




Automatic Code Generation System for Transactional Web Applications

Hector Florez^(✉) , Edwarth Garcia, and Deisy Muñoz

Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia
haflorezf@udistrital.edu.co, eogt04@gmail.com, deisy.dymo@gmail.com

Abstract. Every day new applications appear, but several of these applications usually share a lot of features. Some applications are based on frameworks; however, most features introduce a lot of source code impacting the performance of an application especially when the application is web-based. Thus, automatic code generators have gained attention in the last years because they provide the required elements to create automatically final applications without introducing a source code that is not part of the application model. In this paper, we present an approach to generate transactional web applications based on a conceptual model as the unique input. Through the conceptual model, modelers can specify the entities, attributes, and relations of the application. Then, the approach is able to generate: (a) the source code separated in UI, business, and persistence layers, (b) the DDL (Data Definition Language) scripts for the corresponding relational database, and a script to create corresponding UML diagrams.

Keywords: Code generation · Conceptual model · Web application

1 Introduction

Nowadays, the amount of web applications developed in the world is increasing rapidly. It might produce several issues in the development process. Since projects are developed by several members of a development group, development processes usually take long time even when the project is not too large. Development time issues have been tackled by several authors and practitioners through several strategies such as code reuse, components development, framework-based development, code generation among others.

Code reuse allows developers to save time; however, usually it requires precise knowledge regarding the requirements solved by the code that is going to be reused. Components development is a great strategy because developers can use available components independently of the language; nevertheless, in most cases such components are domain specific; then, sometimes available components do not match to the desired requirements. Framework-based developments have been used in the last years for web application with good results; nonetheless, the final source code of the application usually includes a lot of instructions that

belong to the framework affecting the performance of the application. Code generation is a strategy that has been used for several purposes because it produces pure source code for final applications; however, such final applications have several but specific services; then, usually developers need to complement the generated source code in order to provide the implementation of all requirements.

In this paper, we present our strategy for developing PHP web applications based on code generation. We designed and developed the code generator project called *DevPHP*, which receives as input a conceptual model written in XML and allows generating PHP projects with a 3-layer architecture based on Bootstrap 4¹ as toolkit for supporting responsive web components and JQuery 3² as JavaScript library to support Ajax components. In addition, *DevPHP* generates the DDL (Data Definition Languages) scripts for creating the relational database. Although the script can be used to create the database in any database engine, the generated persistence layer also provides all source code for connections to MySQL³ databases. Finally, *DevPHP* also generates the UML model that includes component, class, and use case diagrams to be opened and manipulated using UML Designer⁴ by Obeo.

In order to validate our approach, we have created various XML conceptual models, which have been run through *DevPHP*. The conceptual models created represents projects in different domains and include different characteristics in order to verify that the generated PHP code corresponds to the modeled project. In addition, we measured the time required to create the model and estimated the time required to create the corresponding PHP source code in order to identify the relevance of this work.

The paper is structured as follows. Section 2 presents the related work. Section 3 illustrates the proposed approach. Section 4 presents the results Finally, Sect. 5 concludes the paper.

2 Related Work

There are some approaches related to automatic code generation of web-based applications. Some of these approaches use XML technologies as the main input for the code generation environment. For example, Li et al. [6] presented an approach using XML, XSLT templates and java language in order to generate automatically the source code of a JSP project, where repetitive patterns in the generated web-based project such as addition, modification, deleting and saving information can be reduced improving the efficiency of developers. Another example is proposed by Turau [16], who presented a framework where its specifications in interfaces for persistence and implementations for the business layer are defined in XML files generating a complete system prototype. Milosavljevic

¹ <https://www.getbootstrap.com/>.

² <https://www.jquery.com/>.

³ <https://www.mysql.com/>.

⁴ <http://www.umldesigner.org/>.

et al. [9] present another approach based on a tool that generates a set of standardized database-oriented JSP pages. The JSP pages are generated from the mapping of JavaBeans components to the database. The mapping is specified as an instance of an XML scheme document. These generated pages allow the visualization of a database table or row, as well as provide the possibility to add a new row, update or delete an existing row. Senthil [14] made a code generator implemented in C# which use as an input a model in XML that generates the data access layer code for Microsoft .NET/SQL server platform. Mbarki et al. [7] applied the Model Driven Architecture (MDA) approach in web applications. In this approach two metamodels are made, the first is a meta model to manage the UML source; the second metamodel is responsible to generate an application using MVC2 architecture. Later, mapping rules are used as a transformation algorithm which allows generating an XML file containing all actions, forms, and the JSP pages to generate the necessary code of the application.

Some other approaches use different strategies for the input of the code generator. For instance, Mgheder et al. [8] describe a different approach to generate web user interfaces. In this approach, the web user interface is generated based on metadata hosted in tables of a generic database. To access the database, they use ADOdb, which is a PHP library that generalizes the database connection. By using the ADOdb library, the application can access to the database metadata that contains the information regarding the tables in the database. The flow to get web user interfaces can be summarized in: (a) create a database based on system requirements; (b) get the metadata from Information Schema or directly from the tables of the database; (c) translate the native type data from the database into a generic meta-type table; (d) map the data from the tables obtained in the last step; and (e) get the user interface without behavior or customize properties.

Nadkarni et al. [10] describe WebEAV, which is a generic framework for web development in applications that possess an entity-attribute-value (EAV) component. This database architecture is widely used in clinical data repositories. It addresses the problem of saving data on several thousand potential parameters for a patient across all clinical specialties. The EAV design has a single table that records data as one row per each action. Each row contains the entity, attribute, and value information about the entity. The main objective of the framework consists in the automatic generation of forms based on EAV data. These forms must have different functionalities and need to be responsive according to the business requirement. Furthermore, this framework generates the forms based on metadata attributes in a schema. Besides, all these data are processed through several rules depending on the system complexity. This approach develops Web forms based on metadata extracted from the database. Sanchez et al. [12, 13] present a framework to develop PHP web applications based on 3-layer architecture and Model View Controller architectural pattern. Albhah and Ridley [1] et al., Propose a framework that allows the generation of Web forms from the use of common sense rules and domain specific rules based on a database metadata. RuleML (Rule Markup Language) is the format used to represent the

rules taken from the metadata. The implementation begins creating a prototype of database, from which the common rules and the domain specific rules will be created taking its database metadata. Then a PHP script tests which rules have to be implemented depending the database metadata tables to generate its respective Web form.

These approaches are just focused in the automatic code generation on concrete points of a web. However, our approach is centred on creating a complete web application where business, persistence and user interface layers are generated from the input of a XML conceptual model.

Finally, some other approaches are based on the concepts related to Model Driven Engineering (MDE), where conceptual models must conform to desired metamodels. For instance, Sanchez et al. [11] propose an approach based on Model Transformation Chains (MTC) to generate source code for configuring peripherals in mobile applications. Although this approach is very interesting, it demands the creation of a metamodel in order to abstract the features of the domain. In addition, Florez et al. [3] present a MDE approach to generate the required source code to connect unitary reusable components in order to produce a particular web application. Our approach does not need the creation of a metamodel and is able to generate the final source code of a web application.

3 Proposed Approach

We proposed a code generation strategy based on conceptual models written in XML presented in Fig. 1. In this strategy, the modeler is the person who creates the XML model. Later, he runs *DevPHP*, which uses the XML model to generate PHP source files that compose the PHP project.

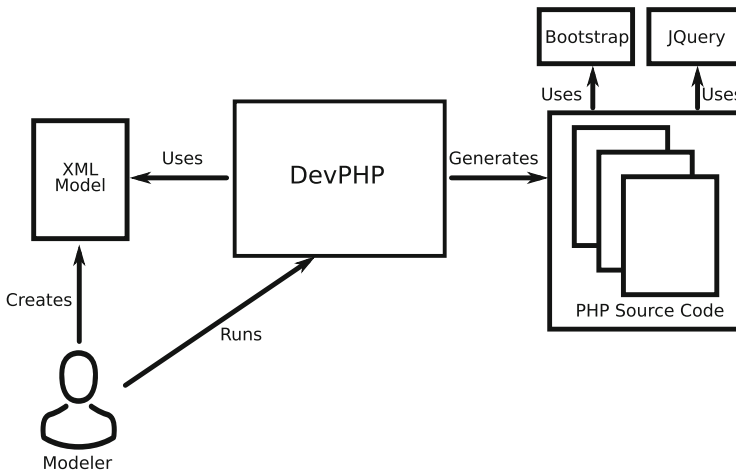


Fig. 1. Code generation strategy.

The XML model follows a specific structure based on the following XML elements:

1. **model**. Is the main XML element that contains the attributes: (a) **name**, which specifies the name of the PHP project; (b) **acronym**; (c) **description** that allows including a short text regarding the project; and (d) **language**, which defines the language of the user interface for the generated PHP project, albeit the source code always is generated in English.
2. **entity**. This element allows modelers to create the concepts involved in the project. It has the attributes: (a) **name**; (b) **actor**, which is boolean and determines whether the entity is an actor; (c) **menuDeploy** that allows excluding an entity from the menu deployed in the user interface; and (d) **delete**, which is also boolean and defines whether the information of this entity in the generated PHP project can be deleted.
3. **attribute**. This element allows including attributes to entities and must be included in the context of the element **entity**. It has the following attributes: (a) **name**; (b) **type** that can be string, text, int, date, email, or password; (c) **mandatory**, which establishes whether the attribute is mandatory in the generated PHP project, when creating or editing information; (d) **length** for setting the length of the attribute; however, when the length is not provided, the attribute will have 45 characters by default; (e) **url**, which is boolean and is used to specify that the attribute contains an url; (f) **visible**, which is boolean and serves to show or hide desired attributes for searching services in the generated PHP project; and (g) **image**, which is boolean and allows including PNG files in the generated PHP project.
4. **relation**. This element must be included in the context of the element **entity** and serves to include one relation to another entity. It has the attributes: **cardinality** that can have the values *1* or *** and **entity** that established the target entity of the relation.
5. **service**. This element must be included in the context of the element **entity** only when the attribute **actor** in the element **entity** has the value *true*. This element serves to define which entities can be controlled by a desired actor through the boolean attributes: **create**, **get**, **edit**, and **delete**.

By default, *DevPHP* includes the *entity Administrator*, which is used to provide all required services to manipulate all information related to the generated PHP project. Furthermore, for each entity that is acting as actor, *DevPHP* includes the **entity Log**. For instance, for the entity *Administrator*, it includes the entity *LogAdministrator*. The entities *Log* allows storing the actions made by actors in the generated PHP project. The log includes by default: action, information regarding the action, date, time, operating system, IP address, and browser.

Listing 1.1 presents a fragment of the XML model for a project that we called *RIS (Research Information System)*. *RIS* is intended to manage research information of a research group. Thus, *RIS* has the entities: *Researcher Role*, *Researcher*, *Book*, *Book Chapter*, *Paper*, *Software*, and *Project*.

Listing 1.1. Fragment of XML Model

```

1 <model name="Research Information System" acronym="RIS"
  description="Research Information System allows managing the
    research information of a research group" language="en" >
2   <entity name="ResearcherRole" delete="true" >
3     <attribute name="name_en" type="string" mandatory="true" />
4     <attribute name="name_es" type="string" mandatory="true" />
5     <relation cardinality="*" entity="Researcher" />
6   </entity>
7   <entity name="Researcher" actor="true" >
8     <attribute name="name" type="string" mandatory="true" />
9     <attribute name="lastName" type="string" mandatory="true" />
10    <attribute name="email" type="email" />
11    <attribute name="password" type="password" />
12    <attribute name="picture" type="string" image="true" visible="false"
13      />
14    <attribute name="isi" type="string" length="200" url="true"
15      visible="false" />
16    <attribute name="scopus" type="string" length="200" url="true"
17      visible="false" />
18    <relation cardinality="1" entity="ResearcherRole" />
19    <relation cardinality="*" entity="PaperResearcher" />
20    <service entity="Researcher" create="false" get="true" edit="false"
21      delete="false" />
22    <service entity="Paper" create="true" get="true" edit="true"
23      delete="false" />
24  </entity>
25  <entity name="Paper" delete="true" >
26    <attribute name="title" type="string" mandatory="true" length="200"
27      deploy="true" />
28    <attribute name="authors" type="string" mandatory="true"
29      length="200" />
30    <attribute name="journal" type="string" length="100" />
31    <attribute name="issn" type="string" nowrap="true" />
32    <attribute name="volume" type="string" visible="false" />
33    <attribute name="pages" type="string" visible="false" />
34    <attribute name="year" type="int" />
35    <attribute name="doi" type="string" length="200" url="true" />
36    <relation cardinality="*" entity="PaperResearcher" />
37  </entity>
38  <entity name="PaperResearcher" delete="true" >
39    <relation cardinality="1" entity="Paper" />
40    <relation cardinality="1" entity="Researcher" />
41  </entity>
42 </model>

```

This fragment of the XML model just describes the entities *ResearcherRole*, *Researcher*, *Paper*, and *PaperResearcher*. The entity *PaperResearcher* is used to make a many-to-many relation between the entities *Paper* and *Researcher*. The fragment of the XML has the following lines:

1. Line 1 includes the element `model` with its corresponding attributes
2. Line 2 has the element `entity` with the concept *ResearcherRole*, which can be deleted by administrators and is intended to classify researchers in the system. In addition, lines 3 and 4 have the attributes *name_en* and *name_es* for including the name of researcher roles in English and Spanish respectively, while line 5 has a relation to the entity *Researcher* with cardinality *** in order to represent that one researcher role can have many researchers.
3. Line 7 has the element `entity` with the concept *Researcher*, which is actor. It includes the attributes *name*, *lastName*, *email*, *password*, *picture*, *isi*, and *scopus* specified in lines 8 to 14. Moreover, lines 15 and 16 describes relations to *ResearcherRole* with cardinality *1* and *PaperResearcher* with cardinality *** indicating that one researcher is related to one researcher role and one researcher can have many papers. Furthermore, lines 17 and 18 have the element `service`. The first one, has the attribute `entity` with the value *Researcher* in order to define that one researcher can consult all researchers, but cannot create, edit, nor delete researcher. The second one, has the attribute `entity` with the value *Paper* in order to define that one researcher can create, consult, and edit papers, but cannot delete any paper.
4. Line 20 has the element `entity` with the concept *Paper*, which can be deleted by administrators. In lines 21 to 28, it has the attributes *title*, *author*, *journal*, *issn*, *volume*, *pages*, *year*, and *doi*. Finally, line 29 has a relation to *Paper-Researcher* with cardinality *** indicating that one paper has been written by many researchers.
5. Line 31 has the element `entity` with the concept *PaperResearcher*, which can be deleted by administrators and includes in lines 32 and 33 relations to *Paper* and *Researcher*

3.1 XML Model Validation

In order to generate a suitable project, the XML model that allows generating the project must be properly organized, i.e., the entities, attributes of these entities and relations have to be written with correct syntax and semantic. *DevPHP* has a function that validates the XML model order, syntax, and semantics. The explanation of this validation service is described as follows:

1. As the XML model contains n number of elements, it must have entities, attributes, and relations (between entities) correctly arranged. The approach used to organize the XML model is the use of a XSD file. This file can establish the labels order of the XML model, the required parameters of the corresponding label, the correct data type and the correct name of these parameters.
2. After this arrangement, the validation service evaluates the correct writing of the parameters that have each general label (e.g the entity label has “name” and “delete”) and the values of the parameters (e.g “actor” parameter’s values must be true or false).
3. Later, it evaluates repetitions of general labels in the XML model.
4. Finally, it validates the correct definitions of the relations between entities.

Listing 1.2 presents a fragment of the XSD file that is used to validate the XML model file.

Listing 1.2. Fragment of XSD validator file

```

1 <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
2   <xs:element name="model">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="customize" type="typeCustomize" minOccurs = "1"
6           maxOccurs = "1"/>
7         <xs:element name="entity" type="typeEntity" minOccurs = "1"
8           maxOccurs = "unbounded"/>
9       </xs:sequence>
10      <xs:attribute name = "name" type = "xs:string" use="required"/>
11      <xs:attribute name = "acronym" type = "xs:string" use="required"/>
12      <xs:attribute name = "description" type = "xs:string"
13        use="required"/>
14      <xs:attribute name = "language" type = "xs:string" use="required"/>
15      <xs:attribute name = "br" type = "xs:string" use="optional"/>
16      <xs:attribute name = "text" type = "xs:string" use="optional"/>
17    </xs:complexType>
18  </xs:element>

```

This fragment of the XSD validation file has the following lines:

1. Line 1 indicates the root element of the schema, which defines elements and data types used in the schema based on the definitions established in the url <http://www.w3.org/2001/XMLSchema>.
2. Line 2 defines the root element of the XML model. The *element* tag defines the main definition of the XML model, which in this case is `Model`.
3. Line 3 defines a complex type, which is an XML that contains other elements or attributes. In this case, the complex type element refers to elements and attributes that the `model` tag contains in the XML model.
4. Line 4 defines a sequence of elements, in this case, the XML model contains n number of elements and m_n number of attributes per element.
5. Lines 5 and 6 define the correct order of `customize` and `entity` in the XML model. Additionally `minOccurs` and `maxOccurs` define the number of elements `customize` and `entity` that the XML model can contain. Since `entity` contains n number of attributes and relations, these elements are defined in `typeEntity` that does not appear in the current fragment.
6. Lines 8 to 13 define `attribute` elements that represent attributes inside element tags in the XML model. The XSD file defines the element name, data type that contains this element and the required use. Besides, the order of the attributes is the same that must appear in the XML model.

4 Results

We created a full conceptual model for *RIS*, which is a system to manage research information that we have introduced in the previous section. This model includes entities with the concepts: *Researcher Role*, *Researcher* (as actor), *Book*, *Book Chapter*, *Paper*, *Software*, and *Project*. However, it includes additional concepts that serve to relate the previous concepts. These entities are: *BookResearcher*, *BookChapterResearcher*, *PaperResearcher*, *SoftwareResearcher*, *ProjectResearcher*, *BookProject*, *BookChapterProject*, *PaperProject*, and *SoftwareProject*.

When running the conceptual model using *DevPHP*, it automatically includes entities with the concepts *Administrator*, *LogAdministrator*, and *LogResearcher*. The generated PHP project will include three main directories for organizing the project in three layers: UI (user interface), business, and persistence. The UI layer includes all PHP files for the front end of the project. In this layer, *DevPHP* generates one directory for each concept of the conceptual model and inside the folder all PHP files related to the corresponding concept will be created. The business layer includes PHP files for the back end with one class for each concept of the conceptual model. Finally, the persistence layer includes PHP files with one class for each concept; however, a class called *Connection* is generated, which includes services to connect the project to MySQL databases. In addition, an additional file is created with the SQL script of the corresponding relational database.

Moreover, additional folders are created to support the generated project. Those folder are: *css* for Bootstrap cascade style sheet files, *js* for Bootstrap, JQuery, and Validator Javascript files, *img* for images, and *uml* for creating the UML model to be used in UML Designer by Obeo.

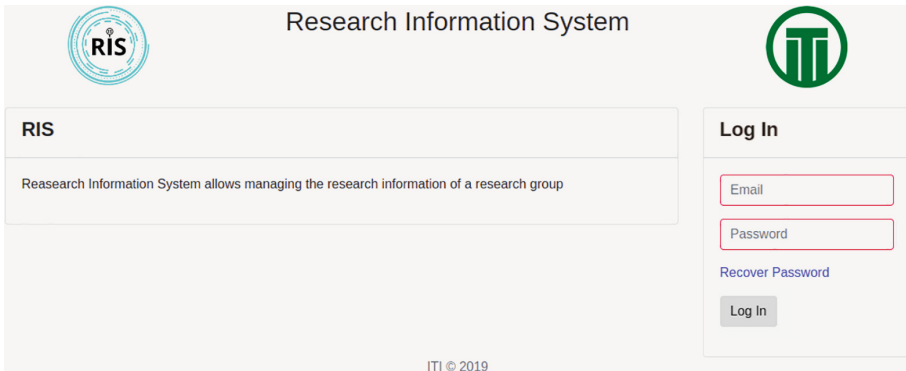


Fig. 2. Index page of *RIS*.

Figure 2 presents the index page of the generated PHP project *RIS*. In this page, actors can log in through their email and password using the card located at the right. It also offers the option to recover password. This option sends

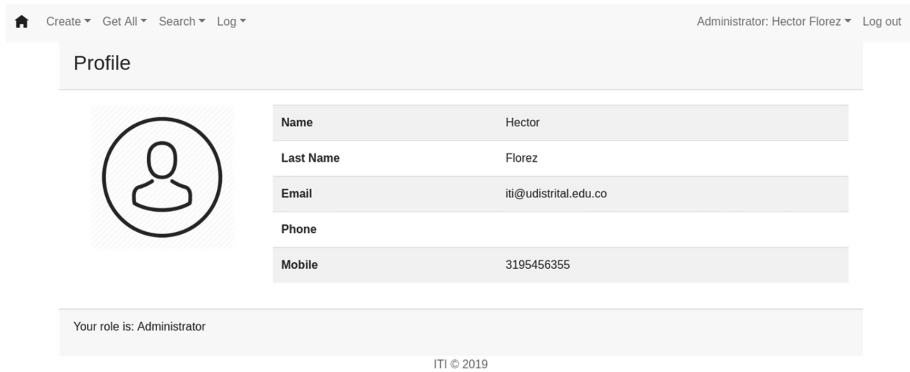


Fig. 3. Session page of administrator.

a new random password to the email registered by the user that is requesting the password recovery. In the card located at the center, it shows the project description that is included in the XML conceptual model. Finally, at the top, it shows the project's name, the project's logo and the logo of our research group. The generated project's logo is a default image that can be replaced in the generated project. Once an actor logs in the system, there is a session page with information of the actor and a menu with the available services included in the XML conceptual model. Figure 3 presents a screenshot of the session page. In particular, the administrator has all services of the project; nevertheless, services for deleting information must be defined in the XML conceptual model. The menu for the administrator includes the following options:

- **Create.** This option allows creating a registry of a desired concept included in the XML conceptual model.
- **Get All.** This option presents all registries of a desired concept included in the XML conceptual model. In this option, for each registry, there are some icons to offer services such as edit, delete, view more information that serves when the concept has a lot of attributes, get all registries of a related concept, and insert a new registry of a related concept.
- **Search.** This option allows finding results that match to one searching word that must have more than 3 characters. These results have the same services presented in the **Get All** option.
- **Log.** This options allows finding the actions made by actors in the systems. Each registry includes the action, data involved in the action, date, time, IP Address, Operating system, and browser.

Another result provided by *DevPHP* is an UML script that contains the corresponding class diagram, use case diagrams, and components diagram. Figure 4 presents a fragment of the generated class diagram for the project *RIS*. It just includes the classes *ResearcherRole*, *Researcher*, *LogResearcher* (not included in the XML conceptual model, but generated because Researcher is an actor),

Paper, and *PaperResearcher*. Every class include all attributes defined through the conceptual model and all methods required to manipulate the corresponding data regarding each concept.

4.1 Results Validation

We have validated our approach by executing five conceptual models of different project contexts. In addition, we analyzed the results of the five generated web applications. The description of the five cases are as follows:

- The first case is a system, which allows managing research information of a research group called *RIS*, which have been introduced in previous sections.
- The second case corresponds to a project for managing information and processes for academic accreditation and self evaluation.
- The third case is a system for managing information of employees fulfillment industrial enterprises.
- The fourth is a document management system specialized for managing thesis documents in a university.
- The fifth is a system for syllabus information of universities academic programs.

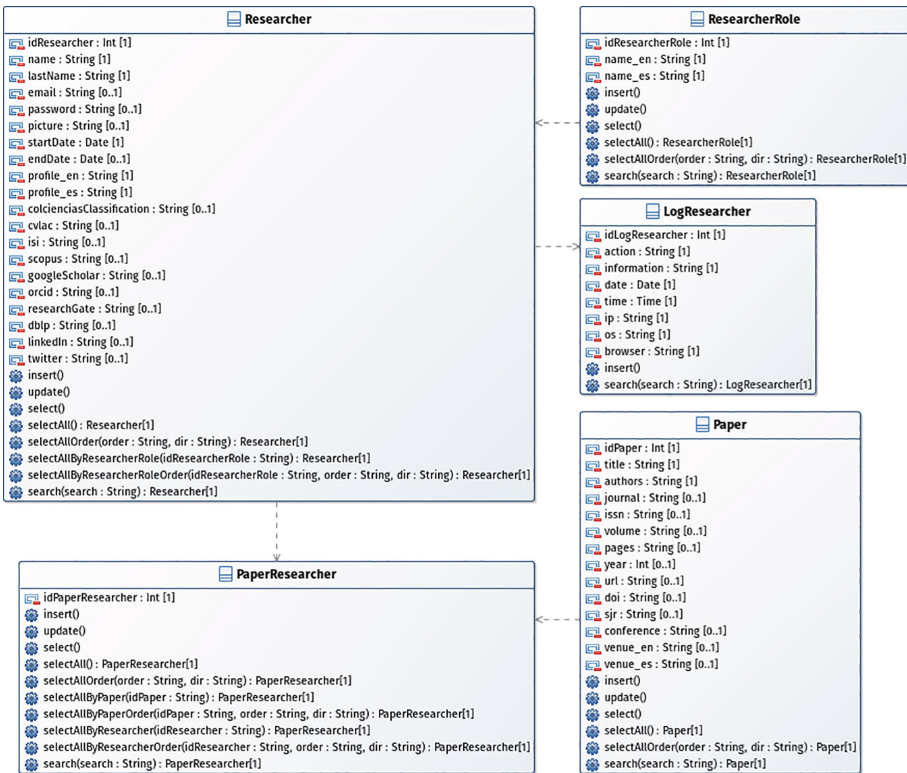


Fig. 4. Generated class diagram.

Table 1 presents the number of entities, attributes, relations and total lines written for the XML model for every case web application, as well as the total number of PHP files and code lines generated by *DevPHP*.

Table 1. Result validation.

Application	Entities	Attributes	Relations	XML lines	PHP Files	Code lines
Web App 1	16	54	18	144	145	16035
Web App 2	15	35	18	117	162	16350
Web App 3	9	32	8	71	94	9791
Web App 4	6	23	5	57	98	9261
Web App 5	6	20	5	49	82	8557

Figure 5 represents the results given in Table 1. Every orange circle represents one of the five web applications generated by *DevPHP*. Inside of each one, there are two light orange circle, where the right one represents the input data included in the XML conceptual model and the left one represents the output data generated by *DevPHP*.

On the one hand, the small yellow circle represents the amount of entities, while the small light purple circle represents the amount of the XML lines. On the other hand, the light green circle represents the generated PHP files, while the big light blue circle represents the total code lines.

We decided to take these values because the PHP files are generated from the different entities features included in the conceptual model. In the same way, we made the comparison with the resulting XML lines and code lines. With this in mind, we can observe how much source code *DevPHP* generates just using the corresponding XML conceptual model in order to offer not only a web application written in PHP, but also SQL, Javascript, CSS and UML scripts.

Also, in order to compare the cost and effort of the previously mentioned web applications, we used the function points model approach to verify the function points of each web application case.

Function Point Analysis (FPA) [15] is a method used for measuring the functional size of a software project. IFPUG [2] is a recognized standard that specifies the use of FPA model. In this model, a software system consists of five components that provide processing information to the user: External Input (EI), External Output (EO), Internal Logical File (ILF), External Logical File (EIF), External Inquiry (EQ). In order to identify these instances, they are classified in complexity levels (Low, Average, High) and the number of each instance is multiplied with the complexity level. For each project, we defined the complexity as Average for every item in the FPA model. Equation 1 is used for calculating the Unadjusted Function Point.

$$UFP = \sum_{i=1}^5 \sum_{j=1}^3 x_{ij} \times w_{ij} \quad (1)$$

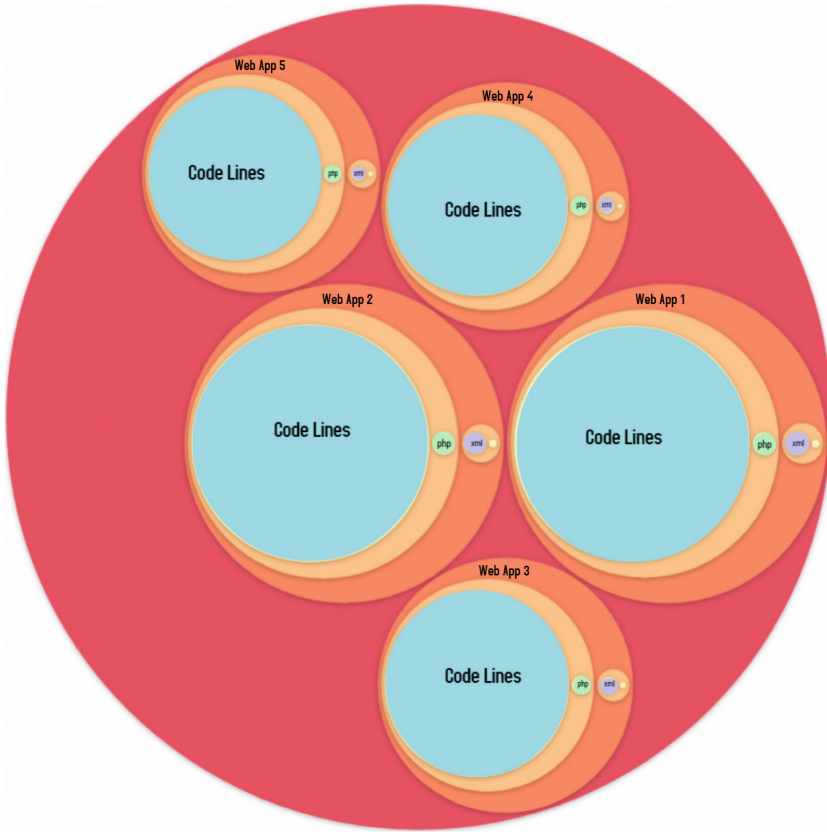


Fig. 5. Result validation

The value for x_{ij} is the number of user function type and w_{ij} is the number of complexity weight. This values can be found in IFPUG table of standard values for the different function type. In this scenario, we define the number of EI, EO and EQ with principals entities for every system with a low complexity.

The calculation of technical complexity adjustment is given by another table that describes features of the operational environment system. This table can be found as well in IFPUG table of Value Adjustment Factors for FPA model, called (VAF) [4]. All web applications cases share similar features related to the environment; consequently, we defined a default number for each of them as the sum of these 14 factors. This number is called *TDI* (Total degree of influence). Equation 2 calculates the Technical Complexity Adjustment, while Eq. 3 calculates the function points.

$$TCA = 0.65 + 0.01 \times TDI \tag{2}$$

$$FP = UFP \times TCA \tag{3}$$

Finally, using the macro-estimate technique Ball-park or Indicative Estimate [5], which estimates the effort by using function points values, we used the Eq. 4 to establish the effort of every web application.

$$Ef = \frac{FP}{150} \times FP^{0.4} \quad (4)$$

Where Ef is the effort measured in staff month. This measure indicates that 1 staff month is equivalent to 174 h.

The relation between function points and effort for each web application is presented in Table 2. The second column designates the function points value calculated through the Eq. 3. The third column shows the effort measured in staff month determined through the Eq. 4. The last column describes the person hours effort through the month staff value.

Since *DevPHP* works based on a XML file that represents a conceptual model of a system, the required time to make a complete project is given by the requirements analysis, the abstract representation of the system, as well as the constructions of every entity, attributes, and relation between entities. Activity for creating entities spend around ten minutes including the definition of its attributes. The time to evaluate and define the relations can spend around ten minutes considering many to many or many to one relations. Thus, the development effort for the web application cases using *DevPHP* are shown in Table 3.

From the estimation of function points and the month staff value shown in Table 2, we determine that the time to develop the current web application cases described in Table 1 can vary between 239.3 and 516 person-hours. This value is given by the complexity of the web application and it can increase depending of the business requirement. In addition, the time spent to develop these web application cases using *DevPHP* was between 2.1 and 5.6 person-hours.

With this in mind, the time to develop some of these web applications using *DevPHP* or pure development can vary between 200 and 500 h (e.g., the time using *DevPHP* to develop the Web Application 1 was 5.6 h; nevertheless, based on the estimated function points, the time to develop the same web application is 516.9 h), which is an important effort saving and clearly a huge cost saving.

Table 2. Development effort estimation.

Application	Function points	Staff month effort	Person-hours
Web App 1	78	3	516.9
Web App 2	62	2.1	370.6
Web App 3	53	1.7	302.9
Web App 4	45	1.4	239.3
Web App 5	45	1.4	239.3

Table 3. Development effort estimation using *DevPHP*.

Application	Person-hours
Web App 1	5.6
Web App 2	5.5
Web App 3	2.8
Web App 4	2.2
Web App 5	2.1

5 Conclusions

DevPHP is a tool that allows the minimization of time and effort in the development of web applications built using PHP. In such a way, from the construction of a consistent conceptual model *DevPHP* can generate a web application that achieves the majority of requirements that a transactional web application could have.

Projects generated by *DevPHP* are scalable and flexible considering that *DevPHP* has been developed to generate projects with a 3-layer architecture; therefore, the project can be edited by the developer depending on specific requirements, either for creating new services or editing and deleting generated project elements.

In order to make the best use of *DevPHP* it is necessary to create a consistent conceptual model. For this the modeler requires to know how to abstract the system requirements in entities, attributes, and relations. Once the web application is generated by *DevPHP*, it might be upgraded using the generated components.

References

1. Albhbah, A.M., Ridley, M.J.: A rule framework for automatic generation of web forms. *Int. J. Comput. Theor. Eng.* **4**(4), 584 (2012)
2. Cuadrado-Gallego, J.J., Rodríguez, D., Machado, F., Abran, A.: Convertibility between IFPUG and COSMIC functional size measurements. In: Münch, J., Abrahamsson, P. (eds.) PROFES 2007. LNCS, vol. 4589, pp. 273–283. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73460-4_25
3. Florez, H., Leon, M.: Model driven engineering approach to configure software reusable components. In: Florez, H., Diaz, C., Chavarriaga, J. (eds.) ICAI 2018. CCIS, vol. 942, pp. 352–363. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01535-0_26
4. Jones, C.: *Applied Software Measurement: Global Analysis of Productivity and Quality*. McGraw-Hill Education Group, New York City (2008)
5. Lawrie, R.: Using functional sizing in software projects estimating. *Charismatek Software Metrics* (2002)
6. Li, L., Yang, J., Liu, Z., Bao, L.: The research and application of web page code automatic generation technology. In: 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), pp. 5246–5249. IEEE (2011)

7. Mbarki, S., Erramdani, M.: Toward automatic generation of mvc2 web applications. *INFOCOMP* **7**(4), 84–91 (2008)
8. Mgheder, M.A., Ridley, M.J.: Automatic generation of web user interfaces in PHP using database metadata. In: 2008 Third International Conference on Internet and Web Applications and Services, ICIW 2008, pp. 426–430. IEEE (2008)
9. Milosavljević, B., Vidaković, M., Konjović, Z.: Automatic code generation for database-oriented web applications. In: Proceedings of the Inaugural Conference on the Principles and Practice of Programming, 2002 and Proceedings of the Second Workshop on Intermediate Representation Engineering for Virtual Machines, 2002, pp. 59–64. National University of Ireland (2002)
10. Nadkarni, P.M., Brandt, C.M., Marengo, L.: WebEAV: automatic metadata-driven generation of web interfaces to entity-attribute-value databases. *J. Am. Med. Inf. Assoc.* **7**(4), 343–356 (2000)
11. Sanchez, D., Florez, H.: Model driven engineering approach to manage peripherals in mobile devices. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 353–364. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_28
12. Sanchez, D., Mendez, O., Florez, H.: Applying the 3-layer model in the construction of a framework to create web applications. In: IMCIC 2017–8th International Multi-Conference on Complexity, Informatics and Cybernetics, Proceedings, vol. 2017-March, pp. 364–369 (2017)
13. Sanchez, D., Mendez, O., Florez, H.: An approach of a framework to create web applications. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 341–352. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_27
14. Senthil, J., Arumugam, S., Kapoor, S.M.A.A.: Automatic code generation for recurring code patterns in web based applications and increasing efficiency of data access code. *Int. J. Comput. Sci.* **9**(3), 473–476 (2012)
15. Symons, C.R.: Function point analysis: difficulties and improvements. *IEEE Trans. Softw. Eng.* **14**(1), 2–11 (1988)
16. Turau, V.: A framework for automatic generation of web-based data entry applications based on XML. In: Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 1121–1126. ACM (2002)