MDA Approach for Laravel Framework Code Generation from UML Diagrams

Mantas Ražinskas[a], Lina Čeponienė[b]

[a]Faculty of Informatics of Kaunas University of Technology, Kaunas, Lithuania
[b]Information Systems Department of Kaunas University of Technology, Kaunas, Lithuania

Abstract

Laravel is a popular PHP framework used in web development. To facilitate and simplify the work of software developers using this framework, this paper proposes an MDA based methodology for developing Laravel based web information systems. This methodology implementation would allow software developers to generate Laravel code from UML diagrams through MDA transformations from PIM to PSM (with Laravel PSM profile applied) and from PSM to code. Developers then could fill the gaps in the generated PHP code for Laravel framework to develop a complete web information system.

Keywords

Laravel, MDA, MVC, UML model, code generation

1. Introduction

The modern need for development of high quality information systems (IS) at minimal cost is rapidly changing their development methodologies, with stronger emphasis on the facilitation of IS development and maintenance [1, 2, 3]. A possible solution for developing systems more efficiently is to use Unified Modelling Language (UML) [4] – a modeling standard developed by the OMG Computer Industry Standards Consortium, and Model Driven Architecture (MDA) [5, 6] – an approach for software development by means of models and transformations between them. MDA enables model driven systems to be built using model transformations from one model type to another. There are different types of models in MDA, for example Platform Independent Model (PIM) which defines implementation independent abstract system functions, and Platform-Specific Model (PSM), which defines the implementation of system functions in a selected platform. UML is specified by its metamodel which can also be extended by using the profile mechanism. In this situation, UML profile extends a referenced metamodel to adapt or customize it with constructs that are specific to a particular platform [4]. In MDA approach, after defining the PSM model, the next step is to generate the source code for the selected implementation platform that corresponds to the elements defined in the PSM [5]. Model transformations and code generation save time in the IS development process, reduce the risk of potential errors, and simplify modifications [7]. Currently, code generation is available in popular UML CASE tools, which usually have the code generation functionality from structural UML diagrams to object oriented programming languages (C#, Java, etc.).

The principles of Model Driven Architecture can be applied in the development of various information systems, including web-based ones. In the context of web development, PHP can be considered the most widely used programming language [8]. Various PHP frameworks are widely used for facilitating the implementation process. Frameworks have a structured architecture, simplify the database connection, are easily extended with additional libraries, and are usually based on the Model View Controller (MVC) architecture principles [8]. MVC gives an effective and proven way of developing modular, structured systems. In addition to benefits of using a framework, introducing the MDA principles and employing PHP code generation from UML diagrams for a certain framework could make web IS development process even more efficient and reduce time-to-market.

Laravel [9] is a popular PHP framework suitable for rapid development of small to large scale systems, even with relatively little experience [8, 10, 11]. This framework is based on the MVC architecture principles and assists in building web systems faster and easier, by providing basic model structure, API access, libraries, and plugins. It also helps developers become more productive by reducing duplicate code in an ongoing project [12].

Based on framework comparison results [11], it is easier to get started with Laravel than with other pop-

IVUS 2020: Information Society and University Studies, 23 April 2020, KTU Santaka Valley, Kaunas, Lithuania
mantas.razinskas@ktu.edu (M. Ražinskas);
lina.ceponiene@ktu.lt (L. Čeponienė)

ular PHP framework, Symfony, and Symfony is more suitable for developing complex systems. In this paper, an MDA based methodology is presented which investigates the application of model transformation principles in the context of PHP programming language and its Laravel framework. The transformations from PIM to Laravel based PSM and from Laravel PSM to Laravel PHP code are analyzed. The proposed concept could help software developers build web information systems more efficiently by applying UML diagrams and MDA principles.

The rest of the paper is organized as follows. In related work section related studies in the area of MDA and code generation are discussed. In the third section, the general process of the proposed methodology for Laravel code generation is presented, along with functional requirements for the methodology implementation and implementation strategy representation. The fourth section discusses guidelines for implementation of proposed concept and presents a short examples of developed models and generated code. Finally, in the conclusions section, conclusions are presented and further research perspectives are discussed.

2. Related Work

In recent years, there has been a numerous research on MDA and code generation [13, 14, 15, 16]. In this paper, the studies associated with framework based code generation [14, 15, 16] were selected for more detailed investigation.

In research paper by Arrhioui, Mbarki, Betari, Roubi, Erramdani [14], the authors explored a model driven architecture approach for PHP CodeIgniter framework. The authors proposed a methodology that enables modeling web systems based on the PHP CodeIgniter framework and used an MDA approach to develop their methodology. This methodology encompasses transformations from PIM to PSM using UML class diagram as a source model to generate an XML file with the essential components of the CodeIgniter framework. The use of model transformations has provided benefits in improving the quality of the system development process while reducing costs and time.

This approach demonstrated efficiency by enabling the creation of a complete CodeIgniter MVC architecture framework with defined metamodel and generated program code based on this metamodel.

The CodeIgniter framework metamodel developed by the authors is divided into packages of models, views, and controllers. Each package has specific metaclasses based on the MVC architecture. When authors completed system modelling, the code generation procedure was followed by the model-to-text transformation (M2T). To generate CodeIgniter code, the authors used Acceleo software and CodeIgniter metamodel template.

The principles of MDA were also applied in research by Srai, Guerouate, Berbiche, Lahsini [15] to create a Spring MVC system using a UML class diagram. First, a metamodel was created that matched the Spring MVC template. The target metamodel consisted of two essential parts: the first part referred to the views package, this package consisted of many JSP pages. The second part consisted of a controller package, and this package encompassed a number of controllers and each controller having one or more actions.

Authors defined transformation rules for model to model and model to code transformations. These rules enabled creation of an XML file containing all the actions, forms, and JSP pages that can be used to generate code. Authors developed a transformation algorithm using the ATL transformation language [17] (this language is part of the Eclipse M2M). Next, the model to text transformation was developed to generate Spring MVC code. For the model to text transformation, a Spring MVC template was defined.

A tool that support the Model Driven Architecture (MoDAr-WA) was introduced by Essebaa, et al. [16], which implements a methodology automating transformations from the highest MDA level (CIM) to the lowest (code). This research is a continuation of the former work in automating transformations from CIM to PIM. The MoDAr-WA authors created sets of metamodels for UML class and sequence diagrams, QVT and Acceleo transformations, as well as Eclipse plugin for MoDAr-WA. In particular, QVT rules for transformations between models (from CIM to PIM, from PIM to PSM) were used. At the PIM level, the authors defined key structural and behavioral aspects in UML class and sequence diagrams. Java source code from PSM was generated with the help of Acceleo tool.

All of the analyzed researches aim to apply MDA principles to improve the process of system development. They provide a detailed definition of their proposed theoretical concept and its implementation, by defining profiles, transformation rules and algorithms, as well as experimental implementation examples. The analyzed methodologies are intended for various frameworks and architectures (PHP CodeIgniter, Spring MVC and Java MVC). The methodology proposed in our paper also applies MDA principles and analyses model transformations and code generation, but in the context of Laravel framework, which is not analyzed in other authors' research.

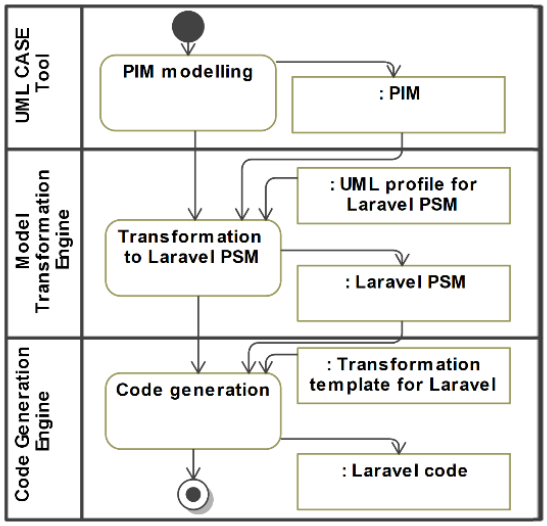

Figure 1: Abstract process of modelling and generating Laravel based IS.

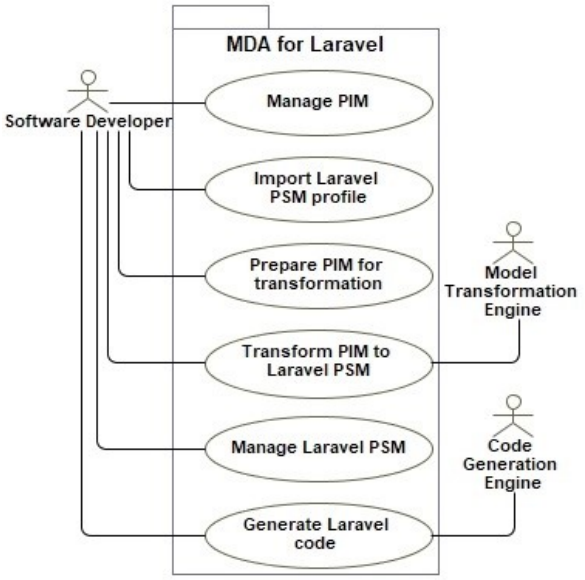

Figure 2: Use Case diagram for methodology of modelling and generating PHP Laravel based IS.

3. The Proposed Methodology for LARAVEL Code Generation

This section presents the MDA based methodology for Laravel framework. This methodology encompasses not only modelling of PIM and PSM, but also model transformations from PIM to PSM and from PSM to Laravel code.

3.1. The General Process for the Proposed Methodology

In the proposed methodology (Fig. 1), the abstract modeling and code generation process is proposed which encompasses three main steps: PIM development, transformation to PSM for Laravel framework and Laravel PHP code generation. First of all, using a UML CASE tool, PIM model should be modeled and exported to the selected model-to-model transformation tool. The XMI model interchange format should be used to enable transfer of models between the steps of the process. Next, transformation from PIM to PSM is performed by applying transformation rules, algorithms and model transformation engine along with UML profile for Laravel PSM. After the transformation, PSM in XMI format is generated.

This PSM model can be either edited in modelling tool or directly transformed into code. PSM to code transformation is performed using transformation template, transformation algorithms and code generation engine. This transformation results in Laravel code which can further be completed by the system developers in order to finalize the system implementation. It should be noted, that our method does not aim to generate the complete PHP code for Laravel framework. It rather concentrates on automating the transition from UML diagrams to code as much as possible, but allowing the software developers to complete the system implementation by themselves, as the full code usually contains more information than it was provided in the UML diagrams of PIM and PSM.

3.2. Functional Requirements for the Methodology Implementation

For a detailed representation of the proposed methodology, the UML use case diagram (Fig. 2) depicts possible actions software developer can perform in order to accomplish the model transformations and generate Laravel code.

In our proposed methodology, platform independent model (PIM) can be managed using any CASE tool selected by software developer, which is able to export the model to XMI file. The PIM model must include a class diagram, without this type of diagram it will not be possible to continue the code generation process. The PIM model has a high level of abstraction and separates logic from technological implementation. This model provides modeled system a structure that will

fit any implementation platform.

In order to perform transformation from PIM to PSM and to edit PSM, the software developer can import the UML profile for the PHP Laravel framework. Having UML Profile and XMI format PIM model file, imported into selected transformations tool, Laravel PSM can be generated by using transformation engine and by applying transformation rules and algorithms. After the transformation, software developer can manage Laravel PSM model – make changes to it if required or import PSM models' XMI file to transformation tool for transformation to code.

Transformation from PSM to code is performed using the selected code generation engine, PSM metamodel and code generation algorithms. As a result of the transformation, software developer will receive the PHP Laravel code with PHP files for models, views, controllers, routes and other required Laravel framework elements.

3.3. The Proposed Implementation Strategy

Possible implementation solution with proposed tools for modeling and generating Laravel based information system are depicted in deployment diagram (Fig. 3). The communication paths (having the name "xmi file") between execution environment nodes in this diagram do not represent the direct communication, but rather the possibility for the software developer to transfer XMI file from one execution environment to another.

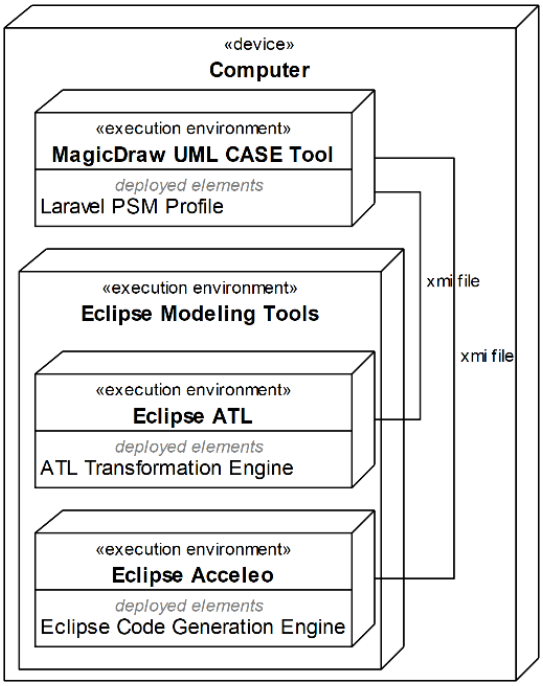

Figure 3: Deployment diagram representing proposed tools for modelling and generating Laravel based IS.

Using MagicDraw CASE tool, software developer can manage PIM. Software developer models the class diagram in PIM and it is possible to save PIM in XMI format for M2M transformation. After M2M transformation, if further changes are required, it is possible to import PSM in XMI format back to MagicDraw CASE tool to modify it before proceeding to code generation. After PSM modification it is possible to save this model in XMI format for further transformation.

Transformation from PIM to PSM can be performed using Eclipse tool and according to the rules and algorithms which have to be defined using ATL transformation language [17]. ATL provides a way to produce a number of target models from a set of source models. An ATL transformation tool is composed of rules, which define how source model elements are used to create the elements of the target model.

To perform transformation from PIM to PSM, XMI format PIM file must be imported into Eclipse ATL tool. Afterwards, transformation option should be selected for the ATL transformation engine to perform the transformation from PIM to PSM according to defined rules and algorithms. As a result, PSM in XMI format will be generated.

Once the M2M transformation has been completed, the PHP code can be generated using the Eclipse Acceleo tool according to established rules and algorithms for code generation [18]. This tool is an open source code generator implementing the OMG's MOF Model to Text Language (MTL) standard that uses any EMF based models to generate any kind of code.

To perform transformation from PSM to code, PIM model in XMI format must be imported to Eclipse Acceleo tool. Then transformation option should be selected for Acceleo code generation engine to perform the transformation from PSM to code according to the defined rules and algorithms. The result of this transformation is PHP Laravel code.

For the proposed transformation methodology, we have chosen to use UML class and sequence diagrams in PIM and in PSM. The implementation plan for the proposed methodology consists of several steps. The first step is the definition of the PIM metamodel, which will consist of relevant class and sequence diagram metaclasses, the next step is the definition of PSM metamodel which will consist of relevant class and sequence

metaclasses enriched with UML profile, encompassing specific Laravel framework stereotypes. After defining both metamodels, we will perform the first transformation using PIM metamodel as input, and by applying transformation rules the PSM will be generated according to the defined PSM metamodel. After defining the first transformation in detail we will proceed to the next step – detail definition transformation to code. The input of this transformation will be the result of previous transformation (PSM), and its output will be PHP Laravel code according to the defined code metamodel and transformation rules.

4. Model Transformation and Code Generation Example

In our proposed methodology, platform independent model should be developed according to MVC architecture principles. When modelling PIM, it is important to have in mind that controller, model and view parts must be separate classes.

Model classes should have attributes and operations declared, controller classes should include operations for navigation between view classes, operations for data manipulation and other additional operations, if required. When adding operations to the controller class, it is important to assign stereotypes to them, which indicate whether it is a route type or a CRUD (Create, Read, Update, Delete) type operation. «Route» stereotype defines an operation which is intended for routing between different views.

«CRUD» stereotype defines an operation intended for manipulating the database. If controller class operation does not have any stereotype applied, this operation is considered as an additional one, which is required by the user.

In order to demonstrate the proposed solution, in Fig. 4 a fragment of PIM model for Bookstore information system is presented. The fragment contains four PIM classes: Book, BookController, AddBookView and BookListView. Each operation of BookController class has its own stereotype applied, which will further be used during transformation to PSM process.

In this case, BookController class has an assigned «Route» stereotype for the showList() and showAdd() operations, the showList() operation provides the BookListView view and the showAdd() operation provides the AddBookView view. The «CRUD» stereotype is assigned to the save() operation, which is intended for creating a new book.

After model transformation from PIM to PSM, with transformation rules and algorithms applied, PSM model should be generated, where previous classes according to the class type would become classes with applied «LaravelBlade», «LaravelModel», «LaravelRoute> or «LaravelRequest» stereotype. «LaravelBlade» stereotype defines Laravel views, «LaravelController» stereotype defines Laravel framework controller type, «LaravelModel» defines frameworks' model type, «LaravelRoutes» defines application route and «LaravelRequest» defines a class used for validation in Laravel framework.

In Fig. 5, a fragment of PSM model for bookstore information system is presented. This fragment gives an example of how the PIM model depicted in Fig. 4 might look like after transformation from PIM to PSM.

Previously defined AddBookView and BookListView views after transformation become views that have «LaravelBlade» stereotype applied, which is specific for Laravel framework. For BookController class, the «LaravelController» stereotype is applied and operations in this controller are replaced with Laravel type operations according to the stereotypes in the PIM model. In this case, PIM showAdd() operation became the PSM create() operation, because in PIM model it had «Route» stereotype assigned, and the showList() operation became index() operation also due to assigned «Route» stereotype. The save() operation was transformed to store(request) operation because of the «CRUD» stereotype, which was applied in PIM model.

The Book model class after transformation became class with «LaravelModel» stereotype. In the PSM model, a new class was created after transformation – the BookRequest class with «LaravelRequest» stereotype applied. This PSM class, after transformation to code will define the validation rules for BookController CRUD operations. The PSM model also has Routes class, that will define routes for controller classes route type methods.

After transformation from PSM to code, PHP Laravel code should be generated. Sample code fragments of BookController, Book, AddBookView, Router and BookRequest classes are presented in Fig. 6-10.

Fig. 6 shows Laravel controller code fragment for PSM model BookController. In this controller class association with Book class was generated because of the association relationship defined between BookController and Book classes in PSM model. Index(), create() and store(BookRequest request) methods were generated in this class because the corresponding operations were also defined for PSM model BookController.

Fig. 7 shows Laravel Book model code fragments for PSM model Book class. Book model would be generated from PSM model Book class which has «Lar-

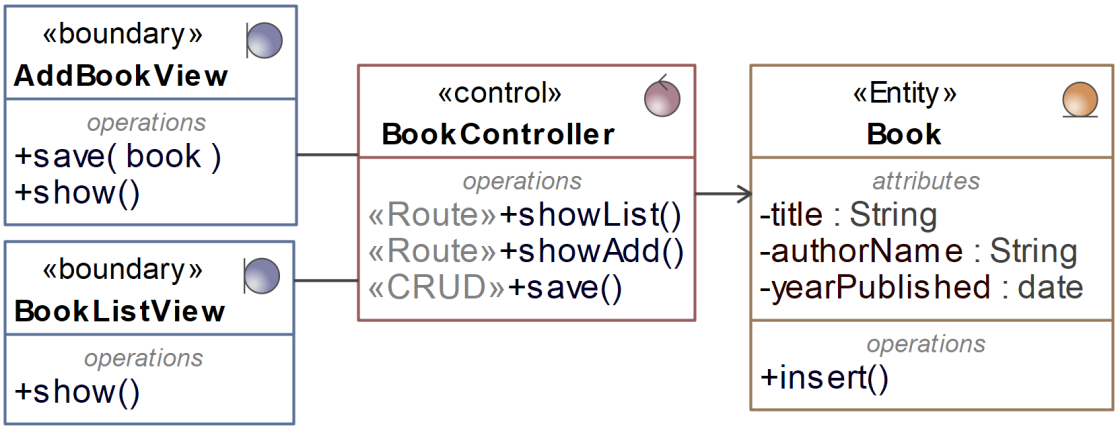

Figure 4: Example PIM fragment for BookStore Web Information System.

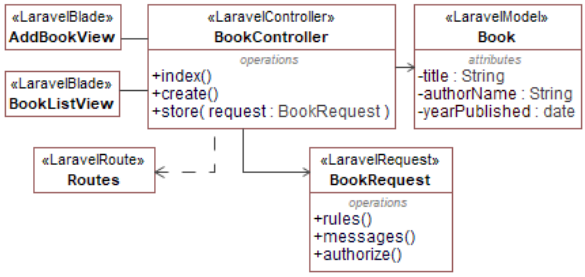

Figure 5: Example Laravel PSM fragment for BookStore Web Information System.

```
<?php
namespace App;
use
Illuminate\Database\Eloquent\Model;
class Book extends Model
{
   protected $fillable = [
        'title',
        'author_name',
        'year_published'
   ];
}…
```

Figure 7: Generated Laravel model code fragment.

```
<?php
use App\Book;
namespace App\Http\Controllers;
use App\Http\Requests\BookRequest;
class BookController extends Controller
{
public function index()
{}
public function create()
{}
public function store(BookRequest request)
{}
…
```

Figure 6: Generated Laravel controller code fragments.

```
<?php
Route::resource('books','BookController');
…
```

Figure 8: Generated Laravel route code.

avelModel» stereotype applied and this generated class would have fillable attributes defined by models' attributes.

Fig. 8 shows generated Laravel route class code fragment of routes class. Routes in Laravel maps requests, basic routing forward the requests to the associated controller. In this case, generated route is associated with BookController controller, and this route code would be generated by in PSM defined relation with BookController controller class.

Fig. 9 shows Laravel view class code fragment for AddBookView view for PSM AddBookView class. As the structure of view type classes is similar, only one of them is presented as an example. In AddBookView code, the form fields would be generated based on the attributes defined in the PSM. In this case, this view class would have title, author name, year published fields because those attributes were defined in model, also this form would be generated with route to store (BookRequest request) method.

```
<form method="post" action="{{
route('books.store') }}">

        <div class="form-group">
            <label
for="title">Title:</label>
            <input type="text"
class="form-control" name="title"/>
        </div>

        <div class="form-group">
            <label
for="author_name">Author Name:</label>
            <input type="text"
class="form-control"
name="author_name"/>
        </div>

        <div class="form-group">
            <label
for="year_published">Year
Published:</label>
            <input type="text"
class="form-control"
name="year_published"/>
        </div>
        <button type="submit" >Add
Book</button>
    </form>
 </div>
</div>…
```

Figure 9: Generated Laravel view code fragment.

Fig. 10 shows generated Laravel request validation class BookRequest code fragment. It is possible to create and use a custom form request for better Laravel application structure or more complex validation scenarios. Form request are custom request classes that contain validation logic. In this case, BookRequest class code fragment could be generated. Validation rules could be generated in rules() method of this class, where variables would be obtained through BookController relation with Book model, and "required" rules would be defined in PIM Book model attribute specification.

The presented model transformation and code generation example demonstrated how PIM and PSM models and generated code should be defined when implemented for bookstore information system. In example case, the generated code consists of five classes and one route: two view classes, one route class, controller class, model class and request validation class. All of these classes have code fragment structure that a software developer can further extend to finish the implementation of the BookStore information system.

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
class BookRequest extends FormRequest
{
   public function authorize()
   {
       return true;
   }
   public function rules()
   {
       return [
           'title' => 'required',
           'author_name' => 'required',
           'year_published' => 'required',
       ];
   }
   public function messages()
   {
       return [
           'title.required' => '',
           'author_name.required'  => '',
           'year_published.required'  => '',
       ];
   }
}…
```

Figure 10: Generated Laravel request validation code fragment.

5. Conclusions

In this paper, a methodology is proposed, which when implemented could enable transformation of web application UML models into Laravel code. While organizations nowadays seek to apply the MDA principles in information system development to reduce high technological migration costs, there is a need for a methodology and algorithms to generate code for Laravel PHP framework based information systems. Laravel framework does not have a complete methodology for managing MDA model transformations and code generation and the proposed methodology for model transformations and code generation for the PHP Laravel framework in the context of the MDA principles should help to improve the process of software development using Laravel. When implemented, the proposed methodology should reduce the migration and development costs of organizations working with PHP Laravel framework.

In the future, we are planning to develop more detailed transformation algorithms and implement the proposed methodology to facilitate the development of Laravel framework based information systems.

References

[1] G. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Matta, M. Re, L. Iess, F. Cialfi, G. De An-

gelis, D. Gelfusa, A. Pulcinelli, L. Simone, Hardware prototyping and validation of a w-i dor digital signal processor, Applied Sciences (Switzerland) 9 (2019).

[2] F. Beritelli, G. Capizzi, G. Lo Sciuto, C. Napoli, M. Woźniak, A novel training method to preserve generalization of rbpnn classifiers applied to ecg signals diagnosis, Neural Networks 108 (2018) 331–338.

[3] F. Beritelli, G. Capizzi, G. Lo Sciuto, C. Napoli, F. Scaglione, Rainfall estimation based on the intensity of the received signal in a lte/4g mobile terminal by using a probabilistic neural network, IEEE Access 6 (2018) 30865–30873.

[4] Unified Modeling Language, 2017. https://www.omg.org/spec/UML/.

[5] S. J. Mellor, K. Scott, A. Uhl, D. Weise, Model-driven architecture, in: International Conference on Object-Oriented Information Systems, Springer, 2002, pp. 290–297.

[6] C. Napoli, E. Tramontana, An object-oriented neural network toolbox based on design patterns, in: International Conference on Information and Software Technologies, Springer, 2015, pp. 388–399.

[7] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, Synthesis lectures on software engineering 3 (2017) 1–207.

[8] M. Laaziri, K. Benmoussa, S. Khoulji, M. L. Kerkeb, A comparative study of php frameworks performance, Procedia Manufacturing 32 (2019) 864–871.

[9] Laravel Documentation, 2020. https://laravel.com/.

[10] K. Benmoussa, M. Laaziri, S. Khoulji, K. M. Larbi, A. El Yamami, A new model for the selection of web development frameworks: application to php frameworks, International Journal of Electrical and Computer Engineering 9 (2019) 695.

[11] M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi, A. El Yamami, A comparative study of laravel and symfony php frameworks, International Journal of Electrical and Computer Engineering 9 (2019) 704.

[12] A. Kilicdagi, H. I. Yilmaz, Laravel Design Patterns and Best Practices, Packt Publishing, 2014.

[13] G. Sebastián, J. A. Gallud, R. Tesoriero, Code generation using model driven architecture: A systematic mapping study, Journal of Computer Languages 56 (2020) 100935.

[14] K. Arrhioui, S. Mbarki, O. Betari, S. Roubi, M. Erramdani, A model driven approach for modeling and generating php codeigniter based applications, Transactions on Machine Learning and Artificial Intelligence 5 (2017).

[15] S. D. Rathod, Automatic code generation with business logic by capturing attributes from user interface via xml, in: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), IEEE, 2016, pp. 1480–1484.

[16] A. Srai, F. Guerouate, N. Berbiche, H. D. Lahsini, Applying mda approach for spring mvc framework, International Journal of Applied Engineering Research 12 (2017) 4372–4381.

[17] I. Essebaa, S. Chantit, M. Ramdani, Modar-wa: Tool support to automate an mda approach for mvc web application, Computers 8 (2019) 89.

[18] "ATL | The Eclipse Foundation, 2020. https://www.eclipse.org/atl/.