ADM-Based Hybrid Model Transformation for Obtaining UML Models from PHP Code

https://doi.org/10.3991/ijes.v7i1.10052

Abdelali Elmounadi (✉), Naoual Berbiche, Nacer Sefiani
University Mohammed V, Rabat, Morocco
a.elmounadi@gmail.com

Nawfal El Moukhi
University Ibn tofail, Kenitra, Morocco

Abstract—In this paper, we present a hybrid-based model transformation, according to the Architecture Driven Modernization (ADM) approach, intended for getting UML (Unified Modeling Language) models from the PHP (Hypertext Preprocessor) code. This latter has been done by offering a tool support for automated generation of UML platform independent models from PHP ASTM (Abstract Syntax Tree Metamodel) representations, which are specific platform models. The model transformation rules are expressed in ATL (Atlas Transformation Language), which is a widely used model transformation language based on the hybrid approach. This work aims to fill the gap between the web-based applications maintenance, especially PHP-based implementations, and the model transformation processes in the ADM context.

Keywords—ADM, model transformation, ATL, UML, PHP

1 Introduction

Among the various web development technologies, PHP language (Hypertext Preprocessor) [1] is the most popular server-side scripting language especially suited for web development and dynamic web pages creation. This programming language has become the basis for many web applications thanks to its ease of use and management of the development, deployment and integration lifecycle. However, in order to support the ever-increasing complexity of user needs, the PHP web-based application maintenance is becoming ever more critical. This challenge becomes more and more tangible in proportion to the gap that can emerge between the initial documentation (if any) and the late implementation versions of the PHP web-based application. Thus, reverse engineering is supposed to solve this kind of problem. However, classical reverse engineering tasks can also be very complex and incur additional costs.

In this paper, we propose a new model transformation process that aims to perform reverse engineering of PHP web-based applications in the context of Architecture-driven Modernization (ADM). The model transformation process aims to provide model representation in a higher level of abstraction from PHP web-based applica-

tions assets. The obtained models shall be expressed in UML. The model transformation rules are expressed in ATL [2] (Atlas Transformation Language), which is based on the hybrid model transformation approach.

The rest of this paper is organized as follow: section 2 presents the research background. It presents related concepts to the Architecture-driven Modernization (ADM). Section 3 presents the adopted methodology in this work to achieve the hybrid-based model transformation. Finally, section 4 concludes this work and gives hints about future work and perspective.

2 Architecture - Driven Modernization

Architecture Driven Modernization (ADM) is the process of understanding and evolving existing software assets. According to [3], ADM is an OMG (Object Management Group) standard which addresses the integration of MDA and reverse engineering.

In fact, MDA encourages the separation of concerns, i.e. it preconizes the model transformations between different levels of abstraction, beginning with platform independent models (PIMs) which do not contain any specific information about the implementation platform, arriving to platform specific models (PSMs) that include specific information about implementation platforms. Thus, ADM is for MDRE what represents MDA for MDE. It also preconizes the use of PIM, PSM [4, 5] and model transformations concepts to facilitate the systematic analysis of existing systems, to gather their corresponding models.

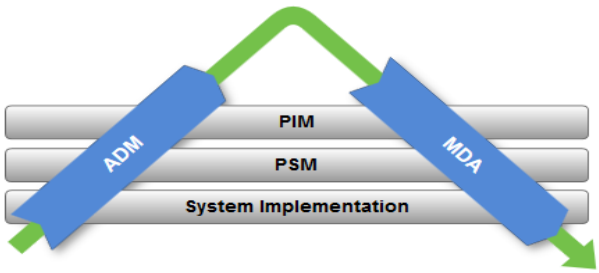

Fig. 1. Process for evolving existing software assets using ADM/MDA approaches

With the advent of ADM, OMG presented a new set of meta model relatively to this context: Knowledge Discovery Meta model (KDM) [6] and Software Metrics Meta model (SMM) [7], which are both platform-independent meta-models, and ASTM (Abstract Syntax Tree Meta model) [8] which is more platform-specific.

Firstly, KDM is designed as the OMG's foundation for software modernization and offers a common intermediate representation for existing software systems and their operating environments, which defines common metadata required for semantic integration of application lifecycle management tools. It represents a meta model for knowledge discovery in software and defines a common vocabulary of knowledge

related to software engineering artifacts, regardless of the implementation programming language and runtime platform. KDM is designed to enable knowledge-based integration between tools and defines an interchange format for those that work with existing software as well as an abstract programming interface (API) for the next-generation assurance and modernization tools.

Secondly, SMM defines a meta-model intended for representing software measurement information. Indeed, a standard for the exchange of measures is very important, given the role that measures play in software engineering and design stage. SMM is part of the ADM roadmap and fulfills the metric needs of the ADM roadmap scenarios.

Finally, ASTM is a meta model from the OMG that describes the set of elements used for composing abstract syntax trees (ASTs) [9]. The main purpose of ASTM specification is to enable easy interchange of detailed software metadata between software development and software modernization tools. This specification defines a meta model for representing information about existing software assets in the form of abstract syntax trees. Indeed, an AST Meta model describes the elements used for composing AST models. An AST model is a formal structure that describes the manner how the statements of a software asset are structured, and reflects the grammar of a particular programming language. Furthermore, the ASTM specification is organized into three levels of abstraction:

Gastm: Generic Abstract Syntax Tree Meta model is a generic set of language modeling elements common across numerous languages establishes a common core for language modeling called the Generic Abstract Syntax Trees. In this specification, the GASTM model elements are expressed as UML class diagrams.

Sastm: Language Specific Abstract Syntax Tree Meta models constitute a set of meta models for particular languages such as PHP, C++ or Java. These meta models are derives from the GASTM along with modeling element extensions sufficient to capture the language.

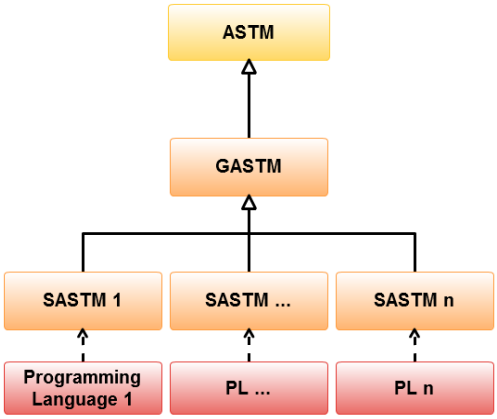

Fig. 2. SATSM - GASTM relationship

Pastm: Proprietary Abstract Syntax Tree Meta models express AST representations for different languages modeled in formats that are not consistent with MOF (Meta-Object Facility), the GASTM, or SASTM. For such proprietary AST this specification defines the minimum conformance specifications needed to support model interchange.

In this article, the KDM and SMM will not be used at this stage. However, we use ASTM to create a SATSM for the PHP language. The input PSMs will be in compliance with this meta model in the model transformation process.

3 The Hybrid - Based Model Transformation

The model transformation performed takes as input a PSM model (a SASTM PHP compliant model instance) to provide a PIM model as output conforming to the UML meta model. Then, the transformation engine is implemented based on the ATL language which adopts the hybrid model transformation approach.

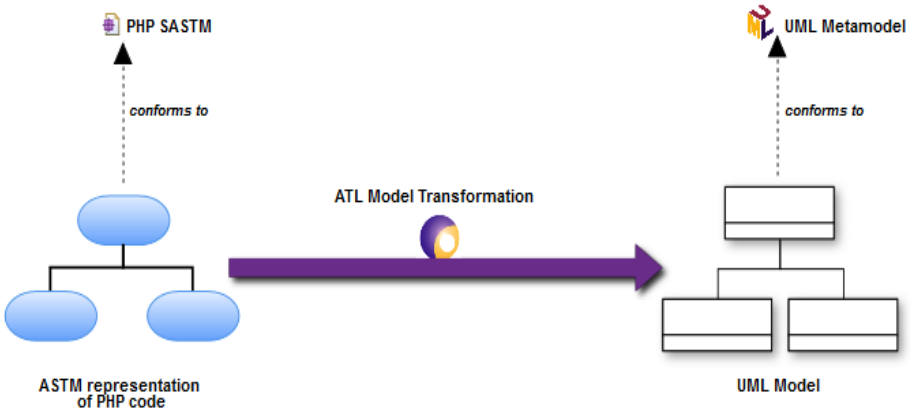

Fig. 3. Description of the model transformation process

Through this work, we are seeking to fill the gap between the ADM approach and web development fields. Indeed, the ADM approach offers better ease of use. Unlike existing commercial tools, it offers better control of reverse engineering processes, by offering more visibility, eliminating the black-box effect, and also offering better traceability for the transformation steps. This latter can be done by offering a tool support for automated generation of UML platform independent models (PIMs) from PHP ASTM (Abstract Syntax Tree Meta model) representations, which are specific platform models (PSMs). In the following, we present in detail the elements of the model transformation.

3.1 PSM and PIM meta models

We present the different meta-classes that make up the PSM and PIM meta-models used in our model transformation process. In this work, we focused on the structural elements of programming on the PHP language side as well as the structural modeling aspect on the UML Meta model side. Figure 4 illustrates the source meta model (PSM) that represents the elements of the PHP language. This meta model, which represents an SASTM for PHP language, was developed on the basis of the PDT (PHP Development Tools) library [10] of the Eclipse platform, by using ECORE [11] of the EMF platform (Eclipse Modeling Framework).

Indeed, according to the PDT library, Table 1 provides the definition of each element present in Figure 4.

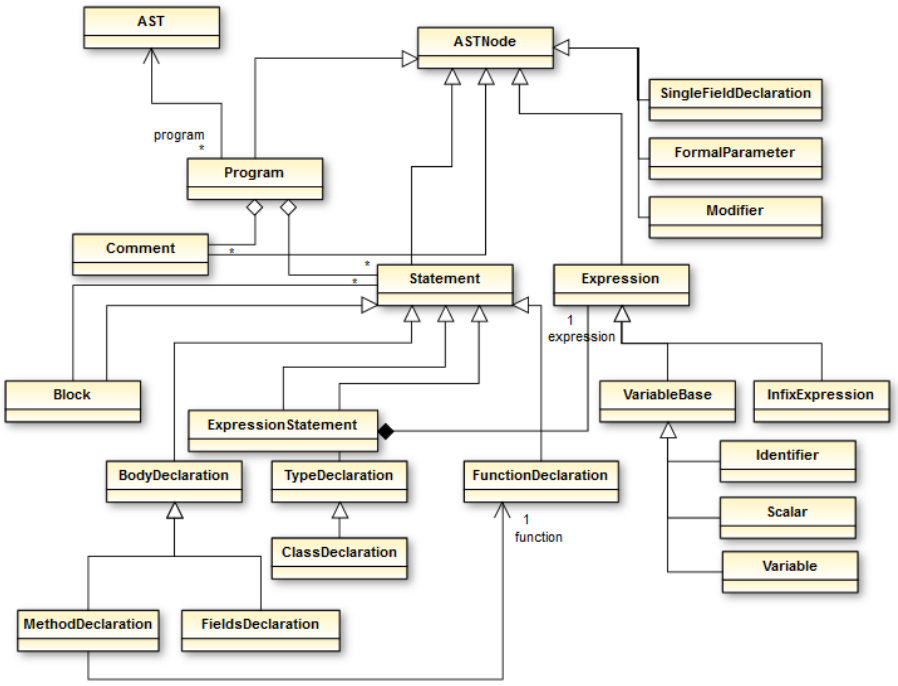

Fig. 4. SASTM for PHP language

Table 1. Overview of the PHP SASTM elements

Name	Description
AST Node	Abstract meta-superclass of all AST nodes
Program	Representation of a PHP script
Statement	Notion of abstract statement, the smallest standalone element of programing languages
Comment	Represents comment lines
Block	Represents code block type

Type Declaration	Type declaration AST node type
Class Declaration	Class declaration AST node type
Body Declaration	Abstract base meta-class of all AST nodes that represent body declarations that may appear in the body of some kind of class or interface declaration as fields declaration or method declaration
Method Declaration	Method declaration AST node type
Function Declaration	Function declaration AST node type
Fields Declaration	A body declaration AST node type dedicated to fields declaration
Expression Statement	A kind of statement AST node type that wraps an expression
Expression	Abstract base meta-class of AST nodes that represent expressions
Infix Expression	Infix expression AST node type
Variable Base	Base meta-class for representing variables
Identifier	Identifying element for the AST nodes
Scalar	Scalar AST node type
Variable	Variable AST node type
Single Field Declaration	A single field declaration AST node type
Formal Parameter	A formal parameter AST node type
Modifier	Access specifier of a body declaration

However, Figure 5 represents the target meta-model (PIM) whose elements constitute the class diagram of the UML modeling language, based on the OMG specification in UML-Infrastructure [12]. UML Package describes the concept of UML package, and is related to the Classifier meta-class. The Classifier meta-class represents both the concept of UML class and the concept of data type. The Property meta-class expresses the concept of properties of an UML class. The Operation meta-class describes the notion of method, which is closely linked to the notion of parameter (Parameter meta-class). Finally, the meta-model highlights the notion of generalization that includes aspects of a higher level of abstraction referring to the concerned classifier. In addition, a classifier could have either no generalizations or many generalizations.

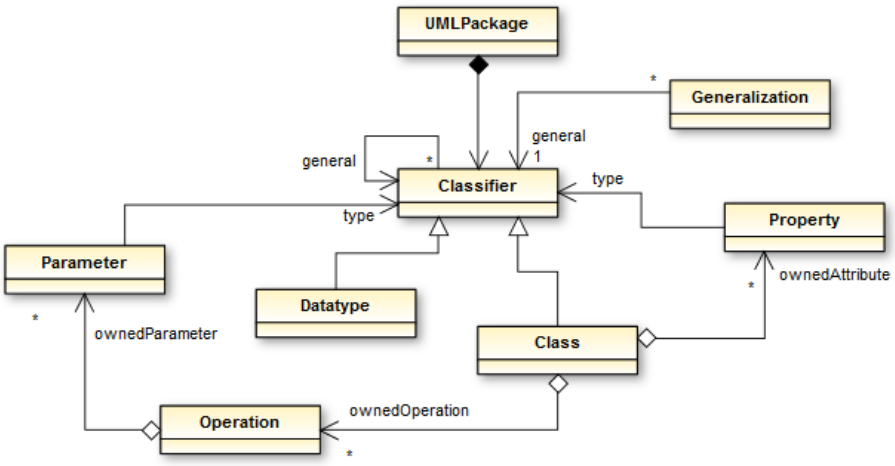

Fig. 5. UML meta model

3.2 Atlas transformation language

Atlas Transformation Language (ATL) [13] is a model transformation language designed to express model transformations as described by the hybrid approach in an MDA/ADM context. This language is the result of research by the INRIA Atlas and the LINA research group in response to the OMG MOF/QVT Request for Proposal.

ATL is specified both as a meta-model and as a concrete textual syntax that relies on the syntax of OCL (Object Constraint Language) [14] formalism for the expression of transformation rules. It is a hybrid language that adopts a declarative and imperative approach. The declarative style is recommended for writing the model transformation. However, mandatory constructs are provided to further specify some transformations that are too complex to be processed in a declarative manner only. The ATL transformation program consists of a set of rules that define how the elements of a source model will match the elements of the target model. The implementation of ATL-based transformations is provided through an integrated plugin into the Eclipse IDE.

3.3 ATL transformation rules

In this section, we present some of the transformation rules that are involved in generating the UML model from a SASTM representation of the PHP code. In this work, we consider structural aspect only. By analogy, an AST will be converted to a UML model as a first step. Then, each class declaration in the PHP code will be translated into a UML class with the corresponding fields and methods. Specific order transformations are then performed to map the remaining elements to complete the model transformation process as described in Table 2

Table 2. Set of the ATL-based transformations

	Transformation Rule description	Transformation Rule in ATL

1	The Abstract syntax tree (AST) is converted to an UML Model	rule PHPAST2UMLModel {
		from a: PHP!AST
		tom: UML!Model(
		name <- 'default model',
		visibility <- #public,
		packagedElement<- a.program-
		>collect(p | p.statement),
		Owned Comment<- a.program->collect(p | p.comment)
		)
		}

2	Each class declaration becomes an UML class with the correspondent attributes, operations and generalization.	rule PHPClassDeclaration2UMLClass {
		from p:PHP!ClassDeclaration
		to
		u:UML!Class(
		name <- p.identifier.name,
		visibility <- #public,
		isAbstract<- (p.modifier = 'abstract'),

		isFinalSpecialization<- (p.modifier = 'final'), ownedAttribute<- p.body.statement->select(o | o.oclIsTypeOf(PHP!FieldsDeclaration))->collect(s | s.field), ownedOperation<- p.body.statement->select(o | o.oclIsTypeOf(PHP!MethodDeclaration)), generalization <- p.superClass) }

3	Each single field declaration becomes an UML property.
	(Remaining part of rule number 3)
		rule PHPFieldDeclaration2UMLProperty {
		from p: PHP!SingleFieldDeclaration
		to u: UML!Property(
		visibility <- p.refImmediateComposite().getVisibility(p.refImmediateComposite().modifier),
		isStatic<-p.refImmediateComposite().isStatic(p.refImmediateComposite().modifier),
		
		isReadOnly<-p.refImmediateComposite().isFinal(p.refImmediateComposite().modifier),
		name <- p.getName,
		defaultValue<- p.value,
		type <- OclUndefined
		)
		}

4	Each method declaration is converted to an UML operation.	rule PHPMethodDeclaration2UMLOperation {
		from p: PHP!MethodDeclaration
		to u:UML!Operation(
		name <- p.function.identifier.name,
		type <- OclUndefined,
		isAbstract<- p.function.body.oclIsUndefined(),
		isStatic<- p.isStatic(p.modifier),
		ownedParameter<- p.function.formalParameter
		)
		}

5	Each formal parameter of a method declaration becomes an UML operation parameter.	rule PHPformalparameter2UMLparameter {
		from p: PHP!FormalParameter
		to u: UML!Parameter(
		name <- p.parameterName.name.name,
		type <- p.parameterType
		)
		}

6	A PHP comment is transformed to an UML comment.	rule PHPComment2UMLComment {
		from p: PHP!Comment
		to u: UML!Comment(
		body <- p.body.trim()
		)
		}

7	An extended PHP class in a class declaration becomes an UML generalization related to the container class.	rule PHPNamespaceName2UMLGeneralization { from p: PHP!NamespaceName(p.refImmediateComposite().oclIsTypeOf(PHP!ClassDeclaration)) to u: UML!Generalization(general <- UML!Class.allInstancesFrom('OUT')->select(c | c.name = p.segment->first().name)->first())

		}
8	A concrete rule that converts a PHP scalar to an UML literal string.	rule PHPScalar2UMLLiteralString extends PHPScalar2UMLLiteral { from p: PHP!Scalar to u: UML!LiteralString(value <- p.value

4 Conclusion and Future Work

Thanks to the Eclipse platform, notably via PDT and EMF-Ecore, we were able to implement a Meta model of the PHP language to be able to manipulate the elements of this language as part of the ADM approach. We then performed a model transformation based on the ATL language that allows us to obtain UML models from code written in PHP language. Through this achievement, we have experienced the strength of the ASTM Meta model as well as the power of the hybrid approach in the context of model transformations. Our objective is now to be able to expand this contribution towards other programming languages to improve the usability of model transformations using the same approach with further development technologies.

5 References

[1] J. Maras, A. Petričić, and M. Štula, "phpModeler–an approach to Reverse Engineering legacy Web applications," in *DICES workshop at the 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2011)*, 2010.

[2] M. Rahmouni and S. Mbarki, "MDA-Based ATL Transformation To Generate MVC 2 Web Models," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 4, pp. 57–70, Aug. 2011. https://doi.org/10.5121/ijcsit.2011.3405

[3] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot, "MoDisco: A model driven reverse engineering framework," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 1012–1032, 2014. https://doi.org/10.1016/j.infsof.2014.04.007

[4] K. Arrhioui, S. Mbarki, and M. Erramdani, "Applying CIM-to-PIM Model Transformation for Development of Emotional Intelligence Tests Platform," *Int. J. Online Eng. IJOE*, vol. 14, no. 8, p. 160, Aug. 2018. https://doi.org/10.3991/ijoe.v14i08.8747

[5] O. Betari, S. Filali, A. Azzaoui, and M. A. Boubnad, "Applying a Model Driven Architecture Approach: Transforming CIM to PIM Using UML," *Int. J. Online Eng. IJOE*, vol. 14, no. 9, p. 170, Sep. 2018. https://doi.org/10.3991/ijoe.v14i09.9137

[6] Object Management Group, "Knowledge Discovery Metamodel (KDM)." [Online]. Available: http://www.omg.org/technology/kdm/. [Accessed: 24-Apr-2018].

[7] Object Management Group, "About the Structured Metrics Metamodel Specification Version 1.1.1." [Online]. Available: https://www.omg.org/spec/SMM/1.1.1/. [Accessed: 24-Apr-2018].

[8] Object Management Group, *Architecture-driven Modernization: Abstract Syntax Tree Metamodel (ASTM)*. 2011.

[9] A. Elmounadi, N. Berbiche, F. Guerouate, and N. Sefiani, "Eclipse JDT-based method for dynamic analysis integration in Java code generation process," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 24, 2017.

[10] Eclipse Foundation, "Eclipse PHP Development Tools." [Online]. Available: https://www.eclipse.org/pdt/. [Accessed: 20-Jul-2018].
[11] Eclipse Foundation, "Ecore - Eclipsepedia." [Online]. Available: http://wiki.eclipse.org/Ecore. [Accessed: 24-Apr-2018].
[12] Object Management Group, "OMG Unified Modeling Language TM (OMG UML), infrastructure." 2011.
[13] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, Jun. 2008. https://doi.org/10.1016/j.scico.2007.08.002
[14] E. Seidewitz and J. Tatibouet, "Tool Paper: Combining Alf and UML in Modeling Tools–An Example with Papyrus–," in *OCL 2015–15th International Workshop on OCL and Textual Modeling: Tools and Textual Model Transformations Workshop Proceedings*, 2015, p. 105.

6 Authors

Abdelali Elmounadi was born in Rabat, in 1988. He is a PhD student at Mohammed V University in Rabat, Morocco. He received a Master's degree in Computer Science from Sidi Mohammed Benabdallah University in 2012. His research interests are Software Engineering, Reverse Engineering, Model-driven Engineering and code generation.

Naoual Berbiche is a Research professor at LASTIMI laboratory and Professor in the Department of Computer Sciences at the Superior School of Technologies of Salé, Mohammed V University in Rabat, Morocco. She is interested in models transformation, systems interoperability, computer network security and web application security.

Nacer Sefiani was born in Rabat, Morocco. He received the Master degree and the Doctorate in instrumentation and measurements from university of Bordeaux, France, respectively in 1992 and 1995. He received the ability, degree and grade Professor in the Graduate School of Technology at Salé in the University Mohammed V, Rabat, Morocco respectively in 2003 and 2011. His research interests include identification and control of nonlinear systems. He is the author or co-author of many papers in international journals and conferences.

Nawfal El Moukhi was born in Salé, in 1987. He is a PhD student in computer science at Ibn Tofail University in Kenitra, Morocco.

Article submitted 2018-12-26. Resubmitted 2019-01-27. Final acceptance 2019-01-27. Final version published as submitted by the authors.