EMF on Rails

Rosa López-Landa[1], Julieta Noguez[1], Esther Guerra[2] and Juan de Lara[2]
[1]*Computer Department, Tecnológico de Monterrey, Mexico City Campus, Mexico*
[2]*Computer Science Department, Universidad Autónoma de Madrid, Spain*
{*atzimba.lopez, jnoguez*}*@itesm.mx,* {*Esther.Guerra, Juan.deLara*}*@uam.es*

Keywords: Model-Driven Engineering, Spring Roo, Code Generation

Abstract: In this paper we propose leveraging existing frameworks for automated web application development, in the style of Ruby on Rails, Grails and Spring Roo, for their use within a Model-Driven Engineering process. Our approach automates the construction of domain-specific generators for web applications in particular domains. These generators are able to synthesize web applications using Spring Roo, starting from annotated models. In this way, designers of web applications do not need to be proficient in web automation frameworks, but they can benefit from the use of domain-specific, intuitive models. We illustrate our approach by generating an application to edit Eclipse Modelling Framework (EMF) models through the web.

1. INTRODUCTION

Web applications have become popular recently because, unlike standalone systems, they are distributed and accessible to a large number of users. Structurally, web applications are based on a client-server architecture where clients make requests to one or more service and resource providers. Rapid Application Development (RAD) is a methodology that improves the productivity of software development by allowing the construction of new systems from existing components (Poulin, 1993). Reusing software components eliminates the need of investing time and effort in developing components that can be generated automatically. This reduces development costs.

The rapid changes and the obvious results of the globalization have forced web applications to adopt new technologies, methodologies and approaches. In this context, Model-Driven Engineering (MDE) (Völter and Stahl, 2006) effectively helps in improving the overall maintenance of web applications and promotes a better manageability of the technological changes they may go through. MDE relies on model abstractions of the applications rather than on implementation details. Developers are provided with customized domain-specific languages, tailored to a specific concern, which promotes higher levels of productivity. From these domain-specific models, the final application is generated, so that the developers do not need to deal with low-level, accidental details of the systems.

Recently, several frameworks for the automated development of web applications have been proposed, built on top of general-purpose languages like Java (Spring Roo[1]), Ruby (Ruby on Rails[2]), Python (Django[3]) and Groovy (Grails[4]). They accelerate web development by using convention over configuration (i.e., using sensible defaults for every aspect of the application), freeing the developer from repetitive tasks, and by generating skeleton code from higher level commands. However, they still require the developer to be proficient in the base language they are built upon, and the different technologies used, in order to complete the generated skeleton code.

To alleviate this situation, we propose leveraging those powerful, but still low-level frameworks for its use in combination with MDE. Our approach is to use these frameworks (in particular Spring Roo) as back-end for web application development. For this purpose, we propose an MDE architecture that automates: (1) the model-based construction of code generators for families of web applications, and (2) the construction of web applications using the previous generators, starting from domain-specific models. The advantage of our approach is that it helps in the construction of domain-specific code generators, from which different web applications of a given domain can be easily generated from models. Instead,

[1]http://www.springsource.org/spring-roo
[2]http://rubyonrails.org
[3]https://www.djangoproject.com
[4]http://grails.org

using web application frameworks like Spring Roo eases the construction of one particular application, but not families of applications for the same domain.

The rest of the paper is organized as follows. Section 2 introduces MDE and Spring Roo. Next, section 3 describes *EMF on Rails*, our MDE approach to develop web applications. Section 4 presents a usage example of this tool. In section 5, we discuss some related work. Finally, section 6 concludes.

2. BACKGROUND

2.1. Model-Driven Engineering

Model-Driven Engineering (MDE) is a software development paradigm that promotes an active use of models to conduct the different phases of software projects. Hence, models are used to specify, analyse, verify, test and generate code for the final applications. In MDE, models are frequently built using Domain-Specific Languages (DSLs) that gather abstractions and primitives of the domain. These domain-specific models describe the systems from the *problem domain* perspective, instead of focussing on the technological solution space. The syntax of DSLs is normally defined through a model, called metamodel, which describes the primitives, abstractions and relations of the domain. The manipulation of models, e.g., for simulation or optimization purposes, is defined using model transformation programming languages. Moreover, DSLs are complemented with code generators to produce most or all the code for the final application.

Similar to annotations for Java, models can be annotated with elements providing extra information, conveying special semantics to particular elements. Annotations are frequently used as a previous step to code generation. They are defined in so-called profiles, which can be seen as an extension mechanism for models. Profiles have been widely used to annotate the UML language, and have been recently proposed as an extension mechanism for DSLs as well (Langer et al., 2011).

The Object Management Group (OMG) supports a flavour of MDE, called Model-Driven Architecture (MDA) (Kleppe et al., 2003). MDA uses standards like the MetaObject Facility (MOF) for metamodelling, and the QVT (Query/View/Transformations) language (OMG, 2005) for model transformations. Widely used (partial) implementations of the MOF exist, most notably the Eclipse Modeling Framework (EMF) (Steinberg et al., 2008), which indeed is the *de-facto* standard meta-modelling framework for MDE.

While EMF is very successful, there are currently no widely adopted QVT implementations. Thus, in this work, we use the Atlas Transformation Language (ATL) (Jouault et al., 2008) as it is one of the most popular transformation languages nowadays. ATL is a rule-based declarative model transformation language, where transformations are specified by mapping object patterns from the source model into patterns of the target model.

2.2. Spring Roo

Spring Roo (Long, 2011; Rimple and Penchikala, 2012) is a Java productivity tool for building enterprise applications. This project provides a command-line shell where special commands can be issued to automatically create high quality web applications. The importance of this project lies in its ability to facilitate the development of high quality web applications, based on a set of architecture patterns and best practices, just in minutes.

The aim of Spring Roo is to improve Java developer productivity without compromising the integrity or flexibility of the solution (Spring Source Community, 2012). Figure 1 depicts the Spring Roo approach, where developers type commands that shape the web application (a). These commands are interpreted by the command-line shell (b) to produce a Java web application (c) with a database (d). If there are special requirements, such as securing the access to the web application, developers need to use *add-ons*. Spring Roo is based on a modular architecture that supports the installation of third-party *add-ons*. In this way, users can extend the capabilities of the project (Spring Source Community, 2012). Each *add-on* incorporates a set of commands and functions to the shell of Spring Roo that allow building different types of applications depending on the used modules. If an application has special requirements for which there is no *add-on* available, then developers must create their own *add-ons* (e) and add their own commands to the

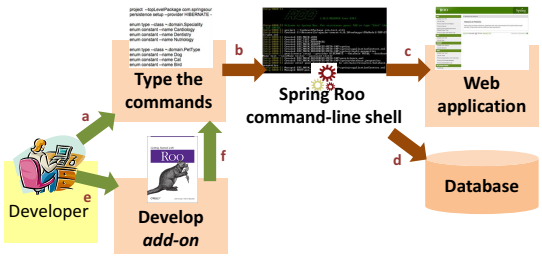

Figure 1: Web application development with Spring Roo.

script of Spring Roo (f). Then, these commands can be used to produce a web application that takes into account the special requirements.

3. PROPOSED ARCHITECTURE

The aim of MDE is to develop software from models and model transformations, rather than by handcrafted code. Taking this into consideration, we propose *EMF on Rails*, an approach for web application development that is integrated within the Spring Roo project and the Eclipse Modeling Framework. In this way, we consider the automated development of customized environments and generators for families of web applications (see Figure 2). Such generators are then used to automatically generate web applications for specific domains, as can be seen in Figure 4.

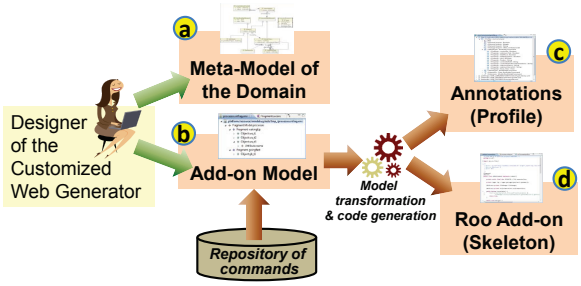

Figure 2: *Step 1*: Building a customized web generator.

Figure 2 shows the envisioned architecture of our proposal. In particular, the design of a customized web generator comprises building a meta-model or DSL of the domain (e.g., of an e-learning system), as well as an *add-on* model with specific commands that the code generation using Spring Roo will need. The DSL (marked as (a) in the figure) will be used later to build models from which the final web application will be generated. The *add-on* model (marked as (b) in the figure) includes a description of the domain-specific functionality (*commands*) used by the web applications to be generated.

Figure 3 depicts the meta-model that permits building *add-on* models. This is made of:

Addon. This class contains the metadata of the Spring Roo *add-on*. It has a name which corresponds to the *add-on* identifier, the URI of the Ecore meta-model of the domain that is being used, and a set of commands.

Command. This class represents a command of the Spring Roo shell. Each command has a name, a category, a help message, an eObjectType and a set of parameters. The name and the category determine how the command will be typed on the shell. The eObjectType stores the relationship between the command and an object from the meta-model.

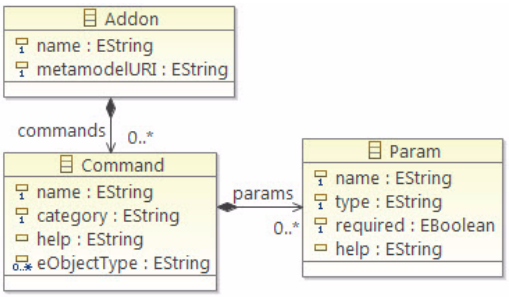

Figure 3: Meta-model for Spring Roo *add-ons*.

Param. This class represents a parameter of a command. Each parameter has a name, a type, a help message and a boolean flag to indicate if the parameter is required or not.

From the *add-on* model, we use an ATL transformation and a code generator to produce: a profile that will be used to automatically annotate the models defined by the user of the generator framework (see (c) in Figure 2), and a set of Java classes with a skeleton of the Roo *add-on* that will be used for code generation (see (d) in Figure 2). The generated Java classes are skeletons that the designer needs to complete by hand with the specific functionality. We foresee having a repository of typical Roo commands that can be reused when designing a new generative framework. Such a repository is currently being developed.

Once the domain-specific generative framework is built, it can be used to generate families of web applications within a particular domain, as Figure 4 shows. The designer of the web application only needs to describe the application using a model, which is an instance of the meta-model of the domain (see (a) in Fi-

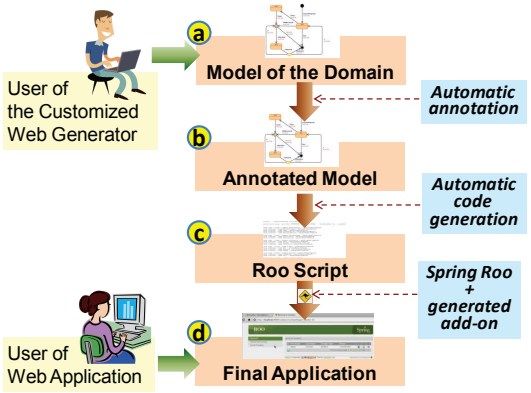

Figure 4: *Step 2*: Using the customized web generator to synthesize the final web application.

gure 4). Then, this model is automatically annotated using the profile that was generated from the *add-on* model (see (b) in the figure), and a Roo script is subsequently generated based on these annotations (see (c) in the figure). The Roo script utilizes the *add-on* previously defined, and is used to synthesize the final web application using Spring Roo (see step (d) in the figure). The automatic annotation process and the Spring Roo script generation are detailed on the *Model2Roo* project (Castrejón et al., 2011).

4. EXAMPLE

To test the *EMF on Rails* approach, we have developed an editor for building models through a web application. The application allows the instantiation of meta-models, as well as the serialization of models as XMI files in the same way as it is done within Eclipse using EMF. Additionally, the application stores models in a database, so that model objects can be easily reused across different models. This has the advantage that not only models, but also model fragments, can be serialized into XMI files. Indeed, this functionality goes beyond what is available in the standard EMF model tree editor.

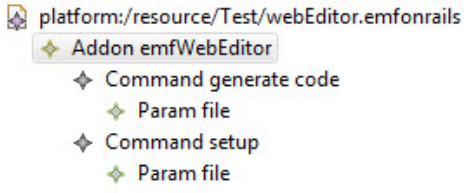

Figure 5: *Add-on* model of the EMF web editor. This model is an instance of the meta-model shown in Figure 3.

As explained in the previous section (Figure 2), to develop this application we needed to define (a) the meta-model of the domain and (b) the *add-on* model of the EMF web editor. Step (a) was not necessary in this case because we directly used the meta-meta-model of EMF, so that the user of the customized web generator can generate web applications from Ecore meta-models (step (a) in Figure 4). For step (b), we built an instance of the *add-on* meta-model shown in Figure 3. Figure 5 shows this instance, which defines (1) a command to generate the Java classes from an Ecore meta-model; and (2) a command to serialize, within the web application, instances of a meta-model in XMI format.

Starting from the *add-on* model, we generated the annotations profile (see Figure 6) and the code skeletons of the two Spring Roo commands that make up the *add-on*. Then, we added the code for the Java clas-

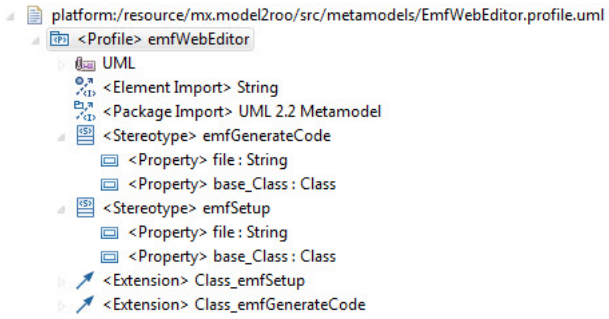

Figure 6: Annotations from the EMF web editor.

ses of the Spring Roo *add-on* commands. With these two steps, we were able to generate the customized web generator for the family of web editors for EMF models.

To test the web generator, we have developed a meta-model for describing e-learning systems that include an Intelligent Tutoring Systems (ITS) (Self, 1999; Devedzic and Harrer, 2005). This corresponds to step (a) in Figure 4. Intelligent tutoring systems are tools that incorporate artificial intelligence techniques to help students gain knowledge during the teaching-learning process. Figure 7 shows the most significant elements of our *tutor*, which is composed of learning objects (such as lessons and tests) and Bayesian networks (Pearl, 1988; Jensen, 2001) to infer knowledge.

Once we had an initial version of the ITS meta-model, it was automatically annotated using the annotations generated previously. This step corresponds to (b) in Figure 4. Finally, we executed the transformation that generates the Spring Roo script (step (c) in Figure 4). The transformation transforms the annotations into Spring Roo commands, including the commands defined in the EMF *add-on* model descri-

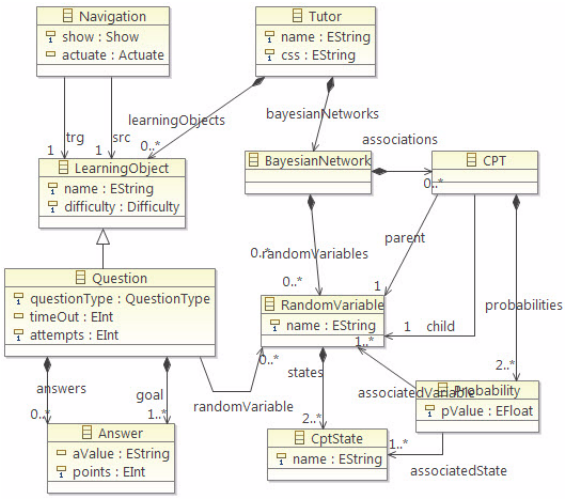

Figure 7: Most significant elements in the *tutor* model.

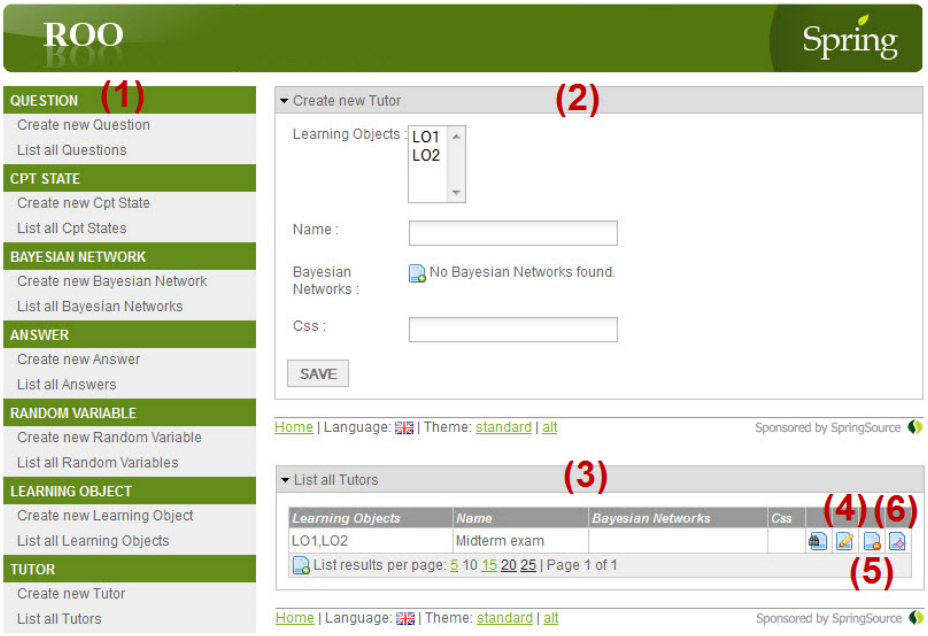

Figure 8: Graphical user interface generated by Spring Roo.

bed at the beginning of this section. Then, this script was processed within the Spring Roo command-line shell (step (d) in Figure 4) to produce the final web application shown in Figure 8. Altogether, each class of the *tutor* model was transformed into an element of the menu (1). Moreover, Spring Roo generated operations to create (2), read (3), update (4) and delete (5) objects. The layout of the generated web application is the same for any application built with Spring Roo. However, as a result of using the EMF *add-on*, the generated web applications can generate XMI files (6) from the data stored in the database.

Figure 9 shows an instance of the *tutor* meta-model generated using the EMF web editor, which comprises two learning objects. It can be seen that our web editor is equivalent to use the EMF editor, in the sense that both editors generate the same XMI files. Nevertheless, with the database of the web application generated with Spring Roo, users can easily reuse object instances. Also, if the web application is deployed on a web server (such as Tomcat, Glassfish or Jetty), users can edit it through mobile devices and share model objects.

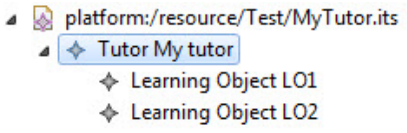

Figure 9: A *tutor* instance generated with the web editor and shown through the EMF editor.

To summarise, once the generator is built, we can use it to synthesize a customized web application for any Ecore meta-model. It should be noted that the generation of the web application from the *tutor* meta-model shown in Figure 7 did not require any coding or developer intervention. Users of the generator are only required to provide the model of the domain (step (a) in Figure 4). In this way, *EMF on Rails* expedites and facilitates the development of web applications providing significant savings in time and effort without requiring expertise in system development.

5. RELATED WORK

There has been a lot of supporting work for models to source code transformations. Regarding web application development, we can mention the work of Hou *et al* (Hou et al., 2006), where a modelling approach for web applications models, based on UML extensions, is proposed. We can also highlight the work described by Abdella (Daissaoui, 2010) where an approach based on the extension of UML class diagrams, and the use of the EMF framework to build Java web applications, is proposed.

Several authors have proposed domain-specific languages for web modelling, like WebML (Ceri et al., 2000), WSDM (Troyer and Leune, 1998), OO-METHOD (Pastor et al., 2001) or Labyrinth (Díaz et al., 2001). Apart from the last proposal, the other modelling languages include a dedicated language to

model the concepts of the domain, and follow model-driven approaches to generate the final application from models describing different aspects of the application, such as presentation and navigation.

Our work differs from the previous ones in that we support the automated construction of generators for families of web applications, and not only single web applications.

6. CONCLUSIONS

In this paper, we have proposed an approach that combines MDE with automation frameworks for web development like Spring Roo. Our approach automates the creation of code generators for families of web applications. This enables the rapid generation of domain-specific web applications from annotated models by generating Spring Roo scripts. We have illustrated our approach with the generation of a web application for editing EMF models, thus enabling, e.g., editing models through mobile devices. Similarly we have shown that *EMF on Rails* provides significant savings in time and effort for web application development.

EMF on Rails is currently in development. Therefore, the future work includes (1) adding a command library that facilitates the reuse of commands; (2) developing an *add-on* to propagate evidence and infer knowledge in Bayesian networks; and (3) improving the *tutor* model.

Acknowledgments

This work was supported by a grant provided by CONACyT and Tecnológico de Monterrey, Mexico City Campus. This research is part of the project "Dynamic Probabilistic Graphical Models and their Applications", number 95185, funded by CONACyT and the European Union through FONCICyT. This work is also supported by the Spanish Ministry of Economy and Competitivity (TIN2011-24139) and the R&D programme of the Madrid Region (S2009/TIC-1650).

REFERENCES

Castrejón, J. C., López-Landa, R., and Lozano, R. (2011). Model2Roo: A model driven approach for web application development based on the Eclipse Modeling Framework and Spring Roo. In *CONIELECOMP'11*, pages 82 –87.

Ceri, S., Fraternali, P., and Bongio, A. (2000). Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, 33(1-6):137–157.

Daissaoui, A. (2010). Applying the MDA approach for the automatic generation of an MVC2 web application. In *RCIS'10*, pages 681 – 688.

Devedzic, V. and Harrer, A. (2005). Software patterns in ITS architectures. *Int. J. Artif. Intell. Ed.*, 15(2):63 – 94.

Díaz, P., Aedo, I., and Panetsos, F. (2001). Modeling the dynamic behavior of hypermedia applications. *IEEE Trans. Software Eng.*, 27(6):550–572.

Hou, J., Wan, J., and Yang, X. (2006). MDA-based modeling and transformation approach for web applications. In *ISDA'06*, pages 867–874. IEEE CS.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31 – 39. See also http://www.emn.fr/z-info/atlanmod/index.php/Main_Page . Last accessed: Nov. 2010.

Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc.

Langer, P., Wieland, K., Wimmer, M., and Cabot, J. (2011). From UML profiles to EMF profiles and beyond. In *TOOLS'11*, volume 6705 of *LNCS*, pages 52–67. Springer.

Long, J. (2011). *Getting Started with Roo*. O'Reilly.

OMG (2005). *MOF QVT Final Adopted Specification*. Object Modeling Group.

Pastor, O., Gómez, J., Insfrán, E., and Pelechano, V. (2001). The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Inf. Syst.*, 26(7):507–534.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Mateo, California.

Poulin, J. S. (1993). Integrated support for software reuse in computer-aided software engineering (case). *SIGSOFT Softw. Eng. Notes*, 18(4):75–82.

Rimple, K. and Penchikala, S. (2012). *Spring Roo in Action*. Manning Publications.

Self, J. (1999). The defining characteristics of intelligent tutoring systems research: Itss care, precisely. *Int. J. Artif. Intell. Ed.*, 10:350 – 364.

Spring Source Community (2012). Spring roo project. http://www.springsource.org/spring-roo/ .

Steinberg, D., Budinsky, F., and Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley Professional, second edition.

Troyer, O. D. and Leune, C. J. (1998). WSDM: A user centered design method for web sites. *Computer Networks*, 30(1-7):85–94.

Völter, M. and Stahl, T. (2006). *Model-driven software development*. Wiley.