An UML profile for modeling RESTful services

Emilio Ormeño, María Lund, Laura Aballay, and Silvana Aciar

Institute of Informatics
National University of San Juan,
Meglioli Sur 1051, Rivadavia, San Juan, Argentina
{eormeno,mlund,laballay,saciar}@iinfo.unsj.edu.ar

Abstract. Usually, RESTful services have been used as technology platform for web services and for accessing web resources. More recently, these services have been used as controllers in applications that implement MVC pattern. Tools such as Spring and Spring Roo, greatly facilitate the development and maintenance of such applications. However, despite the maturity of the solutions, the few proposals for modeling RESTful services, focus only on REST Architecture concepts, leaving aside the various uses of these services, and the technological elements that support them. Based on the controllers and artifacts generated by Spring Roo, in this paper we present an UML profile for modeling RESTful services. The models using the profile can represent the various uses of the services, and also they can be transformed into a working Spring Web Application.

1 Introduction

In recent years, REST [1] services were preferred over the traditional Web Services using SOAP. This is because their attributes of scalability, evolvability [2], interoperability, and their relative ease of implementation. More recently, RESTful services have begun to be used as controllers in Web applications that implement the Model-View-Controller (MVC) pattern. This new functionality allows you to include within the same class of service, RESTful services that operate as a controller with RESTful services that operate as Web services.

In the current state of the art, to work with RESTful services, there are two lines: (1) development tools focused on the programmer, and (2) modeling techniques of RESTful services from an academic perspective:

1. The development of the services is partly solved by frameworks and generative programming tools [3] which facilitate many of the implementation tasks. Spring Framework [4, 5] stands as one of the most successful frameworks for Java to automate much of the development of RESTful services [6]. More recently, Spring Roo [7, 8] facilitates the construction of a Spring project, using the best practices for producing the RESTful controllers, the views for the domain entities, and even their ORM [9]. This generative programming technique, that produces the entities CRUDs (CReate-Update-Delete), is also known as scaffolding.

2. Regarding RESTful service modeling, several authors [10–13] propose various techniques mainly focused on the concepts of REST architectures and access to resources.

Both lines do not address the overall problem: on the one hand, generated code that lacks of a clear architecture of the involved artifacts, and on the other hand, modeling from a purist perspective, which does not consider the technology involved, and the various uses of RESTful services.

Focused on reducing the gap between both perspectives and the design effort, in this paper we present a set of UML elements and stereotypes, conforming an UML profile, that allow a software designer, design RESTful services, the interactions with the user interface, and the controllers of a MVC web application.

These elements and stereotypes emerge as a result of applying a metamodeling process that, through successive stages of refinement and abstraction, helps an advanced designer, to stereotype the Java code and the artifacts generated by Spring Roo. At the current stage of development, these stereotypes are packaged in UML profiles [14, 15] of the Enterprise Architect [16] tool alongside a set of transformation code templates that provides the same tool [17].

This paper proceeds as follows: The next section describes related work. Section 3 discusses the technologies used to prepare the proposal. Section 4 describe the UML stereotypes. Section 5 shows the design and implementation of a study case using the stereotypes and templates of our proposal. Section 6 describes the metamodeling process we follow to obtain the UML profile. Finally, Section 7 concludes and discusses future research.

2 Related Work

Silvia Schreier [13] comments the lack of mechanisms for modeling UML RESTful services and proposes a metamodel and a terminology for this type of design services.

On the other hand, Porres and Rauf et al. [11, 12], propose the use of class diagrams and UML state machines to model REST services. The authors model interfaces with state diagrams, and proposes the use of functions based on the HTTP protocol that test the presence or absence of a resource.

Laitkorpi and Koskinen [18], define a mediation mechanism between the definition of a service and a RESTful API for RESTful services. This mechanism uses a tool chain based on UML for helping developers in modeling and implementing REST services.

Concerning the use of templates and code generation for Spring Roo, highlights the work done by Castrejon, López-Landa and Lozano, who present a tool called Model2Roo [19]. They generate scripts for Spring Roo from entity classes with this tool including the Roo command to generate the CRUDs controllers for each entity.

Our approach differs from these proposals in the sense that we model with simple UML elements the interaction and the structure, without introducing into

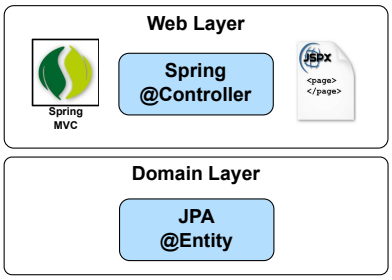

Fig. 1. Target technologies of the modeling strategy.

the logic of the service itself as is proposed by Porres and Rauf. Regarding the work of Schreier, our proposal is more integrated into the MDA [20, 21] process. Finally, although we generate code in a similar way than the work of Castrejon, our proposal can model the entire MVC web application.

3 Related Technologies

Below, we describe in detail the technologies employed.

3.1 Spring Roo

Spring Roo (or simply Roo) is a Maven-based RAD tool that generates Java applications on Spring Framework. Roo takes domain entities defined by a programmer and can generate a complete and working web application with the corresponding CRUD for the selected entities. Later, the development and customization of the generated project could be continued from the Eclipse environment.

While Spring Roo allows the developer to select different frameworks to build the project, our modeling approach attempts to abstract elements of the frameworks: Spring MVC [22], JPA [23] for domain classes, and JSPX [24] for views. Figure 1 shows an architecture with the considered frameworks.

The statements that make up the Roo application can be defined in a script. For example, the script in Listing 1.1 creates a Web application for a guestbook. In lines 3 and 4 is defined the GuestBook entity with the message attribute. Line 5 specifies a RESTful controller called GuestBookController which does not allow or modification or deletion. Line 2 defines Hypersonic as database and Hibernate as ORM. The resulting Web application can be seen in the figure 2.

```
1 project --topLevelPackage com.foo
2 persistence setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY
3 entity --class ~.domain.GuestBook
4 field string message
5 controller scaffold --class ~.web.GuestBookController
      --disallowedOperations update,delete
```

Listing 1.1. Guest Book Roo Script

4

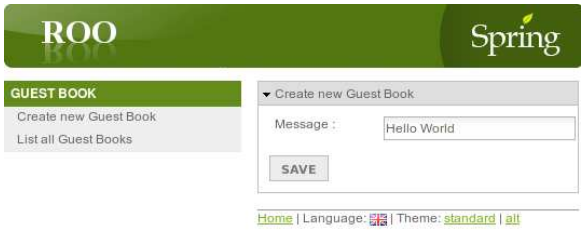

Fig. 2. The Guestbook CRUD generated by Roo.

Listing 1.2 shows an excerpt of the RESTful controller generated by Spring Roo for the script of the listing 1.1.

- Lines 1 and 2 show the annotations that configure the class as a RESTful controller, whose resources will be accessed from the URI `/guestbooks`.
- The annotation of line 5, indicates that the `createForm` method is executed when the parameter `form` is present in an URL via the HTTP GET method. The method instantiate an object GuestBook and send it to the view `create`.
- Between lines 8 and 9, the `create` method receives a GuestBook object from a view, makes the persistence of the object and redirects to a view that shows the new object. The `RequestMapping` annotation indicates that the method is executed with HTTP POST.
- The `show` method of line 12, receives as a parameter the ID of the object to be displayed, after found the object, the method redirects to a view called `show`. Note the pattern {`id`} defined in the URI.

```
1 @Controller
2 @RequestMapping("/guestbooks")
3 public class GuestBookController {
4
5  @RequestMapping(params="form", method=GET)
6  public String createForm(Model uiModel) {...}
7
8  @RequestMapping(method=POST)
9  public String create(GuestBook guestBook){...}
10
11  @RequestMapping(value="/{id}", method=GET)
12  public String show(@PathVariable("id") Long id){...}
13 }
```

Listing 1.2. Roo Generated Guestbook RESTful controller pseudo code

In short Spring Framework and Spring Roo are of particular interest in our study because, first, the technologies are mature and widely disseminated. Second, code generation is based on templates developed by experts. Finally, the generated application implements design patterns that enhance modifiability, scalability, and performance.

3.2 Enterprise Architect Design Tool

In our initial study, Enterprise Architect (EA) has features that make it suitable due to:

- It facilitates the complete cycle management of software projects.
- It allows to design with UML 2.3 and is compatible with MOF.
- It provides Java libraries to read or modify the models. In our current and future work, this is important for decoupling the generation of Roo applications of the EA design environment.
- It Provides support for MDA transformations. In this regard:
  - It provides mechanisms for metamodeling.
  - It has a code generation framework (CFT) that allows to customize the source code generation of the models, from the analysis of the UML metamodel.

Because these last two features are of particular interest in our study, we will describe below in detail.

3.3 Metamodeling with Enterprise Architect

EA offers agile mechanisms for domain metamodeling. Figure 3 shows the definition of two stereotypes of our proposal: the stereotype "rest-controller" extending the metaclass Class, and the stereotype "rest-service" as an extension of the metaclass "Interface".

The graphical notation of a stereotype can be defined with a language for this purpose. The figure 4 shows the graphical notation of the "rest-controller" stereotype in the shape editor window.

In general, in order to use the stereotyped UML elements of a profile in an EA project, the software designer have to import a XMI file with the profile and its resources (shapes, colors, templates). Figure 5 shows a model that uses the stereotypes of the profile "RESTful.xmi." In the same figure, notice the tool bar with elements of the profile.

3.4 Enterprise Architect's Code Template Framework

EA provides a framework called Code Template Framework to generate source code files. To generate code for a specific metamodel element, its template can be customized. Figure 6 shows the template to generate Java code for "Class" elements stereotyped as "rest-controller".

4 The UML Profile

With the aim of reducing the modeling effort of RESTful services, we consider necessary to extend both the syntax and semantics of certain elements of the UML metamodel. To do this, we used the standard extension mechanism known

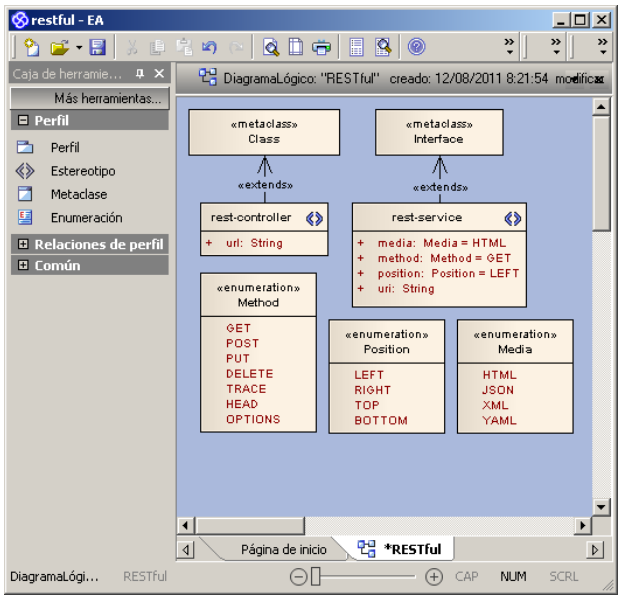

Fig. 3. The definition of stereotypes in EA.

Fig. 4. The shape editor of EA.

as profiling or stereotyping, which has been widely used to extend the set of UML elements to facilitate the modeling for specific domains.

The following subsections describe the stereotypes to model controllers, views and RESTful services.

4.1 Extension for controllers

A RESTful controller is represented as a class with a stereotype called "rest-controller". This stereotype adds the attribute `url` containing the base URL to access the exposed services. Figure 7-a shows the definition of the stereotype `rest-controller` as a specialization of the metaclass `Class`. Figure 7-b shows

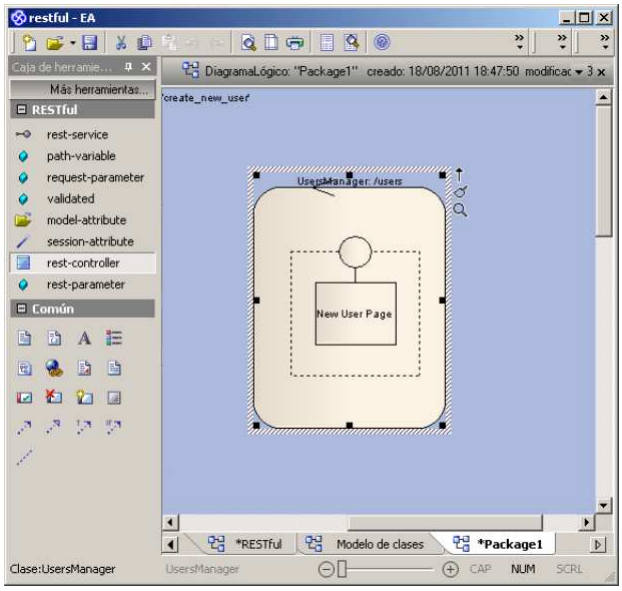

Fig. 5. A diagram using the profile.

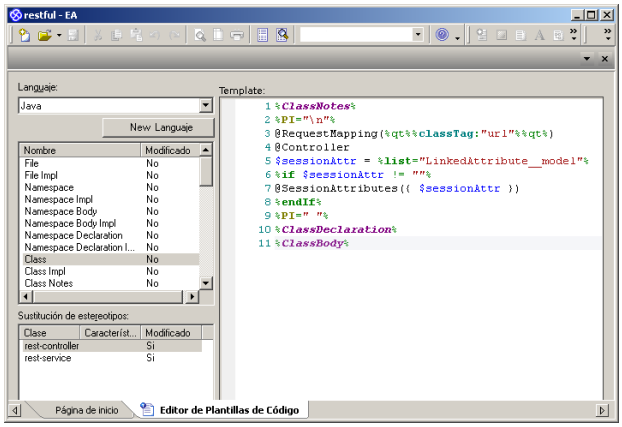

Fig. 6. The Template for the rest-controller stereotype.

the graphical notation of the stereotype with the name of the controller's class and its url (both definitions can also be seen in figures 3 and 4).

4.2 Extensions for the RESTful services

From an architectural point of view, the key concepts in a REST service are: resources have a uniform interface (a), these resources are addressed by resource identifiers (b), with representation (c) and hypermedia links between them (d);

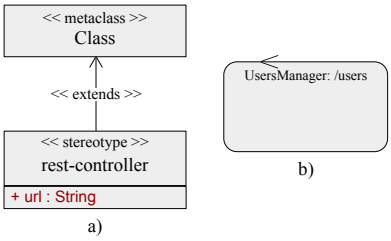

Fig. 7. The `rest-controller` stereotype definition.

and the communication between client and server is stateless (e). Figure 8-a shows the formal definition of the stereotype `rest-service`. Figure 8-b shows how to use this extension for designing a rest service for listing users. The stereotype includes the attribute `position` that specifies the position of the lollipop interface for the graphical notation. In resume, our RESTful stereotype characteristics are:

- The stereotype `rest-service` abstracts a RESTful service which is modeled as an extended UML interface. This element should be embedded within a controller.
- The URI of the service is modeled as an attribute of the stereotype "rest-service".
- URI variables. Also called variable path, is a variable defined by a pattern within the URL of the resource. For example, a variable to access a particular user might be "/users/{userNumber}" and a valid URI for this mapping, would be `/users/30401`.
- HTTP parameters. The parameters included in the URI.
- The service's HTTP method (`GET`, `POST`, `PUT` or `DELETE`) is defined in the `method` attribute.
- The invoked service may have different answers depending on the client. For example, the resource `/users/30401` could return a stream JSON [25], YAML [26], XML, or just redirect to a HTML view.

The variables and parameters of the service are modeled as stereotyped attributes. Figure 9 shows the definition of the stereotypes `path-variable` and `request-parameter`. The stereotypes `validated` and `model-attribute` are described in section 5.

4.3 Extensions for the views

Both views and GUI elements are modeled with stereotyped classes. Each view has a container element called `ViewPage` that defines: its name (`viewName` attribute), the associated URL (the name of the class), and the state of the session (`session` attribute) for indicating whether the view begins a session, the session is intermediate, or it is final. The graphical notation of the stereotype `ViewPage` changes according this state (see figure 10).

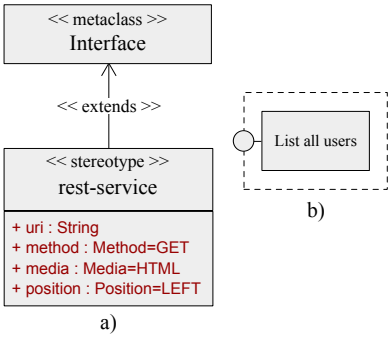

Fig. 8. The `rest-service` stereotype definition.

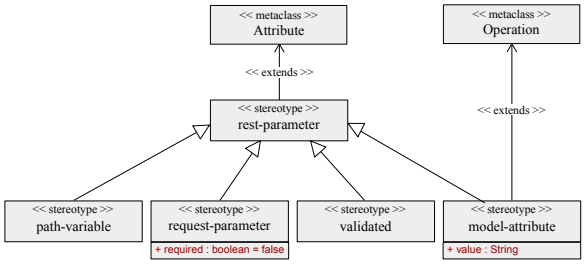

Fig. 9. The `rest-parameters` stereotypes.

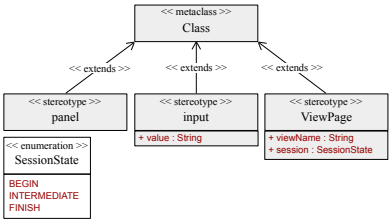

Fig. 10. The stereotypes for GUI elements.

In the current state of development we have not yet implemented full code generation for views. Figure 10 shows the definition of stereotypes for view elements, figure 11 shows the design of a GUI views that includes some of the stereotyped elements.

5 Study Case

To exemplify the use of the profile, we model an user registration functionality. Then, we apply the code generation templates to generate: (1) the stubs for the controller, (2) the RESTful services, and (3) the Roo script for the model's entities and views. In short, the functionality to be modeled is as follows:

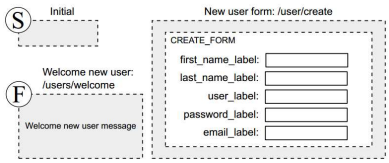

Fig. 11. GUI design example using the defined view stereotypes.

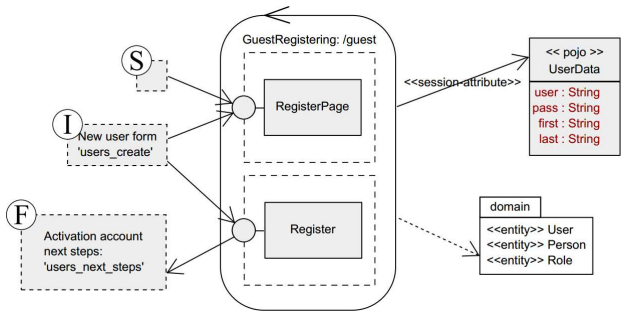

Fig. 12. The registration subsystem designed with the extensions.

1. A guest require registration in the system.
2. The system asks the guest to complete a form with: last name, first name, user name, password, and email.
3. Once the guest submits the form with correct data, the system persists the first name, the last name and the email into a Person entity, while the user name and the password into an User entity with an inactive state.
4. In order to validate the user entered a valid e-mail, the system sends an email to the specified e-mail with instructions to activate the account (the user can not perform any operation if the account is not active).
5. Finally, the system responds the user with a page that indicates that he/she should have received an e-mail with instructions to activate his/her account.

The figure 12 showing the registration subsystem design deserves some comments:

- The whole structure and interaction concerning a controller is in a single class diagram. We hope this approach could reduce the modeling effort and enhance the visibility of the whole interaction.
- The example displays the three states of the `ViewPage` stereotype: `START`, `FINISH` and `INTERMEDIATE`.
- The direction of the associations explicit a sequence, so we think it is no necessary a sequence or activity diagram.
- The class with the stereotype `POJO` (Plain Old Java Object), will become a typical JavaBean with getters and setters for each attribute [27].
- The association with stereotype `session-attribute` define an object with 'interaction' scope. For example, an object of type `UserData` is passed between

the interface and the RESTful services. Here, the designer is free for using the session state (violating the stateless of the REST style) or not.

Listings 1.3 and 1.4 shows part of the Roo script and part of the Java code, both generated from the EA environment with the templates.

```
project --topLevelPackage edu.idei.example
persistence setup --provider HIBERNATE --database MYSQL --databaseName
    ...

entity --class ~.domain.Person
entity --class ~.domain.User

focus --class ~.domain.Person
field string first
field string last

focus --class ~.domain.User
field string user
field string password
field string email
...
web mvc view --path /guest/users_create --title "New user form"
```

Listing 1.3. Generated Roo script for entities and views

```
package edu.idei.example.web;
...
@RequestMapping("/guest")
@Controller
@SessionAttributes({ "userData" })
public class GuestRegistering {
 @RequestMapping(value="register",
 method=RequestMethod.GET)
 public String registerPage(Model uiModel){
  // if (condition) {
  // return "users_create";
  // }
  return "";
 }

 @RequestMapping(method = RequestMethod.POST)
 public String register(Model uiModel){
  // if (condition) {
  // return "users_next_steps";
  // }
  return "";
 }
 public UserData m_UserData;
 public ActivationData m_ActivationData;
}
```

Listing 1.4. Generated code for controller and RESTful services

6 The Metamodeling process

The previous extensions emerge as a result of applying a metamodeling process that, through successive stages of refinement and abstraction, helps an advanced designer, to stereotype the Java code and the artifacts generated by Spring

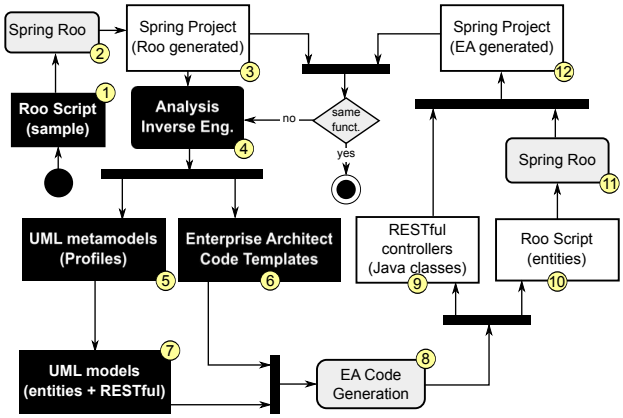

Fig. 13. The process followed to obtain the metamodel.

Roo. Before explaining the details of our process, it is necessary to clarify some premises:

1. We tried to abstract the Roo generated elements in order to obtain a set of platform independent models (PIM)[28].
2. In the case of entity classes, we used the stereotype "entity" already present in almost all design tools.
3. Extensions to the interactions are based on the MVC pattern [29].
4. In order to reduce the modeling effort, we represent in the same structural diagram, both structural and interaction elements.

Figure 13 shows an activity diagram with the process we follow to define the metamodels and the proposed extensions. In the figure, black elements with white text, represent manual artifacts or processes. The rectangles represent artifacts (files), while the rounded rectangles represent processes. Arrows can represent sequence, or the entry of an artifact to a process.

The following describes the process shown in Figure 13 referring to the numbers of each item:

– First, in the Spring Roo shell (1 and 2) we generate a working project of a simple application (3) (see figure 2).
– Then, manually we make the reverse engineering [30] of Java code and generated elements, and get a first generalization and abstraction of the observed elements (4). This abstraction is documented in the EA environment in the form of profiles (5) (packages of stereotypes) and code generation templates (6) (see figure 3).
– Using the profiles, we design the entities and the controller of an application similar to the original script (7) (see figure 12).
– Using the engine of transformation of EA, we obtain the source code of the previous design by applying the templates (8) for code generation. This

transformation generates two codes: a Java RESTful controller (9), and a Roo script from entity classes (10) (see listings 1.3 and 1.4).
– Roo script (10) is executed on the Roo environment (11) to generate another Spring Roo project (12) mixed with the RESTful controller (9).
– Finally, we compare the functionality of the original project regarding the obtained from the stereotypes and transformation templates; if the functionality is not similar, we review the analysis (4); on the contrary, if the functionality is similar, we terminate the process, and we begin again with a new Roo script (1) to incorporate more modeling elements.

7 Conclusion and future work

In this paper we have presented a proposal for modeling RESTful controllers using extended UML elements. Regarding other models, our proposal introduces a set of a stereotyped classes and relationships, conforming both a new UML structural diagram. In figure 12 the controller, the services, and the interaction process, are shown in one structural diagram. We hope this approach could reduce the modeling effort and promote the modifiability given that all the elements and its MVC interaction is shown in a sight.

Currently we are working on two aspects: in first place, we are trying to formalize the profiles using OCL; in second place, we are working in developing this proposal in free tools and platforms such as Eclipse [31] with plug-ins like EMF [32] or [33].

Furthermore, in the absence of proposals easy to use and effective for the generation of source code, we think that ours could be framed in a MDE environment with SPEM [34], to generate pieces of software, to distribute tasks to main stakeholders, and maintain synchronized the models with the source code, in a non-intrusive way.

Our proposal is still under development and the scope of our testing is very limited, so we can not predict whether their use can be extrapolated to larger systems with many controllers, views and services, or whether it constitutes a PIM. However, we consider this approach could represent a new paradigm, and so deserves further research.

Acknowledgment

This work was developed as part of the research project ForCupido financed by CICITCA.

References

1. Fielding, R.: Architectural styles and the design of network-based software architectures. http://www.ics.uci.edu/˜fielding/pubs/dissertation/top.htm [Online; accessed 19-apr-2012].

2. Riebisch, M., Bode, S.: Software evolvability. Informatik-Spektrum **32** (May 2009) 339–343
3. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. 1 edn. Addison-Wesley Professional (June 2000)
4. Walls, C.: Spring in Action. Third edition edn. Manning Publications (March 2011)
5. Springsource: SpringSource.org. http://www.springsource.org/ [Online; accessed 9-apr-2012].
6. Mak, G., Rubio, D., Long, J.: Spring Recipes: A Problem-Solution Approach. 2nd edn. Apress (September 2010)
7. Rimple, K., Penchikala, S., Dickens, G.: Spring Roo in Action. Manning Publications (December 2011)
8. Springsource: Spring Roo-Reference Documentation. http://www.springsource.org/roo/guide [Online; accessed 9-apr-2012].
9. Roebuck, K.: Object-relational mapping (Orm): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. Tebbo (June 2011)
10. Lendak, I., Varga, E., Erdeljan, A., Gavric, M.: RESTful web services and the common information model (CIM). In: Energy Conference and Exhibition (EnergyCon), 2010 IEEE International, IEEE (December 2010) 716–721
11. Porres, I., Rauf, I.: Modeling behavioral RESTful web service interfaces in UML. In: Proceedings of the 2011 ACM Symposium on Applied Computing. SAC '11, New York, NY, USA, ACM (2011) 1598–1605
12. Rauf, I., Ruokonen, A., Systa, T., Porres, I.: Modeling a composite RESTful web service with UML. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ECSA '10, New York, NY, USA, ACM (2010) 253–260
13. Schreier, S.: Modeling RESTful applications. In: Proceedings of the Second International Workshop on RESTful Design. WS-REST '11, New York, NY, USA, ACM (2011) 15–21
14. Frankel, D.S.: Chapter 6. extending and creating modeling languages. In: Model Driven Architecture: Applying MDA to Enterprise Computing. 1 edn. Wiley (January 2003) 145–160
15. Favre, L.: Well-Founded metamodeling for Model-Driven architecture. In Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O., eds.: SOFSEM 2005: Theory and Practice of Computer Science. Volume 3381. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 364–367
16. Sparx Systems: Enterprise Architect UML modeling tool. http://www.sparxsystems.com/ [Online; accessed 21-apr-2012].
17. Sparx Systems: Code Template Framework. http://www.sparxsystems.com/enterprise_architect_user_guide /8.0/software _development /codetemplates.html [Online; accessed 3-apr-2012].
18. Laitkorpi, M., Koskinen, J., Systa, T.: A UML-based approach for abstracting application interfaces to REST-like services. In: 13th Working Conference on Reverse Engineering, 2006. WCRE '06, IEEE (October 2006) 134–146
19. Castrejón, J.C., López-Landa, R., Lozano, R.: Model2Roo: A model driven approach for web application development based on the eclipse modeling framework and spring roo. In: Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference on. (March 2011) 82–87
20. Frankel, D.S., Parodi, J., Soley, R.: The MDA Journal: Model Driven Architecture Straight From The Masters. Meghan Kiffer Pr (November 2004)

21. Object Management Group (OMG): MDA Guide Working Page. http://ormsc.omg.org/mda_guide_working_page.htm [Online; accessed 9-apr-2012].
22. Springsource: Spring Web Flow [Online; accessed 9-apr-2012].
23. Yang, D.: Java(TM) Persistence with JPA. Outskirts Press (March 2010)
24. Zambon, G., Sekler, M.: Chapter 2. JSP explained. In: Beginning JSP, JSF and Tomcat Web Development: From Novice to Professional. 1 edn. Apress (November 2007) 25–27
25. Json.org: JavaScript Object Notation. http://www.json.org/ [Online; accessed 11-apr-2012].
26. Ben-Kiki, O., Evans, C., Net, I.: The official YAML web site. http://www.yaml.org/spec/1.2/spec.html [Online; accessed 21-apr-2012].
27. Fowler, M.: Pojo. http://www.martinfowler.com/bliki/POJO.html [Online; accessed 12-apr-2012].
28. Lano, K.: 5. Platform-Independent design. In: Advanced Systems Design with Java, UML and MDA. 1 edn. Butterworth-Heinemann (June 2005) 97–128
29. qiang Huang, S., ming Zhang, H.: Research on improved MVC design pattern based on struts and XSL. In: International Symposium on Information Science and Engineering, 2008. ISISE '08. Volume 1., IEEE (December 2008) 451–455
30. Chikofsky, E., Cross, J.H., I.: Reverse engineering and design recovery: a taxonomy. Software, IEEE **7**(1) (jan 1990) 13 –17
31. Eclipse Foundation: Eclipse - the eclipse foundation open source community website. http://www.eclipse.org/ [Online; accessed 22-Sep-2011].
32. Eclipse Foundation: Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/ [Online; accessed 21-apr-2012].
33. Eclipse Foundation: Graphical Modeling Framework. http://www.eclipse.org/modeling/gmp/ [Online; accessed 22-apr-2012].
34. Object Management Group: SPEM 2.0. http://www.omg.org/spec/SPEM/2.0/ [Online; accessed 26-apr-2012].