Generating a PHP Metamodel using Xtext Framework

Article *in* Procedia Computer Science · April 2020

DOI: 10.1016/j.procs.2020.03.147

2 authors:

Amine Moutaouakkil
Université Ibn Tofail
4 PUBLICATIONS **1** CITATION
SEE PROFILE

Samir Mbarki
Université Ibn Tofail
104 PUBLICATIONS **346** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 170 (2020) 838–844

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on the Advancements in Model Driven Engineering (AMDE 2020)
April 6-9, 2020, Warsaw, Poland

Generating a PHP Metamodel using Xtext Framework

Amine Moutaouakkil[a,] *, Samir Mbarki

[a]*MISC Laboratory, Faculty of Science, Ibn Tofail University, BP133, Kenitra, Morocco*

Abstract

PHP language has become the most used language for developing web applications. The representation of source code in form of models conform to a metamodel is the center of the MDA approach. This research project aims to find a way to get, using the Xtext framework and Model-driven Engineering Techniques, a PHP language metamodel and parser.

Keywords: Model Driven Architecture (MDA); Xtext; Model-driven Engineering; Web Applications; PHP

1. Introduction

MDA [1] approach provides a leading role to the models in the RE processes. PHP is de facto standard language in web development, other than websites, more web applications are developed using PHP language too. The need for modelizing PHP code has increased.

We defined an approach to get a PHP meta-model and a PHP parser in order to handle the PHP source code. This approach has taken advantage of the potential of the Xtext framework [2].

In this paper, we introduce the generation of PHP metamodel which involves the definition of PHP language elements and the writing of the PHP syntax.

The first part gives a brief presentation of the MDA concepts related to this domain. In the second part, the writing of the PHP language syntax and the generation of the PHP language meta-model and PHP language parser

* Corresponding author.
E-mail address: amine.moutaouakkil@hotmail.fr

10.1016/j.procs.2020.03.147

constitute the key concepts of the current study. A conclusion will take over most of the paper as well as future prospects.

2. Context

2.1. Model Driven Engineering

MDE [3] is a Software Engineering paradigm. MDE introduces to the Software Engineering models-based approaches instead of code-centric ones.

2.2. Model Driven Architecture

The MDA (Model Driven Architecture) is an OMG concept which suggests basing the software development on specific models using standards. With these models, we can focus on the logical conception of a program, and from them it is possible to realize transformations which generate code or other models for a particular technology or with higher abstract level. MDA defines a framework to realize these models. The models are instance of meta-models. A meta-model is the definition of a set of concepts and their relationship using a class diagram. The structuring of a meta-model is it-self provided by a meta-meta-model. A meta-meta-model is defined by MOF language [4]. The MDA is based on three modeling levels: Computation Independent Model (CIM) [5], Platform Independent Model (PIM) [6] and Platform Specific Model (PSM) [7]. To obtain a model in a level (target model) from another model from other level (source model), model transformations can be used as shown in Fig. 1.

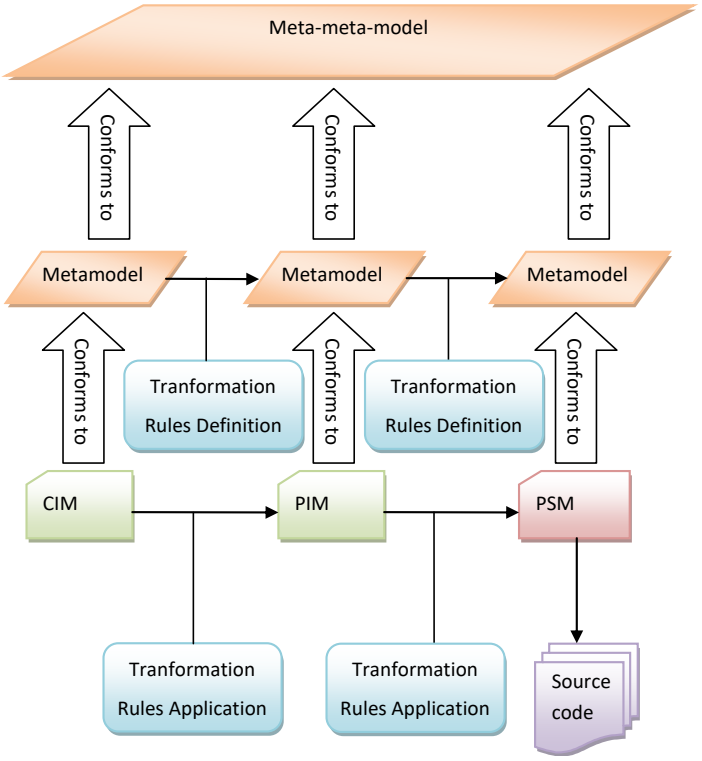

Fig. 1. Models levels and transformations between them

2.3. XText

Xtext is a framework for developing programming languages and domain specific languages (DSLs). Xtext contains a language infrastructure including parsers, compiler and interpreter.

3. Contribution

In the current study, the aim is to bridge the gap between new reengineering technologies and the exploitation of PHP considered as the de facto language and platform in web engineering. However, Establishing a PHP meta-model to represent the information in PHP code in the form of models is the first step in.

3.1. The Approach

This approach can be resumed in the use of the Xtext framework which allows defining the syntax of language, the PHP syntax was written in Xtext Grammar, after; the framework has tool that allows generating the meta-model and a parser for the PHP language.

Following this approach as shown in Fig. 2 we have written a simplified PHP syntax in Xtext Grammar, and using the Xtext plugin for eclipse to generate a simplified PHP metamodel and PHP parser.

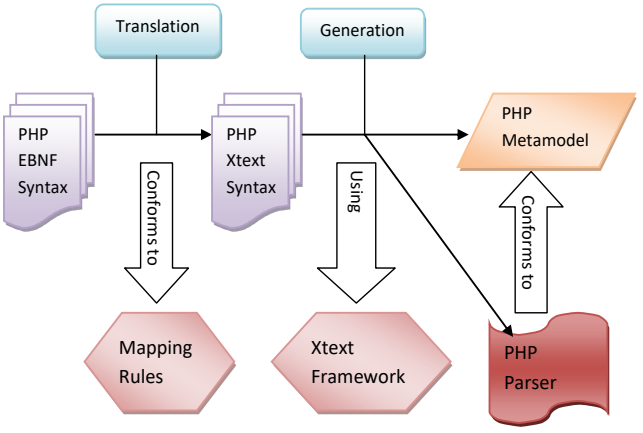

Fig. 2. Xtext based Approach

3.2. PHP EBNF syntax

In [8] a minimalist PHP syntax was written in EBNF. This EBNF syntax needs to be translated to the Xtext grammar form in order to use it in Xtext.

```
IDs report:
PHP_SOURCE_TEXT: defined in line 14, NOT USED
T_INLINE_HTML: NOT DEFINED, used 1 times
T_ENCAPSED_AND_WHITESPACE: NOT DEFINED, used 1 times
T_CHARACTER: NOT DEFINED, used 1 times
1. PHP_SOURCE_TEXT = { inner_statement3 | halt_compiler_statement2 } ;
2. halt_compiler_statement = "__halt_compiler" "(" ")" ";" ;
3. inner_statement = statement5 | function_declaration_statement8 |
class_declaration_statement9 ;
5. statement = "{" inner_statement_list4 "}" | "if" "(" expr54 ")" statement5 {
```

```
elseif_branch23 } [ else_single25 ] | "if" "(" expr54 ")" ":" inner_statement_list4 {
new_elseif_branch24 } [ new_else_single26 ] "endif" ";" | "while" "(" expr54 ")"
while_statement22 | "do" statement5 "while" "(" expr54 ")" ";" | "for" "(" for_expr40 ";"
for_expr40 ";" for_expr40 ")" for_statement16 | "switch" "(" expr54 ")" switch_case_list20 |
"break" [ expr54 ] ";" | "continue" [ expr54 ] ";" | "return" [ expr_without_variable41 |
variable58 ] ";" | "global" global_var31 { "," global_var31 } ";" | "static" static_var32 {
"," static_var32 } ";" | "echo" echo_expr_list39 ";" | T_INLINE_HTML? | expr54 ";" | "use"
use_filename7 ";" | "unset" "(" variable58 { "," variable58 } ")" ";" | "foreach" "(" (
variable58 | expr_without_variable41 ) "as" foreach_variable15 [ "=>" foreach_variable15 ] ")"
foreach_statement17 | "declare" "(" declare_list19 ")" declare_statement18 | ";" | "try" "{"
inner_statement_list4 "}" catch_branch6 { catch_branch6 } | "throw" expr54 ";" ;

6. catch_branch = "catch" "(" fully_qualified_class_name43 T_VARIABLE82 ")" "{"
inner_statement_list4 "}" ;

7. use_filename = T_CONSTANT_ENCAPSED_STRING95 | "(" T_CONSTANT_ENCAPSED_STRING95 ")" ;

8. function_declaration_statement = "function" [ "&" ] T_STRING80 "(" parameter_list27 ")" "{"
inner_statement_list4 "}" ;

9. class_declaration_statement = class_entry_type10 T_STRING80 [ extends_from11 ] [
implements_list13 ] "{" { class_statement33 } "}" | "interface" T_STRING80 [
interface_extends_list12 ] "{" { class_statement33 } "}" ;

10. class_entry_type = [ "abstract" | "final" ] "class" ;

11. extends_from = "extends" fully_qualified_class_name43 ;

12. interface_extends_list = "extends" interface_list14 ;

13. implements_list = "implements" interface_list14 ;

14. interface_list = fully_qualified_class_name43 { "," fully_qualified_class_name43 } ;

...
```

Fig. 3. : PHP EBNF syntax

3.3. Transition from EBNF to Xtext

In [9] the authors defined a method to translate EBNF to the Xtext grammar form. The transition method follows certain rules, which as shown in Tab1 map the EBNF grammar elements to Xtext elements.

Table 1. EBNF Notations to Xtext Notations Mapping

Usage	EBNF Notation	Xtext Notation
Definition	=	:
Concatenation	,	(Directly concatenate)
Termination	;	;

With obtained Xtext code, the Xtext help us to generate the meta-model and the DSL's IDE that can parse the DSL language.

3.4. Obtained PHP Xtext syntax

Fig. 4 shows the Obtained PHP xText syntax.

3.5. Obtained artifacts

The two generated artifacts are PHP parser and PHP metamodel Fig. 5.

```
grammar org.xtext.example.php.Php with org.eclipse.xtext.common.Terminals

generate php "http://www.xtext.org/example/php/Php"

PHP_SOURCE_TEXT : ( inner_statement1 = inner_statement | halt_compiler_statement )*;

halt_compiler_statement : "__halt_compiler" "(" ")" ";" ;

inner_statement : statement1 = statement
       | function_declaration_statement
       | class_declaration_statement ;

inner_statement_list : ( inner_statement2 = inner_statement )* ;

statement : "{" inner_statement_list1 = inner_statement_list "}"
       | "if" "(" expr1 = expr ")" statement2 = statement (elseif_branch1 = elseif_branch)*
(else_single1 = else_single)?
       | "if" "(" expr2 = expr ")" ":" inner_statement_list2 = inner_statement_list
(new_elseif_branch1 = new_elseif_branch)*
         (new_else_single1 = new_else_single)? "endif" ";"
       | "while" "(" expr3 = expr ")" while_statement1 = while_statement
       | "do" statement "while" "(" expr4 = expr ")" ";"
       | "for" "(" for_expr1 = for_expr ";" for_expr2 = for_expr ";" for_expr3 = for_expr ")"
for_statement1 = for_statement
       | "switch" "(" expr5 = expr ")" switch_case_list1 = switch_case_list
       | "break" (expr6 = expr)? ";"
       | "continue" (expr7 = expr)? ";"
       | "return" (expr_without_variable | variable)? ";"
       | "global" global_var ("," global_var2 = global_var)* ";"
       | "static" static_var ( "," static_var2 = static_var )* ";"
       | "echo" echo_expr_list ";"
       | expr8 = expr ";"
       | "unset" "(" variable1 = variable ("," variable2 = variable)* ")" ";"
       | "foreach" "(" (variable|expr_without_variable)
         "as" foreach_variable1 = foreach_variable ("=>" foreach_variable2 =
foreach_variable)? ")"
         foreach_statement1 = foreach_statement
       | "declare" "(" declare_list ")" declare_statement1 = declare_statement
       | ";"
       | "try" "{" inner_statement_list3 = inner_statement_list "}" catch_branch1 =
catch_branch (catch_branch2 = catch_branch)*
       | "throw" expr9 =expr ";" ;

function_declaration_statement : "function" ("&")? T_STRING
       "(" parameter_list1 = parameter_list ")" "{" inner_statement_list5 =
inner_statement_list "}" ;

class_declaration_statement : class_entry_type T_STRING
       (extends_from1 = extends_from)? (implements_list1 = implements_list)? "{"
(class_statement1 = class_statement)* "}"
       | "interface" T_STRING (interface_extends_list1 = interface_extends_list)? "{"
(class_statement2 = class_statement)* "}" ;

class_entry_type : ( "abstract" | "final" )? "class" ;
...
```

Fig. 4. Obtained PHP xText syntax

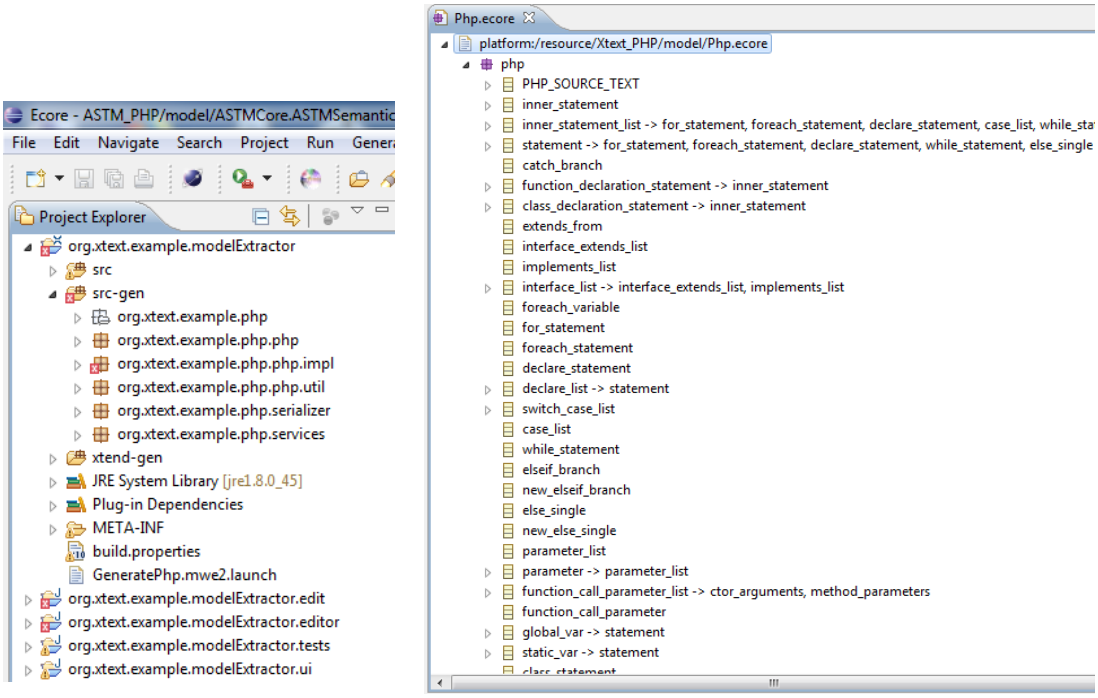

Fig. 5. Obtained artifacts : PHP parser and PHP metamodel

4. Related Works

Due to new horizons opened by the MDA, More and more research projects use the mechanisms offered by the MDA, in among these projects include eg:

- A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications [10]
- MoDisco Project [11]

1. A Model Driven Approach for Modeling and Generating PHP Codeigniter based Application (2017):

This paper applies a Model Driven approach to model the CodeIgniter PHP framework and generate CRUD applications based on this framework.

2. **MoDisco Project (2014):**

« Modisco » framework provides the capability of extracting information from Java software artifacts. The model resulting will conform to meta-model included in Modisco. Java Model for example can be obtained from the source code (based on a java meta-model). Extracted models by Modisco are Ecore models. Modisco is one of tools that have allowed applying the MDA principles in real. Unfortunately, the current Modisco version does not include any specific support for PHP code.

According to the related works we can conclude that projects that give PHP meta-model are rare. As for example the Modisco project which works just with java language metamodel.

5. Conclusion & Future Work

This paper presented an Xtext framework based approach that allows obtaining PHP language meta-model and parser from PHP language syntax.

This approach is composed of two phases: 1) definition of the PHP language elements and writing of PHP language syntax using EBNF language and the transformation of EBNF to the Xtext language, 2) Generation of PHP laguage metamodel and parser.

For the implementation of both of phases, we have used Xtext framework which has allowed us to write the syntax of PHP language and the generation of the PHP language meta-model and parser.

As a future work, we will perform the definition of other languages syntax and then do the generation of their metamodels and parsers.

References

[1] Model Driven Architecture (MDA) . [Online]. Available: http://www.omg.org/mda/

[2] Xtext project. [Online]. Available: http://www.eclipse.org/Xtext

[3] Kent. (2002) “Model driven engineering.” *Integrated Formal Methods volume 2335 of Lecture Notes in Computer Science Springer*: 286–298.

[4] Meta-Object Facility specification of the OMG. [Online]. Available: http://www.omg.org/spec/MOF/2.0/

[5] Rhazali, Hadi, Mouloudi. (2016). “Model Transformation with ATL into MDA from CIM to PIM Structured through MVC. ” Procedia Computer Science **83** : 1096-1101.

[6] Rhazali, Hadi, Mouloudi. (2015). “A methodology for transforming CIM to PIM through UML: From business view to information system view.” Third World Conference on Complex Systems (WCCS), Marrakech, Morocco : 1-6.

[7] Rhazali, Hadi, Mouloudi. (2015). “Transformation approach CIM to PIM: from business processes models to state machine and package models.” Open Source Software Computing (OSSCOM) International Conference on, Amman: 1-6.

[8] Salsi. “PHP EBNF Syntax.” [Online]. Available: http://www.icosaedro.it/articoli/php-syntax.html

[9] Yue. (2014) “Transition from EBNF to Xtext.” *MODELS-PSRC* **1258 :** 75-80.

[10] Arrhioui, Mbarki, Betari, Roubi, Erramdani. (2017) “A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications.” *Transactions on Machine Learning and Artificial Intelligence (TMLAI)* **5** (**4**) : 259-266.

[11] Modisco project. [Online]. Available: https://eclipse.org/MoDisco/