An Aspect Oriented Approach for the Synchronization of Instance Repositories in Model-Driven Environments

Juan Castrejón

ITESM, Campus Ciudad de México
Calle del Puente 222, 14380, México, D.F., México
A00970883@itesm.mx

Abstract. Software development based on the transformation of meta-model abstractions into particular models and code artifacts is an active research line within software engineering. However, there has been less emphasis on the interoperability of the applications in charge of producing runtime instances of these models. As a consequence, model instances are tied to particular combinations of models and generation tools. If these combinations are to be changed, the data associated to the model instances is at risk of being lost. In this paper, an aspect oriented approach is proposed in order to allow the synchronization of the instance repositories associated to a common meta-model, by using a model bus in charge of receiving and distributing notifications of updates made to the particular object instances. Finally, in order to demonstrate the benefits of the proposed approach, an implementation based on the Eclipse Modeling Framework and Spring Roo is presented.

Keywords: Metamodeling, Software engineering, Software maintenance.

1 Introduction

Model-Driven Software Development (MDSD) has recently gained a great deal of attention in the software engineering area. Reasons for this situation include a perceived effectiveness in avoiding the lack of compliance between design documents and implementation artifacts [1], as well as an increased productivity due to its focus on model abstractions rather than on implementation details [1].

According to this discipline, development teams should aim their attention at modeling the highest abstraction levels of software systems, and then rely on transformation procedures to generate low-level abstractions. In doing so, the model abstractions generated by the development teams do not depend on any specific platform, language or implementation details. This mechanism allows the generation of a set of low-level models that depend on a common high-level model. This is especially useful when a system targets a heterogeneous environment, involving different platforms, tools and programming languages [1].

In this scenario, each environment provides its own model definition (for example UML or Ecore), instance repository technology and access mechanisms (for instance Relational Databases or XML records). These differences can potentially lead to interoperability issues among the different model abstractions [2]. This in turn may prevent an effective synchronization of the data generated by these low level models

Advances in Computer Science and Electronic Systems
Research in Computing Science 52, 2011, pp. 179-189

Received 23/01/11
Accepted 20/02/11
Final version 07/03/11

and the orchestration of the services that their associated tools provide [2]. One of the consequences of this situation is that the changes made in each environment are only kept locally. Therefore, if a particular environment becomes unavailable or if its instance repository is corrupted, the set of instance data associated to this environment is at risk of being lost.

In this paper, an approach based on Aspect Oriented Programming (AOP) techniques is proposed to allow the synchronization of the object instances associated to the repositories of a MDSD environment. It should be noted that the use of AOP is intended in order to conduct the synchronization mechanisms in a non-intrusive manner. Moreover, the addition of the AOP constructs to the application base code is performed during the model transformation procedures, in accordance to the MDSD approach. In this regard, there is a growing set of tools and frameworks intended to support software development based on MDSD principles [1]. Two of these tools are of particular interest for this study, the Eclipse Modeling Framework [3] (EMF) and Spring Roo [4].

EMF represents one of the most matured efforts to support the MDSD approach. It is built on top of the Eclipse platform [3] and is considered by many researches as one of the main environments for model-driven development, due to the size of its community and the number of experimental tools developed around it [3].

Spring Roo is an open source project intended to generate enterprise web applications by means of a set of commands executed through a command-line shell [4]. These commands deal not only with functional requirements, but also with a subset of the non-functional requirements usually associated to web applications [4].

The remaining of this paper is organized as follows. Section 2 introduces work related to the integration and synchronization of models in a MDSD environment. Section 3 contains the full description of the approach proposed by this study. In Section 4, a case study is presented in order to analyze the effectiveness of the proposed approach. Finally, Section 5 summarizes the ideas presented in this paper and introduces future work that might derive from this study.

2 Related Work

Regarding the integration of models and their associated tools in MDSD environments, we can refer to the work by Hein et al. [2], where a model bus is proposed in order to facilitate the orchestration of modeling services. These services represent automated operations over a set of models, such as *creation*, *editing*, *transformation*, *verification* and *execution* [2]. The model bus provides an abstraction layer intended to separate implementation details from the modeling services. This allows for *model representation*, *access* and *location* transparency between the different tools that provide these services. Unlike the approach proposed in this paper, the model bus deals with high-level model abstractions, instead of object instances generated from these models, that is, the lowest-level abstractions in MDSD.

In Breu et al. [5], the requirements and architectural concepts of an infrastructure to support model evolution are discussed. This infrastructure allows the cooperation of different tools that act upon a common system model. Its main functionalities include *model versioning*, *change identification*, *propagation* and *notification* [5]. Its change-

driven nature is similar to the one proposed in this paper. However our main intention is to keep up-to-date instance repositories as opposed to high-level models.

Regarding model evolution, we can mention the work by Herrmannsdoerfer [6]. In this work, the author proposes a workbench to handle *meta-model adaptation*, that is, the evolution of models due to changing requirements and technological progress [6]. The workbench includes the recording of changes made to meta-models, and the corresponding migration processes that the associated models require to conform to these changes. Based on this history of changes, the intention is to automatically migrate the models associated to a common meta-model, during its evolution. The general idea of maintaining a coupled evolution between models is similar to the one proposed in this study. However, the approach described in this paper is more limited in scope, because it does not directly consider the evolution of meta-models.

3 Instance Model Bus

This section introduces the *Instance Model Bus* (IMB), an approach to synchronize instance repositories in model-driven environments. First, the general concepts of this approach are presented, followed by an implementation using EMF and Spring Roo.

3.1 General Approach

As discussed previously, when different models are generated from a common meta-model in a MDSD environment, there is risk of incompatibility between the object instances that each of these models generate. In order to overcome this limitation, a general approach is described in this section in order to automatically synchronize instance repositories. The approach is divided in two main phases, as follows. The first phase is in charge of generating an IMB instance, along with the constructs that allow the modeling tools to send and receive notifications of changes made to the object instances that they manage, using this particular IMB instance. In the second phase, the notification constructs are invoked during the execution of the modeling tools. Fig. 1 summarizes the set of elements, and their interactions, required in a MDSD environment using the IMB.

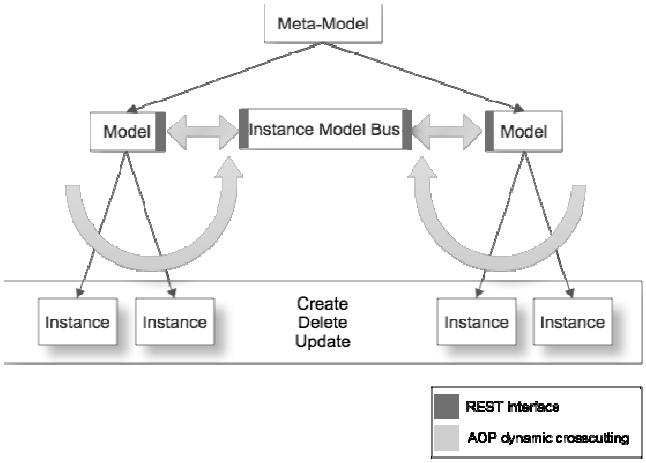

Fig. 1. Elements and activities associated to the Instance Model Bus.

The notification constructs are implemented with the aid of AOP crosscutting techniques. These constructs are generated as part of the transformation procedures from the common meta-model to each of the low-level models. In particular, these constructs advice the tools associated to each of the generated models, in regard to the following operations over the object instances: *create*, *update* and *delete*. When these AOP constructs are executed, they notify the change data to the IMB associated to this particular MDSD environment. In this regard, the IMB instance provides a set of REST interfaces [7] that the modeling tools can invoke for this purpose.

In order to avoid interoperability issues between the data formats used by each instance repository, the definition of transformation procedures that can convert each of these formats into a common data representation is required. In this regard, the use of XML [7] is proposed as the interchange data format for all of the communications to and from the IMB. The combination of REST interfaces and XML data interchange provides a great deal of flexibility, which makes it relatively easy to provide support to a wide range of modeling tools and instance repositories.

Regarding the runtime execution phase, when the IMB receives a notification of change by any of the modeling tools, it propagates the notification through all of the remaining modeling tools registered in the particular MDSD environment. In order to do so, each of the modeling tools should also provide REST interfaces to allow this interchange of data. These interfaces should also be generated during the transformation procedures that generate each low-level model.

Finally, when a modeling tool receives a notification of change by the IMB, it transforms the notification data to the particular format used in its associated instance repository, and performs the requested operation, that is, *create*, *update* or *delete* a particular object instance.

Considering that the previous activities rely on REST communications, the change notifications need to be made persistent by all of the participants. This allows a fallback mechanism in the event of failures preventing communication between the modeling tools, such as network delays or physical failures in their environments.

It should also be noticed that since the operations of the modeling tools are advised through AOP crosscutting, there is no need to manually modify their base code. This

is particularly helpful if development teams have access to the source code of the modeling tools and are in charge of their maintenance. An example of such scenario would be a generated web application acting as a modeling tool. In this way, the proposed approach is non-intrusive at the source code level.

3.2 Implementation

An implementation of the general approach discussed in the previous section is now presented for Java based systems. It is based on EMF and Spring Roo, regarding the generation of the modeling tools following a MDSD approach. For the generation of XML data, to and from the instance repositories, the use of the JAXB specification [8] is proposed. Finally, for the implementation of AOP crosscutting techniques, the use of the AspectJ project [9] is intended. Fig. 2 summarizes the set of tools that were selected for this implementation.

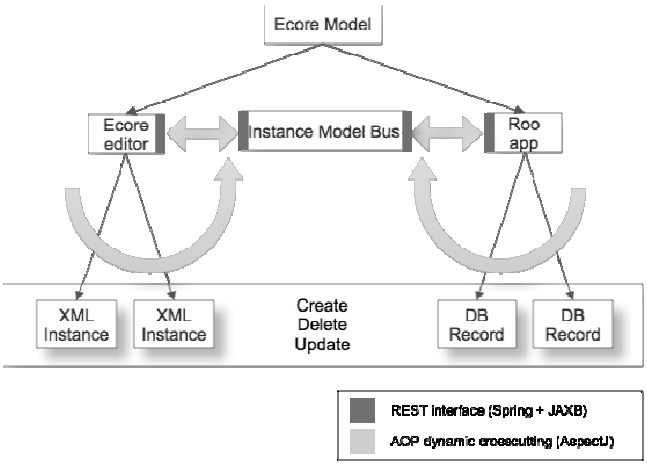

Fig. 2. Elements associated to the Instance Model Bus implementation.

In this implementation, the main meta-model is an instance of the Ecore model, which is the meta-model of the EMF framework [3]. Using the Ecore constructs we can define the static structure of an application. This would be the high-level model in a MDSD environment. We can then generate a set of low-level models taking as base the Ecore model, by using a set of transformation procedures. The current implementation provides two transformations, one to generate an Ecore editor, and a second one to generate a Spring Roo web application. Their details are described next.

The Ecore editor is created using the facilities of the EMF generator projects [3], along with a custom process to customize the data types specified in the associated meta-model. Moreover, in order to generate the REST interfaces and the XML data transformation support, an Eclipse plugin was developed [3]. This plugin is responsible for the creation of a web application associated to the Ecore editor, which

contains the required interfaces and XML support, along with aspect files that advice the *create*, *update* and *delete* operations of the Ecore editor.

The Spring Roo web application is created by generating Spring Roo commands, following a similar approach to the one described in [10]. In order to add REST and XML support to the generated web application, a Spring Roo add-on [4] was developed. This add-on provides a set of Roo commands that generate the aspects files in charge of providing the required interfaces to communicate with the IMB.

It also necessary to generate an IMB instance for this particular MDSD environment. This is achieved by executing an operation of the IMB Eclipse plugin. It should be noted that the generated IMB instance is also a Spring Roo web application. This allows us to reuse some of the Spring Roo generation logic described earlier.

Finally, as appreciated in Fig. 2, the object instances created by the Ecore editor and the Spring Roo application have incompatible data formats. While the editor generates XML files, the web application stores the instance data in database records. The IMB tools allow them to synchronize the data of their instance repositories, by transforming the data from these formats into a common XML definition, and vice versa. This transformation is conducted with the aid of JAXB tools, according to the static structure defined in the common Ecore meta-model.

In order to conclude this section, Fig. 3 depicts the operations of the Eclipse plugin that was developed for the current implementation of the IMB.

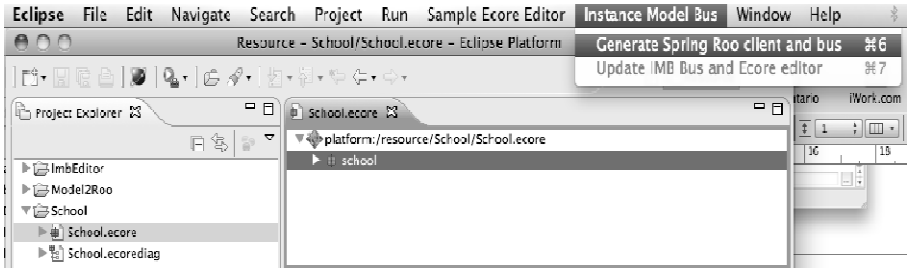

Fig. 3. Operations of the IMB Eclipse plugin.

4 Case Study

This section describes the development of a *Todo list* application, following a MDSD approach. The intention is to demonstrate the benefits of the approach proposed in Section 3, along with the operation of the IMB Eclipse plugin and the Spring Roo add-on described in the previous section.

The *To-do List* is a simple application intended to keep a list of pending tasks for a single user. The modeling of this application can be conducted with the Ecore Diagram editor [3], which is part of EMF. The resulting model is depicted in Fig. 4.

Once the Ecore model of the *To-do list* application is created, we can generate the corresponding Ecore editor, Spring Roo web application and an instance of the IMB for this environment. To this end, we use the IMB Eclipse plugin, and the facilities of the EMF generator projects. The resulting artifacts of this process are described next.

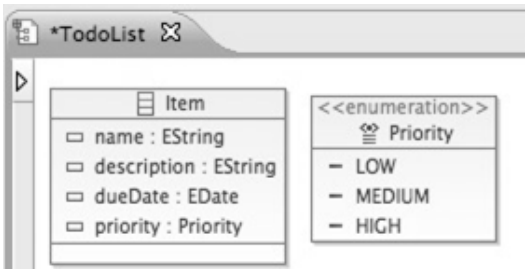

Fig. 4. Modeling of the To-do list application in the Ecore Diagram Editor.

For the creation of the IMB instance, a Spring Roo script is generated. The following program listing depicts the set of commands contained in this script file.

Set of Spring Roo commands that generate the IMB instance for the *To-do list* application.

```
  project --topLevelPackage mx.itesm.todolistbus
  persistence setup --provider HIBERNATE
  --database HYPERSONIC_IN_MEMORY
  jaxb    xsd    compiler    setup    --generateDirectory
src/main/java
  http resource representation setup

  enum type --class ~.domain.ToolName
  enum constant --name Spring
  enum constant --name Eclipse

  entity --class ~.domain.Tool
  field enum --fieldName name --type ~.domain.ToolName
  field string --fieldName description
  field string --fieldName ipAddress
  field string --fieldName port
  field string --fieldName contextPath
  controller all --package ~.web

  enum type --class ~.domain.ImbNotificationType
  enum constant --name CreateEntity
  enum constant --name UpdateEntity
  enum constant --name DeleteEntity
  entity --class ~.domain.ImbNotification
  field string --fieldName url
  field string --fieldName entity
  field string --fieldName entityName
  field string --fieldName returnTypeClass
  field number --fieldName entityId --type java.lang.Long
  field enum --type ~.domain.ImbNotificationType
  --fieldName type
  imb generate notificationScheduling
```

We can appreciate that the Spring Roo commands generate not only the static structure of the application, but also XML and notification scheduling support. In this

case, the static structure represents the modeling tools that can be registered in this MDSD environment, that is, either *Spring* Roo applications or *Eclipse* Ecore editors. The notification scheduling is required in order to persist the change notification data.

The next application to be generated by the IMB Eclipse plugin is the Spring Roo web application. Just like the IMB instance, this application is created from a Spring Roo script, as depicted in the following program listing.

Subset of Spring Roo commands that generate the Spring Roo *To-do list* application.

```
project --topLevelPackage mx.itesm.todolist
persistence setup --provider HIBERNATE
--database HYPERSONIC_IN_MEMORY
jaxb xsd compiler setup
--generateDirectory src/main/java
http resource representation setup

enum type --class ~.domain.Priority
enum constant --name Low
enum constant --name Medium
enum constant --name High

entity --class ~.domain.Item
field string --fieldName name
field string --fieldName description
field enum --fieldName priority
--type ~.domain.Priority
field date --fieldName dueDate --type java.util.Date
controller all --package ~.web
imb update controllers
imb generate schemas
perform command --mavenCommand compile
http resource add oxm
--class imb.domain.todolist.itesm.mx.Item
```

The last application that needs to be generated is the Ecore editor. This is achieved using the standard EMF Generator projects facilities [3]. In particular, we need to create an *EMF Generator Model* from the common meta-model, and then select the *Generate All* option from its associated menu. This last process is depicted in Fig. 5.

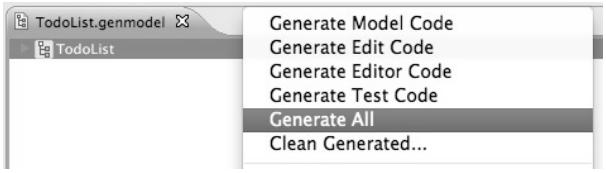

Fig. 5. Generation of the Ecore editor using an EMF Generator model.

Once the generated Spring Roo scripts are executed and the Ecore editor is generated, the aspect files that manage the synchronization of the instance repositories are created. As explained previously, this aspect files depend on the AspectJ project.

The following program listing depicts a subset of the aspect file generated for the Spring Roo application. The first operation advises the execution of the *create* method for instances of the *Item* class. This advice generates a change notification and then registers it with the scheduling program associated to the web application. The scheduling program will in turn deliver this notification to the IMB instance.

The second operation shown in this program listing provides a REST interface that the IMB instance can invoke upon reception of a change notification. This interface transforms from XML to the data format required to create an object instance in the database associated to the web application. Finally, it should be noted that the aspect files generated for the other applications are very similar to the ones described in this section.

Aspect file associated to the Spring Roo *To-do list* application.

```
  after(Item object, BindingResult result, Model model,
        HttpServletRequest request) returning :
  execution (*(mx.itesm.todolist.web.ItemController).
        create(..))   &&   args(object,   result,   model,
request){
      object.setImbId(System.currentTimeMillis()
      + new Random().nextInt());
      object.merge();
      new Thread(new NotificationThread(
          ImbNotificationType.CreateEntity,
          properties.getString("bus.address") +
          "/imb/create/{object}/Spring",
          ItemController_Roo_Imb.transformToImbItem
          (object), "item", object, marshaller)).start();
}

  @RequestMapping(value = "/create/item",
                  method = RequestMethod.PUT)
  public void ItemController.imbCreate(
              @RequestBody Item object) {
      Item modelObject = ItemController_Roo_Imb.
                         transformFromImbItem(object);
      modelObject.persist(); }
```

In order to conclude this section, the interfaces of the applications that were generated in the previous steps are depicted in Fig. 6. These applications synchronize their instance repositories using the aspects advising their operations to *create*, *update* and *delete* instances. For example, if an instance is *created* using the Ecore editor, the Spring Roo application receives a change notification by the IMB, and then creates the instance in its own repository. The same process applies when an object instance is *updated* or *deleted*, either in the Ecore editor or in the Spring Roo web application.

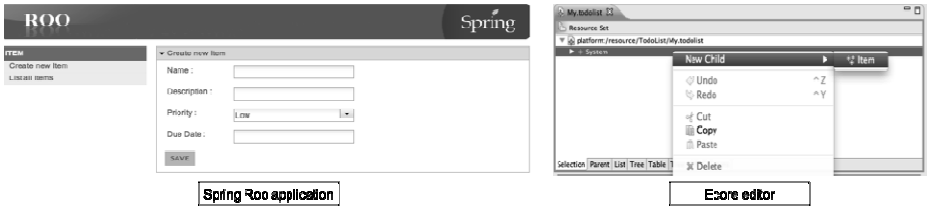

Fig. 6. Applications generated for the To-do list.

5 Conclusions

In this paper, an approach to synchronize the instance repositories within a MDSD environment was proposed. This approach relies on AOP to intercept the operations in charge of maintaining the particular object instances. The change notifications are sent through REST interfaces associated to the modeling tools. Both the aspects and the communication interfaces are generated during the transformation procedures from a common meta-model to particular low-level models.

An implementation of this approach was presented, based on EMF and Spring Roo. The implementation of the REST interfaces was conducted using the facilities of the Spring framework, and the AOP crosscutting was applied using the AspectJ project.

The main contribution of the approach proposed in this paper lies on the ability to automatically synchronize the instance repositories generated from a common meta-model. This is, changes performed over an object instance by any of the modeling tools, are automatically reflected in all of the instance repositories.

It should also be noticed that the current implementation of the *Instance Model Bus* deals with a subset of common non-functional requirements (NFRs), such as maintainability and interoperability of the generated applications. However, support for other types of NFRs, such as debugging and security, will be considered in future implementations. Finally, support for a wider set of programming tools and environments is also intended for future work.

Acknowledgments. The author would like to thank Rosa López-Landa and Dr. Rafael Lozano, both from ITESM Campus Ciudad de México, for their help in discussing and analyzing early ideas that led to the implementation of the approach proposed in this paper.

References

1. Stahl, T., Völter, M.: Model-Driven Software Development. Wiley, New York (2006)
2. Hein, C., Ritter, T., Wagner M.: Model-Driven Tool Integration with ModelBus. Workshop Future Trends of Model-Driven Development (2009)
3. Steinberg, D., Budinsky, F., Paternostro, M.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional, Boston (2008)
4. Spring Roo, http:// www.springsource.org/roo

5. Breu, M., Breu, R., Löw, S.: Living on the MoVE: Towards an Architecture for a Living Models Infrastructure. In: Fifth International Conference on Software Engineering Advances, pp. 290 -- 295. IEEE Computer Society, New York (2010)
6. Herrmannsdoerfer, M.: COPE – A Workbench for the Coupled Evolution of Metamodels and Models. In: 3rd International Conference on Software Language Engineering (2010)
7. Richardson, L., Ruby, S., Heinemeier, D.: Restful Web Services. O'Reilly Media, Sebastopol (2007)
8. Java Architecture for XML Binding, http://jaxb.java.net/
9. Laddad, R.: AspectJ in Action, 2 edition. Manning, Greenwich (2009)
10. Castrejón, J., López-Landa, R., Lozano, R.: Model2Roo: A Model Driven Approach for Web Application Development based on the Eclipse Modeling Framework and Spring Roo. In: 21st International Conference on Electronics, Communications and Computers (2011)