NitroScript: A PHP Template Engine For Customizing of E-Commerce Applications

Jerry Walsh  
NitroSell Ltd  
Cork, Ireland

Donogh Roche  
NitroSell Ltd  
Cork, Ireland

Franclim Foping  
Cork Constraint Computation Centre  
University College Cork  
Cork, Ireland  
Email: f.foping@4c.ucc.ie

Abstract—Customizing Web pages in a shared operating environment is a challenging task since users have different needs and expectations. One way to facilitate the customization process of Web page is to provide to users with a template engine. By using a template engine, the user can specify variability points in the target document that will be replaced with the appropriate piece of information at runtime. This paper presents NitroScript: a new PHP template engine. NitroScript is designed to be used in e-commerce thereby differing from other general purposes template engines such as Smarty. NitroScript allows retailers to easily customize their product Web pages. The code generated by the NitroScript compiler is a PHP bytecode which results in a performance boost.

Keywords—PHP Template engines, Web applications, E-Commerce, NitroScript

I. INTRODUCTION

The model-view-controller architecture is often implemented in Web applications through the use of HTML templates. The ultimate goal of these templates is to dissociate the presentation of the Web page from its business logic. The increasing popularity of this paradigm has paved the way for the model-view-controller architecture to become the de facto standard programming model for Web application development [1].

This paper presents NitroScript, a bespoke template engine,

A. Motivating Example

NitroSell is a company that provides integrated E-commerce solutions to independent retailers. Its mission statement is to make it easy for retailers to increase profits by serving new and existing customers online. In order to do so, NitroSell offers a Software as a Service (SaaS) application that hides complexity, minimizes costs, and provides a highly productive and efficient operating environment. In 2011, NitroSell registered over 700 Web stores with each having about 40 templates. Still in the same year, 700 million page views were recorded1. The SaaS application makes use of a multi-tenant database in order to leverage the economy of scale. This means that all the Web stores are consolidated onto the same database.

Retailers are constantly facing the conundrum of customizing their product pages as well as any other information in order to make it easy for their customers to navigate through their Web stores and locate the products they are looking for. Furthermore, the provider of the SaaS application (NitroSell) ought to provide to their customers (retailers) an interface, simple and easy to learn, to their customers. This highlights the complexity of dealing with the customization of each Web store.

Furthermore, another consideration is that due to the fact that all customers shared the same instance of the database, care should be taken to ensure that customers’ data is stored securely. Indeed, the key question being raised here is related to the way a Web store can be customized without compromising another Web store’s private data in a multi-tenant system. As defined in [2], a multi-tenant Web application is an application that enables different customer organizations (also known as tenants) to use the same instantiation of a system, without necessarily sharing data or functionality with other tenants. The central research question being investigated in this paper can be formulated as follows: How can customization and configurability be realized in Multi-tenant web applications? This paper answers that question by presenting a PHP template engine that allows retailers to easily customize their Web stores.

B. Paper Contribution

This paper presents NitroScript: a novel template engine written in PHP entirely dedicated to e-commerce. To best of the authors’ knowledge, NitroScript is the first template engine written in PHP for the sole use of e-commerce applications. By focusing solely on e-commerce, NitroScript addresses the challenges faced by e-commerce applications much better than any general purpose template engine would have done. The main goal of NitroScript is to seamlessly facilitate the customization of the Web store by retailers.

C. Paper Structure

The rest of the paper is structured as follows: Section II provides the background related to template-based program-
ming as well as the definition of the concept of template. Section III covers the NitroScript template engine and its main building blocks. Section IV presents the related work to NitroScript, this includes the most relevant PHP template engines identified by the literature review. The paper is concluded in Section V where directions for further work are given.

II. ON TEMPLATE-BASED PROGRAMMING

The rationale of template engines is to separate the application logic and data computations from the display of such data in both thought and mechanism [3]. This can be seen as an application of the model-view-controller paradigm first described in [4] as a pattern which aims at isolating ”domain logic” (the application logic for the user) from the user interface (input and presentation), thereby facilitating independent development, testing and maintenance of each (separation of concerns). The motivation behind the template engine was raised by the following key reasons [3]: Encapsulation; Clarity; Division of workload; Component reuse; Single-point-of-change; Maintenance; Interchangeable views; and Security.

A. The concept of template

In this paper, we followed the formal definitions of template and template language provided in [5]. These are summarized as follows:

**Definition** A template $t^o$ is a sentence from a template language.

**Definition** A slot is an area of variability in a document.

**Definition** A slot markup language $\mathcal{S}$ is a non-empty set of features to denote slots within a document.

**Definition** Let $T_\mathcal{S}$ be the set of terminal symbols from the target language $\mathcal{T}$, let $\mathcal{S}$ be a non-empty slot markup language and let $\text{instantiate}$ be an instantiation function. Then the template language $\mathcal{T}^o$ is the smallest language constructed from the target language and the slot markup language $\mathcal{S}$ such that the following two conditions are met: $\forall t \in \mathcal{T} : t \in \mathcal{T}^o \land \text{instantiate}(d,t) = t$ for arbitrary instantiation data $d$ (target language documents are templates and instantiate into itself) and $\forall n \in \mathbb{N}, l_i \in T_\mathcal{S} \cup \{\epsilon\}, b_i \in \mathcal{S} \cup \{\epsilon\}, 0 \leq i \leq n : l_0 b_0 \ldots l_i b_i l_{i+1} \ldots b_n l_n \in \mathcal{T}^o$ (templates are constructed from target language terminal symbols and slot markup language sentences)

**Definition** The component responsible for the instantiation of templates is referred to as template engine.

The relationship between the various definitions provided above is represented in Fig. 1. The conversion of a document in a target language into a template is achieved by introducing elements from the slot markup language, while the instantiation turns templates into documents in the actual target language.

Conceptually, a template defines an artefact that provides built-in support for variability [6]. Therefore, a template is meant to be instantiated to produce different variants given the incoming parameters. Other concepts defined in template languages are placeholders, conditions, and loops which are described as follows:

- **Placeholders** are used to insert elements originating from parameters upon instantiation of a template. The type of these elements varies as it can either be primitive (a string or an integer) or complex (usually in the form of objects);
- **Conditions** offer to template designers the ability to embed parts of the template only if some boolean constraints on the parameters are satisfied. These conditions can either consist of a single branch (an IfThereAre statement) or a two-branch statement (an if elseif);
- **Loops** (forEach construct) can be used to iterate over collections in the parameter model. The body of the loop is then repeatedly inserted into the template instance. Usually, the current element of the iteration is available inside the loop, for example, by accessing a variable.

III. NitroScript

NitroScript is a server-side template engine that looks much like a regular server-side engine that does the template-filling work on the server-side. In this paper, Smarty is used as the reference template engine and PHP as the underlying programming language. However, the design of NitroScript itself should be applicable to any other template engines and programming languages. NitroSell eCommerce enables its users to fully customize their...
WebStore pages by using the NitroScript. Customers can therefore view individual sections of their pages, which are divided into zones as shown in Fig. 2. A template consists of many subsections such as the Header used to alter the relative positioning of the header images, tabs, navigation panel, search (find) panel, and new products panel; the Footer which is used to change the relative positioning of the footer text, shopping cart panel, login panel, and pre order (promotions) panel; Tabs; and panels including the Find, Login, Cart, and Product.

A. Design Principles and Goals

The key design goals of NitroScript are:

1) Security: Creating a Web page by using NitroScript templates should not introduce extra security vulnerabilities;
2) Compliance to Web standards: The implementation should conform to Web standards;
3) Performance: NitroScript should perform better than existing template engines, at least for e-commerce applications.

Security is an important aspect of e-commerce applications. This is particularly true when dealing with publicly accessible applications in the Web environment, where the client platforms cannot be trusted [7]. Compliance to Web standards allows engines to easily access and index Web pages which are then easier to convert to other formats, and easier to access with program code (like JavaScript). NitroScript needs to conform to Web standards in order to handle most of the current Web architectures and existing Web applications that depend on the standards. This includes formal standards such as the HTTP protocol and the de facto standards such as the major browser implementations. Performance is essential since NitroScript only makes sense if it can improve the Web application compared to its original template engine. NitroScript should not change the visible functionality of a Web application, but enhance the server throughput, which is the main reason to use NitroScript. In order to improve efficiency, the code generated by NitroScript is compiled directly into PHP bytecode. This approach contrasts with the way Smarty as it only interprets the code. As stated in [8], compiling source codes into bytecode significantly improves the performance of the application in two ways:

- Avoiding duplicated lexical and syntactical parsing within a source script, and
- Simplifying the interpretation simplification of operations by localizing data to reduce code fetches.

The main drawback of the bytecode technique is the overhead associated to its generation, especially when the runtime execution flow runs those bytecode instructions only once. In NitroScript, this overhead was not considered.

B. Workflow of NitroScript

The user starts by defining his template which contains one or more slots. These slots are defined by using the NitroScript slot markup language & which consists of constants, operators, and functions available in NitroScript and covered in the next section. The NitroScript engine will at runtime compile the template into PHP code. To illustrate these concepts, let us consider the pseudocode in Section VI-A.

The pseudocode prints the product reviews for a given WebStore. In this example, & consists of the following constants: STORENAME, REVIEW_AGE/RATING, REVIEWS_FOUND_USEFUL, REVIEWS_REVIEWER, REVIEWS_STAR, plus the following conditional operators: ifThereAre, endifIfThereAre, if, endif, forEach, endForEach, ifNone, endifIfNone, the following comparison operator: gt and the following functions: getLanguageString, GetStoreImagePath, nl2br. Upon instantiation, the NitroScript engine will produce the output shown in Section VI-B.

It can be observed that the constant STORENAME was substituted with the following PHP code <\?= htmlspecialchars(’The name of My WebStore’)> ?>. The string was automatically sanitized by passing it to the function htmlspecialchars. In this particular case, NitroScript has used MySQL as the underlying database used by this WebStore. This information is provided during the configuration of the WebStore. Covering this process as well as the logic behind it is outside the scope of this paper. The resulting pseudocode is provided here as a sample output. This code is to be rendered in the Custom Content section of the Webpage.

C. NitroScript Language Constructs

This section summarizes the building blocks of NitroScript. For space constraints, we will only be able to describe the main constructs of NitroScript:

- The product fields: The various WebStore panels make use of different fields. The product fields in the list that follows can be used with any product list variable (such as basketitem, product, relateditems). Product list variables are generally in the format: list['field_name']. For example, when the product_name product field is used in conjunction with the basketitem list variable,
the syntax is: \{basketitem[\"product_name\"]\}. Some of the product fields are product_id, product_code;

- The customer fields: These describe the fields that are used by NitroScript to deal with the customer’s data. NitroScript provides a custom function (getCustomerFields(x), with x being the field of interest) to get the value that these fields may hold. Some examples of these fields are customer_id, customer_username;

- The tab fields: As shown in Fig. 2, NitroScript divides the page into several sections. The tab section is accessed in NitroScript using dedicated fields: id is the ID of the tab, link the link to the tabbed page, and name the name of the tab;

- The Page Property fields: These are global fields can be processed in NitroScript using dedicated fields: id is the ID of the tab, link the link to the tabbed page, and name the name of the tab;

- Details fields: Using these fields, customers can edit e-mail templates, such as Customer Registration, Forgotten Password, and Order Confirmation. These fields are used in conjunction with the \{pageproperty[\"x\"]\} list variable, where x is one of the following fields: pageid, title among others;

- Constants: NitroScript defines a number of constants that are available to the developer such as STORENAME which represents the name of the store;

- Comparison operators: These are used to compare two numbers or two text strings. In text string comparisons, lower-case letters are considered to be greater than upper-case letters. Two strings are considered equal if and only if their case (upper or lower) matches exactly. NitroScript defines six comparison operators: GT, which stands for greater than, LT which stands for less than, EQ standing for equal to, NE which stands for not equal to, GTE standing for greater or equal to, LTE standing for less or equal to;

- Functions: NitroScript defines a number of global functions for various purposes. Some of them are GetStor-eImagePath("filename") which returns the currently selected them image path. This function is particularly helpful in achieving customization. Other functions include basketIsEmpty which tells if the basket is empty or not;

- Special Functions and If Statements: For the following special If statements (for example, \{nsIf:SearchPanelEnabled\}), which are closed with matching tags (for example, \{/nsIf:SearchPanelEnabled\}), if the true condition is returned, the action specified in the enclosed block of code is performed. Some of those functions are \{ns:pageTitle\} which prints or generates the title of the Web page used at the header section of the page, and \{ns:PanelPromotionDisplay\} which prints the contents of the Promotions (Pre Order) panel used at the footer section of the page.

D. WebStore Customization

Customization of a product or service means creating or converting a product or service according to the buyers specifications or prior behaviour [9]. This concept is gaining momentum in the e-commerce industry. For instance, direct sales from catalogues offer an excellent opportunity for efficient customization [9]. A given merchant can target his marketing messages to appropriate customers by personalizing the message to the customer’s name, interest or past purchases. This ability to personalize and customize service, or components of services, is a major factor under girding the extremely rapid growth of e-commerce services [10].

NitroScript offers several types of customization features such as moving the horizontal tabs menu to the bottom of the page; Inserting thumbnails on the basket page; Moving the Horizontal Tabs Menu to the left-hand section of the page; Swapping the position of the product search and shopping cart panels; and removing panels from the home page.

E. Security Handling

Security is of paramount concern when dealing with Web applications [11]. Cross-site scripting and SQL injection attacks, both were among the top ten vulnerabilities affecting Web applications in 2010 according to the Open Web Applications Security Project, were handled by the NitroScript engine. In order to achieve that goal, the NitroScript engine automatically calls the htmlspecialchars function to sanitize the data before being sent back to the client. This behaviour was observed in the result listing provided in Section III-B where the constant STORENAME were compiled into htmlspecialchars(‘The Name of My WebStore’).

IV. RELATED WORK

A. PHP Template Engines

There are several template engines for PHP such as Smarty\(^2\) and Twig\(^3\). The workflow of these template engines are as follows:

1) Specify a template to be used when rendering the Webpage. This template contains some placeholders that will be used to fill the computed values;

2) Next, bind values to the parameters as the actual content;

\(^2\)Official Website http://www.smarty.net (last accessed 1\(^{st}\) July, 2013)

\(^3\)Official Website http://twig.sensiolabs.org/ (last accessed 1\(^{st}\) July, 2013)
3) Finally, update the template with the assigned values to get the resulting HTML Webpages that will be sent back to the browser for rendering.

Smarty has been used in several production-quality open source projects such as XOOPS and SugarCRM. Figure 3 illustrates a code snippet showing the different steps of using the Smarty template engine. It consists of three files: The PHP script index.php which acts as the Model, index.tpl which is the Controller, and index.html which represents the View. In the file index.php, the first line creates the actual Smarty object and it creates two variables (name and address) that will be used in the template (file index.tpl) and the file index.html is returned to the client for rendering. The usage process of the Smarty template engine essentially consists of the calls to two methods of the Smarty class: assign() and display(). The former assigns new values to the parameters and the latter is invoked to specify a template and to fill it with the assigned values.

1) FlyingTemplates: FlyingTemplate, a server-side template engine, was presented in [7] whose goals were twofold: reduce the bandwidth consumption in Web application and off-loading the HTML generation tasks to Web browsers. Instead of generating an HTML page, FlyingTemplate only produces a skeletal script that includes only the strap code that runs from a Web browser on the client side. It then retrieves a client side template engine and the payload template separately. The two design goals of FlyingTemplate were the effective browser cache usage, and reasonable compromises which restrict the template usage patterns thereby relaxing the security policies slightly but in a manageable way.

FlyingTemplate has several limitations which prevented it from being used for e-commerce applications: license model, not optimized for e-commerce, and limitations in the programming style of templates. First of all, its license model. As no particular license was specified during its release, this makes very difficult to potential users to decide whether or not to adopt FlyingTemplate. This particular concern is critical to developers of commercial applications. Secondly, as FlyingTemplate is a general-purpose template engine, it does not make the most of PHP abilities in respect to e-commerce. As a result, we decided to develop a specific solution that is optimized for e-commerce applications.

V. CONCLUSION

This paper presents a new PHP template engine to customize Web stores in e-commerce called NitroScript. NitroScript is tailored to be used in e-commerce unlike other template engines such as Smarty which are too generic to be used in the field. Another advantage of NitroScript is due to the fact the result from the template is PHP bytecode which result in a performance boost. NitroScript automatically sanitizes the data before being used in a bid to avoid common security pitfalls such as a SQL injection and the cross-site scripting attacks. As part of future work, we plan to formally compare the performance of NitroScript against a similar Web store written with Smarty. Benchmarking NitroScript would allow us to qualitatively assess its performance over similar approaches. We also plan to extend NitroScript to take into account complex Web stores. This includes creating new panels to hold the list of recently viewed items.

ACKNOWLEDGMENT

The authors would like to thank Walid Trabelsi for his valuable and constructive feedback.

REFERENCES


Figure 3. Sample Usage of Smarty
A. Input Code Snippets

```php
{ifThereAre reviews}
<h1>{STORENAME}</h1>
<p>
{getLanguageString("REVIEWS_AVG_RATING")}
<img src="{GetStoreImagePath(rsum['filename'])}" alt="{rsum['averagerating']}">{getLanguageString("REVIEWS_STAR")}
{if (reviewsummary['avgrate'] > 1)}
{endif}
{ifNone}
{getLanguageString("REVIEWS_NOT_REVIEWED")}
{endifNone}
{endIfThereAre}

B. Output Code Snippets

```php
<?=$htmlspecialchars('The name of My WebStore')?><p>
</p>
<?php @mysql_data_seek($result, 0);
$thisloop['iteration'] = 0;
$thisloop['count'] = (is_resource($result)) ? mysql_num_rows($result) : 0;
if ($thisloop['count'] > 0):
while ($nitroloopreview = mysql_fetch_array($result, MYSQL_ASSOC)):
$thisloop['iteration']++;
$nitroloopreview['filename'] = "star-".$nitroloopreview['rating'].'.gif';
$nitroloopreview['datecreated'] = strtotime($nitroloopreview['datecreated']);
if (($nitroloopreview['useful'] > 0) || ($nitroloopreview['notuseful'] > 0)) :
echo getLanguageString("REVIEWS_FOUND_USEFUL", $nitroloopreview['useful']);
endif;
endif;
</p>