SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM
TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
VOLUME 5 NO 4, 2017
ISSN 2054-7390

A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications

Karim Arrhioui, Samir Mbarki, Oualid Betari, Sarra Roubi, Mohammed Erramdani
MISC Laboratory, Faculty of Sciences, IbnTofail University, Kenitra, Morocco
MATSI Laboratory, Superior School of Technology, Mohammed First University, Oujda, Morocco
arr.karim@gmail.com, mbarkisamir@hotmail.com, beta.oualid@gmail.com, roubi.sarra@gmail.com, m.erramdani@gmail.com

ABSTRACT

During the last decade, web development industry has grown exponentially. Models have been introduced as a solution to face the challenge of both business and technology changes. In this article, we present a Model Driven based approach concerning the design of CodeIgniter based web applications. We describe a meta model of this framework and we also specify a set of transformations to generate the application's source code taking into account the MVC (Model-View-Controller) architecture of CodeIgniter. In this approach, the PHP framework meta model is considered as a platform Specific model (PSM). Its instances are used as inputs to generate the source code through transformation rules carried out by Acceleo. This proposal is validated through the use of our approach to generate CRUD (Create, Read, Update and Delete) applications.

Keywords-CodeIgniter; Model Driven Architecture; Model-View-Controller; PHP; Platform Specific Model

1 Introduction

The express development of web based application has affected the coding methodology. This points toward a higher need of sustainability and maintainability. Besides, PHP is a web scripting language which is used in dynamic interactive web development. It is a general-purpose and an open source tool that requires minimal setup [1]. Furthermore, PHP has become one of the most powerful programing languages for developing web applications. To deal with the problems caused by the increasing projects' complexity, several techniques for programming in PHP such as Procedural PHP coding and Object Oriented Programming have been proposed. Resulting this rapid development, several frameworks such as CodeIgniter PHP framework have emerged to facilitate the development tasks. Indeed, as mentioned in [2],they have been improved and "become handy tools for developers to build complex applications efficiently".

In order to benefit from using these frameworks and also handle frequent changes, the Object Management Group (OMG) proposed the Model Driven Architecture as a solution. This approach supports changing business rules in different application domains by providing an open approach to manage the challenge of interoperability, by using OMG's established modeling standards that are: Unified Modeling Language (UML) and Meta-Object Facility (MOF) [3] [4].

DOI: 10.14738/tmlai.54.3189
Publication Date: 15th August, 2017
URL: http://dx.doi.org/10.14738/tmlai.54.3189

In this paper, we consider the union of the solutions offered by the introduction of the PHP frameworks and the use of MDE (Model Driven Engineering) in developing applications, as we will apply a Model Driven approach to model the CodeIgniter PHP framework and generate CRUD applications based on this framework.

The present paper is structured as follows: In section 2 we outline the MDE principles. Section 3 presents the concepts of CodeIgniter and the MVC (Model-View-Controller) pattern. Section 4 introduces our MDA approach for generating CRUD applications. A case study is presented in Section 5. Finally, section 6 concludes the work and offers further perspectives.

2 Model Driven Architecture

The OMG has introduced the Model Driven Architecture that is based on Modeling and transformation to generate the code and presented a formal statement with several tools and approaches [3]. This architecture focuses on creating models with a high level of abstraction, and promotes transforming the models according to given defined rules [5].

Today's systems are constantly changing and highly networked. In order to face these challenges, MDA provides a platform independence architecture that assures portability and cross-platform Interoperability [3].

MDA consists of three general types of models, structured into three basic layers: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). The three models are defined as follow:

- CIM: This model describes the system's functionalities with a high level of abstraction. It is seen as a business model, as it uses a vocabulary that is familiar to the subject matter experts. It presents what the system is supposed to do, and hides structure details and implementation [6].
- PIM: This model defines the concept of the system without showing the specific details of a target platform. It exhibits a sufficient degree of independence so as to enable its mapping to one or more platforms.
- PSM: It describes the deleted details and characteristics of PIM. It also provides platform specific details that should be considered to implement the system.

The reason for the above model organization is to develop models of the systems' business logic independently from the platforms of execution, then to transform these models automatically to models dependent of the platforms. The complexity of the platforms does no longer appear in the business logic models but it is found in the transformation [7].

The Model Driven development using UML approach requires several steps: at first building the CIM that acquires user requirements. Then, according to this CIM, a PIM is built. Next, the proposed PIM is mapped into one or more PSMs. This type of transition from CIM to PIM and PIM to PSM is called Model To Model (M2M) transformation. Finally, the code of the target platform is generated from the PSM instance. This transition is called Model To Text (M2T) transformation [5] [7] [8]. Fig. 1 shows how the transformations are done [9]:

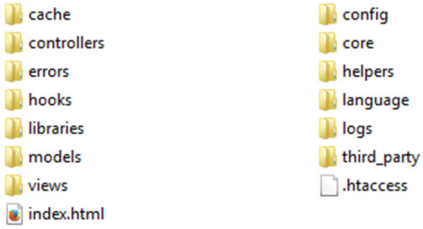


Figure 1. Model Driven Architecture Layers.

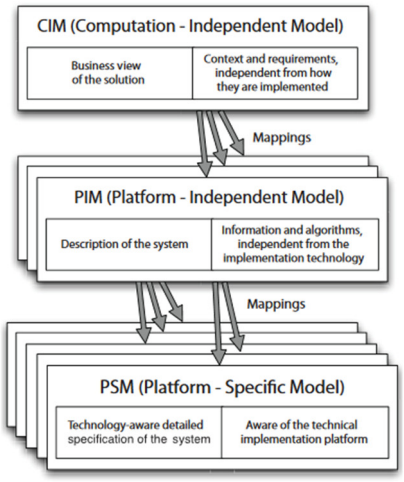


Figure 2. Structure of CodeIgniter applications

CodeIgniter is a PHP Application Development Framework based on a well-structured architecture. It aims to provide necessary tools such as helpers and libraries to implement common tasks. Thus, project development becomes much easier and faster, and developers don't have to write all the code from scratch [2] [10].

CodeIgniter is based on the MVC development approach, MVC is a design pattern that structures the software by separating application logic from presentation [11]. Indeed, PHP scripting related to business elements is separated from web pages [12].

- The Model contains the business logic of the application. It gathers functions related to data accessing and third-party services.
- The View is composed of user interface elements such as HTML, CSS and JavaScript files. In CodeIgniter, a view can be a web page, a fragment of a page, a RSS page...
- The Controller connects views, models and any needed resources to process and respond to the user request. It is the point of entry that instantiates the required views and models [13].

3 Model Driven Approach for generating CRUD applications

The development of web applications has been improved by the integration of frameworks, CodeIgniter being one of these frameworks allows the development of PHP web applications.Using CodeIgniter for the development of web applications, it is necessary to respect a precise structure as shown in Fig. 2:

That's why when we initiated the development of a meta-model for this framework, it was necessary to study this structure and raise the level of abstraction.

The proposed meta model in this paper is the PSM that describes the CodeIgniter PHP framework. This meta-model is used to define instances of models that describe web applications. These instances will be the inputs of the transformation engine that will be developed using Acceleo that respects an approach by template. The primary benefit of our approach is that the long task of coding a CRUD web application is done in a systematic, well-structured and standard way by using MDA principles.

3.1 The proposed CodeIgniter meta model

The developed meta model of CodeIgniter framework, shown in Fig. 3, is represented by the model, view and controller packages. Each package contains specific meta classes according to the MVC pattern.

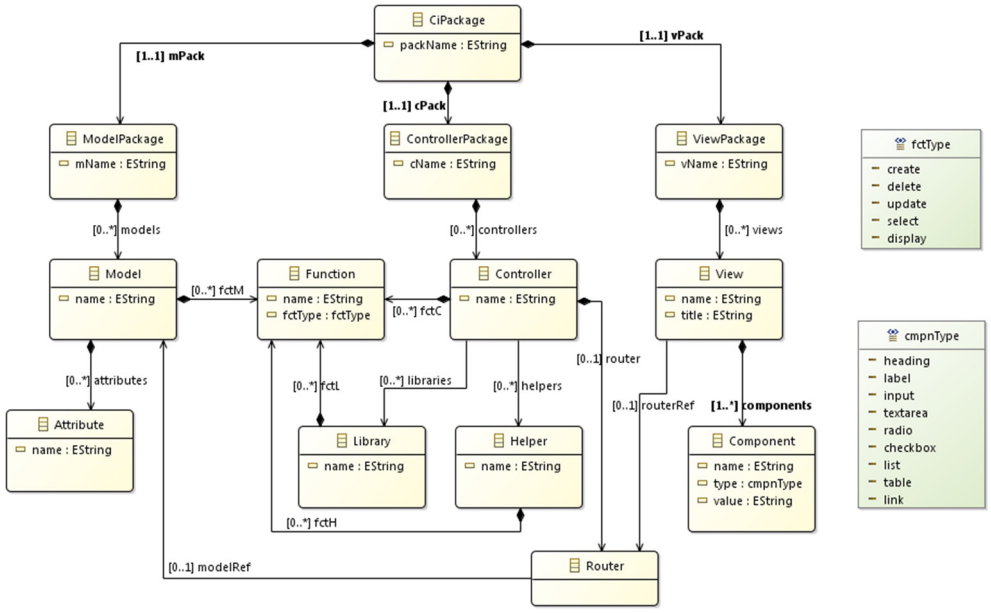


Figure 3. CodeIgniter meta model.

- CiPackage: expresses the concept of package that includes the entire elements of the model; the model, the controller and the view, that are themselves regrouped, respectively, in ModelPackage, ControllerPackage and ViewPackage.
- A helper or a library is a set of functions that can be called from the controller, the view or the model. It is a PHP file that contains functions grouped by theme [12].
- View: contains the components of the View that will be defined bellow.
- Controller: the intermediary between the model and the view. It contains all the functions to manage each user action [14].
- Component: it is a graphical element such as input, text area and heading.

The View/Controller layers are responsible for describing the structure and content of views, while the navigation flow is ensured through the controller's specific functions that are connected to the specified services from the model layer. This layer gathers all the business parts of the application.

3.2 The Model to Text transformation rules

Once the application has been sufficiently modelled, the code generation procedure follows the Model To Text transformation (M2T) to get, in our case, a CodeIgniter application's source code using Acceleo [8]. This transformation follows the template approach; thus, we have developed the needed templates for code generation taking as an input a CodeIgniter meta model instance. The main module for code generation is shownbellow:

```
[comment encoding = UTF-8 /]
[modulegenerateCI('http://cimd/1.0')]
[templatepublicgenerateElement(aCiPackage : CiPackage)]
[comment@main/]
        [for(model : Model|aCiPackage.mPack.model)]
                [generateModels(model)/]
        [/for]
        [for(controller : Controller|aCiPackage.cPack.controller)]
        [generateControllers(controller)/]
        [/for]
        [for(view : View|aCiPackage.vPack.views)]
                        [generateViews(view)/]
        [/for]
[/template]
[templatepublicgenerateControllers(controller : Controller)]
        [file ('/controllers/'+controller.name+'.php',false,'UTF-8')]
<?php
class[controller.name/] extends CI_Controller {
                public function index()
{   $data['['/]'title'[']'/] = "Add [controller.name/]";
            $data['['/]'heading'[']'/] = "-- Add [

controller.name/] --";
            $this->load->view('[controller.name/]_view', $data);
   }
                [for(fct : Function|controller.fctC)]
                        [generateFct(fct)/]
                [/for]
}
?>
        [/file]
[/template]
```

4 Case Study

CRUD operations are often the most commonly used in web applications. In addition, CodeIgniter offers the possibility to manage these operations through its predefined structure. So the approach we propose comes to simplify and further automate this process which converges towards these objectives and allows to generate these CRUD operations automatically in terms of source code.

```php
<?php
class Customer extends CI_Controller {

    public function index()
    {
        $data['title'] = "Add Customer";
        $data['heading'] = "-- Add Customer --";

        $this->load->view('Customer_view', $data);
    }

    public function addCustomer()
    {
        $this->load->helper('form');
        $this->load->model('CustomerModel');

        $Customer_array = array(
        'id'        => NULL,
        'name'      => $this->input->post('name')
      , 'email'     => $this->input->post('email')
      , 'adress'        => $this->input->post('adress')
        );
```

Figure 4. The input model of Customer CRUD operations

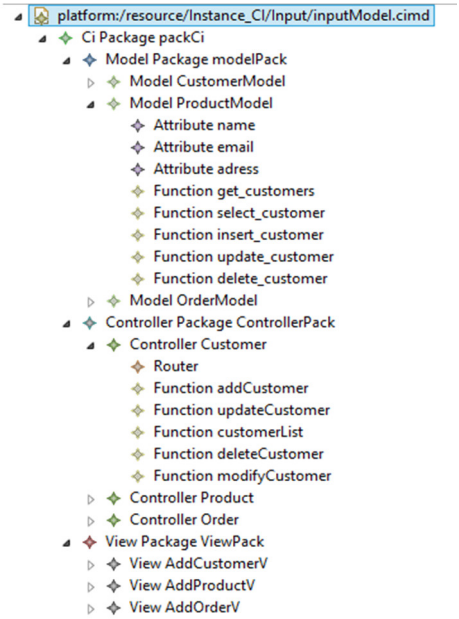


Figure 5

Once the application is modeled,using Acceleo, the code generation procedure follows the templates developed for the M2T engine. Fig. 5 below is an excerpt from the generated code:

To add a new customer, the user should provide a name, an address and an email then submit the form to the customer controller. Fig. 6 shows the generated views containing forms for adding both customer and product. Once the form is submitted, the specific controller calls a function in order to add the new element

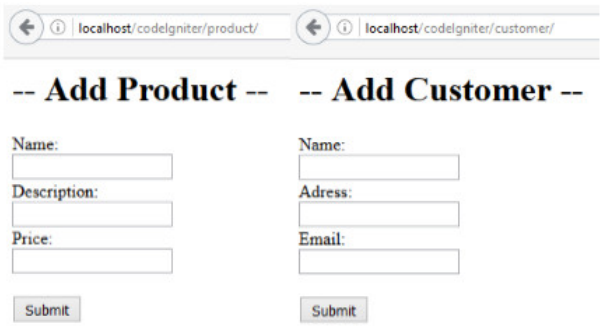
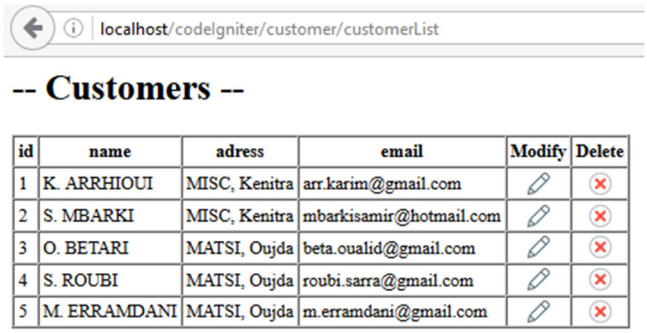


Figure 7. Generated forms for adding customer and product

To list all customers, the customer controller loads the customer model, calls “get_cutomers” function and then sends the result to the customerList view as show in Fig. 7:



id	name	adress	email	Modify	Delete
1	K. ARRHIOUI	MISC, Kenitra	arr.karim@gmail.com		
2	S. MBARKI	MISC, Kenitra	mbarkisamir@hotmail.com		
3	O. BETARI	MATSI, Oujda	beta.oualid@gmail.com		
4	S. ROUBI	MATSI, Oujda	roubi.sarra@gmail.com		
5	M. ERRAMDANI	MATSI, Oujda	m.erramdani@gmail.com		

Figure 8. Customers list from the database

5 Conclusion and Perspectives

In this paper, we proposed an approach that allows modeling web applications based on the CodeIgniter PHP framework. In order to achieve this end, we applied the model driven approach principles to generate these applications. This approach using modeling and transformation process provides advantages in improving applications portability and quality, while minimizing cost and time. This approach has demonstrated its efficiency through enabling the building of a complete meta model that describes MVC CodeIgniter application, then, generating its source code.

The form validation, using jQuery Validation Plugin and CodeIgniter form validation library, will be the subject of a study thereafter. Afterwards, we will consider applying this solution to other PHP frameworks in order to propose a comparative study.

REFERENCES

[1] S. Bergmann, and G. Kniesel, "GAP: generic aspects for PHP," in Proc. EWAS'06, 2006.

[2] O.Betari, M. Erramdani, S. Roubi, K. Arrhioui, and S. Mbarki, "Model transformations in the MOF meta-modeling architecture: from UML to codeIgniter PHP framework," Europe and MENA Cooperation Advances in Information and Communication Technologies, vol. 520, pp. 227-234, 2016.

[3] Object Management Group (OMG), MDA Guide 2.0. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[4] Executive Overview, Model Driven Architecturehttp://www.omg.org/mda/executive_overview.htm

[5] S. Kherraf, É. Lefebvre, and W. Suryn, "Transformation from CIM to PIM using patterns and archetypes," 19th Australian Conference on Software Engineering (aswec 2008), Perth, WA, pp. 338-346, 2008.

[6] S. Roubi, M. Erramdani, and S. Mbarki, "Modeling and generating graphical user interface for MVC rich internet application using a model driven approach," 2016 International Conference on Information Technology for Organizations Development (IT4OD), Fez, pp. 1-6, 2016.

[7] X. Blanc, MDA en Action: Ingénierie Logicielle Guidée par les Modèles, Eyrolles, 2005.

[8] S. Roubi, M. Erramdani, S. Mbarki, "Generating graphical user interfaces based on model driven engineering,"in International Review on Computers and Software (IRECOS), vol. 10, pp. 520-528, 2015.

[9] M. Brambilla, J. Cabot and M. Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool, 2012.

[10] E. Orr, and Y. Zadik, Programming with CodeIgniter MVC, Ed. Birmingham, UK: Packt Publishing, 2013.

[11] R. Foster, CodeIgniter 2 Cookbook, Ed. Birmingham, UK: Packt Publishing, 2013.

[12] CodeIgniter Documentation website. https://www.codeigniter.com/docs/

[13] C. Pitt, Pro PHP MVC, Apress, 2012.

[14] M. Brambilla and A. Origgi, "MVC-Webflow: an AJAX tool for online modeling of MVC-2 web applications," 2008 Eighth International Conference on Web Engineering, pp. 344-349, 2008.