
    my ($self) = @_;

    my $r = $self->r;
    my $ce = $r->ce;          # course environment
    my $db = $r->db;          # database
    my $j;            # garden variety counter

    my $userName = $r->param('user');

    my $user = $db->getUser($userName); # checked
    die "record for user $userName (real user) does not exist."
        unless defined $user;

    ### Check that this is a professor
    my $authz = $r->authz;
    unless ($authz->hasPermissions($userName, "modify_problem_sets")) {
        print "User $userName returned " .
            $authz->hasPermissions($user, "modify_problem_sets") .
    " for permission";
        return(CGI::div({class=>'ResultsWithError'},
        CGI::em("You are not authorized to access the Instructor tools.")));
    }

    my $showHints = $r->param('showHints');
    my $showSolutions = $r->param('showSolutions');

    ##########  Extract information computed in pre_header_initialize

    my $first_shown = $self->{first_shown};
    my $last_shown = $self->{last_shown};
    my $browse_which = $self->{browse_which};
    my $problem_seed = $self->{problem_seed}||1234;
    my @pg_files = @{$self->{pg_files}};
    my @all_db_sets = @{$self->{all_db_sets}};

    my @pg_html;
    if ($last_shown >= $first_shown) {
        @pg_html = renderProblems(
            r=> $r,
```

SetMaker2.pm
Last Saved: 11/28/11 8:20:19 PM
File Path ▾ : /Volumes/WW_test/opt/webw...rator/Instructor/SetMaker2.pm
SetMaker2.pm
body
140 1
Perl
Unicode (UTF-8)
Unix (LF)
7,443 / 844 / 193

```
sub body {
    my ($self) = @_;

    my $r = $self->r;
    my $urlpath =$r->urlpath;
    my $courseID = $urlpath->arg("courseID");

  ##########  Loading Screen
  #print '<div id="loading"></div>';

  ##########  toolbar
  print '<div id="toolbar">';
    print '<span id="messages"></span>';
    print '<span class="actions">
                <button class="button" type="button" id="undo_button">Undo</button
                <button class="button" type="button" id="redo_button">Redo</button
                <button class="button" type="button" id="delete_problem">Remove Se
                <a class="button" href="http://bugs.webwork.maa.org/">BUGS!</a>
          </span>';
  print '</div>';

  #big wrapper div that will hopefully fix theme issues
  print '<div id="app_box">';

        print '<div class="break"></div>';
        print '<b>Libraries:</b><div id="library_list_box"></div>';
        ###########################################
                  # Library repository controls
        ###########################################
            #print '<span class="js_action_span" onclick="selectAll();">all</span>'
             #    '<span style="margin-left:10px;" class="js_action_span" onclick="
                ###########################################

        print '<div id="problem_container">';
          print '<div id="dialog">',
                    '<input type="text" id="dialog_text"></input>',
                    '<button type="button" id="create_set">Create Set</button>',
                '</div>';
          print '<div id="problem_sets_container">';
            print '<h1>Target Sets</h1>';
```

**Figure 3.4: Example duplicate code**

The database also has an issue: A user in the database cannot be shared between courses. For example, if a student is enrolled in several courses in the same WeBWorK server, then the database must contain one user entry for each course (which may or may not have the same username and password). It is desirable for a user to belong to several courses, therefore a change in the database structure is needed.

## 4. CONCEPTUAL ARCHITECTURE

With the current architecture of WeBWorK, developers often have to make changes to several parts of the code to make a single change in functionality. An ideal architecture would avoid this inconvenience and allow developers to make changes to underlying functionality without affecting the flow of the user interface. Therefore we chose to base our reference architecture on a model-view-controller pattern. We will now describe the reference architecture that we propose and how it differs from the concrete architecture.

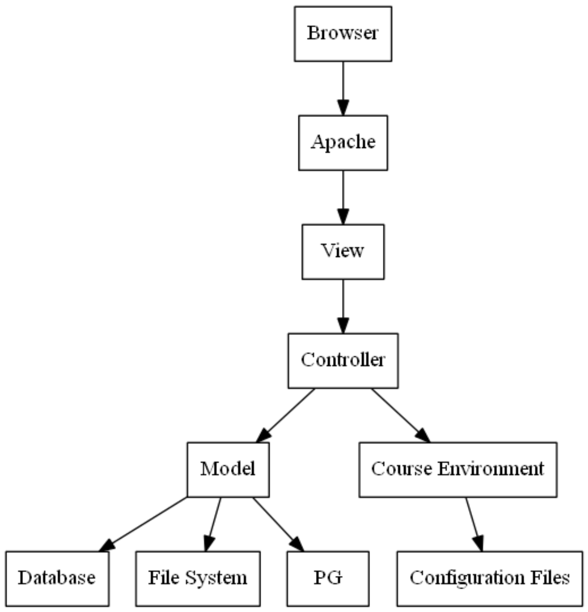


**Figure 4.1: Diagram of an ideal conceptual architecture**

### 4.1 Database

The database will have five tables which should neatly allow the storing of all necessary information. The tables are: courses, problem sets, problems, users, and problem records. The tables and their relations are shown in figure 4.1.

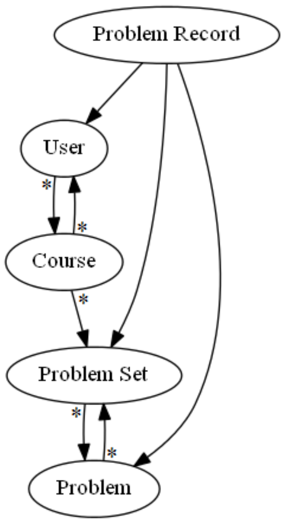


**Figure 4.2: Database diagram for the reference architecture**

*4.1.1 Courses, Users, Problem Sets and Problems*
A course can contain many problem sets. A user can be related (assigned) to many courses, and many users can be assigned to the same course.This allows for a user to login to a WeBWorK server and access several courses related to him/her with the same login information. A problem set can contain many problems, and those problems in turn can be assigned to many problem sets. This way a homework question can be duplicated in several problem sets.

*4.1.2 Problem Record*
A problem record represents an answer to a problem. Each problem record is related to a user (student), a problem set, and a problem. Entries in this table answer the question "What did this student answer for this problem in this problem set?".

This new structure for the database should be very convenient for the developers of WeBWorK to make all the queries that they need to have.

## 4.2 Model

The model is an abstraction layer for the database and filesystem. It provides an interface to the controller so that the controller does not need awareness of the details of the database and filesystem.

## 4.3 Controller

The controller component will receive information from the model component, and return information in the form of variables that are necessary for the view component. Since the information returned will not be in a specific format such as HTML, this should remove the necessity of the duplicate code in Content Generator and Web Services.

## 4.3 View

The view component will send requests to the controller, so that the controller can return information. It then renders the information in a specific format in order to return it to the web browser. The view component can receive the same information from the controller and render it in many different formats.

## 4.4 Differences From The Concrete Architecture

Where the concrete architecture had a Content Generator component and a Web Services component, the ideal architecture would replace these two components with the model, controller and view components.

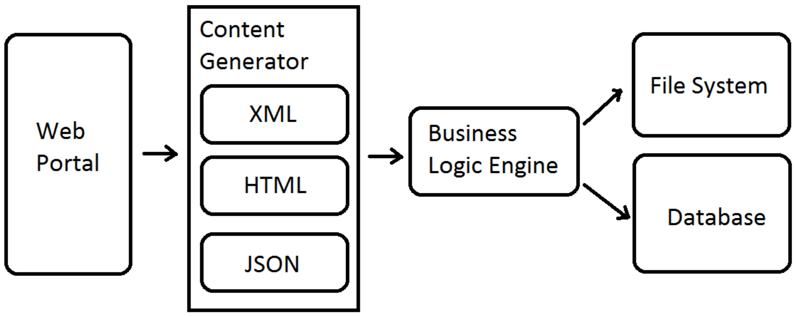


**Figure 4.3: Architecture with querying functionality isolated and consolidated**

## 5. FUTURE WORK

According to Jacobson et al, in order to continue the re-engineering plan, we need to propose a plan for changes and outline how future engineering fits into the proposed conceptual architecture.

## 5.1 Plans to Change

We propose some outlines as to how to evolve the current codebase into the proposed conceptual architecture. We will also show how to gradually change the WeBWorK code so the architecture morphs from the current concrete architecture to our proposed architecture. In order to start this change it should be possible to begin on the front end. This means that the users will see improvements even before the administrators do. The javascript working with the current web service will need to be built carefully, with the understanding that the back end will be changing to a more traditional model view controller set up. In particular there should be as little hard coding and as much modularization in the javascript as possible. For instance any code related to populating a model's information from the server needs to be separate from every other function of that model. This will provide a layer of abstraction that will allow us to change only that function should some fundamental change be required in the way model information is served.

### 5.2 Future Engineering

The move to asynchronous, dynamic pages has been started with Library Browser 2. However this update resulted in duplicate code and was a large project. If the architecture is changed to resemble what we have proposed in this paper, then updating other WeBWorK pages will be simplified significantly. Rather than having to create duplicate code for each updated page, we will simply be able to write a function that finds the necessary data for that page, and a view for each return type that we want to support which renders that data.

## 6. CONCLUSION

We have described a user's point of view of the WeBWorK software, then presented its concrete software architecture, and we have proposed a reference architecture, along with an action plan to migrate WeBWorK's code to the proposed architecture. We expect that this reference architecture will help the developers of WeBWorK to refactor the code to make it significantly more extendable and easy to understand, so that future development of the software will be more efficient and the software can quickly adapt to changing web standards.

## REFERENCES

[1] Bass, L., Clements, P., Kazman, R. Software Architecture in Practice. *Addison-Wesley Longman Publishing Co., Inc.* Boston, MA, USA, 2003.

[2] Jacobson, I., Lindström, F. Re-engineering of Old Systems to an Object-oriented Architecture. *In ACM,* New York, NY, USA. 1991.

[3] Kazman, R., Carriere, S. J., Woods, S. G. Toward a Discipline of Scenario-based Architectural Engineering. *Annals of Software Engineering, Volume 9, Numbers 1-4, 5-33.*

[4] Kazman, R., Abowd, G., Bass, L., Clements, P. Scenario-based Analysis of Software Architecture. *Software, IEEE, Vol. 13, Issue. 6, August 2002.* Waterloo, ON, Canada.

[5] WeBWorK website. http://webwork.maa.org/