Standardizing the Human Interaction in Websites Using Web Application Frameworks

Fernando Arango Isaza and Danny Alvarez Eraso(✉)

National University of Colombia, Medellín, Colombia
{farango,daalvareze}@unal.edu.co

Abstract. Web Application Frameworks (WAFs) are widely used nowadays to build quality web applications. However, developers have to code views one by one because WAFs offer little support for giving uniformity to the whole set of views. We propose the use of a View Code Generator (VCG) to automatize the process and assure views uniformity. Besides uniformity, our approach also reduces error sources, time to market and improves the resources allocation efficiency in the overall software life-cycle.

1 Introduction

Web Application Frameworks are widely used nowadays to build quality web applications [1–3]. WAFs offer developers, a pre-defined Model-View-Controller (MVC) architecture [4,5], improve code reuse [6], and several tools (or Helpers) to implement components. In particular, most WAFs fully support the construction of highly customizable views.

However, most current WAFs offer no means for standardizing the whole set of views. This means that developers must ensure that each application view follows the same data displaying and human interaction strategy; making more difficult the coding process.

This is inconvenient not only because the coding, per se, takes a great portion of the development effort –in a meting with start-up companies they estimated the coding task in 60 % of development effort–, but also because achieving the required view standardization is difficult, if not impossible, in an environment with multiple developers.

We propose the use of a View Code Generator (VCG) to assure views uniformity. In our approach the VCG divides the views specification in two steps. The first one, named *Object's Visible Data Specification* (or OVDS) specify the data to be displayed in a particular view. The second, named *Theme Specification* (TS) is common to all views and specify the data layout and view-user interaction strategy.

To test our approach we coded a VCG in the PISIS Framework, a proprietary WAF prototype. In our prototype the OVDSs were included as part of the classes definition. Meanwhile, the TS was hard coded as part of the VCG.

This paper is presented as follows: Sect. 1 covers the introduction, Sect. 2 presents antecedents to our work, Sect. 3 presents the VCG for displaying data,

C. Stephanidis (Ed.): HCII 2016 Posters, Part II, CCIS 618, pp. 3–7, 2016.
DOI: 10.1007/978-3-319-40542-1_1

Sect. 4 presents our proposal for human interaction standardization and in Sect. 5 we present the conclusions of our work and future work.

2 Antecedents

Rosales et al. in [7] present a recent systematic review on tools for automatic code generation. Computer-Aided Software Engineering (CASE) tools are among the most important tools. CASE tools are know for integrating methodologies and technologies for generating executable source code with the system model as input [7,8].

However CASE tools have disadvantages. Customization is limited in resulting applications, and those initial systems are harder to extend and maintain, and integration with other systems is hard. So, their usability in web applications that highly evolve is not easy.

On the other hand WAFs aid web development providing a flexible skeleton that serves as the base for any application. Every WAF offers a set of helpers for defining views: Sring Framework [9] aimed for java offers the JSP language RubyOnRails [10] uses .erb files, and [11] aimed for PHP offers simple PHP files as the template engine. Also, they offer pre-elaborated components for displaying data and designing forms.

However these tools are far from generating the application using the design diagrams as the only input. In particular, designing views with the available helpers is a manual process that must be done for every view; this is a lot of work. So, ensuring view uniformity is not easy.

As far as our knowledge reaches there is no WAF with automatic view generation are available.

3 View Code Generator and PISIS Framework

To reduce the work involved in designing views we propose the use of an automatic VCG that receives both the data to be displayed and the TS as input to generate all views following a set of conventions.

To test our approach we coded a VCG in the PISIS Framework, a proprietary WAF prototype written in the PHP language. In our prototype the OVDS was included as part of the classes definition. Meanwhile, the TS was hard coded as part of the VCG.

Defining Useful Class Models: We use the classes in the model layer of the MVC architecture as the core of our application. We added meta-data attributes to classes in order to make them useful not only to describe the business entities' data, but also to define what data will be displayed in the class views and the navigation paths; these two constitutes the OVDS.

The class views are, the main class view showing the attribute values for a class object, and the class "facets" that list the objects related to a class

object trough the different class association links. By double clicking one of those objects, the display focus shift to the information of that object. This means that classes are the core of the navigation system, so, navigating from one object to one element in one of his facets also changes the displayed object.

Defining the View Code Generator: After extending classes with view and navigation information, we implemented an interpreter of that information to automatize how requests are managed to display all data contained in the database.

The PISIS Framework uses the Smarty template engine to implement the interpreter. After that, we extended Smarty functionality using plug-ins to code an interpreter of the object meta-data that constructs the corresponding HTML component: labels, tables, lists, urls, etc. This interpreter allocate both the object's primitive and the available facets data based on the OVDS. This plug-in is our VCG.

Even when our VCG is developed inside Smarty, it constitutes a new layer between the controller and views layer. Theoretically, replacing Smarty for other technology makes no difference.

About the Theme Support: Our VCG still needs a new interpreter for sporting multiple themes, which will be defined using a specialized language. However, as a first attempt, we hard-coded the theme inside the VCG.

4 Standardizing the Human Interaction Strategy

The views generated by our VCG cover CRUD (Create, Read, Update and Delete) operations. This is done by, standardizing how HTTP requests are managed by the controller layer and by including the same edition controls for table or data lists.

To do this we implemented a single class that works as a master controller capable of keeping track of the displayed object, managing transition to other object and serving the creation and edition of records. This mechanisms was automatically injected to all views.

This way we ensure that the human interaction strategy remains the same in all views making easier for the final user to learn how to use the application. This also reduces the need of developing multiple controllers and in consequence, this strategy reduces the total development effort and improves maintainability.

So, in Fig. 1 we show PISIS Framework MVC architecture for automatic view generation.

Reading and Displaying Records: The application has a default displayed object from where the user can navigate to all the objects in the system. The navigation elements displayed are the object facets and correspond to the object's visible data.

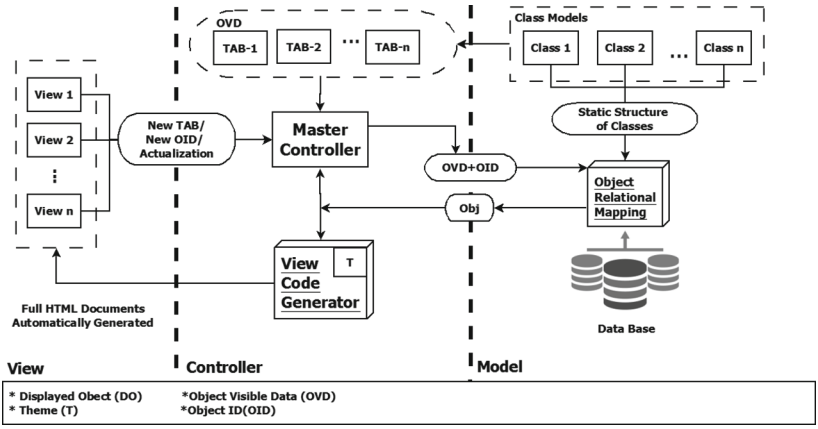

Fig. 1. PISIS Framework MVC architecture for automatic view generation.

We defined a *data manipulation layer* to read database records and transform them into objects. This layer is often referred as the WAF ORM (Object-Relational Mapping) implementation. The current version of PISIS deals with this by manually coding every interaction.

Inserting and Editing Records: Records can be inserted/edited in one of the displayed object facets by using the edition controls available in all views in edition mode, see Fig. 2. This allows the table lines to be used as edition forms, multiple changes can be applied to the list but they are not intermediately committed to database; data is written only after pushing the save button. The reader must note that this behavior is useful to support undo/redo operations, as well as convenient for reuse code for showing and editing records.

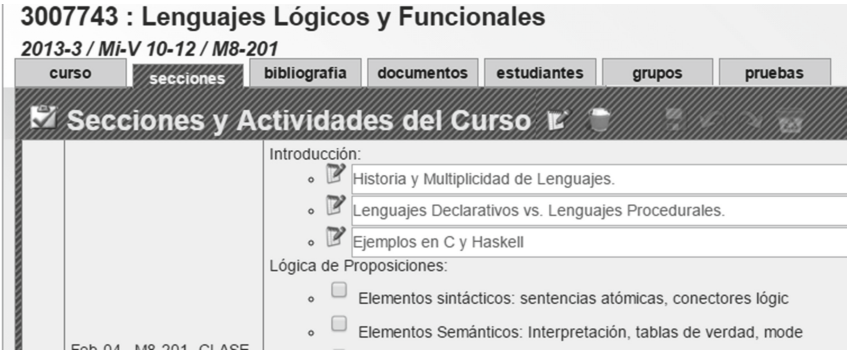

Fig. 2. PISIS Framework Edition controls.

5 Conclusions and Future Work

We proposed a VCG to automatize and standardize the view generation process in web applications using a Framework prototype. Also, we proposed a MCV architecture that eliminates the need of writing controllers using the class models as the core of the application definition. As a consequence, classes are the place where the developer should focus the development effort.

Our approach saves a lot of coding effort as well as it standardizes the whole set of views and the human interaction strategy. This is done by centralizing HTTP request regarding CRUD operations in a master controller that later passes the response to the VCG. Finally, the VCG generates the HTML document and inject standardized edition controls.

Our future research is focused on the definition of a TS interpreter that allows separating the Theme layout from the VCG. We will implement an interpreter for supporting different TSs so that developers can build their own layouts.

We will define an ORM to replace improve our *data manipulation layer* and reduce the need of writing SQL statements.

References

1. Chen, B., Hsu, H.-P., Huang, Y.: Bringing desktop applications to the web. IT Prof. **18**(1), 34–40 (2016)
2. Vuksanovic, I.P., Sudarevic, B.: Use of web application frameworks in the development of small applications. In: MIPRO, 2011 Proceedings of the 34th International Convention, pp. 458–462 (2011)
3. Shan, T.C., Bank, W., Hua, W.W.: Taxonomy of Java Web Application Frameworks, p. 07 (2006)
4. Krasner, G., Pope, S.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. J. Object Oriented Program. **1**, 26–49 (1988)
5. Leff, A., Rayfield, J.: Web-application development using the Model/View/Controller design pattern. In: Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, pp. 118–127 (2001)
6. Schwabe, D., Rossi, G., Esmeraldo, L., Lyardet, F.: Web design frameworks: an approach to improve reuse in web applications. In: Murugesan, S., Desphande, Y. (eds.) Web Engineering. LNCS, vol. 2016, p. 335. Springer, Heidelberg (2001)
7. Rosales, V., Alor, G., García, J., Zatarain, R., Barrón, M.: An analysis of tools for automatic software development and automatic code generation, in Revista Facultad de Ingeniería Universidad de Antioquia, pp. 75–87 (2015)
8. Johns, M., Beyerlein, C., Giesecke, R., Posegga, J.: Secure code generation for web applications. In: Massacci, F., Wallach, D., Zannone, N. (eds.) ESSoS 2010. LNCS, vol. 5965, pp. 96–113. Springer, Heidelberg (2010)
9. Spring.io: Spring Framework (2002). https://projects.spring.io/spring-framework/. Accessed 07 Apr 2016
10. Heinemeier, D.: Ruby on Rails (2005). http://rubyonrails.org/. Accessed 10 Mar 2016
11. EllisLab: CodeIgniter Web Framework (2006). https://www.codeigniter.com/. Accessed 07 Apr 2016