Integrating SPL and MDD to Improve the Development of Student Information Systems

A. Cunha[1][a], S. Fernandes[1][b] and A. P. Magalhães[1,2][c]
[1]*Post Graduated Program in Computing and Systems, Salvador University, Salvador, Brazil*
[2]*State University of Bahia, Department of Exact Sciences and Earth, Salvador, Brazil*

Keywords: Student Information System, Evaluation Criteria, Software Product Lines, Model Driven Development, Domain Specific Language.

Abstract: Software development has become increasingly complex in recent years, with the growing multiplicity of development platforms, the integration between components in heterogeneous environments and platforms, and frequent changes in requirements. Academic systems usually integrate various subsystems, such as student enrolment and class planning which can change almost every semester. To address these issues, different development approaches can be used, for example, Model-Driven Development (MDD) and Software Product Lines (SPL). This paper presents an approach that integrates MDD with SPL for the development of evaluation criteria in a family of educational systems. The solution comprises a modeling language, called DSCHOLAR, for creating the models; and a transformation for C# code generation. This article details the transformation responsible for generating the code of evaluation criteria components for the student evaluations according to different universities scenarios. The transformation was validated using proofs of concepts in which evaluation criteria from three public and private universities were modeled using DSCHOLAR and subsequently converted into C# code.

1 INTRODUCTION

Management software has been widely used to improve institutional processes, and improve decision making, among other purposes. In the educational context, Higher Education Institutions (HEI) use software, either for their core business – to support teaching-learning activities – or in their support activities, such as the management of academic, administrative and financial functions.

In the field of educational systems, it is common for universities to have specificities that lead to significant differences in the information systems that support their processes. In certain situations, business processes or rules in a given institution evolve significantly and frequently. This implies that the implementation of these information system might change significantly over time. The use of a traditional software development process in such situations may be inefficient, because for each specificity of greater complexity, it is necessary to code correspondingly specific software components "manually". Consequently, whenever specificity changes, the corresponding code needs to be updated. One of these specificities is student evaluation criteria, which may vary from one institution to another, from one semester to the next, or even be free enough to be defined by each teacher. Therefore, a more interesting approach is to define some form of representation of the evaluation criteria (for example, a representation at a higher level of abstraction, such as a graphic model), and to create mechanisms for automatic transformation of these models into code.

In order to contribute to the development of educational systems two approaches can be integrated: (i) Software Products Line (SPL), designed to name software families that share common characteristics in which each family member has specific variations of these characteristics (Clements and Northrop , 2001) and

[a] https://orcid.org/0000-0001-5335-1566
[b] https://orcid.org/0000-0002-1118-5560
[c] https://orcid.org/0000-0002-8608-4553

Cunha, A., Fernandes, S. and Magalhães, A.
Integrating SPL and MDD to Improve the Development of Student Information Systems.
DOI: 10.5220/0007711201970204
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 197-204
ISBN: 978-989-758-372-8

(ii) Model Driven Development (MDD), an approach that uses models as the main development artifacts, transforming them (semi) automatically into application source code (Brambilla, et al., 2017). These approaches are strongly encouraged for this context and might contribute to improve productivity: define a SPL comprising the characteristic of the domain and use MDD to specify these characteristics and automatically generate code.

The SPL and MDD approaches have been successfully used by several authors, such as Zhu (Zhu, 2014), who proposed the Engine Cooperative Game Modeling (ECGM) framework to model games and generate source code, showing that it can significantly improve the development process. Sottet, Vagner and García Frey (Sottet, et al., 2015) proposed using the two approaches to improve user interaction with the interface, arguing that the design of this interaction is made difficult by taking into account aspects such as different devices and users and diverse interaction environments. Zarrin and Baumeister (Zarrin and Baumeister, 2018) proposed the construction of a framework to better support semantics in the use of both approaches. Although the two approaches (SPL, MDD) are widely used together, no reference was found to their application in the context of student evaluation criteria in educational systems, nor even in the broader educational domain.

This work proposes a solution that integrates the SPL and MDD approaches for the development of Student Information Systems (SIS). The purpose is to create a product line of a SIS that can be customized through the MDD at its points of variation. Diverse products can be instantiated from this SPL to meet the specific needs of educational institutions.

This paper focuses on a specific point of variability of SIS: the student evaluation criteria. The approach presented here will be further extended to other SIS variability points. For this specific point of variability we developed, a modeling language, called DSCHOLAR, to be used in characteristics modelling by domain specialists, and a transformation program to generate component code from DSCHOLAR models. This article presents an overview of the solution and of the DSCHOLAR (Cunha, et al., 2018), and details the following contribution: the transformation and its validation.

We defined DSCHOLAR due to the higher expressivity of Domain Specific Languages (DSLs) when compared to General Purpose Languages, such as UML (Booch, et al., 2006), making them more suitable for the users, i.e. domain experts, such us academic managers or teachers, not software engineering professionals.

The point of variability focus here, *student evaluation criteria*, was initially selected because of its relevance and non-triviality: relevance because the evaluation criteria may vary significantly in different institutions. Indeed, this variation may occur in different areas or departments of the same institution. In the limit, different teachers might adopt different evaluation criteria; non-triviality because the specification and implementation of an evaluation criteria can be very distinct and expressed through non-trivial rules. This particularity – the possibility that each individual user (typically not an software development expert) may need a specific configuration of the evaluation criteria – is not usual in information systems in general, but a real possibility in the SIS.

The method used to develop this work started with the specification of the characteristics of the SIS product line as well as the classification of them as variable or non-variable. Each variable characteristic was analysed so as to characterize the frequency of changes in their specifications. We use MDD to develop components for variable characteristics that may change considerably over time (e.g. the student evaluation criteria). In these cases, a modeling language and a transformation program must be defined, to enable the modeling of the characteristic and the automatic generation of its code. Finally, both the language and the transformation were validated through proofs of concept.

The rest of the paper is organized as follows: section 2 introduces the concepts necessary for a better understanding of the work and section 3 presents the related works that integrate SPL and MDD. Section 4 presents an overview of the proposed SPL and DSCHOLAR as well as the detailing of the transformation and its evaluation. Finally, section 5 presents the conclusions and future work.

2 BACKGROUND

Several approaches have been proposed to meet the increasing demand and complexity of software. These approaches aim, among other things, to increase the productivity of the development process and software quality. Among them, SPL and MDD stand out in a context such as the one we have: an information system that needs extensive and non-trivial customization for each specific customer.

SPL can be defined as a set of software products with characteristics sufficiently similar to share a common infrastructure and the parameterization of differences among the products (Almeida, 2009). The characteristics, usually called features, of a SPL are classified as mandatory or optional and can be used to specify variabilities and commonalities among software products, as well as to guide the structure, reuse and variations between products in their life cycle. Therefore, the development of software using SPL is based on a set of core assets, defined according to the commonalities and variabilities of a specific domain, used to derive new products of the line.

MDD is a software development approach that uses models as the main developmental artifacts. It changes the focus of the development from code writing to model development. In MDD, high-level abstraction models are automatically / semi automatically transformed through a chain of transformations, into less abstract models and, typically, in the end, into the source code of the application (Brambilla, et al., 2017).

The MDD approach contains two essential elements: the models, artifacts representing the software at the various levels of abstraction; and transformations, which convert models into other models or code (Brambilla, et al., 2017). Models must be formally written using well-defined modeling language syntax and semantics. Transformations are responsible for mapping the models across the various levels of abstraction throughout development (Stahl, et al., 2006).

One of the most widespread techniques in the generation of source code by transformation of models is that of templates. A template is a standardized text file, instrumented mainly with code expansion and selection constructions, and is responsible for performing parameter queries on an entry: a textual file or templates. The information contained in the templates and entries are processed by the transformation, resulting in the source code (Sendall and Kozaczynski, 2003).

3 RELATED WORKS

The development of new products in a SPL has been frequently integrated to the use of MDD, usually in the development of products in telecommunications, banking, embedded systems and automotive sectors (Tolvanen and Kelly, 2016).

Gonzalez-Huerta et al. (González-Huerta, et al., 2014), in a case study in the automotive sector, present a set of guidelines for the development of architectural transformations on a model that represents different points of view of a system, allowing the explicit representation of relationships between architectural patterns and quality attributes.

Lahiani and Bennouar (Lahiani and Bennouar, 2018) performed a case study in the e-Health area to illustrate the transformation process for product generation of an SPL. They used modeling languages to represent the architecture and the application. Then they modeled points of variability according to the needs of some users who used the application and automatically transformed those models into products with the requirements requested by the users.

Sochos et al. (Sochos, et al., 2006) propose the FArM (Feature-Architecture Mapping) method, which provides a stronger mapping from software characteristics into software design. It is based on a series of transformations in the initial model of product line features. During the execution of the transformations, architectural components are derived, encapsulating the business logic of each transformed feature and the interfaces directly reflect the interactions of the feature.

In the same direction of the works presented before, our work defines a SPL and uses MDD to model the variability points of this SPL in order to generate code. However, we apply this strategy to support the development of systems in the education domain, where, to the best of our knowledge, it has not been used before.

4 SPL FOR EDUCATIONAL SYSTEMS

This section introduces the SPL solution for the Student Information Systems (SIS) proposed to enable the customization of systems for different educational institutions (Figure 1).

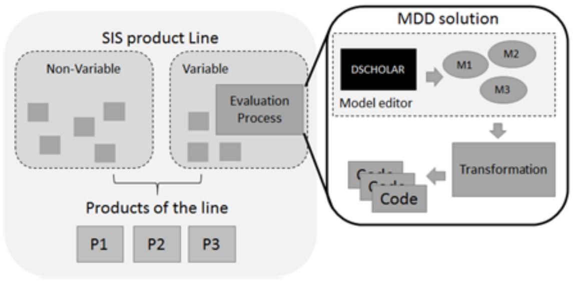

Figure 1: Overview of SIS product line using MDD solution.

The left side of Figure 1 shows an outline of the high-level design of the components of the SIS product line. Some of the components are classified as non-variables, while others are classified as variables. This classification was generated through the analysis of the features (Czarnecki and Eisenecker, 2000) of the product family. Features that do not vary across all members of the product family are classified as non-variable, while those that will need to be customized for each individual product are classified as variables. New products are generated by reusing and/or customizing product family components. Variable component customization can be done using the MDD approach. This is the case of the *Evaluation Criteria* component.

The right-hand side of Figure 1 details the MDD solution provided to enhance the development of the evaluation criteria component according to the specific needs of each SIS. In order to enable the modeling of the specific component, a modeling language was defined for the evaluation domain, DSCHOLAR (presented in section 4.1). Thus, from this language, several evaluation criteria can be defined (in the figure we have M1, M2, M3) and used to automatically generate the application code in the C# language through a transformation (detailed in section 4.2).

This project was implemented in Microsoft DSL Tools, a set of plugins hosted by Microsoft Visual Studio (Warren, 2019). We adopted this technology due to the expertise of the team in using it and because it provides easy integration between the code generated by the transformations and the code manually written in Microsoft Visual Studio.

4.1 DSCHOLAR Modeling Language

DSCHOLAR is a domain-specific language designed to model the Student Evaluation Criteria in the SIS SPL. In (Cunha, et al., 2018), the abstract syntax of DSCHOLAR is presented and discussed. DSCHOLAR encapsulates the necessary knowledge to enable domain specialists, not necessarily software developers, define new evaluation components according to their needs.

In DSL DSCHOLAR, the elements specified for the modeling of the evaluation criteria of universities are: (i) *Entity*, which represents the educational institution and (ii) *Evaluation*, which represents the student evaluations of a certain educational institution.The concept *Entity* is a generic concept that may represent an institution, a course or even a discipline.

An *Entity* has attributes, such as *entityName*, *meanGrade*, *lowestGrade* and *finalMeanGrade*, which indicate how the entity works.

Evaluation is a general concept, specialized in four other concepts: *MandatoryEvaluation*, for evaluations that must be applied; *OptionalEvaluation*, for those that are part of the evaluation process but that may be applied at teachers discretion; *VariableEvaluation*, when teachers may freely define a number of evaluations not predefined by the evaluation process; and *ExtraEvaluation*, which is a special evaluation whose grade is to be added to that of another evaluation grade. All Evaluations have the attributes: (i) *name*, referring to the name that the evaluation will have; (ii) *weight*, referring to the weight of each evaluation within the criteria of a university; (iii) *description*, which describes each evaluation; and (iv) *sequence*, representing the order in which the evaluations will be performed.

Figure 2 illustrates an example of a model created with DSCHOLAR to represent the evaluation criteria of a private higher education institution.

In the model depicted in Figure 2, there are three evaluations, namely *Evaluation 1*, *Evaluation 2* and *Final Evaluation*, and their respective relative *weights* (30, 40, 30). The round-cornered rectangle of the first two evaluations is the concrete syntax used to specify that all of them are mandatory specifications. *Final Evaluation* is an optional evaluation, which is depicted by a conventional rectangle in a different color. Regarding the number of optional evaluations in a model, there are two different modeling options. If the quantity of optional evaluations is already defined, each one of these evaluations is represented by an instance of a specific modeling element in the respective model. Otherwise, each teacher can define the number of optional evaluations as an attribute of a variable evaluation element, so that the model will have only one instance of that evaluation and an attribute quantity is used to define the upper boundary of this quantity.

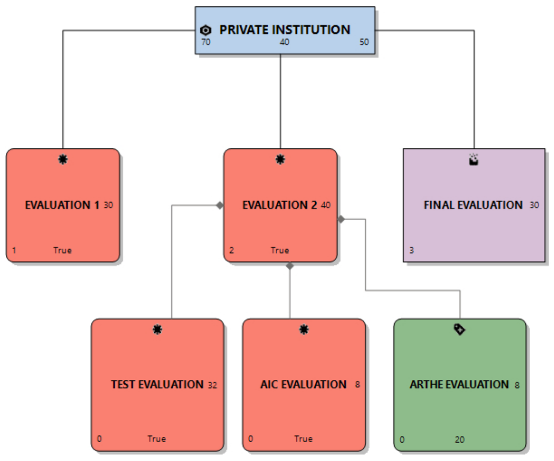

Figure 2: Example of model using DSCHOLAR that represents the evaluation criteria of a private institution.

There are different kinds of connections among the represented elements in the model. Simple connections are represented between an *Entity* and an *Evaluation* (e.g. the *Private Institution* and the *Evaluation 1* in Figure 2). Composite connections are used to link evaluations, i.e. when an evaluation is composed of others (e.g. *Evaluation 2* is composed by *Test* and *AIC* in Figure 2).

In summary, for the example shown in Figure 2, there is an Institution, named *Private Institution* (first square at the top of Figure 2), which has the mean grade 70 (attribute *meanGrade*), the lowest grade 40 (attribute *lowestGrade*), and final grade 50 (attribute *finalGrade*). The evaluation criteria is given respectively by an evaluation (*Evaluation 1*) of weight 30, a second evaluation (*Evaluation 2*) of weight 40 which is composed of three other evaluations, one weighing 32 (*Test Evaluation*), one weighing 8 (*AIC Evaluation*) and one that generates extra point of weight 8 (*Arhte Evaluation*). After completing these 4 evaluations (*Evaluation 1*, *Test*, *AIC* and *Arhte*), students who reach 70 points (*meanGrade* of the entity) will pass without a final evaluation (*Final Evaluation*); those who score more than 40 (*lowestGrade* of the entity) and less than 70 (*meanGrade* of the entity) will have to do *Final Evaluation* weight 30; and, after that, should pass if their weighted average is equal to or greater than 50 (*finalMeanGrade* of the entity). Students who score less than 40 in all evaluations before Final Evaluation and less than 50 of the total after completing *Final Evaluation* will fail.

4.2 Code Generator for Evaluation Criteria in Educational Systems

This section presents the transformation, named *dscholar2Code*, developed to support code generation of the component Evaluation Criteria of our SIS product line. The transformation receives as input a model specifying the education criteria of a specific institution, i.e. a model developed according to DSCHOLAR, and generates as output the correspondent code in C# language.

The transformation *dscholar2Code* was specified in five stages. First, *Product Design* stage, the architecture of the component that will be generated is defined. Then, stage *Implementation Strategy Definition*, defines which part of the component code will be static, i.e. manually implemented, and which one will be dynamic, automatically generated by the transformation. Based on this, the transformation rules are specified (in stage 4), implemented (stage 5) and finally tested (*Transformation Validation* stage).

Following the stages presented above, the first stage concerns about architecture definition and the MVC (Model-View-Controller) pattern (Buschmann, et al., 1996) was used. This is an architectural software pattern that structures the application in three layers. For the component *Evaluation Criteria*, the elements of MVC layers are predefined templates specified according to the information provided by the DSCHOLAR metamodel. Figure 3, for example presents the classes specified for the Model layer.

The second stage, *Implementation Strategy Definition*, deals with the identification, in the class structure modeled by the previous step, of which elements of each class are variable and which elements are static. The variable elements must be dynamically generated, and the statics are generated manually. Based on this it is determined, for each class, which code snippets should be generated automatically, and which snippets should be fixed.

The language used to implement the transformation code is based on templates, i.e. a predefined template contains the code parts which are static as well as the specific points where the dynamic code must be inserted when generated. For the component Evaluation Criteria, the templates defined for the View layer are dynamically customized using the input model data, i.e. the DSCHOLAR model of a specific institution. The Control layer is generated manually as it does not vary according to the evaluation criteria. The Model layer comprises dynamically and statically generated code. The code defined to be statically generated was the structural part of the class *Entity* and the declaration of its attributes, such as *entityName*, *meanGrade*, *lowestGrade and finalMeanGrade*. The part defined to be dynamically generated was the methods *loadEntity()* and *generateEvaluationsList()* because they have information that varies according to the student evaluation criteria of the input model.

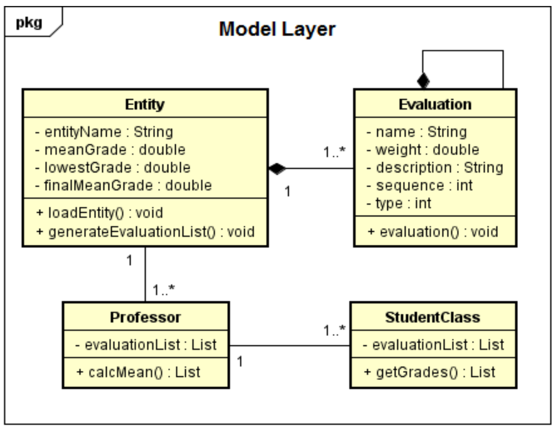

Figure 3: Model layer of the evaluation criteria component.

At stage 3, transformation rules must be specified. They map the elements of DSCHOLAR and the bits of code that are dynamically processed by the transformation *dscholar2Code*. The language used to implement the transformation reads a model and manipulates its elements through tags in order to dynamically generate code. Therefore, in order to specify transformation rules, we map each relevant element of DSCHOLAR metamodel into a tag in the transformation code.

At the *Transformation Implementation* stage, the transformation was coded. Figure 4 shows part of the code of the class *Entity*, which was manually implemented.

```
public class Entity
{
  public String entityName;
  public Double meanGrade;
  public Double lowestGrade;
  public Double finalMeanGrade;
  public  List<Evaluation>  evaluations  =  new
List<Evaluation>();
}
```

Figure 4: Part of the code of the class *Entity*.

For the elements to be dynamically instantiated, a loop-like programming structure was used, which reads a model (as the one shown in Figure 1) in search of instances of the *Entity* and *Evaluations* elements as well as their attributes in the input model. Figure 5 and 7 present respectively the implementation of the methods *loadEntity()* and *generateEvaluationsList ()*.

The method *loadEntity()* assigns a value to each attribute of each object of the *Entity* class existing in the input model. The code in Figure 5 illustrates the search for an instance of type *Entity* in a DSCHOLAR model, and the storing of its attributes *entityName, meanGrade, lowestGrade* and *finalMeanGrade* in the instance of the C# *Entity* class being generated.

```
Public void loadEntity(){
<# foreach(Entity ent in this.x.Entity){ #>
this.entityName = "<#= ent.name #>";
this.meanGrade = <#= ent.meanGrade #>;
this.lowestGrade = <#= ent.lowestGrade #>;
this.finalMeanGrade = <#= ent.finalMeanGrade #>;
<# } #>
}
```

Figure 5: Part of the code of the method *loadEntity()*.

```
Public void generateEvaluationList(){
<# foreach (Evaluation av in this.X.evaluations){ #>
   <# if (av.Targets.Count == 0){#>
     <# if (av.GetType().GetProperty("mandatory") !=
  null){#>
         this.evaluations.Add(new    Evaluation("<#=
  av.name  #>",<#=  av.weight  #>,"<#=  av.description
  #>",<#= av.sequence #>, 1 ));
<# } } }#>
}
```

Figure 6: Code of the method *generateEvaluationList()*.

The code of the method *generateEvaluationList()* (Figure 6) is dynamically generated based on the list of *evaluations* that are part of each entity in the input model. As a result, it will fulfill a list in the C# code (named *evaluations*) which contains all the corresponding evaluations of the input model. When this generated code is executed it will scroll the list instantiating each one of the evaluations.

Figure 7 presents the code generated by the transformation *dscholar2Code* for the class *Entity* considering the input model shown in Figure 2.

```
public class Entity
{
public String entityName;
public Double MeanGrade;
public Double LowestGrade;
public Double FinalMeanGrader;
public  List<Evaluation>  Evaluation  =  new  List<
  Evaluation >();
//Loading Entity
Public void loadEntity(){
this.entityName = "Entity";
this.MeanGrade = 70;
this.LowestGrade = 40;
this.FinalMeanGrade = 50;
}
//Type 1 = mandatory, 2 = variable, 3 = extra, 4 =
  FINAL, TIPO 5 = optional
Public void generateEvaluationList(){
this.Evaluation.Add(new          Evaluation("AV1
 ",30,"description 1",1, 1));
this.Evaluation.Add(new
  Evaluation("Test",32,"Description 2",2, 1));
this.Evaluation.Add(new      Evaluation("AIC",8,"
  DESCRIÇÃO 3",3, 1));
```

Figure 7: Code generated for class *Entity*.

Once the transformation is implemented it is tested (Validation Transformation stage in our method). This is described in section 4.3.

4.3 Validation of the Code Generator

The transformation *dscholar2Code* was validated using a proof of concept, to evaluate the coherence of the generated code in relation to the model input model specified using DSCHOLAR. This goal was defined according to Goal Question Metric (GQM) template [11] in Figure 8.

Analyze the component code generated as output of the transformation dscholar2Code
With the purpose of evaluating its correctness
Regarding its correspondence to the evaluation criteria specified in the input model
In the perspective of the software developer
In the context of models developed with DSCHOLAR modeling language

Figure 8: Goal of the transformation validation.

To guide the evaluation, the following research questions (RQ) were defined: RQ1: are all the evaluations specified on the model present in the component code? RQ2: are the evaluation criteria defined in the input model included in the component code? RQ3: are the mean grades correctly calculated?

The validation was performed using three different input models, i.e. models of three different universities. These models were specified in the case study carried out to validate the DSCHOLAR modeling language.

In order to evaluate the first question we ran the application, i.e. the generated component, and observed if the interface comprises all the evaluations as specified in the model. The metric used was the type of evaluations specified in the input model (EM- evaluation of model) and the type of evaluations presented in the component (EC – evaluation of code). For the second question, we compare the criteria defined in the input model (CM – criteria of model) with the criteria of the generated code (CC – criteria of code). Finally, to evaluate the third question we performed a set of test cases in order to observe the resulting mean grades.

The validation was performed separately for each university, i.e. for each input model. First, we run the transformation using the input model. Then, the codes generated were used to derive a different product of the educational SPL. The product created was a portal to record the grades of the students. We used this portal to execute the test cases previously specified.

For each university used in our validation, ten students had their grades recorded in order to verify if the result achieved was equivalent to the results of the university. We used studies of a real class running in the second semester of 2018 and then compared the results calculated by our system to the results of the system currently used by each university.

At the end of the tests we observed that: (i) the code generator produced the code corresponding to the models in all the cases tested (related to RQ1); (ii) the grades calculated in our system were equal to the ones calculated in the university systems (related to RQ2 and RQ3). Based on these results, we concluded that, for the examples used, the transformation has covered all the evaluation criteria of the three universities, and is therefore satisfactory for the established purpose.

5 CONCLUSIONS

This paper proposed an improvement in the development of student information systems through the integration of SPL and MDD approaches. With this integration, we aim to decrease the development effort to absorb changes in evaluation criteria that frequently occur in this domain.

The MDD solution offers flexibility as it enables the specification of the evaluation criteria in a high abstraction, using DSCHOLAR language, and the generation of code in an automated way, without the need to implement them by hand. The transformation contributed to streamlining the development changes in the final products.

In addition, it is important to observe the potential of the transformation to reduce accidental implementation errors, as all the pertinent information is contained in the models and the generation of the code is automatic. Thus, the impacts of changes in the products diminishes, resulting in lower costs of maintenance of software.

The solution has been tested by proof of concept and although it has been demonstrated to be satisfactory, it has limitations. We are however, working on a case study with professionals from several universities to more accurately assess the solution and reach more generalized conclusions.

The solution was validated to demonstrate its completeness and correctness. Its expected productivity gains were not a goal of the validation and should be the subject of a future work. Another future work will be to define and implement software configuration and deployment processes that enable the solution to be correctly deployed in an institution where different actors use different evaluation criteria.

The student evaluation criteria were the vantage point selected for our study for economic reasons. The costs involved in manually changing this specific functionality into software products deployed without the MDD solution often made customers choose not to evolve its implementation. When this happened, the evaluation criteria supported by the tools differed from the current academic process, generating significant extra work for teachers and others involved. Thus, by automating the modeling and implementation of the evaluation process, we are not only increasing productivity and reducing the cost of software development, but also reducing the effort made by those (usually teachers) who use the solution.

Moreover, at a time when distance learning is expanding, the variability of institutions' assessment criteria need to be more flexible to accommodate new teaching models.

Subsequently, other points of variability of the SIS family of products will be adapted to use the approach presented in this text.

Finally, MDD has been used in the development of embedded systems, in the automotive and aerospace industries, among others. There is a lack of experiences reported using MDD to develop information systems. Therefore, this paper reports a relevant experience in the domain of educational systems which may influence future projects.

REFERENCES

Almeida, E., 2009. Component Reuse in Software Engineering. s.l.:CESAR e-books.

Booch, G., Rumbaugh, J. and Jacobson, I., 2006. UML Guia do Usuário. s.l.: Addison Wesley.

Brambilla, M., Cabot, J. and Wimmer, M., 2017. *Model-Driven Software Engineering in Practice*. 2nd ed. s.l.:Morgan and Claypool.

Buschmann, F. et al., 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. 1st ed. s.l.:Wiley.

Clements, P. and Northrop , L., 2001. *Software Product Lines: Practices and Patterns*. 3rd ed. s.l.:Addison-Wesley Professional.

Cunha, A., Fernandes, S. and Magalhães, A., 2018. A Domain Specific Language for the Domain of Student Evaluation. *Revista de Sistemas de Computação.*

Czarnecki, K. and Eisenecker, U., 2000. *Generative Programming: Methods, Tools, and Applications.* 1st ed. s.l.:Addison-Wesley Professional.

González-Huerta, J., Insfran, E., Abrahão, S. and McGregor, J., 2014. *Architecture derivation in product line development through model transformations.* s.l., Springer, Cham.

Lahiani, N. and Bennouar, D., 2018. On the use of model transformation for the automation of product derivation process in SPL. [Online] Available at: *https:// www.researchgate.net/publication/327269080_On_the _use_of_model_transformation_for_the_automation_o f_product_derivation_process_in_SP*L [Acesso em 24 12 2018].

Sendall, S. and Kozaczynski, W., 2003. Model Transformation: The Heart and Soul of Model- Driven Software Development. *IEEE Software*, 20(5), pp. 42 - 45.

Sochos, P., Riebisch, M. and Philippow, I., 2006. The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. s.l., *IEEE Int'l Conf. on the Engineering of Computer-Based System*s.

Sottet, J.-S., Vagner, A. and Frey, A. G., 2015. Variability management supporting the model-driven design of user interfaces. *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD).*

Stahl, T., Voelter, M. and Czarnecki, K., 2006. *Model-Driven Software Development: Technology, Engineering, Management*. 1st ed. s.l.:Wiley.

Tolvanen, J.-P. and Kelly, . S., 2016. Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry. 2016 4th *International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 711-719.

Warren, G., 2019. OVerview of Domain Specific Language Tools. [Online] Available at: *https:// docs.microsoft.com/pt-br/visualstudio/modeling/ overview-of-domain-specific-language-tools?view=vs-2017*

Zarrin, B. and Baumeister, H., 2018. An Integrated Framework to Specify Domain-Specific Modeling Languages. *Proceedings of 6th International Conference on Model-Driven Engineering and Software Development*, pp. 83-94.

Zhu, M., 2014. *Model-Driven Game Development Addressing Architectural Diversity and Game Engine-Integration*, s.l.: s.n.