Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 314 (2015) 23–44

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Migrating Traditional Web Applications to CMS-based Web Applications

Feliu Trias[1], Valeria de Castro[2], Marcos Lopez-Sanz[3] and Esperanza Marcos[4]

Kybele Research Group, Rey Juan Carlos University, Móstoles, Spain

Abstract

In recent years, Content Management Systems (CMS) have proven to be the best platforms for maintaining the large amount of digital content managed by Web applications. Thus, many organizations have experienced the necessity to base its Web applications on these CMS platforms. To do this, they start a migration process which is complex and error prone. To support this process, we propose a method based on the principles of Architecture-Driven Modernization (ADM) which automates the migration of Web applications to CMS-based Web applications. This article focuses on the implementation of two artifacts of this method: 1) the DSL ASTM_PHP, a modeling language for defining a model from PHP code (ASTM_PHP model) and 2) the model-to-model transformation rules which generate automatically a KDM model from a ASTM_PHP model. To show the feasibility of this implementation, we use a case study based on a widget implemented in PHP which lists the online users of a Web application.

Keywords: Content Management System, Web application, Architecture-Driven Modernization, Software Migration, Reverse Engineering and Model-driven Engineering.

1 Introduction

Over the last years, the volume of digital content managed by Web applications has increased exponentially. Therefore, organizations have experienced the necessity of using powerful management platforms to maintain this digital content in a robust and reliable manner [3] [17].

One of the most popular adopted solutions has been the use of *Content Management Systems (CMS)* as platforms to base their large-scale Web applications [3] [34]. These CMS-based Web Applications provide advantages that distinguish them from the traditional Web applications [41]. According to [40] some of these

[1] Email: feliu.trias@urjc.es

[2] Email: valeria.decastro@urjc.es

[3] Email: marcos.lopez@urjc.es

[4] Email: esperanza.marcos@urjc.es

http://dx.doi.org/10.1016/j.entcs.2015.05.003

advantages are: 1) the possibility of creating dynamically digital content, 2) the explicit separation between content and design or 3) the flexibility of extending the functionality of the Web application.

Considering these advantages, many organizations have migrated their traditional Web applications to CMS-based Web applications. The problem is that this reengineering process is carried out following an ad-hoc manner that entails risks and high costs for the organization [35].

For this reason, the *Object Management Group (OMG)* [24] proposes the *Architecture-Driven Modernization (ADM)* [16], an standard which advocates for the application of *Model-Driven Architecture (MDA)* [19] techniques and tools to systematize the software reengineering process and reduce its high risks and costs. Moreover, ADM develops a set of standard metamodels to represent the information involved in a software reengineering process. Two of these metamodels are: the *Abstract Syntax Tree Metamodel (ASTM)* [23], which allows to represent at a platform-specific level the syntax of the source code implementing a legacy system, and the *Knowledge Discovery Metamodel (KDM)* [26] [30] which allows to represent at a platform-independent level the syntax and semantics of the legacy system.

After the performance of a literature review [37], we found some methods focused on the development of CMS-based Web applications [33] [36][41], but none of them address the migration of traditional Web applications to these CMS platforms, even though the existing necessity and the advantages provided by these platforms.

To solve this gap, we present an ADM-based method for migrating legacy Web applications to CMS platforms. This migration method copes with the three classical reengineering stages following a horseshoe process [7]: 1) *reverse engineering stage*, that consists of the extraction of knowledge from a legacy Web application, 2) *restructuring stage*, to redefine the legacy system taking into account the features of a target CMS platform and 3) *forward engineering stage*, to address the classical top-down implementation of an information system.

The work presented in this paper is focused on the *reverse engineering stage*. On the one hand, we define a *Domain Specific Language (DSL)* called ASTM_PHP DSL [12]. This DSL is a modeling language based on the ASTM standard metamodel which allows the definition of platform-specific models (*ASTM_PHP models*) which represent the syntax of the legacy software artifacts implemented in PHP code. On the other hand, we present the implementation of the *model-to-model (M2M)* transformations which allow to represent at a platform-independent the *ASTM_PHP models*. These resulting models conform to the KDM standard metamodel (*KDM models*).

To show the feasibility of this migration method, we present a case study where we migrate a widget implemented in PHP from a traditional Web application to the CMS platform called Drupal [9]. Drupal is one of the most used open-source CMS platforms, currently. The widget migrated lists the users connected to the Web application.

The rest of this paper is organized as follows: Section 2 provides an explanation of the ADM principles focusing on the ASTM and KDM metamodels. Section 3

explains the ADM-based migration method that we present. Section 4 presents the case study used to show the feasibility of our approach. Section 5 presents the definition of the ASTM_PHP DSL. Section 6 presents the implementation of the M2M transformations between the *ASTM_PHP models* to the *KDM models*. Section 7 presents other ADM-based approaches found with literature review and, finally, Section 8 presents the conclusions and future works.

2 Architecture-Driven Modernization

Model-Driven Development (MDD) [18] has proven its usefulness as top-down software development paradigm, and now it is expanded to software reengineering and migration processes.

In 2003, OMG proposed the *Architecture-Driven Modernization (ADM)* [15] [29] initiative which follows the MDD principles. ADM fosters system modernization based on the use of models at different abstraction levels [21]: *Computation Independent Model (CIM)* level, *Platform Independent Model (PIM)* level and *Platform Specific Model (PSM)* level. Furthermore, it proposes the use of automated transformations to generate new systems from legacy systems by following a horseshoe process.

ADM defines seven standard metamodels, but currently only three of them are available: *Abstract Syntax Tree Metamodel (ASTM)* [23], *Knowledge Discovery Metamodel (KDM)* [26] and *Software Metrics Metamodel (SMM)* [27]. This paper focuses on the use of ASTM and KDM metamodels.

ASTM is the metamodel which represents a low-level view of the system. It allows the definition of models at PSM level representing the source codes syntax of a legacy system. To obtain these models, it is necessary to define *text-to-model (T2M)* transformations that let the mapping between the codes syntax and the elements of the ASTM standard metamodel. This metamodel has two parts: the *Generic Abstract Syntax Tree Metamodel (GASTM)* which factors common elements of most of the programming languages and the *Specific Abstract Syntax Tree Metamodel (SASTM)* which represents the specific properties of a programming language.

Finally, KDM [26] allows to represent in a model the syntax and semantics of a software system at PIM level. The semantics can be found in the code itself, GUI events and business rules. It provides a common interchange format intended to represent existing software assets, allowing the interoperability at PIM level of the different approaches addressing the reengineering process. KDM comprises several packages (*core*, *kdm*, *source*, *code* or *action*) which are grouped in four layers to improve modularity and separation of concerns (*infrastructure*, *program elements*, *runtime resource* and *abstractions*) [30].

3 An ADM-based Migration Method

The migration method presented in this paper is based on the ADM principles. It is defined as a reengineering process composed of three classical stages as it is

shown in Fig. 1: 1) *reverse engineering stage*, 2) *restructuring stage* and 3) *forward engineering stage*. In this section, we present these stages and the tasks which compose them. As we can see in Fig. 1 marked in dotted line, the work presented in this paper is framed in the *reverse engineering stage* of our migration method; concretely, in the second task called *generation of KDM models*.

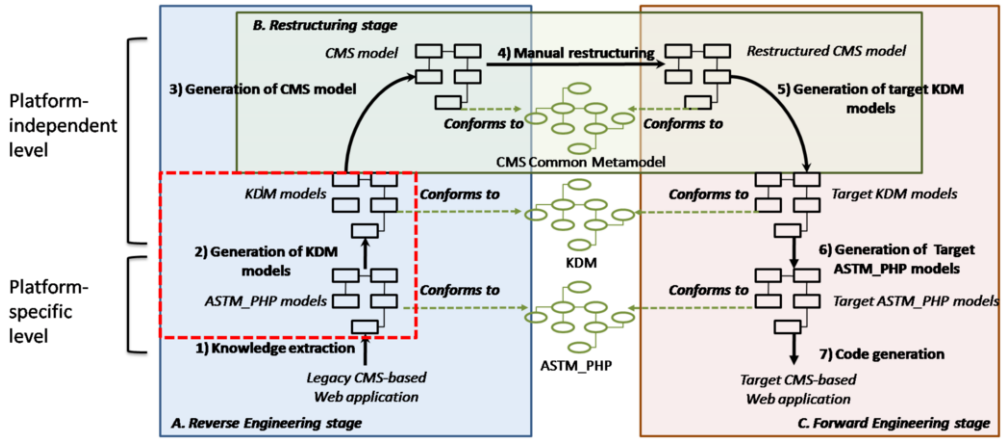

Fig. 1. Our ADM-based Migration Method

Reverse Engineering Stage

This stage is composed of three tasks: 1) *knowledge extraction*, the extraction of *ASTM_PHP models* from a legacy PHP code. To define this models, we have defined the ASTM_PHP DSL, a modeling language which allows the representation of the syntax and semantics of the PHP code in a proper and non-ambiguous way at platform-specific level. The definition of this DSL is presented in Section 5; 2) *generation of KDM models*, from the *ASTM_PHP models* we automatically generate *KDM models* that represent the syntax and semantics of the legacy code but at platform-independent level. These *KDM models* conform to the *code* and *action* packages of the KDM standard metamodel. These models are obtained through M2M transformations implemented in *Atlas Transformation Language (ATL)* [11]. The definition and implementation of these M2M transformations is another objective of this work presented in Section 6; 3) *generation of the CMS model*, using M2M transformations we generate automatically the *CMS model* from the information captured in the *KDM models*. This model represents the knowledge extracted within the CMS domain at platform-independent level .The CMS model conforms to the *CMS Common Metamodel* presented in [37] and introduced at the end of this section. This metamodel has been defined by our research group and it is considered one of the cornerstones of our ADM-based migration method.

Restructuring Stage

In this stage, the CMS model is manually restructured by the developer taking into account the specific features of the target CMS platform. From this restruc-

tured *CMS model*, it is possible to obtain the implementation of the migrated Web application into a new CMS platform.

Forward Engineering Stage

This stage defines the classical top-down software development process into the horseshoe reengineering process. It is composed of three tasks: 1) *generation of the target KDM models*, from the restructured CMS model we generate the target KDM models that represent the implementation of the target CMS-based Web application at platform-independent level; 2) *generation of the target ASTM_PHP models*, during this task we generate the target *ASTM_PHP models*, from the target *KDM models*, that represent the implementation of the target CMS-based Web application at platform-specific level and 3) *code generation*, we generate the software artifacts that implement the CMS-based Web application from the target *ASTM_PHP models*.

3.1 CMS Common Metamodel

As it is said previously, the *CMS Common Metamodel* is one of the main contributions of our ADM-based migration method. It allows the definition of the *CMS model* during the third task of the *reverse engineering stage* of our method (*generation of the CMS model*).

This metamodel represents the key elements for modeling CMS-based Web applications. Thus, it captures elements such as *theme*, *vocabulary*, *module* and other specific elements of this domain. These elements are classified into five views considering the views proposed by the Web Engineering [13] [22]:

- `Navigation`: it considers the elements that define the navigation structure of the Web application. Some of these elements are: *Page*, *Menu* or *MenuItem*.
- `presentation`: it defines the structure and look-and-feel of the pages that compose the Web application. The elements included in this view are: *Theme* and *Region*.
- `content`: it captures the data and data types managed by the CMS-based Web application. In this view, we consider elements such as: *Content*, *Term* or *Vocabulary* among others.
- `user`: it defines the elements related to the roles and permissions assigned to the users of the CMS-based Web application. These elements determine how is the navigation and the use of the Web application by a concrete user. Some of the elements considered in this view are: *Role*, *Permission* or *User*.
- `CMS behavior`: the elements of this view allow the definition of the functionality of the CMS-based Web application. In this view, the elements defined are: *Block*, *Module* and *Function*.

In Fig. 2, we present an excerpt of the CMS Common Metamodel with the elements of the presentation and navigation views. As we can see, some of the elements of this view are: *Theme*, *Region*, *Page* and *MenuItem*.

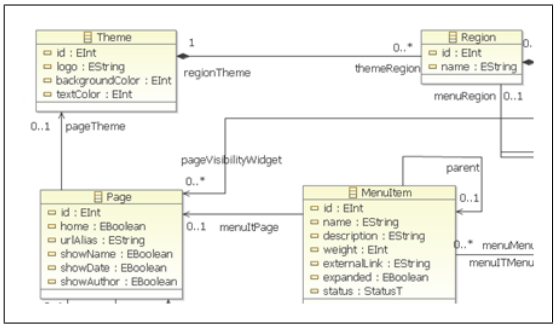

Fig. 2. An excerpt of the CMS Common Metamodel

4 Case Study

To show the usability of our ADM-based migration method, we present a case study where we migrate a widget listing online users from a legacy Web application to a CMS-based Web application implemented in Drupal. It is a Web application of a wellness and nutrition centre called *Websana* which provides users with information about diets, exercises and recommendations about healthy habits. Fig. 3.a shows the widget in the legacy Web application and Fig. 3.b the same widget implemented in Drupal [9].

Fig. 3. a)Legacy Web application b)CMS-based Web application

An *ASTM_PHP model* is extracted from this PHP code. Then, from this model, the *KDM model* is generated automatically at an independent-platform level by a M2M transformation implemented in ATL. Fig 4 shows an excerpt of this legacy PHP code that implements the widget of our case study.

5 Definition of the ASTM_PHP DSL

In this section, we present the ASTM_PHP DSL, a modeling language that allows the definition of models at platform-specific level that represents the syntax and semantics of a PHP code.

The implementation of the ASTM_PHP DSL requires two tasks: 1) *the definition of the abstract syntax* by the ASTM_PHP metamodel and 2) *the definition of the concrete syntax* by implementing a tree-like graphical editor which allows to define graphically models conforming to the ASTM_PHP metamodel. In the following, we explain these tasks in more detail.

```
function listUser_help($path, $arg) {
   switch ($path) {
   case 'admin/help#listuser':
       $output = '';
       $output .= '<h3>' . t('About') . '</h3>';
       $output .= '<p>' . t("A module listing users")
       return $output;
  }
}
function listUser_block_info() {
   $blocks=array();
   $blocks['list_modules']=array(
   'info' => t('A listing of all users'),
   'cache' => DRUPAL_NO_CACHE,
   );
   return $blocks;
}
```

Fig. 4. PHP code implementing the widget migrated

5.1 *Definition of the Abstract Syntax*

The *abstract syntax* is specified by means of the ASTM_PHP metamodel which captures the elements required to represent in a model the syntax and semantics of a PHP code at a platform-specific level. The ASTM_PHP metamodel is as an extension of the ASTM standard metamodel proposed by ADM which is composed of two domains: the *Generic Abstract Syntax Tree Metamodel (GASTM)* which defines the common elements of most of the programming languages and the *Specific Abstract Syntax Tree Metamodel (SASTM)* which represents the specific elements of a concrete programming language, e.g. PHP, Java or C++.

For the definition of the ASTM_PHP metamodel, we extended the SASTM with specific syntax and semantic elements of the PHP programming language. These elements are specializations of the elements defined in GASTM. The elements defined in SASTM are classified into two groups: 1) *those representing a new element* not considered in GASTM (new) and 2) *those representing an existing element in GASTM* but adapted to the PHP specification (redefined).

In the following, we present some of these elements added to SASTM. Fig. 5 shows the specialization of the element Expression (existing in GASTM) into four elements: *ObjectAccess*, *ClassAccess*, *DuplaArray* and *ArrayAccessPHP*.

- `ObjectAccess and ClassAccess`: these two elements represent two expressions not defined within GASTM. The former represents the access to a member or a function of an object, and the latter represents the access to the same items but of a class. As we can see in Fig. 5, the definitions of these two elements are similar, but from the semantic point of view are different so that we decided to define two different elements.
- `ArrayAccessPHP`: this element represents the access to a position of an array. It is defined as the redefinition of the *ArrayAccess* element of GASTM. Concretely, we redefined the cardinality of the relationship *subscripts*. This relationship specifies the positions of the array (e.g. array [1] [2]). The default definition of these positions is required since cardinality of *subscripts* is 1..*. Otherwise, an array access in PHP can be specified without the definition of any position so that we

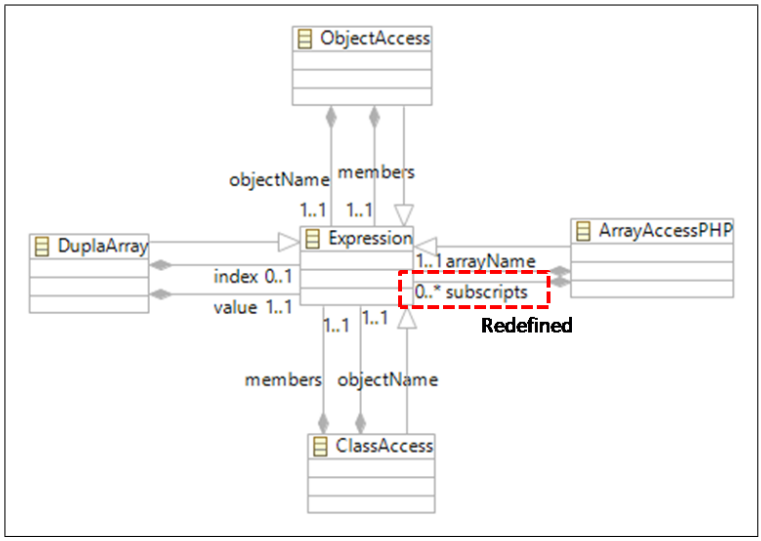

Fig. 5. Specialization of the element Expression

needed to redefine the cardinality of *subscripts* to 0..*. This change is marked in a red dotted line in Fig. 5.

- `DuplaArray`: it is a new element which have been included within the SASTM to represent the key-value pairs. It is composed of two attributes called *index* and *value* defined as two aggregation relationships, as it is shown in Fig. 5. On the one hand, the *index* is a non-required attribute that refers to an expression representing the index of the pair, on the other hand, the *value* is a required attribute linked to an expression representing the value.

Table 1 lists all the elements added into SASTM. The first column refers to the name of the GASTM element that the SASTM element specializes; the second column denotes the name of the SASTM element; the third column indicates whether the SASTM element is *new* or *redefined* and finally, the fourth column is a description of the resulting SASTM element. Most of these SASTM elements represent new operators and expressions. Otherwise, three of them represent redefined elements (two statements and one expression).

5.2 Definition of the Concrete Syntax

As for the *concrete syntax*, it has been implemented with a tree-like graphical editor which allows the representation of models conforming to the ASTM_PHP metamodel. This implementation is based on *Eclipse Modeling Framework (EMF)* [6] which allows to implement automatically a tree-like graphical editor from a generator model (*GenModel*).

This *GenModel*, EMF is able to generate the Java code organized in three differerent packages, as we can see in Fig. 6: the model code (*Java model*), editing model (*Java edit*) and the editor code (*Java editor*). These projects are interrelated. Concretely, the *Java model* is the implementation in Java of the ASTM_PHP

Table 1
Elements defined within SASTM

GASTM element	SASTM element	New or Redefined	Description
BinaryOperator	Xor	New	Boolean Operator
	NotIdentical	New	Boolean Operator
	Identical	New	Boolean Operator
	InstanceOf	New	Boolean Operator
UnaryOperator	New	New	Creates an object
	Clone	New	Creates a copy of an object
Statement	ForStatementPHP	Redefined	For loop
	SwitchStatementPHP	Redefined	Switch Condition
	ForEachStatement	New	For each loop
CompilationUnit	CompilationUnitPHP	Redefined	File containing PHP code

metamodel. This Java code is considered as the logic of the metamodel. It defines an API which allows the access to the metamodel programmatically, the generation of models conforming to this metamodel, and the serialization of these models in XMI. Otherwise, the *Java edit* and the *Java editor* use the *Java model* to generate the Java code to implement the user interface of the resulting tree-like graphical editor.

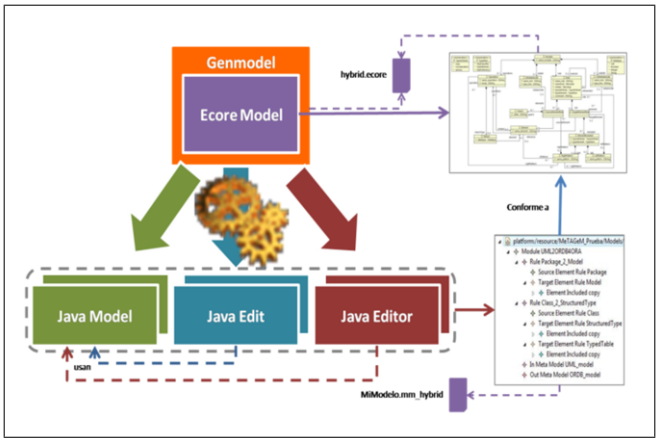

Fig. 6. Overview of the generation of EMF editor

To illustrate the application of the ASTM_PHP DSL, we have extracted an ASTM_PHP model from the PHP code that implements the widget of our case study. This model has been represented with the tree-like graphical editor that we have implemented. Fig.7.a shows the piece of PHP code implementing the widget and Fig.7.b presents the resulting ASTM_PHP model by using the graphical editor.

As we can see in Fig. 7, the *function definitions* have been marked in red. Moreover, the name of these *function definitions* have been marked in green and the parameters passed to them have been marked in blue. Finally, the *statements* that conform the body of the *function definitions* (*switch statement*, *array definition* and *return statement*) have been marked in orange.

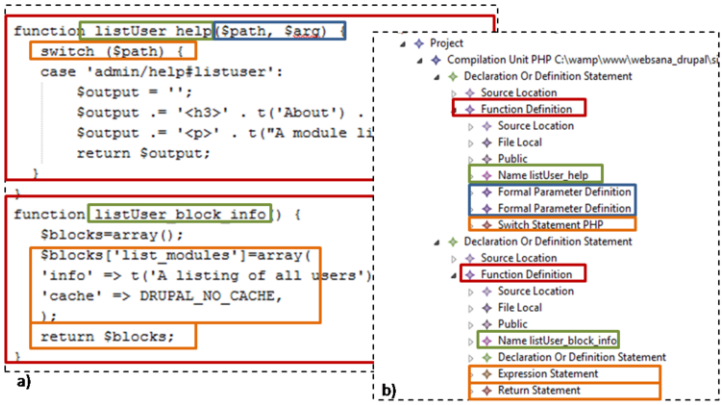

Fig. 7. Correspondence between the PHP code and the ASTM_PHP model

6 Implementation of M2M Transformations: From ASTM_PHP Models to KDM Models

In this section, we explain the implementation of the M2M transformations of our ADM-based migration method. These transformations automate the generation of a *KDM model* from the previous *ASTM_PHP model*.

These transformations represent the second task of our migration method (*generation of KDM models*)(see Fig. 1). The main objective of these transformations is to increase the abstraction level of the *ASTM_PHP models* by generating a platform-independent model.

These M2M transformations have been specified with a set of transformation rules defined at metamodel level (between the ASTM_PHP metamodel and the KDM metamodel). The implementation of these transformation rules is carried out with two tasks: 1)*definition of the transformation rules in natural language* and 2) *implementation of the transformation rules in ATL*. ATL provides a plugin which includes a set of functionality for the implementation of M2M transformations (editor, debugger, etc.). To illustrate these two tasks, we have used the elements captured in Fig. 7.

Table 2 presents the definition of the transformation rules in natural language. The first and second columns refer to the elements of the ASTM_PHP metamodel and their attributes (the source elements of these M2M transformation rules). Otherwise, the third and fourth columns refer to the elements of the KDM metamodel and their attributes (the target elements of the transformation rules).

In the following, we explain the transformation rules presented in Table 2.

- `Transformation rule between the element` ***`FunctionDefinition`*** `and the element` ***`MethodUnit`***: the element *FunctionDefinition* of the ASTM_PHP metamodel is mapped to the element *MethodUnit* of the KDM metamodel. The main attributes of this elements are: *export*, *name* and *codeElement*. The attribute *export* represents the visibility of the method. It takes its value from the attribute *accesskind* of the *FunctionDefinition* element. The attribute *codeElement* allows to store the statements of the body of the function, as well as the parameters

Table 2
Transformation rules between ASTM_PHP metamodel and KDM metamodel

ASTM_PHP element	**Attributes**	**KDM element**	**Attributes**
FunctionDefinition	accessKind identifierName formalParameters body	MethodUnit	export = accessKind name = identifierName codeElement = formalParameters/body
FormalParameter Definition	locationInfo identifierName definitionType	parameter Unit	source = locationInfo name = identifierName type = definitionType
SwitchStatementPHP	SwitchExpression cases	ActionElement	kind = switch codeElement = switchExpression,cases
ExpressionStatement	expression	ActionElement	kind = expression
ReturnStatement	returnValue	ActionElement	kind = return codeElement = returnValue

defined for the *MethodUnit*. The values of this attribute are mapped from the attributes *body* and *formalParameter* of the element *FunctionDefinition*. Finally, the attribute *name* takes the value from the attribute *identifierName*.

- `Transformation rule between the element` *`FormalParameterDefinition`* `and the element` *`ParameterUnit`*: the parameters passed to a function definition are represented by the element *FormalParameterDefinition* of the ASTM_PHP metamodel. This element is mapped to the *ParameterUnit* element of KDM metamodel. The main attributes of the element *ParameterUnit* are: *source*, *name* and *type*. The attribute *source* relates the *ParameterUnit* element with the source code. The value of this attribute is taken from the attribute *locationInfo* of the element *FormalParameterDefinition*. The attribute *name* defines the identifier of the parameter. The value of this attribute is mapped from the attribute *identifierName* of the element *FormalParameterDefinition*. Finally, the attribute *type* defines the data type of the parameter and its value is taken from the attribute *definitionType* of the element *FormalParameterDefinition*.
- `Transformation rule between the element` *`SwitchStatementPHP`* `and the element` *`ActionElement`*: the element *SwitchStatementPHP* of the ASTM_PHP metamodel is mapped to the element *ActionElement* of the KDM metamodel. The main attributes of the element *ActionElement* are: *kind* and *codeElement*. The attribute *kind* is a string that defined the semantic of the action. In this case, the *ActionElement* is defined as a *switch* statement. The attribute *codeElement* allows to store the statements defined within the body of the *switch* statement as well as the value that is assessed to execute the decision of the *switch*. The values of this attribute are taken from the attribute *cases* and *switchExpression* of the element *SwitchStatementPHP*.
- `Transformation rule between the element` *`ExpressionStatement`* `and the element` *`ActionElement`*: the element *ExpressionStatement* of the ASTM_PHP metamodel is mapped to the element *ActionElement* of the KDM metamodel. As it is said previously, the main attributes of the *ActionElement* are: *kind* and *codeElement*. In this case, the attribute *kind* takes the value depending on the expression stored in the attribute *expression* of the element *ExpressionStatement*. For instance, if it is an addition expression(*ADD expres-*

sion) the value of the attribute *kind* will be an *add*. The expression stored within the attribute *expression* will be stored in the attribute *codeElement* of the *ActionElement*.

- `Transformation rule between the element` ***`ReturnStatement`*** `and the element` ***`ActionElement`***: the element *ReturnStatement* of the ASTM_PHP metamodel is mapped to the element *ActionElement* of the KDM metamodel. In this case, the attribute *kind* of the element *ActionElement* takes the expression stored in the attribute *returnValue* of the element *ReturnStatement*.

After defining these transformation rules in natural language, we present the implementation of them in ATL. As we can see in the header presented in Fig. 8, these transformation rules are called *astm2kdm* and they have the *ASTM_PHP model* as the *IN model* and the *KDM model* as the *OUT model*.

```
module astm2kdm;
create OUT: kdm from IN: astm_php;
```

Fig. 8. Header of the M2M transformation rules

The implementation of a these transformation rules in ATL can be defined in three different types: *mapped rules*, *called rules* or *lazy rules* [11]. The mapped rules are the main ones since they are launched directly at the time the M2M transformation rule is executed. They are defined by two patterns: the *source pattern* that represents the element to be transformed and a *target pattern* that refers to the element being generated. The *called rules* and the *lazy rules* are launched from the *mapped rule* that is executed. The main difference between them is that *called rules* do not need to define a *source pattern* however *lazy rules* do not. Table 3 shows how the transformation rules presented previously have been implemented in ATL.

Table 3
Mapping between ASTM_PHP Metamodel and KDM Metamodel

ASTM_PHP element	**KDM element**	**ATL Rule**	**Type**
FunctionDefinition	MethodUnit	FunctionDef2MEthodUnit	matched
FormalParameter Definition	ParameterUnit	CreateParameterUnit	called
SwitchStatementPHP	ActionElement	Switch2Action	matched
ExpressionStatement	ActionElement	ExpressionStatement2Action	matched
ReturnStatement	ActionElement	Return2Action	matched

The first column of the Table 3 presents the source element of the M2M transformation and the second column is the target element. The third column is the name of the transformation rule implemented in ATL. Finally, the last reflects the type of this rule (*matched*, *called* or *lazy*).

Fig.9 shows the implementation in ATL of the *functionDef2MethodUnit* transformation rule. As we can see in the figure, it is implemented as a *matched* rule. The *source pattern* (*from*) defines the variable *funcdef* representing the *FunctionDefinition* element of the ASTM_PHP metamodel. Otherwise, the *target pattern*

(*to*) creates a variable called *meth* representing the new *MethodUnit* element of the KDM metamodel. In the body of the *target pattern*, the attributes *name*, *export* and *codeElement* of the element *MethodUnit* takes their values from the attributes of the *FunctionDefinition* element. It is worth noting that to map the values stored in the attribute *formalParameters* of the element *FunctionDefinition* it is required to create a *Signature* element which is created by using the *called* rule *CreateSignature*.

```
rule funtionDef2MethodUnit {
    from
        funcdef: astm!FunctionDefinition
    to
        meth: kdm!MethodUnit (
            name <- funcdef.declOrDefn.identifierName.getName.debug('path'),
            export <- funcdef.declOrDefn.accessKind.getExportKind,
            codeElement<-funcdef.declOrDefn.body
            codeElement <- thisModule.createSignature
            (funcdef.declOrDefn.identifierName.getName,
                funcdef.declOrDefn.formalParameters),
        )
}
```

Fig. 9. FunctionDef2ActionElement ATL rule

Fig.10 shows the correspondence between the elements of the *ASTM_PHP model* (presented in Fig.10.a) and the elements of the *KDM model* resulted from the execution of the *astm2kdm* M2M transformations (presented in Fig. 10.b).

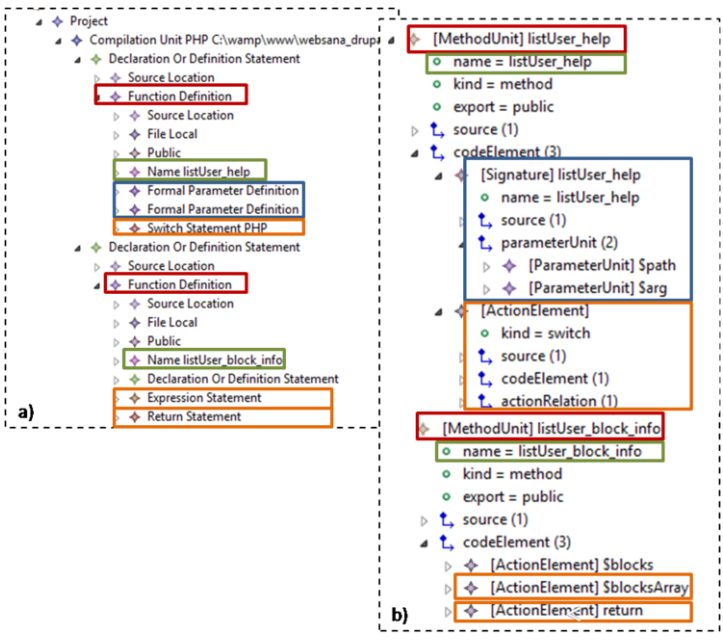

Fig. 10. Correspondence between the ASTM_PHP model and KDM model

The *MethodUnit* elements which have been generated from the *FunctionDefintions* elements by executing the transformation rule *functionDef2MethodUnit* pre-

sented in Fig.9, have been marked in red. In green, we have marked the name of the *MethodUnit* which is mapped from the name of the *FunctionDefinition*. Otherwise, We have marked in blue the *ParameterUnits* generated from the *FormalParameterDefinitions*. As we can see in Fig.10.b, the *ParameterUnits* have been generated within the *Signature* element. Finally, the *SwitchStatementPHP*, the *ExpressionStatement* and the *ReturnStatement*, marked in orange in Fig.10.a, have been mapped to *ActionElements* in the *KDM model*. As for the element *SwitchStatementPHP*, we can observe that the *ActionElement* generated within the *KDM model* defines its attribute *kind* with the value of *switch*.

7 Related Works

In this section, we present some of the current ADM-based approaches found in the literature. In Table 4 we analyze each approach and we compare them with the ADM-based migration method that we propose. To analyze thoroughly these approaches, we defined five criteria presented below:

- `Source code`: this criterion is related to the source code implementing the legacy system from which the approach extracts the models and starts the reengineering process.
- `Metamodel`: this criterion allows to analyze the metamodels used by the different approaches to define the models. Accordingly, the approaches can: 1)define their own metamodel to represent the models at any abstraction level, 2) use existing standard metamodels or 4) use the ADM standard metamodels (ASTM, KDM or SMM).
- `M2M transformations`: this criterion is used to analyze the types of M2M transformations considered by the approaches. It is possible to find two types of M2M transformations, vertical or horizontal [20]. Vertical M2M transformations involve a change of the abstraction level for the resulting model of the transformation, e.g. a PSM is transformed into a PIM. Otherwise, horizontal M2M transformations generate the resulting model at the same abstraction level as the source model.
- `Context`: this criterion analyzes the aspect in which the approach is focused, so that it is possible to find approaches focused on: 1) the reengineering of data bases, 2) the reengineering of Web services or 3) legacy systems (generic context).
- `Toolkit`: this criterion is about the availability of tools supporting the tasks defined in each reengineering approach, such as graphical editors for creating models or frameworks allowing to run the automated transformations.

After presenting these criteria, we analyze the different approaches applying them.

Van Hoorn et al - *DynaMod*

Van Hoorn et al presents in [38] *DynaMod*, a reengineering approach called *DynaMod* which addresses the model-driven modernization of software systems.

This method is composed of three tasks: 1) the extraction of a Java architectural model from Java which conforms to the KDM metamodel; this extraction considers static and dynamic analysis, 2) the Java architectural model is restructured with extra information provided by system experts, finally 3) the automatic generation of the target architectural models for the target system conforming to the KDM metamodel. All the models used by *DynaMod* are defined at PIM level. To generate automatically the target architectural model, they define M2M transformations. *DynaMod* is not focused on any specific domain and it is not supported by any toolkit for systematizing the reengineering process or editing the models.

Sadovykh et al

Sadovykh et al present in [32] an approach for migrating legacy systems implemented in C++ to target systems implemented in Java. It addresses three tasks: 1) the extraction of a platform-specific UML model from the C++ code conforming to the UML metamodel [4], 2) the generation of a platform-independent UML model from the platform-specific UML model, 3) the restructuring of the generated platform-independent UML model eliminating platform dependencies and extracting business logic, finally 4) the generation of a platform-specific Java UML model from the platform-independent UML model. The UML models specified by this method are defined at PSM level and at PIM level. The UML model extracted from the code (the first one) as well as the Java UML model are defined at PSM level, whereas the UML model obtained from the first one is defined at PIM level. To generate automatically the platform-independent UML model and the Java UML model, they define M2M transformations. This approach is focused on Commercial-Off-The-Shelf Software (COTS). Finally, this approach is supported by the UML2 Toolki [10] to generate the UML model at PIM level.

Perez-Castillo et al - *Preciso*

Perez-Castillo et al present in [31] a reengineering process called *Preciso* to recover and implement Web Services in automatic manner from relational databases. This process is based on ADM and it is composed of the four following stages: 1) extraction of a SQL-92 model according to the SQL-92 metamodel from a relational database, 2) transformation of this SQL-92 model into the object model, conforming to UML2 metamodel, which raises the abstraction level of the system, 3) generation of a WSDL model conforming to the WSDL metamodel [8] from the object model and 4) generation of the code from the object model and the WSDL model. This code is the basis for implementing the infrastructure of Web Services. The SQL-92 model and the WSDL model are defined at PSM level, whereas the object model is defined at PIM level. To obtain the object model from the SQL-92 model as well as the WSDL model from the object model a set of M2M transformations are defined. This approach is focused on extracting Web Services from relational databases in automatic manner. Finally, this approach is supported by a comprehensive tool. This tool supports and automates all the process as well as proposes the use of editors to define the models.

Bruneliere et al - *MoDisco*

Bruneliere et al present in [5] *MoDisco* an extensible approach for model-driven reverse engineering which allows extracting platform models from Java, XML and JSP. This approach just considers the reverse engineering stage and concretely the task of knowledge extraction. Therefore, *MoDisco* can extract from legacy systems models at PIM level conforming to KDM as well as at PSM level conforming to a set of platform metamodels such as Java, XML and JSP metamodels. This approach does not define M2M transformations but T2M transformations. It is not constrained to a specific context, thus it can be applied to any legacy system. and it is supported by a comprehensive toolkit.

Blanco et al

The work presented in [2] presents an ADM-based method focused on obtaining conceptual security models from legacy OLAP systems [1]. It allows to analyze OLAP systems in order to include new security requirements to improve the architecture as well as to migrate to other platforms. The process is composed of the following tasks: 1) from an OLAP system they extract automatically a DataWarehouse model which conforms to the Secure Multidimensional Logical Metamodel (SECMDDW) which considers the common structure of OLAP systems, 2) from this logical model they generate automatically a conceptual model which conforms to SECDW metamodel that allows the representation of structural aspects of DataWarehouses and the definition of several kinds of security rules. It is complemented by an Access Control and Audit (ACA) model focused on DataWarehouse confidentiality. The code is generated by executing M2T transformations. This approach models different aspects of an OLAP system at different levels of abstraction. The DataWarehouse model is defined at PSM level, whereas the conceptual model and ACA models are defined at PIM level. The restructuring stage is carried out at PIM level. To extract the DataWarehouse model they define T2M transformation. On the other hand, to obtain the conceptual model and ACA model, they defined M2M transformations. This approach is focused on legacy OLAP systems. It does not take into account any tool that addresses the automation of the approach.

Vasilecas et al

Vasilecas et al presents in [39] a model-driven process for extracting business rules from existing legacy systems. This method is composed of the following stages: 1) the extraction of an ASTM model conforming to the ASTM metamodel from the source code of a legacy, 2) the ASTM model is mapped to a code model conforming to the KDM metamodel, 3) the application of software design recover techniques to the KDM model to extract business rules, this techniques are based on the GUIDE Business Rule project [14] that classifies the business rules into four categories: business terms, facts, constraints, and derivations, finally 4) with the extracted and classified business rules a KDM conceptual model is created. This model conforms to the Semantics of *Business Vocabulary and Business Rules (SBVR)* [25]. The ASTM model is defined at PSM level because describes the source code of the legacy

system, whereas the code model is defined at PIM level. The restructuring of the code model to obtain the KDM conceptual model is carried out at PIM level, but the resulting KDM conceptual model is defined at CIM level. For the extraction of the ASTM model, they define T2M transformations, and for the automatic generation of the code model, they specify M2M transformations. This approach is not focused on any specific context. It is defined as an approach to extract business rules from any legacy information systems. It is supported by a set of tools based on *Eclipse Modeling Framework (EMF)* [6]. The transformations are supported by the ATL IDE, whereas to edit, query and import/export to XMI format the KDM models, they use the KDM SDL toolkit.

Perez-Castillo et al - *Marble*

Perez-Castillo et al presents in [28] *Marble (Modernization Approach for Recovering Business Processes from LEgacy systems)*, an ADM-based framework for recovering business processes from legacy systems. *Marble* defines three transformations tasks: 1) model extraction, where a code model is obtained from the Java code which implements the legacy system. 2) generation of a KDM model, this KDM model is defined considering the code and action packages of the KDM metamodel, 3) obtaining business process model, the business process model which conforms to BPMN [42] represents the business discovered and is manually restructured by business experts. In *Marble* we can find models defined at different abstraction levels. The code model is defined at PSM level, whereas the KDM model is defined at PIM level. Finally, the business process model is defined at CIM level. To obtain the code model they implement T2M transformations. To obtain the KDM model from the Java model and the business process model from the KDM model, they define M2M transformations. This approach is not focused on any specific context. Finally, a complete tool supports the automation of the reengineering process of *Marble* although they do not mention the use of graphical editors to mange the models.

In Table 4 we present the approaches found and we compare them with our ADM-based migration method.

The second column in Table 4 refers to the source code from which the approach extracts the models. Van Hoorn et al, Bruneliere et al (*MoDisco*), Blanco et al and Perez-Castillo et al (*Marble*) extract models from Java code. Moreover, Bruneliere et al (*MoDisco*) extracts models from XML and JSP code. Otherwise, Perez-Castillo et al (*Preciso*) extracts from SQL-92 and Sadovych et al from C++. Finally, Vasileca et al extracts models from Visual Basic code. As we can observe, none of them addresses the extraction of models from PHP code. Thus, our ADM-based migration method may be the unique approach which has implemented T2M transformations to extract models from this programming language.

The third column refers to the metamodel which the extracted models of each approach conform to. Some of the approaches define their owm metamodels to construct their models. For instance, Perez-Castillo et al (*Preciso*) defines the SQL-92 metamodel, Bruneliere et al (*MoDisco*) specifies the Java, XML and JSP

Table 4
ADM-based migration approaches

Approach	Source code	Metamodel	M2M	Context	Toolkit
Van Hoorn et al (DynaMod)	Java	KDM	PIM2PIM (horizontal)	Generic	No
Sadovykh et al	C++	UML	PSM2PIM (vertical)	Generic	Yes
Perez-Castillo et al (Preciso)	SQL-92	SQL-92	PSM2PIM (vertical)	Data base and Web services	Yes
Bruneliere et al (Modisco)	Java XML and JSP	Java XML JSP and KDM	Not defined	Generic	Yes
Blanco et al	Not defined	SECMDDW, SECDW	PSM2PIM (vertical)	Data base	No
Vasilecas et al	Visual Basic	ASTM KDM	PSM2PIM (vertical)	Generic	Yes
Perez-Castillo et al (Marble)	Java	Java, KDM and BPMN	PSM2PIM (vertical)	Generic	Yes
Our ADM-based migration method	PHP	ASTM_PHP KDM CMS Common Metamodel	PSM2PIM (vertical) PIM2PIM (horizontal)	CMS-based Web applications	Yes

metamodels, Blanco et al defines the SECMDDW and SECDW metamodels and Perez-Castillo et al (*Marble*) specifies its own Java metamodel. Some of them consider standard metamodels such as Sadovykh et al taking into account the UML metamodel or Perez-Castillo et al (*Marble*) that uses the BPMN metamodel. Finally, the ADM standard metamodels are used by some of the approaches; for instance, Van Hoorn et al, Bruneliere et al (*MoDisco*), Vasilecas et al and Perez-Castillo et al (*Marble*) uses the KDM metamodel. The ASTM standard metamodel is just considered by Vasilecas et al. For our ADM-based migration method, we have bet for the use of both ADM standard metamodels (ASTM and KDM). We have extende the ASTM metamodel with specific elements of the PHP code creating the ASTM_PHP metamodel. Furthermore, we have defined the CMS Common Metamodel, the cornerstone of our approach.

The fourth column indicates the type of M2M transformation performed in each approach. We can consider two types of M2M transformations depending on the abstraction level of the source model and the target model: vertical and horizontal transformations. The vertical M2M transformations entail a change at the abstraction level, e.g. when a transformation convert a PSM model into a PIM model, whereas the horizontal ones do not entail a change of abstraction level. As it is shown in Table 4, most of the approaches found consider vertical M2M transformations between PSM models and PIM models. In this case, our ADM-based migration method considers both types.

The fifth column in Table 4 specifies the context of the approach. If the approach is focused in the migration of legacy systems without specifying an specific domain, we have considered it as a generic context. Most of the approaches are not defined for a specific context. Two of the approaches are focus on data base context and one of them also in web services context. It is possible to observe that none of the approaches is focused on the migration of CMS-based Web applications. Thus, our

approach becomes the only one addressing this type of migration.

Finally, the sixth column analyzes whether the approach is supported by a toolkit that allows the systematization and automation of the approach. As we can see in Table 4, most of the approaches implement a toolkit that allow the edition of the models or the automation of the transformations. Our ADM-based migration method is supported by a toolkit.

8 Conclusions and Future Works

During the last years, the volume of digital content has increased exponentially. Organizations have experienced the necessity of using powerful platforms to manage all this huge amount of content in a robust and reliable manner. The most adopted solution has been to use Content Management Systems (CMS).

Many organizations have migrated their traditional Web applications to CMS-based Web applications. This migration process entails high risks and costs for organizations. To support this migration process and to mitigate their drawbacks, we propose a method for migrating traditional Web application to CMS-based Web applications. This method is based on the principles of ADM: using models and automated transformations. To define the models, we propose the use of two standard metamodels defined by ADM: Abstract Syntax Tree Metamodel (ASTM) and Knowledge Discovery Metamodel (KDM). Moreover, this migration method is composed of three stages defining a horseshoe process: 1) *reverse engineering stage*, 2) *restructuring stage* and 3) *forward engineering stage*.

The work presented in this article is focused on the *reverse engineering stage* of our method. On the one hand, we have defined a modeling language that allows the definition of platform-specific models representing the syntax of the source code in PHP (*ASTM_PHP models*). This modeling language is defined using a Domain Specific Language (DSL) that we have called ASTM_PHP DSL. On the other hand, we have defined and implemented a set of automated M2M transformations that allow to generate platform-independent models (*KDM models*) from the *ASTM_PHP models*.

To define the ASTM_PHP DSL, we have specified an *abstract syntax* and a *concrete syntax*. The *abstract syntax* is defined by a metamodel called ASTM_PHP metamodel. This metamodel is an extension of the ASTM standard metamodel proposed by ADM. We have extended the ASTM metamodel with specific elements of the PHP code. These new elements have been specified within the Specific Abstract Syntax Tree Metamodel (SASTM) domain. The *concrete syntax* has been defined with a tree-like graphical editor that allows the definition and edition of models conforming to the ASTM_PHP metamodel. This graphical editor has been implemented by the Eclipse Modeling Framework (EMF).

For the implementation of the automated M2M transformations allowing to obtain *KDM models* from *ASTM_PHP models*, we have defined a set of transformation rules. Firstly, these transformation rules have been defined in natural language and then we have implemented them using in Atlas Transformation Language (ATL).

This implementation has been carried out by using matched rules, lazy rules and called rules.

Up to now, our method focuses on the migration of Web applications implemented in PHP. As future work, our research group is working to allow the migration to CMS platforms of Web applications implemented in any programming language. Moreover, our group is working on the definition of other two modeling languages for defining the knowledge involved in the migration process carried out by our ADM-based migration method. Moreover, we work on the definition and implementation of the rest of transformations that automate this method.

9 Acknowledgement

This research has been partially funded by the Project MASAI (TIN-2011-22617) from the Spanish Ministry of Science and Innovation.

References

[1] Berson, A. and S. J. Smith, "Data warehousing, data mining, and OLAP," McGraw-Hill, Inc., 1997.

[2] Blanco, C., R. Pérez-Castillo, A. Hernández, E. Fernández-Medina and J. Trujillo, "Towards a modernization process for Secure Data Warehouses," Springer, 2009.

[3] Boiko, B., *Understanding content management*, Bulletin of the American Society for Information Science and Technology **28** (2001), pp. 8–13.

[4] Booch, G., J. Rumbaugh and I. Jacobson, "The unified modeling language user guide," Pearson Education India, 1999.

[5] Bruneliere, H., J. Cabot, F. Jouault and F. Madiot, *Modisco: a generic and extensible framework for model driven reverse engineering*, in: *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ACM, 2010, pp. 173–174.

[6] Budinsky, F., "Eclipse modeling framework: a developer's guide," Addison-Wesley Professional, 2004.

[7] Chikofsky, E. J., J. H. Cross et al., *Reverse engineering and design recovery: A taxonomy*, Software, IEEE **7** (1990), pp. 13–17.

[8] Christensen, E., F. Curbera, G. Meredith and S. Weerawarana, *Web service definition language (wsdl)*, http://www.w3c.org/TR/wsdl (2001), accessed: 10/02/2014.

[9] Drupal, *Drupal cms*, http://drupal.org (2014), accessed: 14/01/2014.

[10] Eriksson, H.-E., M. Penker, B. Lyons and D. Fado, "UML 2 toolkit," John Wiley & Sons, 2003.

[11] Foundation, E., *Atlas transformation language*, http://www.eclipse.org/atl/ (2013), accessed: 22/05/2013.

[12] Fowler, M., "Domain-specific languages," Pearson Education, 2010.

[13] Ginige, A. and S. Murugesan, *Web engineering: An introduction*, MultiMedia, IEEE **8** (2001), pp. 14–18.

[14] Hay, D., K. Healy and J. Hall, *Defining business rules˜ what are they really? 2000*, The Business Rule Group (2000).

[15] Izquierdo, J. L. C. and J. G. Molina, *An architecture-driven modernization tool for calculating metrics*, Software, IEEE **27** (2010), pp. 37–43.

[16] Khusidman, V. and W. Ulrich, *Architecture-driven modernization: Transforming the enterprise*, in: *Seminar Software Analyse and Trasformation*, 2007, p. 7.

[17] McKeever, S., *Understanding web content management systems: evolution, lifecycle and market*, Industrial management & data systems **103** (2003), pp. 686–692.

[18] Mellor, S. J., T. Clark and T. Futagami, *Model-driven development: guest editors' introduction.*, IEEE software **20** (2003), pp. 14–18.

[19] Mellor, S. J., K. Scott, A. Uhl and D. Weise, *Model-driven architecture*, Computing Reviews **45** (2004), p. 631.

[20] Mens, T. and P. Van Gorp, *A taxonomy of model transformation*, Electronic Notes in Theoretical Computer Science **152** (2006), pp. 125–142.

[21] Miller, J. and J. Mukerji, *Model driven architecture: A technical perspective* (2001).

[22] Murugesan, S., Y. Deshpande, S. Hansen and A. Ginige, *Web engineering: A new discipline for development of web-based systems*, in: *Web Engineering*, Springer, 2001 pp. 3–13.

[23] OMG, *Abstract syntax tree metamodel*, http://www.omg.org/spec/ASTM/1.0 (2005), accessed: 02/02/2014.

[24] OMG, *The Meta Object Facility (MOF) core specification*, http://www.omg.org/mof/ (2006), accessed: 01/06/2006.

[25] OMG, *Business vocabulary and business rules*, http://www.omg.org/spec/SBVR/ (2008), accessed: 23/04/2014.

[26] OMG, *Information technology - Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM)*, http://www.omg.org/technology/kdm/ (2012), accessed: 11/05/2014.

[27] OMG, *Structured metrics metamodel*, http://www.omg.org/spec/SMM/ (2012), accessed: 12/02/2013.

[28] Pérez-Castillo, R., *Marble: Modernization approach for recovering business processes from legacy information systems*, in: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, IEEE, 2012, pp. 671–676.

[29] Pérez-Castillo, R., I. G. R. de Guzmán and M. Piattini, *Architecture-driven modernization*, Modern Software Engineering Concepts and Practices: Advanced Approaches (2010), p. 75.

[30] Pérez-Castillo, R., I. G.-R. De Guzman and M. Piattini, *Knowledge discovery metamodel-iso/iec 19506: A standard to modernize legacy systems*, Computer Standards & Interfaces **33** (2011), pp. 519–532.

[31] Perez-Castillo, R., I. Garcia-Rodriguez de Guzman, I. Caballero, M. Polo and M. Piattini, *Preciso: A reverse engineering tool to discover web services from relational databases*, in: *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on*, IEEE, 2009, pp. 309–310.

[32] Sadovykh, A., L. Vigier, A. Hoffmann, J. Grossmann, T. Ritter, E. Gomez and O. Estekhin, *Architecture driven modernization in practice–study results*, in: *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*, IEEE, 2009, pp. 50–57.

[33] Saraiva, J. d. S. and A. R. d. Silva, *Development of cms-based web-applications using a model-driven approach*, in: *Proceedings of the 2009 Fourth International Conference on Software Engineering Advances*, IEEE Computer Society, 2009, pp. 500–505.

[34] Shreves, R., *Open source cms market share*, Retrieved June **2** (2008), p. 2009.

[35] Sneed, H. M., *Estimating the costs of a reengineering project*, in: *Reverse Engineering, 12th Working Conference on*, IEEE, 2005, pp. 9–119.

[36] Souer, J. and T. Kupers, *Towards a pragmatic model driven engineering approach for the development of cms-based web applications*, in: *Proceedings of the 5th Model Driven Web Engineering Workshop (MDWE09)*, 2009, pp. 31–45.

[37] Trias, F., V. de Castro, M. L. Sanz and E. Marcos, *A systematic literature review on cms-based web applications.*, in: *ICSOFT*, 2013, pp. 132–140.

[38] Van Hoorn, A., S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp et al., *Dynamod project: Dynamic analysis for model-driven software modernization*, in: *Proceedings of the 1st International Workshop on Model-Driven Software Migrations (MDSM 2011)*, 2011, pp. 12–13.

[39] Vasilecas, O. and K. Normantas, *Deriving business rules from the models of existing information systems*, in: *Proceedings of the 12th International Conference on Computer Systems and Technologies*, ACM, 2011, pp. 95–100.

[40] Vidgen, R., S. Goodwin and S. Barnes, *Web content management*, in: *Proceedings of the 14th International Electronic Commerce Conference*, 2001, pp. 465–480.

[41] Vlaanderen, K., F. Valverde and O. Pastor, *Model-driven web engineering in the cms domain: A preliminary research applying sme*, in: *Enterprise Information Systems*, Springer, 2009 pp. 226–237.

[42] White, S. A., *Introduction to bpmn*, IBM Cooperation **2** (2004), p. 0.