PHP modernization approach generating KDM models from PHP legacy code

Article *in* Bulletin of Electrical Engineering and Informatics · February 2020

DOI: 10.11591/eei.v9i1.1269

2 authors:

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

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

Bulletin of Electrical Engineering and Informatics
Vol. 9, No. 1, February 2020, pp. 247~255
ISSN: 2302-9285, DOI: 10.11591/eei.v9i1.1269

PHP modernization approach generating KDM models from PHP legacy code

Amine Moutaouakkil, Samir Mbarki
MISC Laboratory, Faculty of Science, Ibn Tofail University BP133, Kenitra, Morocco

Article Info

Article history:

Received Jul 15, 2018
Revised Oct 7, 2019
Accepted Nov 12, 2019

Keywords:

Abstract syntax tree meta-model (ASTM)
Architecture-driven modernization (ADM)
Knowledge discovery model (KDM)
Model-driven engineering
Reverse engineering

ABSTRACT

With the rise of new web technologies such as web 2.0, Jquery, Bootstrap. Modernizing legacy web systems to benefit from the advantages of the new technologies is more and more relevant. The migration of a system from an environment to another is a time and effort consuming process; it involves a complete rewrite of the application adapted to the target platform. To realize this migration in an automated and standardized way, many approaches have tried to define standardized engineering processes. Architecture Driven Modernization (ADM) defines an approach to standardize and automate the reengineering process. We defined an ADM approach to represent PHP web applications in the highest level of abstraction models. To do this, we have used software artifacts as an entry point. This paper describes the extraction process, which permits discovering and understanding of the legacy system. Moreover, generate models to represent the system in an abstract way.

Corresponding Author:

Amine Moutaouakkil,
MISC Laboratory, Faculty of Science, Ibn Tofail University,
BP133, Kenitra, Morocco.
Email: amine.moutaouakkil@hotmail.fr

1. INTRODUCTION

Basing more and more their business on IT solutions, organizations using web technology systems have to modernize their legacy systems to be aligned with competition. Most of websites and web applications are written in PHP language (82,5% in 2017-02-11). The need to immigrate both from and to this language is increasing. The PHP language is constantly evolving, more and more new frameworks and new standarts are available to use which make users in the need of migrating their PHP Web applications to another language or to a new version of PHP language.

It is necessary to have a standardized an automatic process of reengineering to minimize time and costs. Currently, there is an initiative to standardize and automate the reengineering process. The Architecture-driven Modernization (ADM) [1] is an Object Management Group (OMG) [2] initiative related to the reverse engineering domain. This initiative has been proposed to enhance the classical reverse engineering processes by introducing the Model-driven Architecture (MDA) [3] concepts. In the same way that the MDA approach provides a leading role to the models, the ADM approach introduces several concepts to formalize the RE [4] processes based on models too.

It is necessary to realize methods for migrating PHP Web applications, but currently there are no relied ADM based methodes making it. We defined an ADM based approach to represent PHP source code in form of KDM models. This approach has taken advantage of the potential of the Architecture Driven Modernization (ADM) to modeling the knowledge, which will be extracted from the legacy program artifacts.

In this article, we describe the generation of models to represent the legacy system at the highest level of abstraction. The rest of this article is organized as follows: section 2 describes the process based on the ADM approach and describes their different phases. In section 3, we illustrate our proposed approach by a case study. Section 4 makes the analysis of the process result. In section 5 we list some interesting related works. Finally, section 6 concludes the work and presents the perspectives.

2. RESEARCH METHOD

- MDRE

Model Driven Reverse Engineering (MDRE) [5] is the application of Model Driven Engineering (MDE) principles and techniques to Reverse Engineering (RE) in order to get model based views from legacy systems. The MDRE is based on two main phases: Model Discovery which is extracting information from source code by using parsers, and then represent this information in form of models. And Model Understanding which is applying Model to Model transformations on extracted information to get a higher abstraction level presentation of the information.

- ADM

Architecture Driven Modernization (ADM) is an initiative proposed by OMG to standardize and automate the reengineering process. ADM is based on three standards meta-models to represent the information involved in a software reengineering process. In the current study, only Abstract Syntax Tree Meta-Model (ASTM) [6] and Knowledge Discovery Meta-Model (KDM) standards are useful for the purpose. ASTM allows modeling the legacy code in form of Abstract Syntax Tree. Otherwise, KDM [7] allows defining models at a higher abstraction level representing semantic information about a software system.

2.1. Model discovery

Discovering the legacy system corresponding PSM models by analyze the source code constitutes the first step. PSM models define the structures and relationships between system elements. UML Class Diagram for example is a PSM model that can be generated by many UML tools: PowerAMC [8], Enterprise-Architect [9], Objecteering [10], Modelio [11], Papyrus [12]. Model discovery allows the extraction of information from the system and its representation in concrete models such as ASTM. To represent the information in PHP code in the form of ASTM models, we have found out that we have to write a PHP Discoverer first so that existing PHP code can be represented as a model. After dealing with the creation and operation of discoverers. It turned out that the incorporation and implementation is extremely complex.

We looked for alternative ways to transform PHP code into models. After searching and trying the available parsers for PHP, one of them was interesting, glayzzle/php-parser [13]. This parser is working well, it parses the PHP code and gets an abstract syntax tree (AST), and obtained tree is an intermediate step to get ASTM models after transformations.

2.2. Model understanding

Model understanding consists in the transformation of models to get higher abstract models. In our study, we need to make two transformations: Apply model-to-model transformation on the AST tree to get ASTM model, which is a Generic ASTM model. Apply model-to-model transformation on the ASTM model to get KDM model.

- QVT Transformation Standard

QVT (Query/View/Transformation) [14] is a standard set of languages for model transformation defined by the OMG. The QVT standard defines three model transformation languages. All of them operate on models which conform to Meta-Object Facility (MOF) 2.0 metamodels; the transformation states which metamodels are used. QVT Operational which we use, it is an imperative language designed for writing unidirectional transformations. Models extraction process a shown in Figure 1.

- ASTM

Current OMG ASTM meta-model needs to be simplified, to make the transformation easier for our example. ASTM meta-model provided by OMG and Simplified one a shown Figure 2.

- KDM

KDM meta-model provided by OMG a shown in Figure 3. KDM is a standard defined by OMG for the representation of software systems. This meta-model allows to represent systems artifacts in a high level of abstraction. The KDM specification consists of 12 packages that are arranged into the following four layers [15].

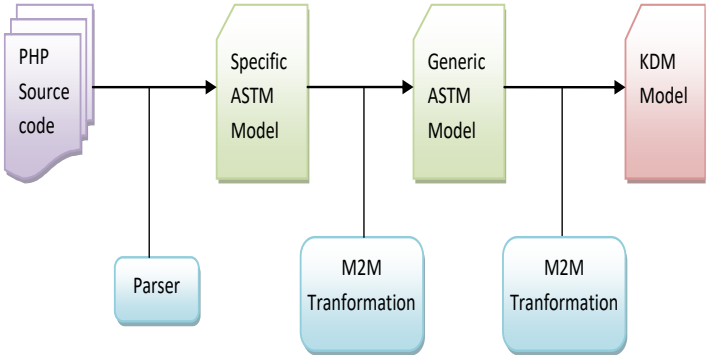

Figure 1. Models extraction process

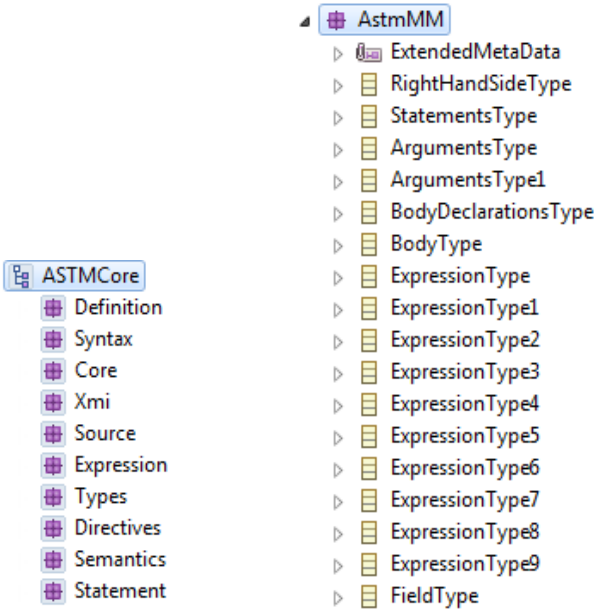

Figure 2. ASTM meta-model provided by OMG and simplified one

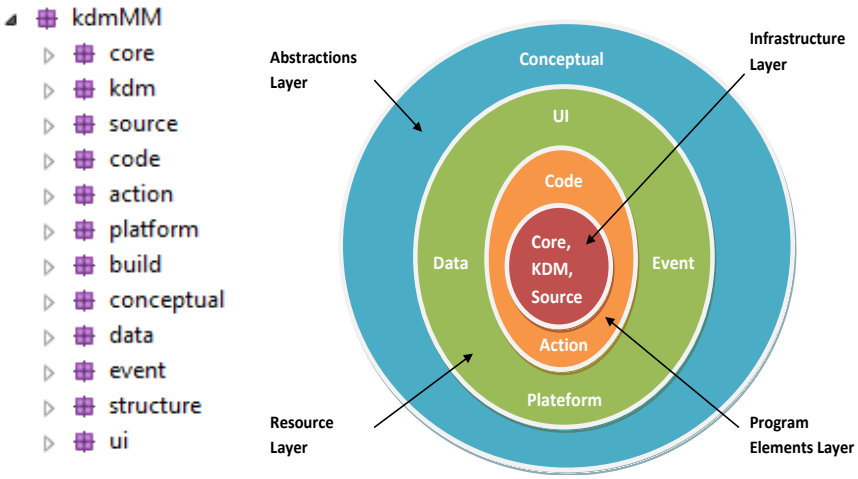

Figure 3. KDM meta-model provided by OMG

In our approach, we focus on Code package. The Code package represents programming elements as used by programming languages: data types, procedures, classes, methods, variables, etc.

- AST to ASTM QVT-O Transformation script

We have defined a mapping table between AST model elements and ASTM model elements. Based on the mapping table, we have written a QVT-O transformation script to map AST model elements to ASTM model elements. AST elements to ASTM elements mapping a shown in Table 1.

Table 1. AST elements to ASTM elements mapping

AST element	ASTM element
ChildrenType	OwnedElementsType1
BodyType	BodyDeclarationsType
ArgumentsType	ParametersType

```
modeltype AST uses 'http://AstMM';
modeltype ASTM uses 'http://AstmMM';
transformation ast2astm(in ast : AST, out ASTM);
main() {
        ast.objects()[AST::ChildrenType]->map R1();
        ast.objects()[AST::BodyType]->map R2();
        …
}
mapping AST::ChildrenType::R1() : ASTM::OwnedElementsType1 {
        if (self.kind<>'inline'){   name := self.name; }
}
mapping AST::BodyType::R2() : ASTM::BodyDeclarationsType {
        if (self.kind='property'){
        type := self.kind;
        modifier := self.map R21();
        fragments := self.map R22();
        }
        else if (self.kind='method'){
        type := self.kind;
        modifier := self.map R21();
        fragments := self.map R22();
        parameters := self.arguments.map R23();
        }
}
…
```

- ASTM to KDM QVT-O Transformation script

We have defined a mapping table between ASTM model elements and KDM model elements. Based on the mapping table, we have written a QVT-O transformation script to map ASTM model elements to KDM model elements. ASTM elements to KDM elements mapping a shown in Table 2.

```
modeltype ASTM uses 'http://AstmMM';
modeltype KDM uses 'http://KdmMM';
transformation astm2kdm(in astm : ASTM, out KDM);
main() {
        astm.rootObjects()[ASTM::Model]->map R0();
        astm.objects()[ASTM::ModifierType]->map R1();
        astm.objects()[ASTM::OwnedElementsType1]->map R2();
        astm.objects()[ASTM::OwnedElementsType1]->map R21();
        …
}
mapping ASTM::Model::R0() : KDM::code::CodeModel { name := self.name; }
mapping ASTM::ModifierType::R1() : KDM::kdm::Attribute {
value := self.visibility.toString();
```

```
}
mapping ASTM::OwnedElementsType1::R2() : KDM::code::CodeElement when {self.type<>'Class'}{
name := self.name; }
mapping ASTM::OwnedElementsType1::R21() : KDM::code::ClassUnit when {self.type='Class'}{
        var a := new KDM::code::ClassUnit();
        a.name := self.name;
        result.codeElement := a;
}
…
```

Table 2. ASTM elements to KDM elements mapping

ASTM element	KDM element
ModifierType	kdm:Attribute
OwnedElementsType1	code:CodeElement
OwnedElementsType1	code:ClassUnit
BodyDeclarationsType	code:CodeElement
ParametersType	code:ParameterUnit

3. RESULTS AND DISCUSSION

3.1. Case study example

Models will be extracted from a simple PHP class: Vegetable Class which has two properties, a constructor and two functions.

```
<?php
class Vegetable {
   private $edible;
   private $color;
   public function __construct($edible, $color) {
      $this->edible = (int) $edible;
      $this-> color = (int) $color;
   }
   public function is_edible () { return $this-> edible; }
   public function what_color () { return $this-> color; }
}
?>
```

3.2. Result

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
 <kind>program</kind>
 <children>
  <kind>class</kind>
  <name>Vegetable</name>
  <isAnonymous>false</isAnonymous>
  <extends/>
  <implements/>
  <body>
   <kind>property</kind>
   <name>edible</name>
   <value/>
   <isAbstract>false</isAbstract>
   <isFinal>false</isFinal>
   <visibility>private</visibility>
   <isStatic>false</isStatic>
  </body>
  <body>
```

```
   <kind>property</kind>
   <name>color</name>
   <value/>
   <isAbstract>false</isAbstract>
   <isFinal>false</isFinal>
   <visibility>private</visibility>
   <isStatic>false</isStatic>
  </body>
  <body>
   <kind>method</kind>
   <name>__construct</name>
   <arguments>
    <kind>parameter</kind>
    <name>edible</name>
    <value/>
    <type/>
    <byref>false</byref>
    <variadic>false</variadic>
    <nullable>false</nullable>
   </arguments>
```

AST tree

- ASTM model

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:AstmMM="http://AstmMM"
xsi:schemaLocation="http://AstmMM
platform:/plugin/org.eclipse.m2m.qvto.ast2astm/metamodel/ASTM.ecore">
  <AstmMM:OwnedElementsType1 name="Vegetable"
type="Class" />
<AstmMM:BodyDeclarationsType>
    <modifier visibility="private"/>
    <fragments name="edible"/>
    <type>property</type>
  </AstmMM:BodyDeclarationsType>
  <AstmMM:BodyDeclarationsType>
    <modifier visibility="private"/>
    <fragments name="color"/>
    <type>property</type>
  </AstmMM:BodyDeclarationsType>
  <AstmMM:BodyDeclarationsType>
    <modifier visibility="public"/>
    <fragments name="__construct"/>
    <parameters name="edible"/>
    <parameters name="color"/>
    <type>method</type>
  </AstmMM:BodyDeclarationsType>
  <AstmMM:BodyDeclarationsType/>
  <AstmMM:BodyDeclarationsType>
    <modifier visibility="public"/>
    <fragments name="isValid"/>
    <parameters/>
    <type>method</type>
  </AstmMM:BodyDeclarationsType>
  <AstmMM:BodyDeclarationsType/>
  <AstmMM:ExpressionType/>
  <AstmMM:ExpressionType operator="="/>
  <AstmMM:ExpressionType operator="="/>
  <AstmMM:ExpressionType/>
  <AstmMM:ExpressionType/>
  <AstmMM:ExpressionType/>
</xmi:XMI>
```

- KDM model

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:code="http://www.eclipse.org/MoDisco/kdm/code"
xmlns:kdm="http://www.eclipse.org/MoDisco/kdm/kdm"
xsi:schemaLocation="http://www.eclipse.org/MoDisco/kdm/code
platform:/plugin/org.eclipse.m2m.qvto.astm2kdm/metamodel/KDM.ecore#//code
http://www.eclipse.org/MoDisco/kdm/kdm
platform:/plugin/org.eclipse.m2m.qvto.astm2kdm/metamodel/KDM.ecore#//kdm">
  <code:CodeModel name="Vegetable_Project"/>
  <kdm:Attribute value="private"/>
  <kdm:Attribute value="private"/>
  <kdm:Attribute value="public"/>
  <kdm:Attribute value="public"/>
  <kdm:Attribute value="public"/>
  <code:CodeElement/>
  <code:CodeElement/>
  <code:CodeElement/>
  <code:CodeElement/>
  <code:CodeElement/>
  <code:ClassUnit>
    <codeElement xsi:type="code:ClassUnit" name="Vegetable"/>
  </code:ClassUnit>
  <code:CodeElement name="edible"/>
  <code:CodeElement name="color"/>
  <code:CodeElement name="__construct"/>
  <code:ParameterUnit name="edible"/>
  <code:ParameterUnit name="color"/>
  <code:CodeElement/>
  <code:CodeElement name="is_edible"/>
```

```
  <code:ParameterUnit/>
  <code:CodeElement/>
  <code:CodeElement name="what_color"/>
  <code:ParameterUnit/>
  <code:CodeElement/>
</xmi:XMI>
```

3.3. Related works

More and more research projects use the mechanisms offered by the MDA, in among these projects include eg:

- ADM-Based Hybrid Model Transformation for Obtaining UML Models from PHP Code [16]
- Improving Consistency of UML Diagrams and Its Implementation Using Reverse Engineering Approach [17]
- Model Driven Approach based on Interaction Flow Modeling Language to Generate Rich Internet Applications [18]
- Java Swing Modernization Approach: Complete Abstract Representation based on Static and Dynamic Analysis [19]
- A Model Driven Reverse Engineering Framework for Extracting Business Rules out of a Java Application [20]
- Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration [21]
- Combining UML Class and Activity Diagrams for MDA Generation of MVC 2 Web Applications [22]
- MoDisco Project [23]

ADM-Based Hybrid Model Transformation for Obtaining UML Models from PHP Code (2019). This paper describes a model transformation process performing reverse engineering of PHP web applications. Model to model transformations are implemented using ATL [24] (Atlas Transformation Language). The obtained models are expressed in UML.

Improving Consistency of UML Diagrams and Its Implementation Using Reverse Engineering Approach (2018). This paper describse the development of a tool that improve the consistency between Unified Modeling Language (UML) design models and its C# implementation using reverse engineering approach. This approach also takes code source as input applying to it the reverse eenginering process, and then it takes the models as input too to verivying the consistency. This approach does not generate models as artifacts.

A Model Driven Approach based on Interaction Flow Modeling Language to Generate Rich Internet Applications (2016). This paper presents a model driven approach to generat GUI for Rich Internet Application. Using the language IFML adopted by the Object Management Group, Query View Transformation (QVT) for model transformations and Acceleo for code generation, the approach allows to generate a RIA focusing on the graphical aspect of the application.

Java Swing Modernization Approach: Complete Abstract Representation based on Static and Dynamic Analysis (2016). This paper defines an ADM-based method to define abstract models representing the GUI knowledge and automate the generation of these models through transformation chains. A Model Driven Reverse Engineering Framework for Extracting Business Rules out of a Java Application (2012). This paper proposes a process of extracting business rules out of a Java application, by identifying business rules from the source code. And presenting the extracted business rules through models.

Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration (2013). This paper defines an ADM-based method for migrating CMS-based Web applications. This method is focused on open-source CMS which are implemented in PHP. It makes the implementation of the reverse engineering phase: ASTM models are extracted from the PHP code by text-to-model (T2M) transformation implemented by a source code parser, KDM models are generated from the previous ASTM models by means of M2M transformations. and by using M2M transformations the CMS Model which conforms to the CMS Common Metamodel is generated. M2M transformations are implemented using ATL Transformation Language.

Combining UML Class and Activity Diagrams for MDA Generation of MVC 2 Web Applications (2013). This paper defines an MDA based process that integrates UML class diagram and Activity diagram, in order to generate MVC2 web model of e-commerce web applications. Model-to-model transformations are implemented using ATL Transformation Language, Model to text transformation are implemented using JET language [25].

MoDisco Project (2014). The eclipse plugin « Modisco » provides the capability of extracting information from Java software artifacts, The model resulting will conform to meta-model included in Modisco. Model of the abstract syntax tree can be obtained first from the program (based on a generic meta-model such as OMG ASTM), After a transformation, Model of KDM meta-model is obtained; KDM allows representing the entire software system and all its entities at both structural and behavioral levels. Extracted models by Modisco are Ecore models. Modisco is one of rare tools that have allowed to apply the ADM principles in real. Unfortunately, the current Modisco version does not include any specific support for PHP code.

3.4. Analysis and discussion

In the ADM approach, KDM plays an important role. It is a meta-model that allows to represent the structural and semantic aspect of the software systems artifacts in a high level of abstraction. As a system is represented in form of KDM models, the immigration to another platform becomes easier. We can notice that obtained result corresponds to our aim goal.

According to the related works we can conclude that ADM approaches that extract ASTM and KDM models from PHP code and others platforms are rare. Modisco approach stops at the Specific ASTM level. "java" models extract by Modisco from Java projects are not generic ASTM models. The approach " Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration" is based on in the use of the Xtext [26] plugin witch allows defining the syntax of language, wich involves to write the PHP syntax in Xtext Grammar. Writing the PHP syntax in xtext language is a hard and time-consuming task, this task is avoided in our approach by using glayzzle php-parser, our approach helps to gain this consumed time.

4. CONCLUSION

This paper presented an ADM based approach that allow obtaining KDM models from PHP source code. This approach is composed of two phases: 1) extracting AST tree from source code, in this phase the trees are extracted from the PHP code by text-to-model (T2M) transformations implemented by a source code parser, 2) Generation of ASTM models, KDM models are generated from the previous ASTM models by means of M2M transformations.For the implementation of the tree extraction phase, we have used Galyzzle PHP Parser which has allowed us to extract AST tree from the source code.For the implementation of the Generation of ASTM and KDM models phase we have implemented transformation rules using QVT-Operational language. The major contribution is the use of the QVT transformation language defined by the OMG. In addition, using this approach can reduce the necessary time to make the PHP web applications migration. As a future work, we will perform similar approach to other platform and compare high-level abstract results.

REFERENCES

[1] Architecture Driven Modernization (ADM). [Online]. Available: http://adm.omg.org/
[2] Object Management Object (OMG). [Online]. Available: http://www.omg.org/
[3] Model Driven Architecture (MDA). [Online]. Available: http://www.omg.org/mda/
[4] E. J. Chikofsky, J. H. Cross. “Reverse engineering and design recovery: a taxonomy," in *IEEE Software*, vol. 7, no.1, pp. 13-17, 1990.
[5] C. Raibulet, F. Arcelli Fontana and M. Zanoni, "Model-Driven Reverse Engineering Approaches: A Systematic Literature Review," in *IEEE Access*, vol. 5, pp. 14516-14542, 2017.
[6] Abstract Syntax Tree Meta-Model specification of the OMG. [Online]. Available: http://www.omg.org/spec/ASTM/1.0/
[7] Knowledge Discovery Meta-Model specification of the OMG. [Online]. Available: http://www.omg.org/spec/KDM/1.3
[8] Comsoft-direct, PowerAMC. [Online]. Available: http://www.comsoft-direct.fr/poweramc
[9] Sparx systems, Enterprise architect. [Online]. Available: http://www.sparxsystems.com.au/products/ea
[10] Objecteering, Objecteering, the model driven development tool. [Online]. Available: http://www.objecteering.com/
[11] Modeliosoft, the open source modeling environment, Modelio. [Online]. Available: https://www.modelio.org/
[12] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P.Tessier, R. Schnekenburger, H. Dubois, F. Terrier. “Papyrus UML: an open source toolset for MDA”. *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, pp. 1-4, 2009.
[13] NodeJS PHP Parser - extract AST or tokens (PHP5 and PHP7) . [Online]. Available: http://glayzzle.com/php-parser/
[14] QVT (Query/View/Transformation) standard of the OMG. [Online]. Available: http://www.omg.org/spec/QVT/1.3/

[15] KDM Technical Overview from KDM Analytics. [Online]. Available: http://kdmanalytics.com/resources/standards/kdm/technical-overview/
[16] A. Elmounadi, N. Berbiche, N. Sefiani, N. El Moukhi, "ADM-Based Hybrid Model Transformation for Obtaining UML Models from PHP Code," *International Journal of Embedded Systems (IJES)*, vol. 7, no. 1, pp. 32-41, 2019.
[17] V. Kaliappan, N. Mohd Ali, "Improving Consistency of UML Diagrams and Its Implementation Using Reverse Engineering Approach," *Bulletin of Electrical Engineering and Informatics (BEEI*). Vol. 7, no. 4, pp. 665-672, 2018.
[18] S. S. Roubi, M. Erramdani, S. Mbarki, "A Model Driven Approach based on Interaction Flow Modeling Language to Generate Rich Internet Applications," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 6, no. 6, pp. 3073-3079, 2016.
[19] Z. Gotti, S. Mbarki, "Java Swing Modernization Approach: Complete Abstract Representation based on Static and Dynamic Analysis," *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT*). Lisbon. Vol. 1, pp. 210-219, 2016.
[20] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, J. Perronnet, "A Model Driven Reverse Engineering Framework for Extracting Business Rules out of a Java Application," *Proceeding RuleML'12 Proceedings of the 6th international conference on Rules on the Web: research and applications*. *Montpellier*, pp. 17-31, 2012.
[21] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration. Evaluation of Novel Approaches to Software Engineering," 8th *International Conference (ENASE)*. Angers, pp. 241-256, 2013.
[22] M. Rahmouni, S. Mbarki, "Combining UML Class and Activity Diagrams for MDA Generation of MVC 2 Web Applications," *International Review on Computers and Software (IRECOS)*. vol 8, no. 4, pp. 949-957, 2013.
[23] H. Brunelière, J. Cabota, G. Dupé, F. Madiot, "MoDisco: a Model Driven Reverse Engineering Framework: Information and Software Technology," *Elsevier*, vol 56, no. 8, pp.1012-1032, 2014
[24] S. Mbarki, M. Rahmouni, "Validation of ATL Transformation to Generate a Reliable MVC2 Web Models," *International Journal of Engineering and Applied Computer Science (IJEACS)*, vol. 2, no. 3, pp.83-91, 2017.
[25] Eclipse Foundation, "JET." [Online]. Available: http://www.eclipse.org/modeling/m2t/?project=jet
[26] Xtext project. [Online]. Available: http://www.eclipse.org/Xtext