Lightweight Web Application Framework and Its Application
Helping Improve Community Bus Timetables after Japan Earthquake

Yu Kitano[1], Hiroki Suguri[2] and Atsushi Togashi[2]
[1]*Graduate School of Project Design, Miyagi University, 1-1 Gakuen, Taiwa-cho, Kurokawa-Gun, Miyagi, Japan*
[2]*School of Project Design, Miyagi University, 1-1 Gakuen, Taiwa-cho, Kurokawa-Gun, Miyagi, Japan*

Keywords: Web Application Framework, Lightweight Methodology.

Abstract: We have identified four typical problems in building web applications: High cost of education for engineers; difficulty of distributed development; scope creep; and coupling of UI and business logic. To solve these problems, the authors developed lightweight methodology and Perl-based framework for web applications. In this paper, we discuss effectiveness of our approach compared with existing tools. We have successfully applied these techniques to development of web application for helping improve community bus timetables, which illustrates real-world usefulness of the methodology and framework.

1 INTRODUCTION

Northeast region of Japan was severely hit by great earthquake on 11th March, 2011. Yamamoto City in Miyagi Prefecture lost two train stations, which were wiped out by tsunami. Nuclear disaster made it impossible to reconstruct the railroads and stations. People who lost their homes were forced to move to temporary dwellings scattered high above hills to keep away from possible another tsunami. Although bus routes were restored in downtown, not all hills could be covered by existing routes and timetables.

Therefore, Yamamoto City rescheduled bus routes by hand to encompass all habitation areas. However, the new routes were not welcomed by citizens because bus stops were placed inconveniently and timetables failed to match their needs. To solve these problems, we have collaborated with Yamamoto City to develop and deploy bus boarding and alighting information system. By using the system, we were able to comprehend the bus usage situation and to propose improved routes and timetables.

The system consists of information gathering subsystem and information presentation subsystem. The information gathering subsystem is an iPhone application that collects location, time, and number of people for boarding and alighting. The data are sent to server and aggregated there. The presentation subsystem is a web application that displays data analysis report to decision makers who revise bus routes, and to citizens who claim improvement of the routes.

Web application is the most popular client-server model information system today. Standard web browser is used as the client, which means no additional component is required to install on the client. Therefore, both decision makers in the office and citizens in their homes can make use of the application via the Internet. Consensus between provider and users of community bus is very important.

In this paper, we focus on the web application, the framework we developed to build web applications, and methodology behind the framework. Section 2 discusses problems in developing web applications. Section 3 details out proposed methodology and framework. In section 4, the application is described. We compare our approach with Ruby on Rails in Section 5. Section 6 concludes the paper by summarizing the outcome and outlining the future work.

2 PROBLEMS IN DEVELOPING WEB APPLICATIONS

Generally speaking, following four challenges must be overcome for effective development of web applications.

2.1 Learning Cost

Learning cost is expensive because wide range of exp-

DOI: 10.5220/0003996402520257
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 252-257
ISBN: 978-989-8565-11-2

ertise is necessary. In the case of developing web applications in Java, it is indispensable to understand HTML, JavaScript, web server administration, and JSP/Servlet in addition to Java language itself. Session management and security issues specific to web application also increase learning cost. Usually, knowledge about frameworks such as Struts and Hibernate is additionally required.

A framework is a set of functionalities commonly required to build applications. Many frameworks arrange large number of components to build feature-rich applications. However, the initial cost of learning to start using such heavyweight framework is huge.

2.2 Distribution and Integration

Division of work and integration of system is necessary in developing web applications. Multiple technologies are involved such as server-side programming, database middleware, and client-side design. Each technology is complicated by itself, and integration is much more difficult.

MVC model is a traditional approach to logically divide the system into three components of Model, View, and Controller. Each component is assigned a specific task and communication between the components drives the system. MVC model is popular in modeling and developing web applications (Mitsuda and Fukuyasu, 2010). However, the model must be fixed at first. Modeling is an intensive and time-consuming task that becomes bottleneck early in the development process before distributing the work to designers and programmers.

2.3 Requirement Changes

System requirements are getting more and more complicated today. Since web is bleeding edge of business, it is common to begin developing systems without fixing the requirements. Even in the middle of development, requirements keep changing and the system must quickly embrace the changes. Traditional waterfall project management is no longer effective in such scope creep situations.

Many web applications are developed using application framework that is based on MVC model, such as Struts. It is true that MVC model is said to resilient to change. However, it is only true when the model is kept intact. If the model is forced to change due to requirement changes, system-wide modification is likely to occur.

2.4 Coupling of User Interface and Business Logic

Web application is divided into HTML page design rendered in client, and background business logic programmed in server side. Design of user interface and program of business logic must be strictly distinguished, allowing designer and programmer to work separately. However, in many existing web application frameworks, programming logic such as field repetition and input validation must be coded in page layout along with HTML. Therefore, changing user interface design requires programmer's involvement as well as UI designer's work.

3 PROPOSED LIGHTWEIGHT METHODOLOGY AND FRAMEWORK

This section explains proposed lightweight methodology and framework to solve the four problems discussed in the previous section. The approach is characterized by the following four advantages: (1) Low learning cost; (2) Easy distribution and integration of work; (3) Robustness and flexibility to requirement changes; and (4) Clear decoupling of view and logic.

The proposed lightweight methodology is targeted for small- and medium-sized applications. Suggested number of pages is less than thirty. With such applications, large framework such as J2EE is unnecessary. Larger systems can be built by integrating such medium-sized applications.

3.1 Lightweight Methodology

3.1.1 Low Learning Cost

Leaning cost is increased by complexity of the programming language and complexity of the development environment. For the programming language, we selected Perl. Perl is one of the most popular lightweight programming languages along with Ruby, Python, and PHP. Learning Perl is much easier than learning Java.

For the programming environment, we wanted to avoid using heavyweight IDE such as Eclipse and NetBeans. These tools are difficult to learn. Difference between local debugging environment and remote deployment environment is also problematic. Instead, we incorporated browser-based development environment (i.e, file browser, text editor, and debugger) inside the framework, which handles server files directly.

3.1.2 Distribution and Integration

Web application is divided into two main components. One is client page design comprising HTML, CSS, JavaScript, and media files. The other is server program and database. The client and the server can be developed asynchronously. In many cases, end users want to see client UI before server programming has been completed. To meet these requirements, our lightweight methodology abstracts web application as a set of page transitions. In our approach, MVC is redefined as follows: Model is a program that corresponds to a page, View is HTML template, and Controller is the web application framework itself. We call it lightweight MVC model.

With the lightweight MVC model and its implementation in our framework, page transitions can be performed on the client without coding the server-side business logic. Of course, server-side programming is necessary to complete the application. However, page design and transitions can be demonstrated to users at early stage of the development before fixing model (server program). In addition, scope of responsibilities is made clear by assigning each programmer a set of pages and programs that correspond to the pages.

3.1.3 Robustness and Flexibility

In our lightweight MVC model, responsibility for each page is clearly defined. If the user requests specification changes, only affected page design (view) and program (model) must be changed. Other views and models, along with controller (the framework itself) are kept intact. Therefore, our methodology is flexible while maintaining robustness.

3.1.4 Decoupling of View and Logic

Most web application frameworks provide HTML template that displays run-time value of variables embedded in page layout. On the other hand, it is also common to use page design tools such as Adobe Dreamweaver to visually design user interface. Thus, page designers do not have to understand the syntax of underlying HTML and CSS. Now, the problem is conflict between HTML template and design tools. Each template engine has its own syntax to embed values, repetitions and control flows, which often cannot be properly handled by design tools.

Therefore, we came up with minimum extension to HTML for our template language that does not conflict with design tools. In addition, Perl code can be executed on the template on the server side.

3.2 Lightweight Framework

Figure 1 depicts the structure of lightweight framework. As illustrated in Figure 2, the framework consists of three Perl modules.

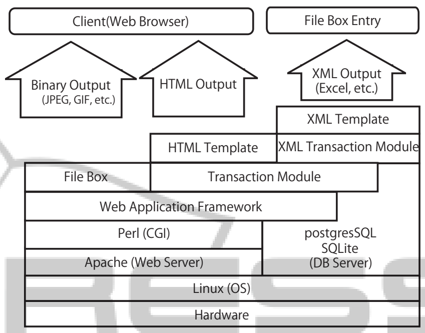

Figure 1: Structure of lightweight framework.

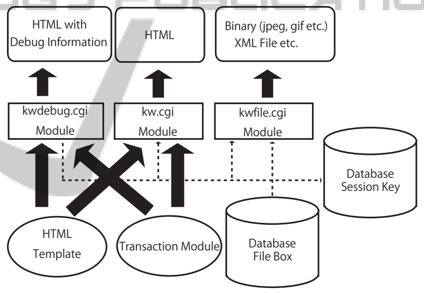

Figure 2: Perl modules.

The module kw.cgi is the core of the framework. It calls back HTML template and transaction module supplied by user, then generates HTML output. The module kwfile.cgi is responsible for binary file output from file box. The module kwdebug.cgi is a superset of kw.cgi for debugging purposes. In addition to all functionalities of kw.cgi, kwdebug.cgi can generate debugging information to browser. It also provides simple text editor of server files to browser client.

HTML template embeds data items in page layout. The data items are defined in the transaction module. In the HTML template, designer can use special tags to be replaced with run-time data, which are specified by hash variable %kw::out_param in transaction module. Example will be shown in Section 5.

In MVC terms, kw.cgi corresponds to Controller, HTML template corresponds to View, and transaction module corresponds to Model. Page transition of the web application controlled by kw.cgi is handled by HTML form parameters sm and ss. The parameter sm specifies transaction module that is executed after page transition. The parameter ss designates HTML template used by the transaction module. If ss is null, default HTML template is used, which is specified in the transaction module. In other words, transaction module run after pressing submit button is determined by the value of sm, while HTML template is chosen by the values of sm and ss.

If transaction module is empty, empty application is generated that only performs page transitions defined by HTML template. This is useful for checking the page transitions before getting to the programming. It is recommended that HTML template (View) is designed and implemented at first, before transaction module (Model) is designed and implemented.

It is common that web application must deal with image files, PDF files and Microsoft Office documents. To output these non-HTML files, file box may contain arbitrary binary files.

Transaction module may register non-HTML files with the file box. These files are sent as a part of HTTP response by specifying the file identifiers in kwfile.cgi. Especially, XML files can be generated and registered as easily as HTML templates. This is useful when the web application produces XML data files that are used by client applications such as Microsoft Excel. An example will be presented in the next section.

Debugging module kwdebug.cgi displays runtime values of variables. It also offers Adobe Flash-based text editor for client to edit server files without using local editor and file transfer tools over ssh or scp. This simplifies the development of web application. Especially in Japan, character code set on server (UTF-8) is different from that of Windows client (Shift JIS). Programmers must always be very careful to convert the character set properly, which is heavy stress. We solved the problem by offering server-based text editor.

In addition, the framework provides miscellaneous functions such as numeric formatting and Japanese Imperial Calendar processing.

4 BUS BOARDING AND ALIGHTING APPLICATION

Figure 3 shows page transition of bus boarding and alighting information system. The application starts from user authentication page, where the user enters user name and password. Global navigation menu branches to boarding and alighting summary page, data analysis page, and map visualization page. The collected data from information gathering subsystem can be summarized in tabular format in web browser, analyzed and graphed in Microsoft Excel via XML intermediate file, and plotted on Google Maps using JavaScript.

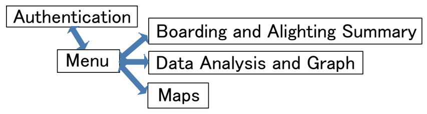

Figure 3: Page transition of the bus application.

During the development of the application, the requirements from users kept changing. Specifications revised repeatedly and new features added frequently. Roughly speaking, we took three steps to develop the application using our lightweight methodology and framework.

In the first step, we started from building empty application that only performs page transitions defined by HTML templates. Then tasks were divided into page design, programming of server-side logic, and design of Microsoft Excel output. These tasks ran concurrently. We used XML template, which is similar to HTML template, to develop XML output to Excel. The generated XML file is registered with file box so that web browser can download the file, which is then passed to Microsoft Excel.

In the second step, we plotted number of people boarded and alighted on Google Maps. Complicated JavaScript programming was necessary to use Google Maps API. Thanks to our framework, JavaScript programming was separated from HTML page design. A skilled JavaScript programmer, who happened to be very bad in user interface design, concentrated on the programming.

In the third step, we refactored many parts of the application to implement user requests and specification changes. When a set of data is analyzed and presented, users requested to add different features and to change existing functionalities. Since no heavyweight MVC structure was adopted, we successfully modified code and user interface without major problems. No “pseudo-MVC” problem hindered the refactoring process.

We are planning to add new features to better address the needs of the customers. For example, gathering and visualizing accelerometer data from iPhone is requested.

5 COMPARISON WITH RUBY ON RAILS

Ruby on Rails (Hansson, 2004) is a popular lightweight web application framework written in

Ruby language. Traditional frameworks such as Struts require massive amount of configuration files, which leads to low productivity and low quality. On the other hand, Ruby on Rails emphasizes "convention over configuration," reducing the need of configuration files and increasing productivity. For example, ActiveRecord O/R mapper generates appropriate class from database schema definition. Ruby on Rails allows easy development and deployment of web application based on MVC model.

In this chapter, we compare our lightweight framework with Ruby on Rails in terms of the four problems identified in Section 2: Learning cost; distribution and integration; requirement changes; and coupling of user interface and business logic.

5.1 Learning Cost

With our lightweight application framework, no programming knowledge is required for page designers. Our lightweight methodology centers on page transitions, with each page corresponding to its database table and business logic. Therefore, heavyweight O/R mapper such as ActiveRecord is not needed. Heavyweight IDE like Eclipse or NetBeans is also unnecessary because editing and debugging support is provided by the lightweight framework itself. As a result, learning cost of our framework is cheaper than that of Ruby on Rails.

5.2 Distribution and Integration

It is mandatory to clearly define the scope of responsibilities of each member in web application development project. With our approach, the scope is defined by page. Once page transitions are fixed, each member clearly understands what to do. On the other hand, modeling is essential prerequisite on Ruby on Rails, which is based on MVC model. Scope of responsibilities are made clear only after the model has been developed and entire class relationship has been fixed. This is typical bottleneck of Ruby on Rails project. Moreover, when it becomes necessary to refactor the implementation of the model and controller, the cost is quite expensive.

5.3 Requirement Changes

As discussed in Section 2, requirement change or scope creep is major reason why it is difficult to maintain MVC structure in the course of web application development project. Ruby on Rails enforces strict MVC model by "convention over configuration." Resulting database schema and page transitions must conform to the model. If requirement specifications are clear and stable from the beginning of the project to the end, high efficiency and quality can be expected.

However, in a project where MVC structure cannot be distinctively defined at the beginning, or where requirement changes occur frequently, it is common that Model, View and Controller are getting mixed during the course of the development. Since Ruby on Rails requires code that conforms to the framework convention and MVC structure, major refactoring must be performed when requirements change. In reality, many projects circumvent the major refactoring and tend to do quick and dirty fix of code, resulting in the deterioration of clean MVC structure. This is sometimes called "pseudo-MVC."

On the other hand, our lightweight methodology abstracts web application as a set of page transitions. In our lightweight framework, MVC is redefined as follows: Model is transaction module, View is HTML template, and Controller is the framework itself. We call it lightweight MVC model. With lightweight MVC model, scope of responsibilities is made clear by assigning each programmer a set of pages. In addition, lightweight MVC model does not prevent developers from using common libraries or object-oriented data model.

5.4 Coupling of User Interface and Business Logic

Now we compare our approach with Ruby on Rails in terms of how much programmer has to get involved in page design. On Ruby on Rails, page template is .rhtml file, which embeds Ruby constructs into HTML. When page designer edits the .rhtml file with design tool such as Dreamweaver, the designer is required to understand the Ruby language and .rhtml tags. For example, Figure 4 is .rhtml form that displays patient name, and accepts entry of birthday.

Figure 5 shows Dreamweaver screen editing the .rhtml file. Even in this simple example, Dreamweaver is unable to render the form correctly because it does not understand .rhtml tags.

On the other hand, using our lightweight framework in Perl language, Figures 6 and 7 show the form code and Dreamweaver screen respectively. Dreamweaver appropriately renders the form because unknown tags or tags in HTML comments are not used.

Since we do not introduce new tags in our HTML template, page designer can use existing tools like Dreamweaver. In addition, no knowledge of Perl is required for the designer.

```
<body>
Edit birthday
<%form_tag :action => 'edit',
 :id => @patient do %>
<%=render :partial => 'form' %>
<br>
<%=link_to 'Patient  List',
:controller => 'patient_id' ,
:action => 'patientList' ,
:id => @birthday %>
<br><br>
<%=submit_tag 'Edit' %>
<%end %>
</body>
```

Figure 4: Example of .rhtml form.

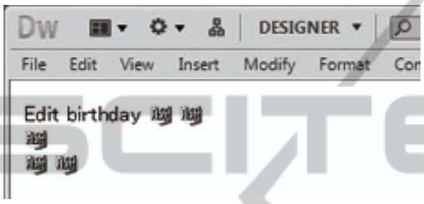

Figure 5: Display of .rhtml in Dreamweaver.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed lightweight methodology and lightweight framework for developing web applications. To develop web applications, orthodox method is to employ highly skilled engineers and to adopt heavyweight methodology, framework and IDE. However, another approach is to use lightweight methodology and lightweight framework with unskilled engineers. We have proven that web application can be developed with limited financial and human resources. The outcome of the research is summarized to the following four points.

(1) The lightweight methodology and framework enabled cost-effective development of web application. Compared to Java-based framework and Ruby on Rails, learning cost is cheap. Even unskilled programmers could take part in the development project of bus information system.

(2) We made distribution and integration of workload effectively by defining scope of responsibilities for each page. Developers and designers cooperated smoothly to finish the system in short time.

(3) The proposed framework allowed rapid modification of the system to satisfy requirement changes from users.

(4) The framework decoupled page design, application logic, and data model thoroughly. Our lightweight MVC model separates View and Model distinctively, and framework itself becomes Controller. The application is resilient to requirement changes.

After successful development and deployment of applications based on the framework, we also identified two issues requiring additional work. The first thing to do is to incorporate test tools in the framework. In unit testing framework such as PerlUnit, dedicated testing program must be coded. However, when specifications change, the testing program must be changed, too. This is not productive. We are planning to develop unit testing library as a part of our framework to solve this problem.

Second issue is version control. Tools like CVS, Subversion, and Git are popular among skilled developers. However, unskilled programmers are not familiar with the notion and operation of version control. In many cases, simple versioning such as those found in Microsoft Office is sufficient for small- and medium-sized project. We are designing easy-to-use versioning tools to incorporate to our lightweight framework.

```
<body>
Edit birthday
<form>
<input type=hidden name=sm
 value="patientlist" >
<input type=hidden name=ss value="" >
$˜name˜$
<input name=birthdayvalue=$˜birthday˜$ >
<input type="submit" value="Edit">
</form>
</body>
```

Figure 6: Example of our HTML template.

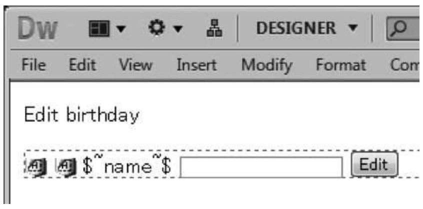

Figure 7: Display of our HTML template in Dreamweaver.

REFERENCES

Hansson, D. H. (2004). Ruby on rails. *http://rubyonrails.org/.*

Mitsuda, N. and Fukuyasu, N. (2010). Issues behind the use of web application frameworks. *Computer Software*, 27(3):3 2–3 12.