

HTML Templates that Fly

A Template Engine Approach to Automated Offloading from Server to Client

Michiaki Tatsubori

Toyotaro Suzumura

IBM Research, Tokyo Research Laboratory
1623-14 Shimo-tsuruma, Yamato, Kanagawa, Japan
{mich,toyo}@jp.ibm.com

ABSTRACT

Web applications often use HTML templates to separate the webpage presentation from its underlying business logic and objects. This is now the de facto standard programming model for Web application development. This paper proposes a novel implementation for existing server-side template engines, FlyingTemplate, for (a) reduced bandwidth consumption in Web application servers, and (b) off-loading HTML generation tasks to Web clients. Instead of producing a fully-generated HTML page, the proposed template engine produces a skeletal script which includes only the dynamic values of the template parameters and the bootstrap code that runs on a Web browser at the client side. It retrieves a client-side template engine and the payload templates separately. With the goals of efficiency, implementation transparency, security, and standards compliance in mind, we developed FlyingTemplate with two design principles: effective browser cache usage, and reasonable compromises which restrict the template usage patterns and relax the security policies slightly but in a controllable way. This approach allows typical template-based Web applications to run effectively with FlyingTemplate. As an experiment, we tested the SPECweb2005 banking application using FlyingTemplate without any other modifications and saw throughput improvements from 1.6x to 2.0x in its best mode. In addition, FlyingTemplate can enforce compliance with a simple security policy, thus addressing the security problems of client-server partitioning in the Web environment.

Categories and Subject Descriptors

C.2.5 [Local and Wide-Area Networks]: Internet; D.2.3 [Software Engineering]: Coding Tools and Techniques; D.3.2 [Programming Languages]: Specialized Application Languages

General Terms

Design Languages Performance Security

Keywords

Template engines, Client-server partitioning, Web applications

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

1. INTRODUCTION

Web applications often implement the advantageous model-view-controller architecture [20, 11] by using *HTML templates*, often seeking to separate the webpage presentation from the business logic and objects. This is a mantra for experienced Web application developers conforming to that architecture [20, 4]. It has become a de facto standard programming model for Web application development [10, 5].

This paper proposes a novel implementation of server-side template engines that exploits the nature of the template-based programming model to enhance existing Web applications to boost their server throughput. Instead of producing fully-rendered HTML pages, the proposed template engine produces *skeletal scripts* that run on a Web browser on the client side. This bootstrap code includes only the template parameter values, which change with each request, and then retrieves the template data and the client-side template engine separately. This allows Web browsers to *cache the template data*, which is relatively static. Other skeletal scripts for the dynamic pages based on the same template can then share the cached template.

This architecture contributes to the reduction of server-side load since a server usually needs only to provide the skeletal scripts even though the dynamic data changes for each request. The improved server needs only to serve a client-side template engine for each new user and then a payload template for each newly visited type of page using the same template. These transmissions may also happen when a client-side cache is either stale or has a cache miss (because the website has evolved, or because files have been flushed out of the cache). Advantageously, the template engines and templates are reduced to static files on the server's file system, which makes it easier for a Web server to serve them compared to dynamically generating large and complex pages.

The differences in the implementations are almost *transparent* and most typical Web applications should run without any modifications except for replacement of the template engine. In fact, we made our prototype emulate the application programming interface of the Smarty template engine library [19] for the PHP language [21, 25], so it can replace the original Smarty libraries used in the SPECweb2005 application that we used for our performance tests. This simple method also works for at least some other Web applications such as SugarCRM.

While naive automatic client-server partitioning exposes *security vulnerabilities*, our implementation addresses these problems by using secure loading to the clients. This in-

volves the automatic enforcement of a simple security policy controlled by the administrator of the Web application. Security is always of concern in publicly available Web applications [29]. Typical automated client-server partitioning technologies such as Hilda [28] have ignored the security problems caused by porting some parts of the server-side logic of a Web application to the untrusted clients.

The contributions of the paper are:

- a proposal for an template engine approach to an automated client-server partitioning system compatible with the existing Web architecture
- a design and implementation with efficient cache usage and security
- experimental results for the proposed template engine with an application in SPECweb2005, which is an industry standard benchmark
- potential promotion of template-based Web application development with the additional advantage of automatic performance gains from using the proposed template engine

The rest of the paper is organized as follows: In Section 2 we motivate our proposal by discussing the programming model and implementation of a reference template engine, used as a representative of current template engines. Section 3 introduces FlyingTemplate, our proposal for the design of a novel server-side template engine. In Sections 4 and 5, we discuss the important implementation issues affecting efficiency and security. We report on our experimental results in Section 6. After discussing related work in Section 7, we conclude the paper in Section 8.

2. TEMPLATE-BASED PROGRAMMING

Template-based Web programming is popular mainly because it separates the page representation, the “views”, from the business logic and data of a program, the “controls and models”. Such templates are available as software libraries [20, 11], as programming or modeling language features [5, 2] or as Web application frameworks [10, 4]. The benefits include encapsulating the look and feel of a website, clearly described views, a better division of labor between graphics designers and coders, component reuse for view designs, unified control over the evolution of the appearance, better maintainability of the runtime, interchangeable view artifacts for different development projects, and security compatible with end-user customizability [20].

2.1 Template Usage

In general, there are three steps in using a template engine:

- Specifying a template to use,
- Assigning values to the parameters as the actual content, and
- Filling in the template with the assigned values to obtain the HTML results.

Though the application programming interfaces may vary for each template engine, they are essentially built around these steps.

Account	Type	Current Balance	Total Deposits	Average Deposit	Total Withdrawals	Average Withdrawal
000002006	Other	7016.06	16.06	16.06	96.06	96.06
000002007	Checking	7016.06	116.06	16.06	136.06	96.06
000002008	Saving	7016.06	216.06	16.06	186.06	96.06
000002009	Other	7016.06	316.06	16.06	236.06	96.06
000002010	Other	7016.06	416.06	16.06	286.06	96.06
000002011	Other	7016.06	516.06	16.06	336.06	96.06
000002012	Other	7016.06	616.06	16.06	386.06	96.06

```

check_login();
$smarty=new SmartyBank;

$smarty=backend_get($_SESSION['userid']);

$smarty->assign('userid', $_SESSION['userid']);
$smarty->assign('summary', $summary);
$smarty->display('account_summary.tpl');

```

Figure 1: A final end-user view and a simplified PHP script using a template engine.

In this paper, we use the Smarty template engine library [19] for the PHP language [21, 25] as the reference implementation of a template engine. This library is used in many production-quality open source Web applications such as XOOPS and SugarCRM. It is even used in SPECweb2005 [22, 26], the industry-standard Web server benchmark kit, with available implementations for both PHP and Java. We used the release version 2.6.7 of Smarty, which is distributed with the SPECweb2005 environment that we used for our experiments.

Figure 1 includes a simple PHP script written using Smarty, extracted from SPECweb2005 and greatly simplified for this explanation. This code first authenticates and authorizes the user by calling a user function `check_login()`. Then it instantiates a template engine object (from the class `SmartyBank`) and assigns that reference to the variable `$smarty`. Next it calls the user function `backend_get()` to access the backend database for the required user data. The `$_SESSION` is a globally accessible associative array variable that holds the session data for the user. Here, the `userid` is a key associated with a user identifier number. The resulting data is stored into the variable `$summary`. Finally, it renders the output result based for the user ID with the data obtained from the backend using the template. The PHP runtime passes the output result to the frontend Web server to be included in its HTTP response.

In this example, the usage process of the Smarty template engine essentially consists of the calls to two methods, `assign()` and `display()`. The `assign()` method is called to assign new values to the parameters, while the `display()` method is called both to specify a template and to fill it with the assigned values. For the example in Figure 1, the template parameter `'userid'` is bound to the user ID while the other parameter `'summary'` is bound to the results constructed from the data obtained from the backend database.

