6ème Conférence Internationale en Automatique & Traitement de Signal (ATS-2021)
Proceedings Book Series –PBS- Vol 1, pp.27-34

An ADM-based approach for generating ASTM models from PHP code legacy

Amine Moutaouakkil #1, Samir Mbarki #2

MISC Laboratory, Faculty of Science, Ibn Tofail University
BP133, Kenitra, Morocco
1amine.moutaouakkil@hotmail.fr
2mbarkisamir@hotmail.com

***Abstract*— Most of websites are written in PHP language (82,5% in 2017-02-11) and some of them are still written in PHP4 legacy language. With the rise of new web technologies such as web 2.0, Javascript based technologies : Jquery, Bootstrap. the need to modernize PHP web sites increases.**

The migration process of a system from an execution environment to another implementation platform is performed for various reasons: more secure platform, having more possibilities such as web services etc. This process constitutes a set of complicated operations that takes time and requires effort. This problematic 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. This research project aims to find a way to represent, using the ADM approach, PHP web applications in form of ASTM models.

***Keywords*— Architecture driven Modernization (ADM), Abstract Syntax Tree Meta-Model (ASTM), Reverse Engineering, Parsing, Model-driven Engineering, web applications.**

I. INTRODUCTION

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 processes based on models too.

PHP is de facto standard language in web development, other than websites, more web applications are developed using this language. The need to immigrate both from and to this language has increased.

We defined an ADM based approach to get ASTM models from PHP models and then adapt the ASTM meta-model in order to handle the PHP concepts which do not exist in current version of the meta-model. 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 paper, we introduce the extraction of ASTM models from PHP source code which involve the adaptation of the ASTM meta-model.

The first part gives a brief presentation of the ADM initiative and presents some concepts related to this domain. In the second part, the adaptation of the ASTM meta-model and T2M/M2M transformations constitute the key concepts of the current study. A conclusion will take over most of the paper as well as future prospects.

II. CONTEXT

A. Software Reverse Engineering Process

Reverse engineering is the process of examining an already implemented software system in order to represent it in a different form or formalism and at a higher abstraction level [4].

B. Model Driven Engineering

MDE [5] is a Software Engineering paradigm. MDE introduces to the Software Engineering models-based approaches instead of code-centric ones.

C. Model Driven Reverse Engineering

MDRE [6] is the application of Model Driven Engineering (MDE) principles and techniques to 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.

D. Model Driven Architecture

The MDA (Model Driven Architecture) is an OMG concept which suggests to base 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-

ISSN: 2961-6611

model is it-self provided by a meta-meta-model. A meta-meta-model is defined by MOF language [7]. The MDA is based on three modeling levels : Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). To obtain a model in a level (target model) from another model from other level (source model), model transformations can be used.

E. Architecture Driven Modernization

Architecture Driven Modernization (ADM) is an initiative proposed by OMG to standardize and automate the reengineering process. ADM is based on seven standards meta-models to represent the information involved in a software reengineering process, but currently only three of them are available: Abstract Syntax Tree Meta-Model (ASTM) [8], Knowledge Discovery Meta-Model (KDM) [9] and Structured Metrics Meta-Model (SMM) [10]. In the current study, only ASTM and KDM standards are considered useful for the purpose. ASTM allows modeling the legacy code in form of Abstract Syntax Tree. Otherwise, KDM allows defining models (KDM models) at a higher abstraction level representing semantic information about a software system.

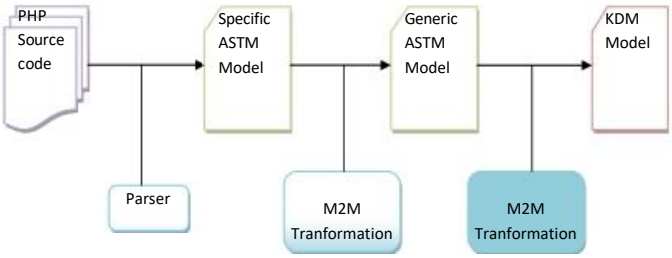

Fig. 1 Models Extraction Process

F. QVT Transformation Standard

QVT (Query/View/Transformation) [11] 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. A transformation in any of the three QVT languages can itself be regarded as a model, conforming to one of the metamodels specified in the standard.

- QVT-Operational is an imperative language designed for writing unidirectional transformations.
- QVT-Relations is a declarative language designed to permit both unidirectional and bidirectional model transformations to be written.
- QVT-Core is a declarative language which has not yet a full implementation.

III. CONTRIBUTION

In the current study, the aim is to establish the link between new technologies involved in the reengineering context and the exploitation of PHP technologies considered as the de facto language and platform in web engineering.

A. The Approach

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 an Ecore 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 XMI, or ecore models.

After searching and trying the available parsers for PHP, one of them was interesting, glayzzle/php-parser [12]. This parser is working good, it parses the PHP code and gets an abstract syntax tree (AST), obtained tree is an intermediate step to get ASTM models after transformations.

The approach is covering the two phases of Model Driven Reverse Engineering : Model Discovery and Model Understanding.

- Model Discovery : Get the AST tree, which is a Specific ASTM Model, from PHP source code
- Model Understanding : Apply model to model transformation on the AST tree to get ASTM model which is a Generic ASTM model

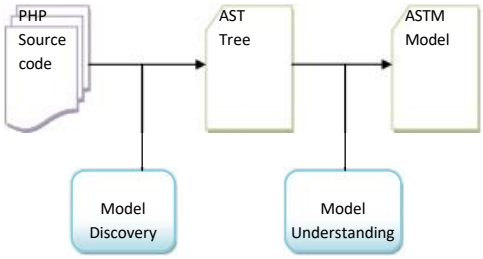

Fig. 2 : Approach MDRE Phases

Detailed steps of our approach :

- Parse PHP code to extract Abstract syntax Tree (AST)
- Transform the AST tree from Json format to XML format (xmi)
- Get the XSD schema from the AST in XML format
- Generate the AST Metamodel in ecore format from the XSD schema
- Write an adapted minimalist ASTM metamodel based on the OMG ASTM metamodel
- Write a transformation script in QVT-Operational language to transform the AST model to ASTM model

ISSN: 2961-6611

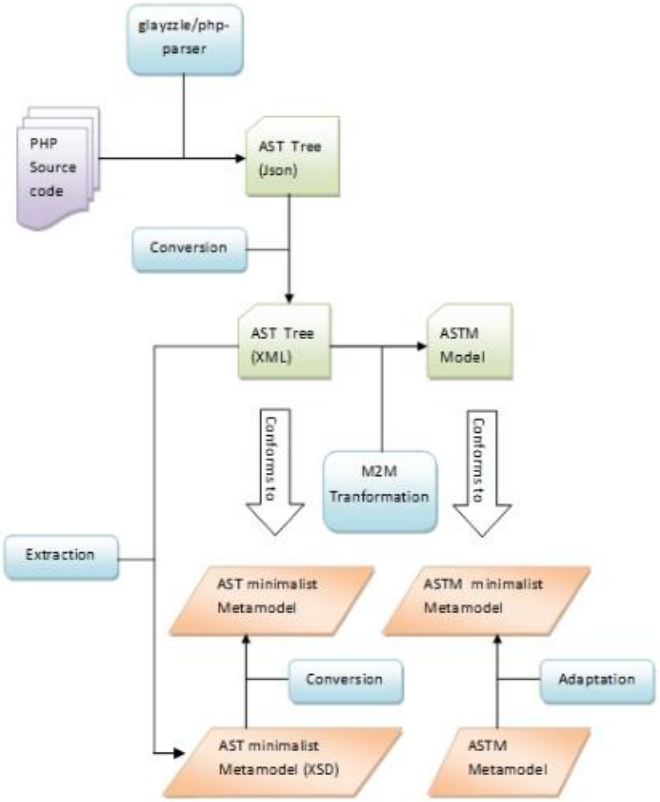

Fig. 3 : ADM-based Approach

B. *The Approach Case Study*

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

```php
<?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;
    }
}
?>
```

Fig. 4 : Vegetable Class

C. *Obtained AST tree*

Using Glayzzle PHP Parser, we obtain this Json AST tree from the class above.


```
{
"kind":"program",
"children":[
{"kind":"class","name":"Vegetable","isAnonymous":f
alse,"extends":null,"implements":null,
        "body":[
        {"kind":"property","name":"edible","value":
null,"isAbstract":false,"isFinal":false,"visibilit
y":"private","isStatic":false},
        {"kind":"property","name":"color","value":n
ull,"isAbstract":false,"isFinal":false,"visibility
":"private","isStatic":false},
                {

        "kind":"method","name":"__construct",
                "arguments":[
        {"kind":"parameter","name":"edible","value"
:null,"type":null,"byref":false,"variadic":false,"
nullable":false},
        {"kind":"parameter","name":"color","value":
null,"type":null,"byref":false,"variadic":false,"n
ullable":false},
                ],
…
```


Fig. 5 Obtained AST tree

D. *XML AST tree*

Using a Json to XML converter, we obtain this XML AST tree from the Json AST tree above.


```xml
<?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>
```


ISSN: 2961-6611

```
    <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>
    …
```

Fig. 6 XML AST tree

E. AST extracted XSD schema

From the XML AST tree, we have generated an XSD Schema.

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="unqualified"
elementFormDefault="qualified">
<xs:element name="root" type="rootType"/>
  <xs:complexType name="childrenType">
    <xs:sequence>
      <xs:element type="xs:string" name="kind"/>
      <xs:element type="xs:string" name="value"
minOccurs="0"/>
      <xs:element type="xs:string" name="operator"
minOccurs="0"/>
      <xs:element type="leftType" name="left"
minOccurs="0"/>
      <xs:element type="rightType" name="right"
minOccurs="0"/>
      <xs:element type="exprType" name="expr"
minOccurs="0"/>
      <xs:element type="xs:string" name="name"
minOccurs="0"/>
      <xs:element type="xs:string" name="name"
minOccurs="0"/>
      <xs:element type="xs:string"
name="isAnonymous" minOccurs="0"/>
      <xs:element type="xs:string" name="extends"
minOccurs="0"/>
      <xs:element type="xs:string"
name="implements" minOccurs="0"/>
      <xs:element type="bodyType" name="body"
maxOccurs="unbounded" minOccurs="0"/>
      <xs:element type="xs:string"
name="isAbstract" minOccurs="0"/>
      <xs:element type="xs:string" name="isFinal"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="bodyType">
    <xs:sequence>
      <xs:element type="xs:string" name="kind"/>
      <xs:element type="xs:string" name="name"
minOccurs="0"/>
      <xs:element type="xs:string" name="value"
minOccurs="0"/>
      <xs:element type="argumentsType"
name="arguments" maxOccurs="unbounded"
minOccurs="0"/>
      <xs:element type="xs:string" name="byref"
minOccurs="0"/>
      <xs:element type="xs:string" name="type"
minOccurs="0"/>
      <xs:element type="xs:string" name="nullable"
minOccurs="0"/>
      <xs:element type="bodyType" name="body"
minOccurs="0"/>
      <xs:element type="xs:string"
name="isAbstract" minOccurs="0"/>
      <xs:element type="xs:string" name="isFinal"
minOccurs="0"/>
      <xs:element type="xs:string"
name="visibility" minOccurs="0"/>
      <xs:element type="xs:string" name="isStatic"
minOccurs="0"/>
      <xs:element type="childrenType"
name="children" maxOccurs="unbounded"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  …
```

Fig. 7 : AST extracted XSD schema

F. AST generated Metamodel

Using EMF Eclipse model importer from schema, we obtain the simplified AST meta-model in ecore format from the XSD schema above.

The ASTM meta-model is in EMOF language Meta-Object Facility (MOF). A supporting standard of MOF is XMI, which defines an XML-based exchange format for models.

The meta-model is shown with help of Sample Ecore Model Editor in Eclipse.

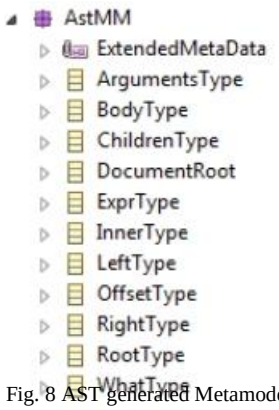

Fig. 8 AST generated Metamodel

ISSN: 2961-6611

G. OMG ASTM Meta-model

ASTM meta-model provided by OMG.

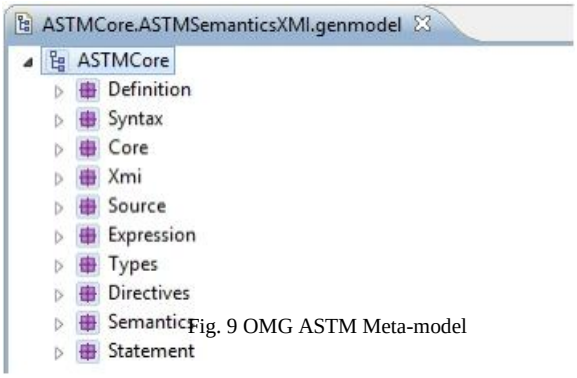

Fig. 9 OMG ASTM Meta-model

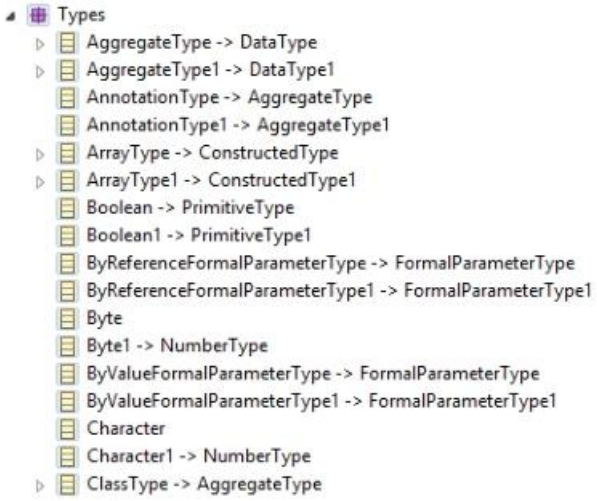

Fig. 10 OMG ASTM Meta-model Types

H. Minimalist ASTM Meta-model

Current OMG ASTM meta-model needs to be simplified, to make the transformation easier for our example.

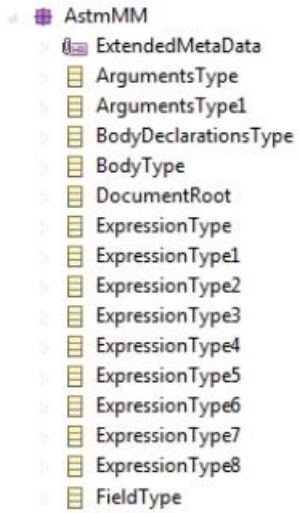

Fig. 11 ASTM Target Metamodel

I. QVT-O Transformation project

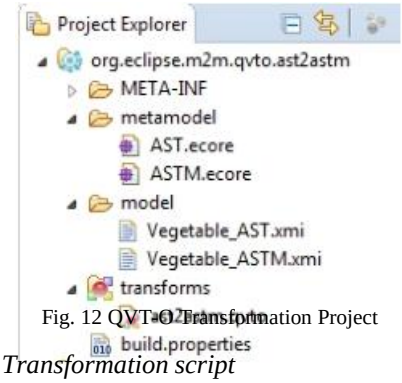

Fig. 12 QVT-O Transformation Project

J. QVT-O Transformation script

We have defined a mapping table between AST model elements and ASTM model elements.

TABLE I
AST ELEMENTS TO ASTM ELEMENTS MAPPING

AST element	ASTM element
ChildrenType	OwnedElementsType1
BodyType	BodyDeclarationsType
ArgumentsType	ParametersType
…	…

Based on the mapping table, we have written a QVT-O transformation script to map AST model elements to ASTM model elements.

```
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;
```


ISSN: 2961-6611

```
        modifier := self.map R21();
        fragments := self.map R22();
        parameters := self.arguments.map R23();
}       }

…
```

Fig. 13 QVT-O Transformation Script

K. Case Study Results

The XML AST tree obtained above is used as input in the transformation.

```
<?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="http://AstMM">
<RootType>
  <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>
…
```

Fig. 14 Source AST Tree Model

L Obtained ASTM Model

After applying the approach, we obtain the following result. Obtained ASTM models are Ecore Models in XMI format.

```
<?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/met
amodel/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>
```

Fig. 15 Obtained ASTM Model

We can notice that obtained result corresponds to our aim goal.

IV. 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:

- Java Swing Modernization Approach : Complete Abstract Representation based on Static and Dynamic Analysis [13]

ISSN: 2961-6611

- A Model Driven Reverse Engineering Framework for Extracting Business Rules out of a Java Application [14]
- Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration [15]
- MoDisco Project [16]

1) ***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.

2) ***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.

3) ***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.

4) ***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.

According to the related works we can conclude that ADM approaches that extract Generic ASTM models from PHP code and others platforms too are rare, some approaches stop at the Specific ASTM level. As for example the "java" models extract by Modisco from Java projects which are not true ASTM models.

V. Conclusion & Future Work

This paper presented an ADM based approach that allow obtaining ASTM model 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 models phase we have implemented transformation rules using QVT-Operational language.

As a future work, we will perform the transformation of extracted ASTM models above to KDM models, By transforming models to KDM, the modeled program will reach the top level of abstraction. Then, extending the KDM meta-model to modeling the MVC structure and behavior of the web application.

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 and J. H. Cross II, Reverse engineering and design recovery: A taxonomy, IEEE Software 7 (1990), pp 13–17.
[5] S. Kent, Model driven engineering, in: Integrated Formal Methods, volume 2335 of Lecture Notes in Computer Science, Springer, 2002, pp. 286–298.
[6] C. Raibulet, F. Arcelli Fontana, and M. Zanoni : Model-Driven Reverse Engineering Approaches : A Systematic Literature Review
[7] Meta-Object Facility specification of the OMG. [Online]. Available: http://www.omg.org/spec/MOF/2.0/
[8] Abstract Syntax Tree Meta-Model specification of the OMG. [Online]. Available: http://www.omg.org/spec/ASTM/1.0/
[9] Knowledge Discovery Meta-Model specification of the OMG. [Online]. Available: http://www.omg.org/spec/KDM/1.3/
[10] Structured Metrics Meta-Model, http://www.omg.org/spec/SMM/1.1/
[11] QVT (Query/View/Transformation) standard of the OMG. [Online]. Available: http://www.omg.org/spec/QVT/1.3/
[12] NodeJS PHP Parser - extract AST or tokens (PHP5 and PHP7) . [Online]. Available: http://glayzzle.com/php-parser/
[13] Z. Gotti, S. Mbarki : Java Swing Modernization Approach : Complete Abstract Representation based on Static and Dynamic Analysis
[14] V. Cosentino, J. Cabot, P. Albert, P. Bauquel and J. Perronnet : A Model Driven Reverse Engineering Framework for Extracting Business Rules out of a Java Application
[15] 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
[16] Modisco project. [Online]. Available: https://eclipse.org/MoDisco/

ISSN: 2961-6611