A Toolkit for ADM-based Migration: Moving from PHP Code to KDM Model in the Context of CMS-based Web Applications [1]

Feliu Trias *feliu.trias@urjc.es*
Kybele Research Group, Rey Juan Carlos University
Móstoles, Spain

Valeria de Castro *valeria.decastro@urjc.es*
Kybele Research Group, Rey Juan Carlos University
Móstoles, Spain

Marcos López-Sanz *marcos.lopez@urjc.es*
Kybele Research Group, Rey Juan Carlos University
Móstoles, Spain

Esperanza Marcos *esperanza.marcos@urjc.es*
Kybele Research Group, Rey Juan Carlos University
Móstoles, Spain

Abstract

In the last few years, many organizations have based their Web applications on Content Management Systems (CMS) because of the advantages they provide to manage their huge amount of digital content. The objectives of these organizations change, for this reason they may see the necessity of migrating their CMS-based Web applications to other CMS platforms meeting better their needs. Thus, we propose a method based on Architecture-Driven Modernization (ADM) to automate this migration process. In this paper we present the toolkit supporting this ADM-based migration method. For space restrictions, we focus on the implementation of two modules of this ADM-based toolkit: i) the ASTM_PHP DSL, a modeling language which allows to model the code of a system implemented in PHP (ASTM_PHP models) and ii) the model-to-model transformation rules which allow to generate KDM models from the information captured in the ASTM_PHP models. To show its usability, we present a case study where a widget listing online users of a CMS-based Web application is migrated from Drupal to Wordpress.
Keywords: Content Management System; Web Application; Architecture-Driven Modernization; Reverse Engineering and Model-driven Engineering.

1. Introduction

In the last decade, the volume of digital content managed by Web applications has grown dramatically as well as the processes supported have become more complex. Thus, organizations have experienced the necessity of using powerful management platforms to maintain their large-scale Web applications and manage all their content in a robust and reliable manner [12]. One of the most adopted solutions has been to base Web applications on Content Management Systems (CMS) [1]. These CMS-based Web Applications provide with some features such as, dynamic creation of content or flexible functionality extension [24].

Currently, a considerable number of different CMS platforms is found in the market. This fact, along with the changes in the objectives of organizations, can cause organizations to see

1 Research funded by the project *Definición de un proceso de desarrollo centrado en la arquitectura para el alineamiento de servicios de negocio* (ref. 2013/00194/001) from Rey Juan Carlos University.

the necessity of migrating their CMS-based Web applications to other CMS platforms meeting better their needs.

This migration process entails a complex, time-consuming and error-prone reengineering process [5]. Currently, the Architecture-Driven Modernization (ADM) [15] is considered one of the most effective approaches to systemize this process and to mitigate their drawbacks. ADM advocates for the application of MDA (Model-Driven Architecture) [13] techniques and tools in the migration process. Furthermore, it develops a set of standard metamodels to represent the information involved in this process. Two of these metamodels are: the Abstract Syntax Tree Metamodel (ASTM) [14], which allows to represent at platform-specific level the syntax of the code implementing a legacy system, and the Knowledge Discovery Metamodel (KDM) [10] allowing to represent at platform-independent level the syntax and semantics of this code.

To the best of our knowledge, there is not any model-driven reengineering method in the literature [20] systemizing the migration process from a CMS-based Web applications to other CMS platforms. To solve this gap, we define an ADM-based migration method to support this process [21]. It is composed of three reengineering stages defining a "horseshoe" process [5]: *reverse engineering stage*, *restructuring stage* and *forward engineering stage*. Up to now, this method is focused on the migration of Web applications based on open-source CMS platforms such as Drupal [6], Joomla! [11] or Wordpress [25] because of the their spread use and relevant acceptance in the market [18]. Most of these open-source CMS platforms are implemented in PHP, so that we pay special attention on this code.

This paper presents the implementation of the toolkit supporting this ADM-based migration method. For space restrictions, we focus on the implementation of two modules: i) the ASTM_PHP Domain Specific Language (ASTM_PHP DSL), a modeling language based on the standard metamodel ASTM_PHP which allows to define platform-specific models which represent the code of a system implemented in PHP (ASTM_PHP models) and ii) the model-to-model (M2M) transformation rules which allow to generate KDM models, from the information captured in the ASTM_PHP models, which represent the syntax and semantics of the PHP code at a platform-independent level.

To show the usability of this ADM-based toolkit we present a case study where a widget listing online users of a CMS-based Web application is migrated from Drupal to Wordpress.

The rest of this paper is organized as follows: Section 2 provides an explanation of the ADM-based migration method which is supported by the toolkit presented. Section 3 presents the implementation of the two modules of the ADM-based toolkit. Section 4 presents the related works and finally, Section 5 presents the conclusions and future works.

2. An ADM-based Method for Migrating CMS-based Web Applications

The toolkit presented in this paper supports the ADM-based migration method presented in [21]. It is composed of three stages (Fig. 1 shows this process).

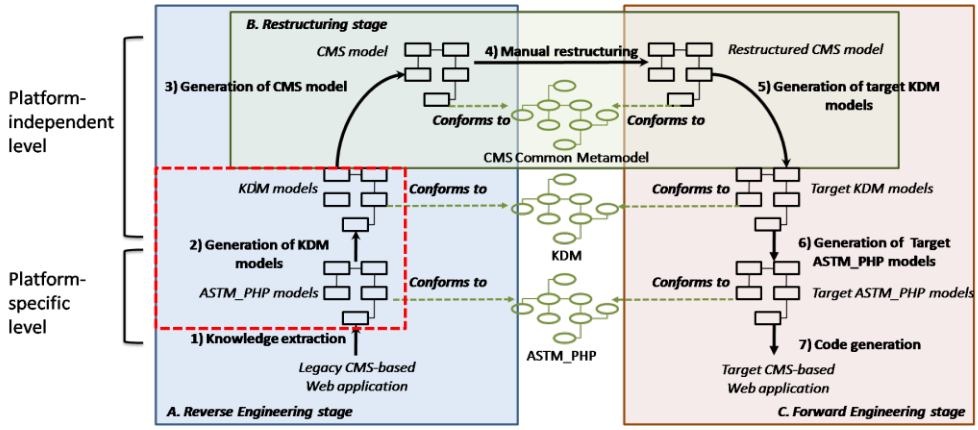

Fig. 1. ADM-based method for migrating CMS-based Web Applications.

A. Reverse engineering stage, this stage is composed of three tasks: 1) *the knowledge extraction*, focused on the extraction of ASTM_PHP models from PHP code by defining text-to-model (T2M) transformations; 2) *the generation of the KDM model*, focused on the M2M transformations generating the KDM models from the ASTM_PHP models.; 3) *the generation of the CMS model*, focused on the M2M transformations generating the CMS model from the KDM models. The CMS model contains the information involved in this migration process but represented into the CMS domain. It conforms to the CMS Common Metamodel [22].

B. 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 to which is intended to migrate.

C. Forward engineering stage, this stage represents a top-down development process. It is composed of three tasks: 5) *the generation of the target KDM model*, is focused on the M2M transformations generating the target KDM models from the restructured CMS model; 6) *the generation of the target ASTM model*, is focused on the M2M transformations generating the target ASTM_PHP models from the target KDM models and finally, 7) *the code generation*, generates the code implementing the target CMS-based Web application by defining automatic model-to-text (M2T) transformations.

The two modules presented in this paper are framed in the reverse engineering stage. Concretely, in *the generation of KDM models* task (see in Fig. 1 the part marked in a red dotted line).

3. The ADM-based Toolkit

In this section we present the implementation of the two modules of our ADM-based toolkit. Fig. 2.a shows the tasks composing the *reverse engineering stage* of our ADM-based migration method and Fig. 2.b presents the two modules of the ADM-based toolkit framed in *the generation of KDM models* task (see Fig. 1). On the one hand, we have the ASTM_PHP DSL, a modeling language which allows to define ASTM_PHP models representing the syntax of PHP code in a proper and non-ambiguous way (Section 3.2 presents its implementation); on the other hand, we present the implementation in ATL of the M2M transformation rules which automate the generation of KDM models from ASTM_PHP models (Section 3.3 presents its implementation).

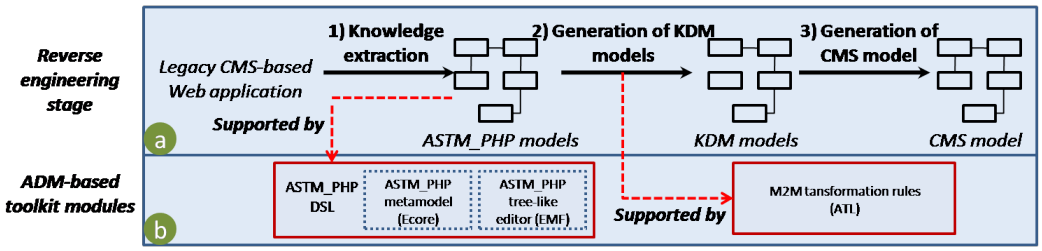

Fig. 2. a) Reverse engineering stage, b) ADM-based toolkit modules.

3.1. Case Study

To show the usability of our ADM-based toolkit, we present a case study where we migrate a widget listing online users from a CMS-based Web application implemented on Drupal to Wordpress. 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 this widget implemented in Drupal and Fig. 3.b in Wordpress.

Fig. 3. a) Plugin Online users on Drupal, b) Plugin Online users on Wordpress.

3.2. Implementation of the ASTM_PHP DSL

As we can see in Fig. 2.b, the implementation of the ASTM_PHP DSL requires two tasks: the definition of the ASTM_PHP metamodel (abstract syntax) and the implementation of a tree-like editor (concrete syntax) [8] which allows to define graphically models conforming to the ASTM_PHP metamodel.

The ASTM_PHP metamodel captures all the required elements and their relationships in a model to represent the syntax of the PHP code which implements a system. It is defined as an extension of the standard metamodel ASTM and implemented as an Ecore model [4]. The ASTM metamodel has two parts: the Generic Abstract Syntax Tree Metamodel (GASTM) and the Specific Abstract Syntax Tree Metamodel (SASTM) [14]. For the definition of the ASTM_PHP metamodel, we have extended the SASTM with specific PHP elements. These elements can be classified in those which were not considered in GASTM and those even existing in GASTM did not fit well with the specification of the PHP element and need to be redefined. GASTM provides a set of abstract classes to be extended for defining the specific PHP elements within SASTM.

Table 1 presents all the classes included within SASTM. The first column refers to the abstract GASTM class which has been extended; the second column denotes the name of the SASTM class; the third column indicates whether the SASTM class is a new or redefined element and finally, the fourth column is a description of the SASTM class. Most of these SASTM classes represent new operators and expressions. Otherwise, three of them represent redefined elements (two statements and one expression).

Table 1. Classes composing SASTM.

GASTM class	SASTM class	New/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
Expression	ObjectAccess	New	Access to a item of an object
	ClassAccess	New	Access to a item of a class
	DuplaArray	New	An entry of an array
	ArrayAccessPHP	Redefined	Array access in PHP

The implementation of the tree-like editor is based on the Eclipse Modeling Framework (EMF) [4]. This framework allows implementing automatically the tree-like editor from a generator model (GenModel). From this GenModel, EMF generates the Java code organized in three different packages: the model code, editing model, the editor code and code for the testing. To illustrate the ASTM_PHP DSL, we define in Fig. 4 a model by using the tree-like editor which represents the PHP code of the widget of our case study.

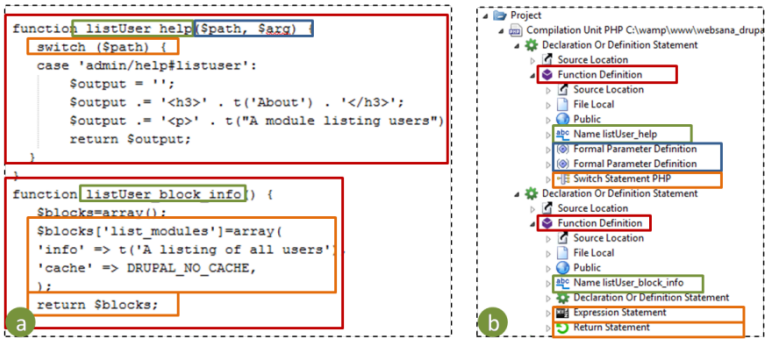

Fig. 4. a) PHP code, b) ASTM_PHP model.

Fig. 4.a shows the PHP code and Fig. 4.b represents the corresponding ASTM_PHP model. As we can see, we have marked in red the function definitions and in green the name of these functions. Furthermore, we have marked in blue the parameters passed to the functions. Finally, we have denoted in orange some of the sentences composing the body of the functions (switch sentence, an array definition and a return sentence).

3.3. Implementation of the M2M Transformation Rules

As we can see in Fig. 2.b, the other module is the M2M transformation rules which support the generation of KDM models from ASTM_PHP models. Their main aim is to raise the abstraction level generating platform-independent models (KDM models) from platform-specific models (ASTM_PHP models). These M2M transformation rules are firstly defined in nature language (see Table 2) and then implemented in Atlas Transformation Language (ATL) [7].

The transformation rules presented in Table 2 correspond to the elements found in the ASTM_PHP model in Fig. 4.b. On the one hand, the first and second column refers to the ASTM_PHP classes and their attributes. On the other hand, the third and fourth column refers to the KDM classes and their attributes.

Table 2. Definition of M2M transformation rules: ASTM_PHP → KDM.

ASTM_PHP class	Attributes	KDM class	Attributes
Function Definition	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
SwitchStatement PHP	switchExpression cases	Action Element	kind = switch codeElement = switchExpression, cases
Expression Statement	expression	Action Element	kind = expression
ReturnStatement	returnValue	Action Element	kind = return codeElement = returnValue,

For space limitations, we just explain the transformation rule from the *FunctionDefinition* class to *MethodUnit* class. The main attributes of the *MethodUnit* class are: *export, name* and *codeElement*. The *export* attribute represents the visibility of the method and takes its value from the attribute *accesskind*. The *codeElement* attribute allows storing the sentences defined within the body of the function as well as the parameters passed. The values of this attribute are taken from the *body* and *formalParameters* attributes. Finally, the attribute *name* takes the value from the attribute *identifierName*.

A transformation rule implemented in ATL can be of three types: *mapped rules*, *called rules* and *lazy rules* [7]. Table 3 shows the implementation in ATL of the transformation rules defined in Table 2. The first column shows the ASTM_PHP classes (source class) and the

second column the KDM classes (target class). The third column presents the name of the transformation rule implemented in ATL and finally, the fourth column shows the type of the transformation rule (*matched, called* or *lazy rule.*)

Table 3. Implementation of transformation rules: ASTM_PHP → KDM.

ASTM_PHP class	KDM class	Name	Type
FunctionDefinition	MethodUnit	functionDef2MEthodUnit	matched
FormalParameterDefinition	ParameterUnit	CreateParameterUnit	called
SwitchStatementPHP	ActionElement	Switch2Action	matched
ExpressionStatement	ActionElement	ExpressionStatement2Action	matched
ReturnStatement	ActionElement	Return2Action	matched

Fig. 5 shows the implementation in ATL of the transformation rule *functionDef2MethodUnit*. It is implemented as a *matched rule*. The source pattern (*from*) defines a variable called *funcdef* representing the *FunctionDefinition* class (line 3). Otherwise, the target pattern (*to*) creates a variable called *meth* representing the *MethodUnit* class (line 5). In the target pattern the attributes *name, export and codeElement* are mapped from the attributes of the *FunctionDefinition* class (lines 6-11). To map the parameters from the attribute *formalParameters* it is required to create a *Signature* element created with the called rule *CreateSignature* (line 9). The execution of this matched rule has allowed us to represent correctly in the KDM model all the function definitions in PHP required to implement the widget listing online users of our case study.

```
1  rule funtionDef2MethodUnit {
2      from
3          funcdef: astm!FunctionDefinition
4      to
5          meth: kdm!MethodUnit (
6              name <- funcdef.declOrDefn.identifierName.getName.debug('path'),
7              export <- funcdef.declOrDefn.accessKind.getExportKind,
8              codeElement<-funcdef.declOrDefn.body
9              codeElement <- thisModule.createSignature
10             (funcdef.declOrDefn.identifierName.getName,
11                 funcdef.declOrDefn.formalParameters),
12         )
```

Fig. 5. FunctionDef2ActionElement ATL rule.

4. Related Works

In this section, we present some of the existing ADM-based approaches found in the literature and compare them with our ADM-based migration method [21] (see Table 4). Due to space limitations only we present the most representative ones.

Van Hoorn et al presents in [9] DynaMod, a method which addresses the model-driven modernization of software systems. It considers static and dynamic analysis for extracting models conform to KDM from Java code. Sadovykh et al presents in [19] a method to extract an UML (Unified Modeling Language) [2] models containing the most persistent part of a system from C++ code. Perez-Castillo et al presents in [16] a reengineering process called PRECISO to recover and implement Web Services in automatic manner from relational databases. They extract a model conform to a SQL-92 metamodel from SQL-92 code. Bruneliere et al presents in [3] MoDisco an extensible approach for model-driven reverse engineering which allows extracting platform-specific models from Java, XML and JSP code. They define three metamodels one for each type of code. Reus et al presents in [17] a reverse engineering process for recovering UML models from PL/SQL code. Vasilecas et al presents in [23] a process which derives business rules from a legacy system. They extract ASTM models from Visual Basic code.

The second column in Table 4 refers to the source code from which the ADM-based approach extracts the models. We can find approaches extracting from Java, C++, SQL-92, XML, JSP and VisualBasic, but none of them addresses the extraction of models from PHP code. Thus, our ADM-based migration method would be the unique existing approach implementing text-to-model (T2M) transformations to extract a code from PHP code.

Table 4. Comparison ADM-based approaches.

Approach	Source code	Metamodel	Context	Toolkit
Van Hoorn et al (DynaMod)	Java	KDM	Generic context	Yes
Sadovykh et al	C++	UML	Generic context	Yes
Perez-Castillo et al (Preciso)	SQL-92	SQL-92	Data base / Web services	Yes
Bruneliere et al (Modisco)	Java, XML, JSP	Java, XML, JSP	Generic context	Yes
Reus et al	PL/SQL	UML	Data base	Yes
Vasilecas et al	Visual Basic	ASTM, KDM, SBVR	Generic context	Yes
Our ADM-based migration method	PHP	ASTM_PHP, KDM	CMS-based Web applications	Yes

The third column refers to the metamodels to which the models conform to. Two of them (Perez-Castillo et al and Bruneliere et al) bet for defining their own metamodels. The former propose a SQL-92 metamodel and the latter three metamodels the Java, XML and JSP. On the other hand, two approaches use the UML metamodel. Finally, two of the approaches (Van Hoorn et and Vasilecas et al) use the ADM standard metamodels. The former defines models conforming to KDM and Vasilecas et al defines models according to ASTM and KDM. For our ADM-based migration method, we bet for the use of both metamodels.

The fourth column specifies the context of the approach. If the approach is framed in a generic reengineering context we have categorized it as a generic context. Thereby, three of the approaches have been categorized as such. On the other hand, two of them are focused on data base context. It is worth noting that none of them is focused on the context of CMS-based Web applications apart from our ADM-based migration method.

Finally, the fifth column denotes if the approach is supported by a toolkit. As we can see, all of the approaches found are supported by a toolkit to a greater or lesser extent.

5. Conclusions and Future Works

In the last years, organizations have experienced the necessity of using powerful management platforms to maintain their large-scale Web applications and manage all their content in a robust and reliable manner. One of the most adopted solutions has been to base Web applications on Content Management Systems. These CMS-based Web Applications provide organizations with many advantages.

The evolving objectives of organizations may cause them to experience the necessity of migrating their CMS-based Web applications to other CMS platforms meeting better their needs. This migration process entails a complex, time-consuming and error-prone reengineering process. We propose an ADM-based migration method to automate this process and to mitigate these drawbacks. Up to now, this method is focused on open-source CMS platforms which most of them are implemented in PHP. Furthermore, this method is supported by an ADM-based toolkit. In this paper we present the implementation of two of its modules: i) the ASTM_PHP DSL and ii) M2M transformations rules which allow to generate KDM models from ASTM_PHP models.

To define the ASTM_PHP DSL, we have defined an ASTM_PHP metamodel (abstract syntax). This metamodel extends the standard metamodel ASTM (concretely its SASTM part) with specific PHP elements. Moreover, a tree-like editor (concrete syntax) which allows defining graphically models conforming to the ASTM_PHP metamodel has been implemented using EMF. On the other hand, the M2M transformation rules have been defined firstly in natural language and then implemented in ATL.

To the best of our knowledge, the migration process of CMS-based Web applications is not supported by any model-driven reengineering method. Thus, we propose an ADM-based migration method supported by a toolkit to support this process. As for the application of

ADM, we can state that it is pretty applied by the reverse engineering approaches as well as its standard metamodels, mainly ASTM and KDM. Concretely, the use of KDM is more widespread than ASTM. Finally, we think that in the market there is a lack of comprehensive toolkits allowing the extraction of models from PHP code. So far, our migration method is focused on CMS-based Web applications implemented in PHP, so that as a future work, we plan to address the migration of these Web applications implemented in any code.

References

1. Boiko, B.: Understanding Content Management. Bull. Am. Soc. Inf. Sci. Technol. 28, 1, 8–13 (2001).
2. Booch, G. et al.: The Unified Modeling Language User Guide. Longman (1999).
3. Bruneliere, H. et al.: MoDisco : A Generic And Extensible Framework For Model Driven Reverse Engineering. ASE ’10. pp. 173–174 (2010).
4. Budinsky, F.: Eclipse Modeling Framework, (2008).
5. Chikofsky, E.J., Cross, J.H.: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Softw. 7, 1, 13–17 (1990).
6. Drupal: Drupal CMS, http://drupal.org/.
7. Eclipse Foundation: Atlas Transformation Language, http://www.eclipse.org/atl/.
8. Fowler, M.: Domain Specific Languages. Addison Wesley Professional (2010).
9. Hoorn van, A. et al.: DynaMod Project : Dynamic Analysis for Model-Driven Software Modernization. 1st International Workshop on Model-Driven Software Migration. pp. 12–13 , Oldenburg, Germany (2011).
10. ISO/IEC: Information technology - Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM). (2012).
11. Joomla!: Joomla! CMS, http://www.joomla.org/.
12. McKeever, S.: Understanding Web content management systems: evolution, lifecycle and market. Ind. Manag. Data Syst. 103, 9, 686–692 (2003).
13. Mellor, S.J. et al.: Model-Driven Architecture. In: (Eds.), J.-M.B. and Z.B. (ed.) Advances in Object-Oriented Information Systems. pp. 233–239 Springer-Verlag (2002).
14. OMG: Abstract Syntax Tree Metamodel specification of the OMG, http://www.omg.org/spec/ASTM/1.0.
15. OMG: ADM task force.
16. Perez-Castillo, R. et al.: PRECISO: A Reverse Engineering Tool to Discover Web Services from Relational Databases. 16th Working Conference on Reverse Engineering. pp. 309–310 IEEE (2009).
17. Reus, T. et al.: Harvesting Software Systems for MDA-Based Reengineering. Second European Conference, ECMDA-FA. pp. 213–225 (2006).
18. Ric Shreves: Open Source CMS Market Share. (2008).
19. Sadovykh, A. et al.: Architecture Driven Modernization in Practice: Study Results. 2009 14th IEEE International Conference on Engineering of Complex Computer Systems. pp. 50–57 IEEE (2009).
20. Trias, F. et al.: A Systematic Literature Review on CMS-based Web Applications. ICSOFT. (2013).
21. Trias, F. et al.: An ADM-based Method for migrating CMS-based Web applications. 25th SEKE. , Boston, EUA (2013).
22. Trias, F.: Building CMS-based Web Applications Using a Model-driven Approach. Sixth RCIS. pp. 1 – 6 (2012).
23. Vasilecas, O., Normantas, K.: Deriving Business Rules from the Models of Existing Information Systems. 95–100 (2011).
24. Vidgen, R. et al.: Web Content Management. 14th Bled Electronic Commerce Conference. , Slovenia (2001).
25. Wordpress: Wordpress CMS, http://wordpress.org/.