Generating Applications
Framework Reuse Supported by Domain-Specific Modeling Languages

Matheus C. Viana, Rosângela A. D. Penteado and Antônio F. do Prado
Department of Computing, Federal University of São Carlos (UFSCar), P.O. Box 676, 13565-905, São Carlos - SP, Brazil

Keywords: Reuse, Framework, Domain-Specific Modeling Language, Template, Domain Feature.

Abstract: Applications can be developed with efficiency and quality when supported by frameworks. However, framework reuse is a complex task and its advantages may not be reached if it is not correctly done. In order to mitigate this difficulty, this work proposes an approach that identifies the domain features of a framework to build a Domain-Specific Modeling Language (DSML) for this framework. This DSML can be used to create application models, whose information is mapped into templates aiming to generate the code of these applications. Framework reuse supported by a DSML provides greater efficiency and quality on the development of applications, since it provides a more abstract way to instantiate the framework and generates code from application models. We illustrate our approach using the GRENJ framework, which can be reused in applications in the domain of rental, purchase, sale and maintenance transactions.

1 INTRODUCTION

Frameworks are reusable artifacts which act as skeletons that can be instantiated in applications (Johnson, 1997). The reuse of frameworks provides higher quality and efficiency to software development process, since the classes of frameworks have been previously tested and the applications are not developed from scratch. However, frameworks require the developer to have detailed knowledge about their internal structure and their hot spots so that they can be properly used (Abi-Antoun, 2007)(Fayad and Schmidt, 1997).

Some solutions have been applied in order to ease the difficulties in reusing frameworks, such as manuals, cookbooks and pattern languages. These solutions can guide the application developer through framework instantiation. However, the task of identifying and configuring the hot spots according to the application requirements is still executed by the developer and relies on his/her skills and knowledge.

In order to use a more effective solution, this paper proposes an approach promoting the building of a Domain-Specific Modeling Language (DSML) to facilitate the reuse of frameworks on the development of applications. DSML can protect the developer from the framework hot spots complexity. The use of a DSML is focused on creating models that consider the features of a domain. Instead of trying to identify which classes must compose an application, when the developer uses a DSML, he/she concentrates his/her efforts on identifying which domain features such application should have.

The proposed approach is divided in two phases: Domain Engineering and Application Engineering. In Domain Engineering, a DSML and a set of templates are created based on the domain features and the hot spots of the framework. In Application Engineering, this DSML is used to create an application model, which is combined with the templates of the framework in order to generate the application code and other artifacts. Although the developer has to implement some parts of the application code manually, due to the fact that the framework and its DSML cannot predict application-specific functions, this approach improves the efficiency and the quality on the development of the applications that reuse a framework by generating most of the code of the applications and avoiding mistakes made by the developer while reusing the framework.

The idea of dividing an approach into Domain and Application Engineering is not new (Weiss and Lai, 1999). There are also other works which propose approaches for building DSML of frameworks (Oliveira et al., 2007)(Antkiewicz et al., 2009)(Amatriain and Arumi, 2011). However, they do not explain clearly how they identify the features of the frameworks. Our approach presents a

Viana M., Penteado R. and Prado A..
Generating Applications - Framework Reuse Supported by Domain-Specific Modeling Languages.
DOI: 10.5220/0003990000050014
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 5-14
ISBN: 978-989-8565-11-2
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

systematic way to identify the domain features of the framework and the information which is necessary to reuse its classes.

The remainder of this paper is organized as follows: Sections 2 and 3 present, respectively, frameworks and Domain-Specific Modeling Languages. Section 4 presents some related works. Section 5 presents the approach of framework reuse supported by Domain-Specific Modeling Language. Section 6 presents the use of the proposed approach to build the GRENJ Framework DSML. Section 7 presents the conclusions about this work.

2 FRAMEWORKS

Frameworks support software development by providing reuse of code and design. They are composed of two parts (Brugali and Sycara, 2000): frozen spots, which constitute unchangeable parts, regardless of the application that is being developed; and hot spots, which represent classes that are directly reused by the applications. The classes that are hot spots and their superclasses can contain hook methods that need to be overridden or invoked in order to customize the framework according to the application requirements.

A framework is classified according to the way its hot spots are accessed: 1) white box, when such access occurs through inheriting their classes and overriding their methods, 2) black box, when the access occurs through composition, and 3) gray box, when it occurs through the two previous ways. According to their purpose, frameworks can also be classified as: 1) System Infrastructure Frameworks (SIF), which simplify the development of software that controls low-level operations; 2) Middleware Integration Frameworks (MIF), which increase the modularization of applications; 3) Enterprise Application Frameworks (EAF), which are used to instantiate applications for specific domains (Abi-Antoun, 2007)(Fayad and Schmidt, 1997).

In this paper, we used the GRENJ Framework to exemplify our approach. The GRENJ Framework is a white box EAF that addresses the domain of rental, purchase, sale and maintenance transactions of goods or services (Durelli et al., 2010). The main reasons for choosing the GRENJ Framework are: 1) it contains all kind of features, e. g., mandatory, optional and variable; 2) applications within its domain are widely adopted by industry and academia; 3) its code is open; 4) we have developed applications reusing it, so we have full knowledge about its code, hot spots and functionality.

3 DOMAIN-SPECIFIC MODELING LANGUAGES

Modeling is the act of creating a model to get a better understanding of what is to be built and a more abstract view of a problem solution. It aims at facilitating the comprehension of the solution and serves as documentation (Gronback, 2009).

A modeling language is formed by: an abstract syntax, that defines the elements and the rules of the language and usually is represented by metamodel; and a concrete syntax, which defines its notation. The most common notations for modeling languages are textual, graphical and tree-view. (Cuadrado and Molina, 2009).

In place of general-purpose modeling languages, such as UML, a Domain Specific Modeling Language (DSML) can be used to model applications with elements that are appropriate to the addressed domain. Since there is a greater similarity between the DSML elements and the application features, mapping the application requirements is easier (Turki et al., 2004) and transformations from application models are more effective. Templates can be used as skeletons to generate several kinds of artifacts (Gronback, 2009), such as application code, documentation, test code and other models.

The main advantages provided by the use of DSMLs are (France and Rumpe, 2007): 1) better separation between the logic of the software and the details of the technologies employed in its implementation, 2) greater facility in creating reusable artifacts, and 3) better quality of the software that are produced since their features are well defined and their code can be generated.

4 RELATED WORKS

In this section, some studies reporting the difficulties during the reuse of frameworks and suggesting possible solutions to this problem are presented and compared with the approach proposed in this paper.

Braga and Masiero (2003) proposed building a wizard for the development of applications reusing an EAF. In this process, the developer fills out the wizard forms according to the application requirements and the wizard generates the application code. It is similar to the approach supported by a DSML. However, it does not generate code directly from the application model.

Oliveira et al. (2007) presented a systematic approach for framework reuse based on the Reuse Description Language (RDL), a language designed by these authors to specify framework instantiation processes. In their approach, RDL is used to register the framework hot spots in an XML-format file and a tool, denominated RDL execution environment, accesses this file for the execution of reuse processes and framework instantiations that lead to domain-specific applications. Therefore, their approach depends on the RDL and the RDL execution environment. The main advantage of our approach over theirs is that our approach does not depend on any specific tool or language so that developers can use their favorite tool to construct the DSML.

Antkiewicz et al. (2009) proposed a method for engineering new framework DSMLs by specializing existing approaches to domain analysis, software development, and quality evaluation of models and languages. Besides their work was focused on the reuse of MIF, it does not explain in details how the features of the framework are identified.

Amatriain and Arumi (2011) proposed an approach in which the construction of a DSML occurs in parallel to the development of its framework through iterative and incremental activities. Our approach differs from theirs, mainly, because our approach can be applied on frameworks that were previously developed by other people.

5 THE PROPOSED APPROACH

Figure 1 shows the two phases of the approach that promotes framework reuse supported by a DSML: Domain Engineering and Application Engineering. Subsections 5.1 and 5.2 present the processes of Domain Engineering and Application Engineering in details.

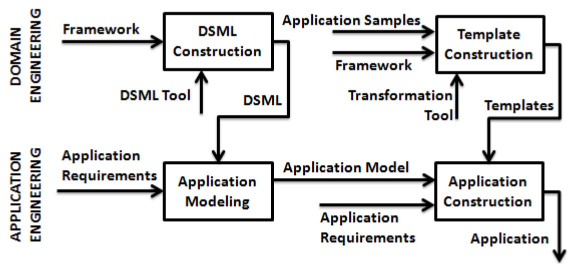

Figure 1: The proposed approach phases and their activities.

5.1 Domain Engineering

The Domain Engineering process aims at preparing an environment that facilitates framework reuse in the development of applications. In this process, DSML Construction and Template Construction activities are carried out.

DSML Construction activity starts with the creation of a metamodel which contains metaclasses representing the domain features of the framework and stores the information required by its hot spots. The domain features of the framework are identified by analyzing its documentation and its code through the following steps:

1. Identify a class that represents a domain feature. In an EAF, a domain feature is identified by a class which can be directly extended (white box) or instantiated (black box) by the applications. This class is a hot spot and define a functionality of the framework;
2. Identify the superclasses of the class identified in step 1;
3. Identify the classes targeted by association relationships coming from the classes identified in steps 1 and 2;
4. Analyze the hook methods of the class identified in step 1 in order to identify the information that is necessary to reuse it;
5. Include a metaclass in the metamodel to represent the class identified in step 1. Also include attributes and/or relationships in this metaclass to store the necessary information to reuse the class it represents;
6. Repeat the steps 1 to 5 until all domain features of the framework have been identified.

Class models are the first artifacts to be analyzed, because they provide a wide vision of the framework classes. A framework may also be documented by a Pattern Language (PL), which specifies the classes representing the domain features of the framework (Kirk et al., 2007). The framework code also needs to be analyzed to confirm the classes identified in the documentation, as models may be inconsistent with the current state of the code. Moreover, class models in analysis level contain only the main classes of the framework and the hot spots cannot be identified from these models.

Figure 2 shows part of the GRENJ framework DSML metamodel as an example. Feature, Attribute, Operation and Parameter metaclasses are domain-independent and provide common properties to domain-specific features. The applications which reuse GRENJ framework need to contain a class that extends Resource class. This class is a hot spot of the GRENJ Framework and represents a feature which is concerned with the goods or services involved in transactions. Therefore, Resource metaclass was included in the metamodel to represent it. As one of the information required by the hook methods of Resource class is the classes which represents the resource types in the applications, the types relationship was created to store this information.

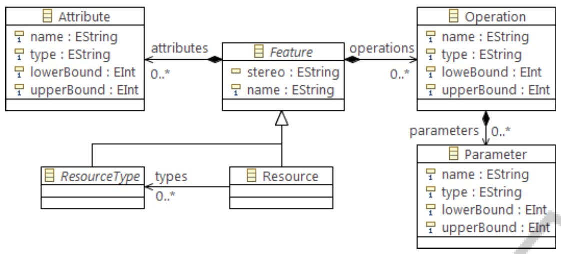

Figure 2: An example of a framework DSML metamodel.

The metamodel represents the abstract syntax of the DSML in which the domain features are defined. The concrete syntax, which defines the graphical notation of the DSML, must also be built so that developers can use it to model applications (Cuadrado and Molina, 2009). How the concrete syntax is built depends on the tool used to construct the DSML.

After DSML Construction activity, Template Construction activity begins with the use of a transformation language. The templates depend on both the DSML metamodel and the content that they may originate. Therefore, each template is usually linked to an existing metaclass in the DSML metamodel.

A template is formed by: fixed parts, consisting of parts of code that remains the same in all applications generated from the template; variable parts, corresponding to parts that change from one application to another. Usually, in XML-based template languages, such as Java Emitter Templates (JET) (Gronback, 2009), the fixed parts consist of texts and the variant parts consist of tags. The best tactic to create templates is to analyze the classes of applications that reuse the framework in order to identify its fixed and variable parts.

Taking the GRENJ Framework as an example, an application class that extends the Resource class should be similar to this:

```
public class Movie extends Resource {

   private int year;

   public Class[] typeClasses() {
      return new Class[]
        { Category.class, Genre.class };
   }
```

The JET template that was implemented to generate subclasses of Resource class is:

```
public class <c:get select="$feature/@name"/>
 extends Resource {
  <c:iterate
    select="$feature/attributes" var="attr">
  private <c:get select="$attr/@type"/>
    <c:get select="$attr/@name"/>;
  </c:iterate>

  public Class[] typeClasses() {
     return new Class[] {
     <c:iterate select="$feature/types"
       var="rtype" delimiter=", ">
     <c:get select="$rtype/@name"/>.class
     </c:iterate> };
  }
```

In the template for Resource subclasses, the fixed parts consist mainly of Java keywords, annotations, operators, punctuations and signatures of required methods, while its variant parts consist of tags indicating the attributes and references of the metaclasses presented in Figure 2.

In addition to the templates which are responsible for generating application code, other templates can indicate to the transformation tool to generate the database script, create the application package, include a copy of the framework, and so on. These templates are essential for the functioning of the generated application code.

Templates can be tested to check if they generate the correct code and if the generated code is correct. It can be done by creating automated unit tests for the classes that are generated by these templates. There can also be tests which verify if the code generated by a template is similar or even equal to the code of the classes from which this template was created. The technique applied to create these unit tests is beyond the scope of this work.

5.2 Application Engineering

The Application Engineering process comprehends Application Modeling and Application Construction activities with the use of the DSML and the templates constructed in the Domain Engineering phase, as shown in Figure 1.

In Application Modeling activity, the developer uses the DSML to create an application model based on the application requirements. Due its domain-specific aspect, DSML prevents the developers from creating relationships that are not defined in its metamodel and it provides a validation mechanism of the application models, which verifies if all obligatory classes were instantiated among other things.

In Application Construction activity, templates are implemented to generate code and other artifacts from the application model. As code generation is automated, it provides the efficiency in the application development. Although all code responsible to reuse the framework can be generated, it may be necessary to add code because a framework cannot predict all the particularities of the applications, whether using a DSML or not. Generally, the code which is added is responsible for behavioral aspects of the application such as the content of application-specific operations.

Application-specific attributes and methods should be inserted into the application model to avoid inconsistency between the model and the code. Moreover, methods which are added or modified need protection so that their contents can remain in case of the application code is generated again. Usually, transformation tools provide mechanisms that keep the alterations made by the developer, such as the implementation of the body of a method, even when the application code is regenerated.

6 USE CASE: GRENJ FRAMEWORK

We used the GRENJ Framework in order to exemplify the use of the approach of framework reuse supported by a DSML. The GRENJ Framework DSML was built by using the Graphical Modeling Framework (GMF) and Java Emitter Templates (JET), both available in the Eclipse IDE. Although we have chosen these tools, others could be used as well, such as xPand (Gronback, 2009) and Generic Modeling Environment (GME) (Institute for Software Integrated Systems, 2012). The proposed approach is intended to be tool-independent.

This section is divided as follows: Subsection 6.1 presents the GRENJ Framework Domain Engineering process; Subsection 6.2 presents the Car Rental Shop Application Engineering process; Subsection 6.3 presents a study with the development of applications in the domain of rental transactions reusing the GRENJ Framework with and without its DSML.

6.1 GRENJ Domain Engineering

In DSML Construction activity, the features were identified by analyzing models with the GRENJ framework classes that must be extended in the applications. The information required by the GRENJ Framework hot spots were identified by analyzing the code of the hook methods.

Figure 3 shows a GRENJ Framework model with the classes related to the resource feature. The hook methods that need to be overridden by an application

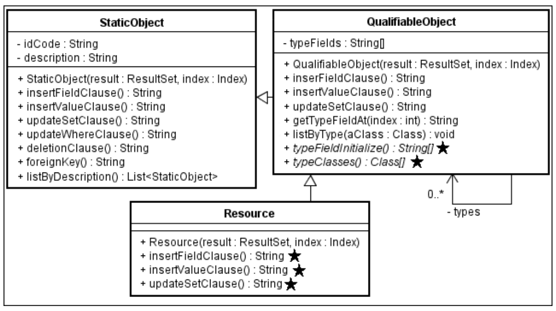
StaticObject
- idCode : String
- description : String
+ StaticObject(result : ResultSet, index : Index)
+ insertFieldClause() : String
+ insertValueClause() : String
+ updateSetClause() : String
+ updateWhereClause() : String
+ deletionClause() : String
+ foreignKey() : String
+ listByDescription() : List<StaticObject>
QualifiableObject
- typeFields : String[]
+ QualifiableObject(result : ResultSet, index : Index)
+ inserFieldClause() : String
+ insertValueClause() : String
+ updateSetClause() : String
+ getTypeFieldAt(index : int) : String
+ listByType(aClass : Class) : void
+ typeFieldInitialize() : String[] ★
+ typeClasses() : Class[] ★
0..*
- types
Resource
+ Resource(result : ResultSet, index : Index)
+ insertFieldClause() : String ★
+ insertValueClause() : String ★
+ updateSetClause() : String ★

Figure 3: Classes related to the Resource feature.

class which extend the Resource class are highlighted by stars.

Hook methods, such as the insertFieldClause method, were analyzed in order to identify the information required to the hot spots.

```
public String insertFieldClause() {

   StringBuilder strb = new StringBuilder(
      super.insertFieldClause() );
   String clause =
      quantification.insertFieldClause();
   if ( clause != null ) {
     strb.append( ", " + clause );
   }
   return strb.toString();

}
```

Table 1 shows a list with some of the features identified in the GRENJ Framework domain. These features are the minimum necessary to develop applications that deal with rental transactions reusing the framework GRENJ. Resource type is used to define classes that classify the resources. If the resource type has subtypes, it is a nested type. Otherwise, it is a simple type. Resource quantification identifies whether the resource is unique or can have copies. Resource rental transaction represents the leasing transaction of resources and destination party represents who orders the resource rental.

Figure 4 shows the GRENJ Framework DSML metamodel with the features described in Table 1. All metaclasses that represent domain features extend the Feature metaclass shown in Figure 2. The multiplicity of the relationships defines how many subclasses of the targeted feature can associate with a subclass of the source feature in the application models.

GRENJ Framework Template Construction activity was accomplished with use of JET transformation language (Gronback, 2009). This language allows the domain engineer to implement a set of templates and compile it as an Eclipse IDE plug-in that can access the application models and generate code and other artifacts.

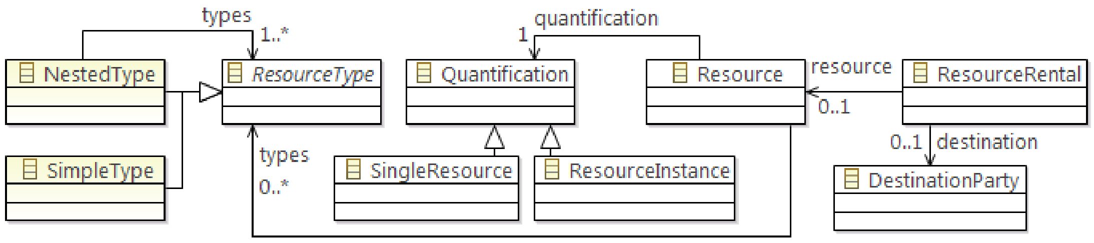

Figure 4: DSML metamodel which addresses the subdomain of resource rental of GRENJ framework.

Table 1: Some of the GRENJ Framework features.

Feature	Hot Spots	Instantiation
All features	-	The name of the class which represents the feature, its attributes, its operations and their parameters.
Resource	Resource	Resource types and quantification.
Resource type	SimpleType, NestedType	-
Resource quantification	SingleResource, ResourceInstance	Is the resource single or instantiable?
Resource rental transaction	ResourceRental	Resource and destination party.
Destination party	DestinationParty	-

A JET template is a XML-format file whose text represents the fixed part of the file that it originates and the variant part is formed by tags which refer to the information to be obtained from the models. Samples of classes of applications that reuse the GRENJ Framework were analyzed, such as it was described in Section 5.1, so that the fixed and the variant parts of the templates could be identified.

6.2 GRENJ Application Engineering

In order to exemplify the Application Engineering process, the Car Rental Shop application were developed by using the DSML and the templates that were created in the Domain Engineering process described in Section 6.1.

The GRENJ Framework DSML and templates originate a set of plug-ins for the Eclipse IDE. The integration of these plug-ins in Eclipse IDE result in a CASE tool that supports the developing of applications with the reuse of the GRENJ Framework.

In Car Rental Shop Application Modeling activity,

Table 2: Car Rental Shop Application requirements.

#	Description
1	The shop rents cars for customers registered in the application. The attributes of a car are: code, description and number of doors.
2	There can be one or more vehicles for each car registered. A vehicle has license plate, year, color and it can be available or not.
3	A car is classified by a category that determines its rental price.
4	Code, name, address and phone number are the data that need to be registered about the customers.
5	The information to register a car rental is: number, date, expected returning date, real returning date, customer, vehicle and total price. Total price is calculated on the number of days between the rental date and the real returning date of the vehicle multiplied by the rental price determined by the car category of the vehicle.

the developer creates a model selecting the domain features of the GRENJ framework based on the application requirements listed in Table 2. This is similar to create a class model in analysis level.

Figure 5 presents the model of the Car Rental Shop Application which was created with the use of the GRENJ Framework DSML based on the requirements of Table 2. In this model, the classes of the GRENJ framework that are being reused by the Car Rental Shop application are identified by the stereotypes and the names of the subclasses are in bold.

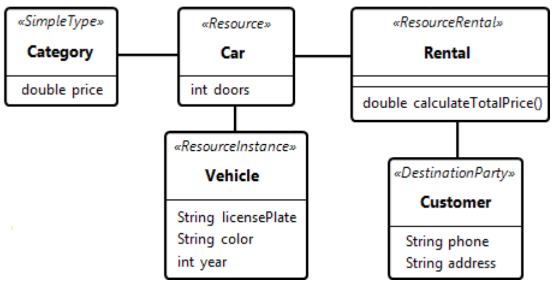

Figure 5: Car Rental Shop Application model created with the GRENJ Framework DSML.

Figure 6 shows a class model of the Car Rental

Shop Application, in which the classes highlighted in grey are from the GRENJ Framework. This class model is equivalent to the model created with the GRENJ Framework DSML shown in Figure 6. Most of the attributes specified in the requirements in Table 2 were not added to the classes of the application model shown in Figure 5 because these attributes are inherited from the GRENJ Framework classes and makes no sense to repeat them on the models. Our approach assumes that the application developer knows the framework.

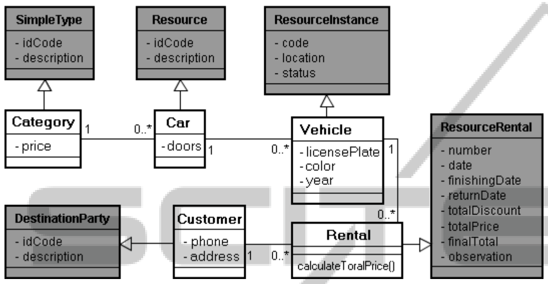

Figure 6: Car Rental Shop Application class model.

In Application Construction activity, the Car Rental Shop application model was integrated with the templates. Thus the code and all the file structure of the application was generated. The code of the framework was also copied to the application code. A combination of the application model presented in Figure 5 with the JET template for Resource subclasses, presented in Section 5.1, generates the code of the Car class:

```
public class Car extends Resource {

   private int doors;

   public Class[] typeClasses() {
      return new Class[] { Category.class };
   }
```

The total price of the rental in this application is based on the price established by the category multiplied by the number of days the customer rented the car. As the framework cannot predict this calculation, it needs to be manually implemented.

The calculateTotalPrice method was generated in the Rental class with an empty body by the transformation tool. It was modified in order to calculate the total price in the instances of the Rental class. The method was marked as “generated NOT” to avoid its code to be erased in case of the application code is generated again.

The code of the calculateTotalPrice method after it was manually modified is:

```
/**
 * @generated NOT
 */
public double calculateTotalPrice() {

  Category category = (Category)
    getResource().getTypes().get(0);
  int numDays =
     Period.numberOfDays(
   getDate(), getReturningDate());
  return category.getPrice() * numDays;

}
```

6.3 GRENJ DSML Evaluation

In order to evaluate the proposed approach, a study was performed to compare the time spent on the development of applications with the reuse of the GRENJ Framework. Two approaches were applied in this study: 1) an approach in which application-specific code was manually implemented; and 2) the approach supported by DSML, in which the application-specific code was generated with the use of the GRENJ Framework DSML and templates, such as it was described in the Section 6.1.

26 Computer Science undergraduate students were trained to develop applications reusing the GRENJ Framework through both approaches in order to get knowledge about the framework and the tools used in each approach. The students that participated in this study had previous knowledge about Java programming and UML modeling.

In the study, the students should develop two applications involving resource rental transactions: the first one to check books out from a library and the second one for checking guests in a hotel. These applications have the same level of complexity of the Car Rental Shop application (Section 6.2). The students were divided into two teams, T1 and T2, with 13 students each one. The teams should carry out the following tasks:

- T1 development of the application for the hotel with the reuse of the GRENJ Framework through manual programming and development of the application for the library with the reuse of the GRENJ Framework supported by its DSML;
- T2 development of the application for the library with the reuse of the GRENJ Framework through manual programming and development of the application for the hotel with the reuse of the GRENJ Framework supported by its DSML.

To perform the study, the students received a document containing the requirements of the applications they had to develop. Each student also received a class model in analysis level of the application he/she had to develop by manual programming. The applications were developed by using the Eclipse IDE, whose workspace were configured with one project for each application with set of JUnit tests. All the computers the students used had the same configuration of software and hardware.

Each student has developed the applications referent to his/her team individually and measured the time spent in the development of each application. After finishing the development of an application, the students had to pause the chronometer and run its JUnit tests to verify whether or not the application worked correctly. In case of test fail, the students had to write in a form a description of the fail and activate the chronometer again to correct the defects they found. The application was considered finished only after the tests had shown no fail.

To evaluate the approaches, the study worked with two hypotheses:

- Null Hypothesis, H0 - the DSML does not make the process of application development with the reuse of frameworks easier and faster; and
- Alternative Hypothesis, H1 - the DSML make the process of application development with the reuse of frameworks easier and faster.

The applications developed by the students had approximately 18,400 lines of code, from which 18,000 correspond to the GRENJ Framework code and only 400 belong to the code generated or implemented during the application development. Due to idiomatic patterns imposed by the GRENJ Framework, the codes of the applications developed by the students are similar, whether they were manually implemented or generated. Moreover, the number of lines of the code generated or implemented during the application development was small due to the reuse of code provided by the GRENJ Framework.

Table 3 presents the averages (AVG) and the standard deviation (SD) of the time spent by students of each team (T1 and T2) in the development of the applications. It was possible to observe that there was an average reduction of about 89.3% for Library Application and 87.5% for Hotel Application. It results in a general reduction of 88.5% in the time spent on developing applications with reuse of the GRENJ Framework through its DSML when compared to time spent on the manual implementation approach. The times in Table 3 are measured in minutes (‘) and seconds (“) and consider the development of the applications and the successful execution of the application tests.

This study shown that the development of applications with reuse of frameworks spends less

Table 3: Average of time spent on the development of the applications.

App.	Manual Programming			DSML		
	Team	AVG	SD	Team	AVG	SD
Library	T2	60’31”	18’59”	T1	6’28”	1’23”
Hotel	T1	44’14”	14’21”	T2	5’32”	0’38”
General Average	52’53”			6’00”		

time when performed with the use of a DSML than when it is done through manual programming. However, a long time is spent in the execution of the Domain Engineering process in order to construct the DSML and the templates of the framework. Therefore, the approach supported by DSML represents an advantage only when several applications are developed, making the sum of the time spent on Domain Engineering phase and the time spent on the engineering of the applications lesser than the time spent developing these same applications manually. For example, the Domain Engineering process of the GRENJ Framework was accomplished in, approximately, 5 hours. The sum of 5 hours and the times shown in Table 3 demonstrates that, considering the GRENJ Framework, the approach supported by DSML becomes worthwhile when 6 or more applications are developed.

Besides the time spent in the development of the applications, the problems shown by the application tests were analyzed in order to identify when the students made more mistakes: with or without the use of the DSML. This analysis were based on the information the students wrote in the form after they had run the tests.

Table 4: The number of times which some problems was found in the applications developed by the students.

Problem	Number of times it happened	
	Without DSML	With DSML
A hook method was missing.	8	0
A hook method was not implemented correctly.	13	0
Hot spots that require the same information were inconsistent.	6	0
A framework class was misused.	2	2
Total	29	2

Table 4 shows the number of times which some problems was found in the applications developed by all students in the study. These problems does not include compilation errors, because the applications were tested only when they could be run. The students made more mistakes while reusing the framework without the use of the DSML. This occurred due to the fact that, in the approach through manual programming, there is a higher probability of defect insertions in the code, e.g., mistyping, misuse of the controls, lack of methods required by the framework and so on. In some cases, multiple hot spots of the GRENJ framework require the same information, usually, when many classes has the same hook method. Some students implemented these hook methods returning different values, resulting in inconsistencies. It cannot occur with the DSML because it requires each information only once. However, the correct selection of a framework class to implement a requirement of the application relies on the knowledge the developer has about the framework. This statement could be confirmed, when some students misused the same framework class both in the manual programming approach and in the DSML approach.

Some restrictions and threats to the validity of this study should be taken into consideration:

- In the development of the applications with the DSML, all students used the DSML and the templates created in the Domain Engineering process described in Section 6.1. In this manner, there would be no difference among students on the way that the applications were modeled and the codes were generated.
- It can be argued that the examples developed by the students are simple. However, the applications which were described access most types of GRENJ Framework hot spots. If two or more hot spots request similar information, for example, class names, then these hot spots are the same type.
- The addition of functionality not predicted by the GRENJ Framework was not verified in this experiment because this activity can be done only by a manual programming approach.

7 CONCLUSIONS

In this paper, an approach to framework reuse supported by Domain-Specific Modeling has been presented. This approach improves the efficiency and the quality on the development of applications in a specific domain, since it generates code from application models and prevents the developer from incorrectly accessing the hot spots of the framework.

The proposed approach requires effort in Domain Engineering phase to build the environment for the development of applications that reuse the framework. However, in Application Engineering phase, the total effort comes down to the effort for modeling the application as well as the effort for the refinement of the code, if it occurs. The time to generate the application code is irrelevant compared to the total effort, because it is generated.

The quality of the DSML depends on the identification of the domain features of the framework in the same way as the quality of an application depends on the identification of its classes. The proposed approach mainly aims to provide a sequence of steps which the developer should follow in order to identify the domain features of the framework, model them and create templates. The use of tools only supports these activities, so that the developer can choose his/her favorite tools for each activity.

The DSML does not eliminate the necessity of the developer to know the framework. He/She has to know the hot spots and the attributes and the operations provided by them in order to add to the classes of the applications only properties that are not provided by the framework or even modify the generated code correctly. The DSML is useful to liberate the developer from implementing the code to reuse the framework, saving time and avoiding some mistakes. Thus, the developer can dedicate more effort to implement the functionality not provided by the framework.

Our work focused on developing DSMLs for Enterprise Application Frameworks, whose domain features are related to social and economic aspects. Although we have not applied our approach to develop DSMLs for System Infrastructure Frameworks or Middleware Integration Frameworks, there is no theoretical impediment to do it. The most important thing is identify the features from the hot spots and the information necessary to instantiate the framework.

In future works, we intend to extend the proposed approach to include the development of the framework from domain features models. We also intend to include in the proposed approach mechanisms to generate behavioral aspects, such as the code of application-specific operations.

ACKNOWLEDGEMENTS

We would like to thank the students of Computer Science Course of UFSCar for participating in our study. We also thank CAPES for financial support.

REFERENCES

Abi-Antoun, M. (2007). Making Frameworks Work: a Project Retrospective. In *Companion to the 22nd ACM SIGPLAN conference on Object-Oriented Programming Systems and Applications*, OOPSLA '07, pages 1004–1018, New York, NY, USA. ACM.

Amatriain, X. and Arumi, P. (2011). Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain. *Software Engineering, IEEE Transactions on*, 37(4):544–558.

Antkiewicz, M., Czarnecki, K., and Stephan, M. (2009). Engineering of Framework-Specific Modeling Languages. *Software Engineering, IEEE Transactions on*, 35(6):795–824.

Braga, R. and Masiero, P. (2003). Building a Wizard for Framework Instantiation Based on a Pattern Language. In Konstantas, D., Léonard, M., Pigneur, Y., and Patel, S., editors, *Object-Oriented Information Systems*, volume 2817 of *Lecture Notes in Computer Science*, pages 95–106. Springer Berlin / Heidelberg.

Brugali, D. and Sycara, K. (2000). Frameworks and Pattern Languages: an Intriguing Relationship. *ACM Comput. Surv.*, 32(1).

Cuadrado, J. and Molina, J. (2009). A Model-Based Approach to Families of Embedded Domain-Specific Languages. *Software Engineering, IEEE Transactions on*, 35(6):825–840.

Durelli, V. H. S., Borges, S. S., Penteado, R. A. D., and Viana, M. C. (2010). An Iterative Reengineering Process Applying Test-Driven Development and Reverse Engineering Patterns. *INFOCOMP Journal of Computer Science*, Special Edition(1):1902–1929.

Fayad, M. and Schmidt, D. C. (1997). Object-Oriented Application Frameworks. *Communications of ACM*, 40(10):32–38.

France, R. and Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering, 2007. FOSE '07*, pages 37–54.

Gronback, R. C. (2009). *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley.

Institute for Software Integrated Systems (2012). Generic Modeling Environment.

Johnson, R. E. (1997). Frameworks = (Components + Patterns). *Communications of ACM*, 40(10):39–42.

Kirk, D., Roper, M., and Wood, M. (2007). Identifying and Addressing Problems in Object-Oriented Framework Reuse. *Empirical Software Engineering*, 12(3):243–274.

Oliveira, T. C., Alencar, P. S. C., Lucena, C. J. P. D., and Cowan, D. D. (2007). RDL: A language for framework instantiation representation. *Journal of Systems and Software*, 80(11):1902–1929.

Turki, S., Soriano, T., and Sghaier, A. (2004). An MDA Application for a Virtual Reality Environment. In *Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on*, volume 2, pages 807–812.

Weiss, D. M. and Lai, C. T. R. (1999). *Software Product Line Engineering: A Family-Based Software Development Process*. Addison-Wesley.