Model2Roo: A Model Driven Approach for Web Application Development based on the Eclipse Modeling Framework and Spring Roo

Juan-Carlos Castrejon-Castillo, Rosa López-Landa, Rafael Lozano

▶ **To cite this version:**

Juan-Carlos Castrejon-Castillo, Rosa López-Landa, Rafael Lozano. Model2Roo: A Model Driven Approach for Web Application Development based on the Eclipse Modeling Framework and Spring Roo. CONIELECOMP 2011 - International Conference on Electrical Communications and Computers, Feb 2011, Cholula, Puebla, Mexico. pp.82-87, 10.1109/CONIELECOMP.2011.5749344 . hal-00921673

HAL Id: hal-00921673
https://hal.science/hal-00921673

Submitted on 2 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Model2Roo: A Model Driven Approach for Web Application Development based on the Eclipse Modeling Framework and Spring Roo

Juan Carlos Castrejón, Rosa López-Landa, Rafael Lozano
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México
{A00970883, atzimba.lopez, ralozano}@itesm.mx

Abstract

Inherent complexity in web application development is continually increasing due to changes in both functional and non-functional requirements, as well as to technological changes like new programming languages, tools, frameworks and development processes. An adequate management of this complexity is required in order to generate high quality software systems. In this paper, an approach based on model-driven techniques is proposed to guide the development of web applications, by focusing on model abstractions rather than implementation details. In order to do so, we propose a set of model extensions, such as profiles and annotations, to describe both the static structure and a subset of functional and non-functional requirements usually associated to web applications.

1. Introduction

Web application development has been one of the most evolving industries in recent software engineering [1]. This evolution has also been associated with an increased complexity in the set of both functional and non-functional requirements that these systems are expected to fulfill [1]. In addition to this added complexity, the emergence of new programming languages, tools, frameworks and methodologies for web application development represent additional concerns that software developers must address. The administration of software complexity and the adequate selection of development tools, becomes vital for the change management process associated to web software development. This is because a defective process can lead to productivity and maintainability problems [2].

These kind of problems are very common not only in web application development, but in software engineering in general. As stated by Kleppe et al [3], one of the reasons for the high recurrence of these issues lies in the way the traditional software development process is usually conducted. As shown in Figure 1, this process is composed by six general stages, that range from the requirements of the system to the deployment of an application. We can appreciate that early stages deal primarily with design documents, while later ones deal with code artifacts. In theory, when a change has to be performed to the system, all of the corresponding documents and artifacts have to be updated accordingly to reflect this change. However, software developers tend to consider programming stages more important than their design counterparts, and, as a result, changes are usually applied only to code artifacts. This generates a lack of compliance between design documents and implementation artifacts, which can cause a detriment in the overall quality of the software system [2].

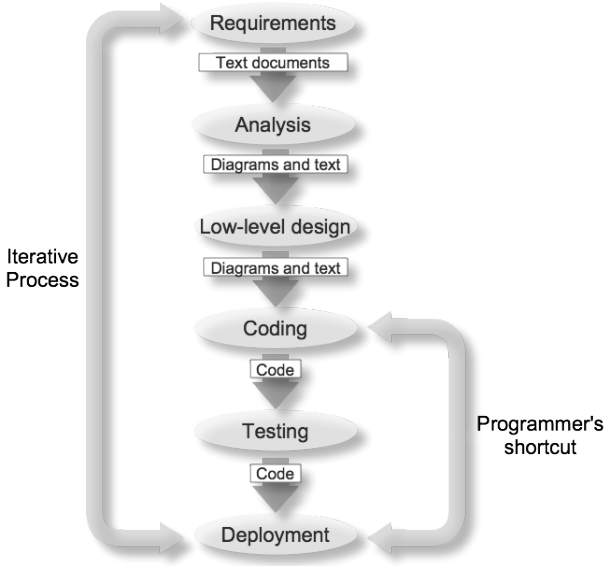

Figure 1. Traditional software development process [3]

A relatively recent approach to cope with the aforementioned problems is represented by the Model-Driven Software Development (MDSD) discipline [5]. MDSD heavily relies on model abstractions rather than implementation details, with the intent of avoiding a lack of compliance between design documents and implementation artifacts. This effectively enhances the overall maintainability of software systems and promotes a better manageability of the technological changes they may go through [5]. Taking this into account, we propose a MDSD approach to model the domain knowledge of web applications in platform independent models.

The remaining of this paper is organized as follows. Section 2 introduces the Model Driven Architecture (MDA) initiative, along with the Eclipse Modeling Framework (EMF), the ATL Transformation Language (ATL) and the Spring Roo project. In section 3 we describe related work. Section 4 contains the definition of the Model2Roo approach. Finally, in section 5 we describe an example application of our approach, and present our conclusions and future work in section 6.

2. Model-driven technologies

2.1 MDA

MDA [3] is a MDSD initiative proposed by the Object Management Group [4]. It relies on a layered architecture of models that specify system functionality, independently of its implementation in any given platform. In order to achieve this separation, the use of models that contain either platform dependent (PSM) or independent information (PIM), as well as transformation procedures between them, is proposed.

The base of the MDA initiative is the MetaObject Facility (MOF), an specification provided by the OMG to describe meta-meta models. By using an implementation of MOF, such as the Unified Modeling Language (UML), we can develop a PIM meta-model that may later be transformed into a particular PSM model. Finally, instances of these platform dependent models can be generated. In order to do so, the definition of model transformation procedures between models of different layers is required. Table 1 depicts the four layered architecture of the MDA initiative.

Table 1. MDA layered architecture

Name	Layer	Example
M3	Meta-metamodel	MOF
M2	Metamodel	UML
M1	Model	UML model
M0	Instance data	Instance data

2.2 EMF

EMF is a modeling framework built on top of the Eclipse platform [6]. Its core component is the Ecore meta model. Ecore is based on Essential MOF (EMOF), which is a subset of MOF that is used for the definition of simple meta models. By using Ecore we can in turn define additional models or meta models, using a set of integrated Eclipse editors [6]. Another key MOF based meta model, within the EMF framework, is the UML2 project [6]. This metamodel is an implementation of the UML 2.x specification.

Finally, EMF is considered by many researches as one of the main environments for model-driven development, specially considering the size of its community and the number of experimental tools developed around it [5].

2.3 ATL

ATL is a model transformation engine integrated as part of the Eclipse platform that is aimed to produce target models from a set of source models [7]. It provides two main units of operation: ATL modules and ATL queries. The former specify a set of operations required to transform from one model to another, while the latter compute a single primitive value, such as a number or a String, from a set of source models. These transformation procedures are aligned with MDA definitions regarding model transformations.

2.4 Spring Roo

The Spring Roo project [8] provides a Java productivity tool for building enterprise applications. It does so by providing a command-line shell where special commands can be issued to create high quality web applications. The importance of this project lies in its ability to specify not only the static structure of an application, but also comprehensive support for the functionality provided by the Spring framework [8], as well as integration to other relevant technologies. One of its key advantages is the ability to generate code and test artifacts from the same domain model [8].

It should also be noted that web applications generated by Spring Roo are automatically built with a set of architecture patterns and best practices, in order to favor the maintainability of the system [8]. It is particularly helpful for this study that the Spring Roo commands can be saved into a script file, and then be executed at any given time.

3. Related work

In recent years there has been a lot of supporting work for models to source code transformations [9-12]. Regarding web application development, we can mention the work of Hou et al [13], where a modeling approach for web applications PIMs, based on UML extensions, is proposed. These PIMs are later transformed into PSMs using an abstract algebra method. Zhuang et al [14] describe a method for the development of e-commerce web applications, based on the WebML language and the transformation facilities provided by the WebRatio environment [14].

We can also highlight the work described by Abdella [15], where an approach based on the extension of UML class diagrams, and the use of the EMF framework, is proposed to build Java web applications. Closer to our approach is the work of Jeanneteau [16], an extension to an UML case tool that allows the generation of Spring Roo scripts that represent the static structure and the persistence layer of a software system. However, the generated scripts only contain the most basic Roo commands, without taking full advantage of the potential of the Spring Roo project.

4. Model2Roo

As stated in the previous section, the focus on software development in a MDA environment should be on models and model transformations, rather than specific programs or programming languages details. Taking this into consideration, we propose Model2Roo, an approach for web application development that is integrated within the EMF framework and that relies on MOF based meta models. Our intention is to develop web applications by transforming models built using these meta models into Spring Roo scripts, using tools associated to the EMF framework. By doing so, software developers will not need to worry about implementation details, but only on modeling the domain characteristics of the required application. A set of extensions to the source meta models is also proposed in order to take advantage of the full potential of the Spring Roo project.

The two main EMF meta models, Ecore and UML2, will be used as the meta models of the Model2Roo approach. Nonetheless, considering that Ecore is the main building block of the EMF framework, it will also be the key component of our approach. This is, we propose Ecore to be the source meta model for the transformation procedures to Spring Roo scripts. Figure 2 depicts the Model2Roo approach, along with its associated models and model transformations.

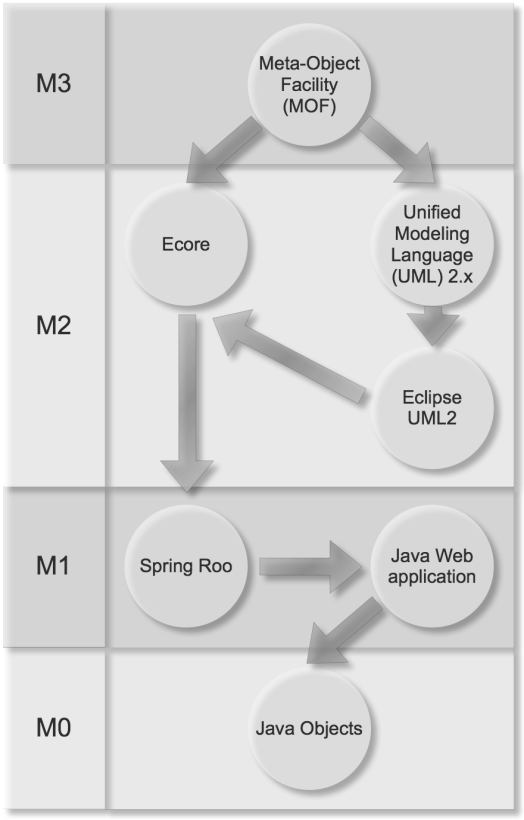

Figure 2. Model2Roo approach

In our approach, the model transformation from UML2 to Ecore is done through the UML importer application [6], built-in the UML2 project. The transformation from Ecore models into Spring Roo scripts is achieved by means of an ATL query, described in section 4.2. Finally, the transformation from Spring Roo scripts into Java Web applications is conducted from within the Spring Roo command-line shell [8], using its integrated scripting facilities.

By restricting the transformation to Spring Roo scripts only from Ecore models, we can centralize the associated transformation logic and details. Also, since Ecore is the meta model of the EMF framework, any EMF based meta model may define a transformation to Ecore, and thus benefit from the transformation to Spring Roo scripts. For instance, this is the case of the UML2 meta model.

4.1 Model extensions

For each of the transformation processes defined in the previous section there is risk of loss of information from the source to the target model. For instance, during the model transformation from UML2 to Ecore, all of the UML profile details are lost. We also need to consider that the commands associated to the Spring Roo project provide a set of options that cannot be specified in the base constructs of the Model2Roo meta models. For example, we could require the generation of integrated tests for all of the entities defined in a particular model. However, since neither Ecore nor UML2 base constructs provide a property to describe such requirement, we have no standard way to specify this option.

In Table 2 we describe the extensions that we propose to implement in the Model2Roo source metamodels, in order to overcome the aforementioned limitations:

Table 2. Meta models extensions

Limitation	Extension
Loss of UML profiles data during UML to Ecore transformation	Extension to the UML Importer application
Inability to specify full set of Roo command options	Development of Ecore Annotations and UML Profiles

The main building block for these meta models extensions is a set of Ecore annotations [6]. These annotations will describe details that allow the exploitation of the full potential of Spring Roo, while also taking advantage of modeling in MOF meta models. Taking into consideration that one of the main extension mechanisms in UML based models is the use of UML profiles [5], we propose to align the set of Ecore Annotations to the corresponding set of UML profiles. In doing so, we will be providing means to specify Spring Roo details both in Ecore and UML2 models, the two source meta models of our approach.

The set of Model2Roo annotations are first modeled as UML profiles using facilities of the UML2 project,. These profiles can then be used when the starting metamodel is UML, by using stereotypes applications. However, they cannot be used directly when the source metamodel is Ecore, since it lacks the concept of profile constructs. This is why we propose a set of aligned Ecore annotations, generated from the UML2 profiles, that can be used to extend Ecore base constructs.

Considering that the transformation process to Spring Roo relies on Ecore, we also define a transformation process from UML2 to Ecore models. This is conducted by an extension of the UML2 importer application. Our extension analyzes the UML stereotypes applied to a model, and associates corresponding Ecore annotations to the resulting Ecore model. Figure 3 depicts the process for the generation of the Model2Roo annotations, as well the association between UML and Ecore models.

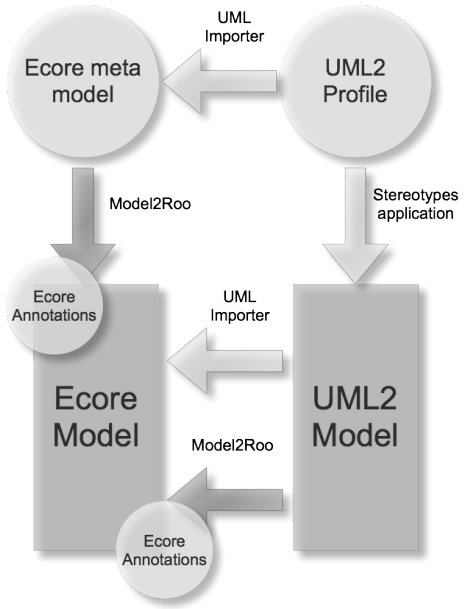

Figure 3. Spring Roo details as Ecore Annotations

Having explained the definition process of the Model2Roo annotations, we will now discuss their details. We have decided to make a distinction between Spring Roo commands that affect the static structure of web applications, from those commands that allow integration with other technologies, or that provide additional functionality to the base structure of the system. When either the Ecore annotations or the UML profiles are applied, this first group of commands is identified with the prefix *rooStructure*, while the second group uses the *rooCommand* prefix.

As stated before, the *rooStructure* annotations are used to create the static structure of the system. This includes, the system definition, along with the model entities, enumerations, properties, and references between them. Nonetheless, in order to generate a more comprehensive web application we need more than the static structure of the system, which is why the *rooCommand* annotations include support for the definition of system logging configuration, web MVC controllers, along with their associated web and integration tests, data on demand and finder methods for the model entities. Tables 3 and 4 depict the details of the *rooStructure* and *rooCommand* annotations, along with their associated Ecore and UML elements. Also shown are the Spring Roo commands that will be issued during the transformation from Ecore models to Spring Roo scripts.

Table 3. rooStructure annotations

Ecore element	Uml element	Annotations	Roo commands
EPackage	Model	RooModel	project
EClass	Class	RooEntity	entity
EAttribute, EReference	Property	RooField {Boolean, Date, Enum, Number, Reference, Set, String}	field, set, reference
EEnum	Enumeration	RooEnumType	enum
eLiterals	Enumeration Literal	RooEnumConstant	enum

Table 4. rooCoomand annotations

Ecore element	Uml element	Annotations	Roo commands
EPackage	Model	RooModelCommand	web, controller, persistence
EClass	Class	RooEntityCommand	dod, test, finders, logging, selenium, controller

4.2 Ecore2Roo ATL query

In order to transform from Ecore models to Spring Roo scripts we propose the use of an ATL query. This query traverses the elements of Ecore source models, analyzes them and generates the corresponding Spring Roo commands. This set of commands can then be saved into a Spring Roo script file, that can in turn be used to generate the target web application. The Ecore2Roo ATL query has been designed to generate Spring Roo commands both from Ecore models annotated with the set of Model2Ecore annotations, as well as with unannotated models. The fact that the Ecore2Roo query does not depend on Spring Roo details allows for a seamless integration of the Model2Roo approach with existing Ecore models. In this regard, our approach can be seen as an alternative to the current code generation facilities provided by the EMF framework [6]. Finally, it should also be noted that the only requirements to execute the Ecore2Roo query are an ATL environment, the Ecore meta model definition, and an Ecore source model.

4.3 Model2Roo eclipse plugin

We have developed an Eclipse plugin that allows the integration of the Model2Roo operations as part of the EMF framework. It provides all of the functionality described in the previous sections, for Ecore and UML2 models, easily accessible from within the Eclipse environment. This is, by selecting an ECore EPackage element within an Eclipse editor, we may associate the set of Model2Roo annotations. We can also generate Spring Roo scripts by selecting either ECore EPackage or UML Package elements.

Figure 5 depicts the menu of the Model2Roo plugin that is added to Eclipse's menu bar. As we can appreciate, it provides three menu items that allow the execution of the operations proposed by our approach.

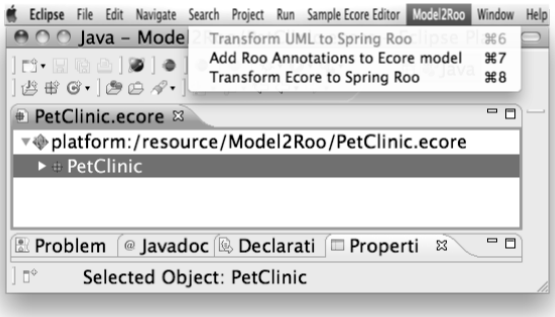

Figure 4. Model2Roo Eclipse plugin

5. Model2Roo example: PetClinic

In order to demonstrate the use of the Model2Roo approach we will develop a Pet Clinic application, a classic example designed to show the strengths of Spring application frameworks [17]. Considering that a sample script of this application is also provided along with the Spring Roo distribution, our intention is to compare the script file generated with the Model2Roo approach, against the script provided by the Spring Roo distribution, so as to verify if the corresponding web applications can be considered equivalent or not.

The Pet Clinic application is an information system that requires web browser access. It is intended to be used by employees of the clinic in order to manage information regarding veterinarians, clients and their pets. The use cases for this system include managing veterinarians and their specialities, managing clinic clients, their pets and their visits to the clinic [17]. As explained in the previous section, we can decide to model this application either with Ecore or with UML2. In this case, we have decided to directly use the Ecore support. It should be noticed that at this stage no special Model2Roo annotation is required to model the application. Figure 5 depicts the Pet Clinic Ecore model, developed within the Ecore Diagram Editor [6], an application that is part of the EMF framework.

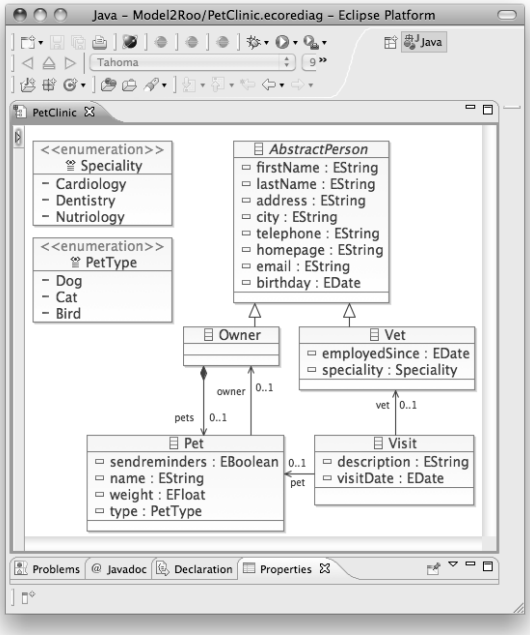

Figure 5. Unannotated Pet Clinic application

By executing the *Transform Ecore to Spring Roo* operation, from the Model2Roo plugin, we could generate a Spring Roo script from this model. However, it would only reflect the static structure of the system. In order to generate a more comprehensive application, we can apply the annotations provided by the Model2Roo approach. To do this, we execute the *Add Roo Annotations to Ecore Model* operation, from the Model2Roo plugin, and then specify values for the relevant annotations. Figure 6 depicts this annotated Pet Clinic Ecore model.

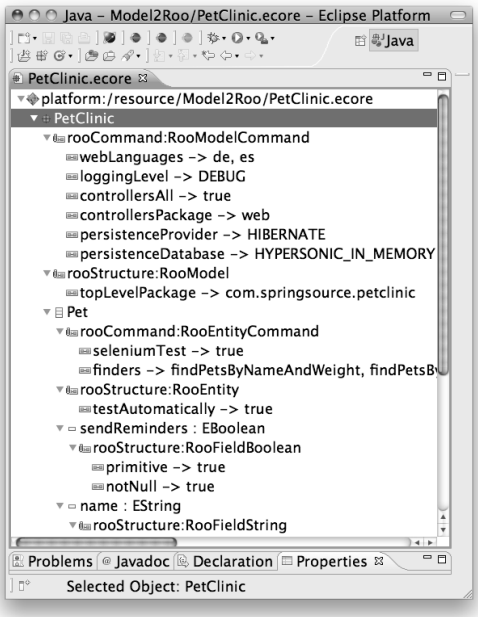

Figure 6. Pet Clinic with Model2Roo annotations

The Spring Roo script generated from this model will describe the web application with a greater level of detail, taking full advantage of the Spring Roo constructs. Figure 7 depicts representative fragments of this generated script.

```
project
--topLevelPackage com.springsource.petclinic

persistence setup --provider HIBERNATE
--database HYPERSONIC_IN_MEMORY

entity --class ~.domain.Pet
--testAutomatically
field string --fieldName name --notNull
--sizeMin 1
field number --fieldName weight
--type java.lang.Float --notNull --min 0
field reference --fieldName owner
--type ~.domain.Owner
field enum --fieldName type --notNull
--type ~.domain.reference.PetType

enum type --class ~.reference.PetType
enum constant --name Dog
enum constant --name Cat
enum constant --name Bird
controller all --package ~.web
```

Figure 7. Fragments of the Spring Roo script

Through the application of our approach we have successfully generated a Spring Roo script that is functionally equivalent to the one provided by the Spring Roo distribution. It should be noted that the associated web applications will provide the same set of functional and non-functional requirements.

6. Conclusions and Future Work

In this paper, we have proposed Model2Roo, an MDA approach for the development of high quality web applications. This approach is integrated within the EMF framework and relies on extended MOF meta models for the generation of Spring Roo scripts, that describe both the static structure of the system, as well as associated functional and non-functional requirements, such as the generation of web controllers, integration tests, and the definition of a persistence layer.

The meta model extension mechanism is based on Ecore annotations and UML profiles. However, its use is not mandatory and our approach can be applied both to annotated and unannotated models, though in order to take full advantage of the Spring Roo project, the use of the Model2Roo annotations is encouraged.

The main contribution of our approach lies in the change management process associated to web applications. This is, changes need not be applied directly over the implementation artifacts of the system. Instead, the system model is updated using the EMF framework and the generated Spring Roo script will in turn contain the appropriate implementation changes. In this way, we avoid the lack of compliance between design documents and implementation artifacts.

For our future work, we would like to develop a Spring Roo add-on, in order to be able to provide the Model2Roo functionality outside the Eclipse platform. Support for additional MOF based meta models and their associated tools is also intended.

7. References

[1] M. Jazayeri, “Some Trends in Web Application Development”, *Future of Software Engineering FOSE '07*, IEEE Computer Society, Minneapolis, MN, 2007, pp. 199-213

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, Second Edition*, Addison-Wesley Professional, 2003

[3] A. Kleppe, J. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise: Addison-Wesley Professional, 2003.

[4] OMG. (2010, October). Unified Modeling Language. Available: http://www.uml.org/

[5] T. Stahl, M. Völter, Model-Driven Software Development, Wiley, 2006

[6] D. Steinberg, F. Budinsky, and M. Paternostro, EMF: Eclipse Modeling Framework, Second ed.: Addison-Wesley Professional, 2008.

[7] F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev, “ATL: A model transformation tool”, *Science of Computer Programming*, Volume 72, Issues 1-2, Special Issue on Second issue of experimental software and toolkits (EST), 2008, pp. 31-39

[8] SpringSource. (2010, October). SpringRoo. Available: http://www.springsource.org/roo

[9] Y. Liu and Y. Ma, “An Approach for MDA Model Transformation Based on JEE Platform,” *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, 2008, pp. 1-4.

[10] K. Ma and B. Yang, "A Hybrid Model Transformation Approach Based on J2EE Platform," *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, 2010, pp. 161- 164.

[11] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli, "Automatic Performance Model Transformation from UML to C++" *Parallel Processing - Workshops, 2008. ICPP-W '08. International Conference on*, 2008, pp. 228-235.

[12] A. M. Reina-Quintero, J. Torres-Valderrama, and M. Toro-Bonilla, "Improving the adaptation of web applications to different versions of software with MDA", *7th International Conference on Web Engineering. Workshop Proceedings. Workshop on Adaptation and Evolution in Web Systems Engineering (Aewse'07) (2)*, 2007, pp. 101-107

[13] J. Hou, J. Wan, X. Yang, "MDA-based Modeling and Transformation Approach for WEB Applications”, *Sixth International Conference on Intelligent Systems Design and Applications ISDA'06*, 2006, pp. 867-874

[14] G. Zhuang, and J. Du, “MDA-Based Modeling and Implementation of E-Commerce Web Applications in WebML”, *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, *vol.2*, 2009, pp. 507-510

[15] D. Abdellah, “Applying the MDA approach for the automatic generation of an MVC2 web application”, *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*, 2010, pp. 681-688

[16] D. Jeanneteau. (2010, October). RooPlugout. Available: http://gna.org/projects/roo-plugout/

[17] SpringSource. (2010, October). The Spring PetClinic Application. Available: http://static.springsource.org/docs/petclinic.html