A Domain Specific Language for the Domain of Student Evaluation

Anderson Cunha Santana do Vale, Sérgio Martins Fernandes, Ana Patrícia Magalhães

Universidade Salvador – Brasil

andersoncunh@gmail.com, sergio.fernandes@unifacs.br, anapatriciamagalhaes@gmail.com

Abstract. *Software development has become increasingly complex over the years. It might run on different platforms, integrate with other software and accept constant changes in requirements. Academic systems, although less complex than other categories of software, such as embedded systems, for example, need to integrate different subsystems, such as student enrollment and class planning and may change almost every semester. To deal with such complexity, different development approaches might be used, for example, Model-Driven Development (MDD). MDD is an approach that focuses on modeling an application and then (semi) automatically generating code to improve productivity and quality. This paper presents DSCHOLAR, a Domain Specific Language (DSL) to support the development of models of student evaluation processes at several universities. This DSL is part of a solution for the development of academic applications using the MDD approach. Our DSL can model different student evaluation scenarios to then (semi) automatically generate application code. This language was validated in a case study performed at four different universities and was efficient in modeling their student evaluation processes.*

1. Introduction

Model-driven development (MDD) is an approach for software development based on higher abstraction level models and (semi) automatic translation of these into lower level models and, frequently, into textual programming language code [Brambilla, et al., 2012] [Chen, et al., 2005]. Among other possible benefits, MDD aims to improve software development processes by increasing process productivity and product quality and facilitating multiplatform software development.

MDD is gaining acceptance in many domains (e.g. automotive, aerospace, railways and others.), but it is still evolving, and further research is needed – and is being performed – to improve its effectiveness and reach. An important research area in MDD concerns model creation. Here we present an MDD case study for the educational domain, an area where it is not widely used.

In MDD models must be expressed in modeling languages with a well-defined syntax and semantics, i.e. in General Purpose Languages (GPL), e.g. Unified Modeling Language (UML), or using Domain Specific Languages (DSL) [Chen, et al., 2005] [Schmidt, 2006]. A DSL is a modeling language or executable specification language that has the power of expression normally restricted to a particular domain of interest through notations and appropriate abstractions [Deursen, et al., 2006].

There are other software development approaches, such as Software Product Lines (SPL) [Clements, et al., 2001], that use strategies and techniques concerning productivity and quality as well as MDD. SPL is a development approach that typically focuses on the development of product families. Such products share common characteristics related to a given domain and have *points of variability*, which must be customized to attend a specific customer needs.

MDD can be integrated to SPL to improve the development of those specific points of variability. If so, models are designed – typically using DSLs – for each variability point, and (semi) automatically transformed into code.

The context for this paper is a software product family in the educational domain, named Student Information System (SIS). It manages all academic aspects of the lifecycle of students at the university, such as six-monthly enrollment, class planning and registration, course evaluations, among others. In SIS product family, each product of the family is customized to be used in a specific university or some specific area of a university. This paper focuses specifically on the student evaluation process, which is an important point of variation in the SIS product family.

Different universities (or departments in a university) have very different evaluation processes, which can also evolve significantly over time. The use of a traditional software development process in these situations would be highly inefficient, because for each specific process a non-trivial software code, manually written, would be necessary. Every time the process changes, the corresponding code would need to be updated. This is a typical case where an MDD solution applies. Therefore, we propose a solution that comprises: (i) a DSL, named DSCHOLAR, which is able to graphically represent each specific evaluation process of a university or department; (ii) a transformation program, to generate the high-level textual code based on the models created with de DSL; and (iii) a tool, to support modeling tasks using our DSL as well as the transformation execution for the code generation.

The integration of MDD and SPL as well as the use of DSL to assist this integration is not novel, especially for embedded systems or for the controls domain. On the other hand, this solution is seldom used in enterprise information systems due to aspects such as every-change requirements, weak architecture constraints, variable platforms and others [Ishida, 2007]. Particularly, this paper presents a DSL to support the modeling of students' evaluation processes. For the best of our knowledge, a DSL for this domain has never been proposed before in literature.

This paper describes DSCHOLAR DSL and presents some graphical models developed using DSCHOLAR. These models describe the evaluation process at two different universities. Moreover, we show the results of the DSL evaluation through a case study.

The rest of this paper is organized as follows. First, Section 2 introduces the development approaches related to our solution;provides an overview of our SIS product family; the student evaluation domain; and briefly describes the software tool used to support the work. Section 3 presents related works. Section 4 and 5 describe the Student Information System (SIS), the DSL DSCHOLAR proposed in this paper and how it was validated. Finally, Section 6 presents our conclusions and future work.

2. Background

Model Driven Development (MDD) [Brambilla, et al., 2012] is an approach that makes intensive use of models to represent systems at different levels of abstraction (e.g. specification, design and code). These models are used to (semi) automatically generate other models and application code in different languages.

The main concepts of the MDD approach are abstraction and automation. Models are abstract representations of the structure and behavior of systems [OMG, 2014] and automation is achieved using several model management operations – in essence, model-to-model and model-to-code transformations [Brambilla, et al., 2012].

The MDD approach uses two main elements: models, abstract representations of the structure and behavior of systems [OMG, 2014]; and transformations, programs that are responsible for model conversion into code Brambilla, et al., 2012].

Models in MDD are not mere documentation; they are the first artifacts in code generation. Therefore, they must be formally expressed in a modeling language with a well-defined syntax and semantics. In the MDD context, Domain Specific Languages (DSLs) are usually adopted – instead of UML models – because they better encapsulate the concepts of the domain, enabling the construction of more expressive models.

The definition of a DSL involves (i) an abstract syntax, which defines the constructors of the language; (ii) a static semantics, with the well-formed rules and constraints of the language; (iii) and concrete syntax, with the concrete notation for the language constructors.

The abstract syntax and static semantics of a language in MDD context are usually defined using metamodels [Stahl, et al., 2010]. Therefore, models must be in conformance with metamodels. Analogously, metamodels are defined according to a metalanguage, represented in meta-metamodels [OMG, 2017].

The concrete syntax of a language specifies how to represent its constructors during modeling and can be expressed in many ways, for example in graphical and textual notation [Stahl, et al., 2010].

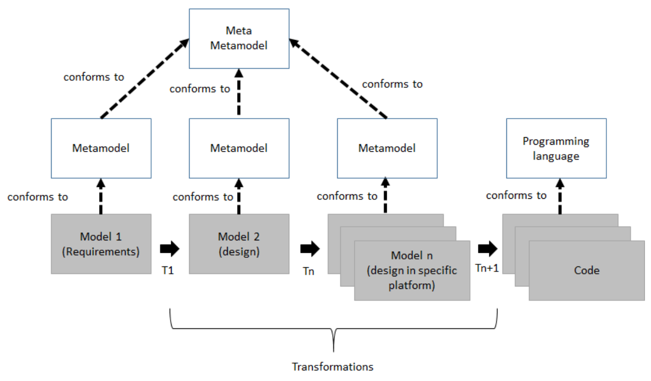

Figure 1 - Example of application development using MDD

Initially, Model 1 is specified, which could be for example a requirements model. Then it is converted, through a transformation, e.g. T1, into another model, e.g. a design model, and so on until an application code is generated. The number of models (e.g. abstraction levels) may vary in each scenario. Each model is specified according to a metamodel, which represents the domain-specific modeling language used. At a certain point in the development, a platform independent model (e.g. *Model 2* in Figure 1) can be used as input to generate models on specific platforms. (e.g. *Model n* in Figure 1) and/or in code in different programming languages.

MDD may be integrated with several other development approaches to (semi) automate the development process improving productivity.

Greenfield, et al. (2004) and Czarecki, et al. (2000) among others, propose to integrate MDD with Software Product Lines (SPL), which is an approach used to develop a software family – a range of software products with a high degree of similarity [SEI, 2012].

In SPL a set of domain artifacts can be reused for the composition of different applications in the same domain. They are common artifacts that might be customized according to applications specific needs.

SPL uses domain knowledge to identify common parts within a family of related products. These common parts form the basis of a product platform and are used in all products of a product family. Differences between products of a family are represented as variability points – unique parts of each product – which are individually developed [Istoan, 2014].

Greenfield, et al. (2004) integrates MDD and SPL and proposes the use of MDD to develop the variability points of each product in a SPL, instead of the whole product family.

It also advocates the use DSLs – instead of pure UML – to model each variability point, since UML would yield a lower fidelity description. Furthermore, multiple DSLs may be needed focusing on different aspects of the product family.

2.1 Evaluation Process Domain

A Student Information System (SIS) "consists of several basic functional modules to support features such as system setup (e.g. managing users, roles, countries, buildings, rooms, faculties, and departments), academic setup (e.g. managing courses, course sections, and study plans), admissions, student record management (e.g. managing personal information, scholarships, schedules, grades, transcripts, major transfers, etc.), registration (i.e. adding and dropping courses), final exam scheduling, grade processing (i.e. entering grades, computing Grade Point Averages (GPA), and viewing transcripts), graduation, and reporting." [Al-Hawari, et al., 2017].

Each university has its specific evaluation process and differences can be significant. To illustrate this, two real evaluation processes are briefly described below.

At a public university, for example, there is a minimum number of evaluations each class must do, but there is no fixed number of evaluations, nor a maximum number. Therefore, a teacher is free to define the number of evaluations (over the minimum) and free to define the weight given to each evaluation.

At another university, a private one, the number of evaluations and the weight of each one is predefined: there are three evaluations for each class, and their weights are 3, 4 and 3 respectively. However, only students that do not reach GPA 7 (on a scale of 0 to 10) in the two first evaluations go on to do the third one. Furthermore, Evaluation 2 is a composition of sub-evaluations.

Based on the evaluation processes described above (and others omitted for brevity), the evaluation process module of the SIS was defined as a point of variation of the product family. An analysis of the specific variations in this model made it a candidate for an MDD development process based on a DSL. Therefore, we defined the DSCHOLAR DSL for this domain as well as the necessary transformations to generate components code. The DSL, which is the main goal of this paper, is detailed in section 4.

2.2. Software tool to support the MDD / DSL solution

There are many tools suited to support the development of MDD solutions, some based on UML, others on DSLs; some open source, other proprietary.

The team working on this project had previous experience with some of these tools, such as Generic Modeling Environment (GME) [Molnár, et al., 2007], an open source DSL based modeling tool that is integrated with a transformation definition environment; Eclipse Modeling Tools, plugins for the Integrated Development Environment (IDE) Eclipse that supports the whole MDD process, based on DSLs [Eclipse Foundation]; MetaEdit+ a proprietary DSL based tool from a company named MetaCase, which also supports a complete MDD process based on DSLs [Tolvanen, 2016], among others.

The MDD / DSL software tool selected for this project was Microsoft DSL Tools, a set of plugins hosted by Microsoft Visual Studio, that comprise four tools [Microsoft Corporation, 2016]:

- A project wizard to help initiate the creation of the DSL. This wizard provides DSL templates (such as a well-formed subset of UML class diagrams; workflows, component models, among others).
- A graphical environment for DSL creation and editing.
- A validation engine that analyzes the DSL syntax and guarantees that it is well-formed.
- A transformation generator – called code generator – that translates DSL models into high-level programming language code.

Microsoft DSL Tools has strengths and some weaknesses. Among the strengths, its meta-metamodel can be based not only on a subset of UML class diagrams but, as mentioned above, the tool wizard allows for other kinds of meta-metamodels structures. It is also a relatively stable tool, with a friendly user interface, and it is integrated to MS Visual Studio – in the present case, an advantage because Visual Studio is the tool used by the software house involved in this project.

Among MS DSL Tools weaknesses, the main one is that the transformation tool is, in fact, a code generator tool, which means that it does not allow model to model transformation. It is also part of a proprietary tool – Microsoft Visual Studio – and its use implies some cost.

DSL Tools was selected for this project essentially because the academic team involved in the project had previous expertise in using it, while the software industry team already uses MS Visual Studio as standard IDE. Beyond that, it provides for easy integration between code generated by the transformations and code manually written in Microsoft Visual Studio – although this, in particular, will only be useful on the second phase of the project, not for DSL development.

The main tool weaknesses were not relevant for this project since the project does not intend to do model-to-model transformations and the organizations involved (research team and software house) supplied product licenses available to all involved.

3. Related Work

The development of new products in a software product line has been aided by the use of MDD in variation points modeling and code generation for quite some time [Durelli, et al., 2012].

Embedded systems are one of the domains which frequently adopt MDD and SPL in development. According to Bunse (2007), the use of MDD in embedded system development promotes higher than normal reuse. Therefore, many authors use this strategy. [Polzer, et al., 2009], for example, integrates SPL techniques and MDD to develop control systems, where product lines practices are used to define variabilities in the behavior of microcontrollers and MDD to improve the development of these variabilities. In the same direction, [Braga, et al., 2011] proposes the ProLiCES, an approach for the development of safety-critical embedded applications using a product line for unmanned aerial vehicles and MDD for modeling and code generation.

To assist the integration between SPL and MDD, Domain Specific Languages are usually adopted [Tokumoto, 2010]. Ivanova [Ivanova, et al., 2014], for example, proposes a DSL to develop portable embedded systems which allows rapid modeling and generation of code in different platforms, and [Durelli, et al., 2012] proposes a DSL to develop a SPL applied to mobile robot applications.

Another domain where MDD and SPL are commonly used is game design, typically using DSLs, as was presented in a survey of state of art in game development [Tang, et al., 2011]. This survey focuses on to identify strategies used in game development, including MDD, as well as on identifying modeling languages specific for this domain, e.g. DSLs that adapt state chart diagrams, use case diagrams and class diagrams to generate models in the game domain. Furthermore, the work presented by Zhu (2014) proposes the framework Engine Cooperative Game Modeling (ECGM) to model games and generate code and data based on DSLs and shown that it can significantly improve the productive.

In summary, the integration of MDD and SPL as well as the use of DSL to assist this integration is not novel, especially for embedded systems or for the controls domain. On the other hand, this solution is seldom used in enterprise information systems due to aspects such as every-change requirements, weak architecture constraints, variable platforms and others [Ishida, 2007]. Our work differs from the aforementioned ones since it applies SPL and MDD approaches to develop enterprise information systems in the academic domain. Particularly, this paper presents a DSL to support the modeling of students' evaluation processes. To the best of our knowledge, a DSL for this domain has never been proposed before in literature.

4. SIS product family

This section presents an overview of our SPL for the development of the Student Information System (SIS) product family. The MDD solution to develop the components of the product family related to student evaluation process in universities is also presented (Figure 2).

On the left of Figure 2, there is a sketch of the SIS product family high-level design. Some of the components are classified as non-variable while others are classified as variable. This classification was generated through an analysis of the features [Czarnecki, et al., 2000] in the product family. Features with no variation in all members of the product family are deemed non-variable, while those requiring customization for each product are deemed variable. New products are generated reusing and/or customizing components of the line.

The customization of variable components is made using an MDD approach. This is the case of the component *Evaluation Process*.

The right side of Figure 2 details the MDD solution to improve the development of different *Evaluation Process* components, according to the specific needs of the universities. Using the modeling language that we defined for this domain (named *DSCHOLAR* in Figure 2) models are designed (e.g. *M1*, *M2* and *M3*) representing each component variability, i.e. different student evaluation processes. These models are processed through transformations automatically generating new components code. Therefore, if a change occurs in the evaluation process of a university, a new model will be designed, and its respective component is automatically generated enabling the development of a new product in the line.

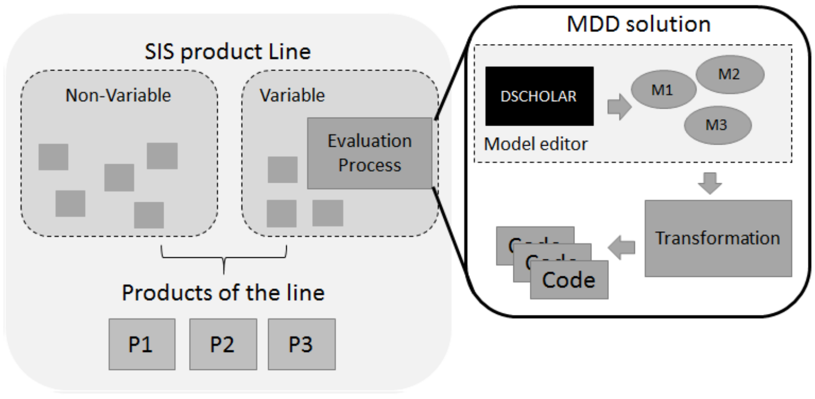

Figure 2 - Overview of SIS product line

4.1 DSCHOLAR DSL

This section presents the DSL defined to support the development components for evaluation processes using the MDD approach.

DSCHOLAR is a domain-specific modeling language defined as part of our MDD solution to represent the domain of student evaluation processes used in our SIS product line. To assist in the definition of the DSCHOLAR metamodel, we used the process proposed by [Magalhães, et al., 2015]. Moreover, to implement both the abstract and concrete syntax of the DSL, we used Microsoft DSL Tools [Microsoft Corporation, 2016].

Figure 3 shows the abstract syntax and Figure 4 shows concrete syntax of DSCHOLAR. Microsoft DSL Tools represents these two parts of the language in a single diagram. Here we split the diagram in two figures for clarity. For each element of the abstract syntax, there is a specific relationship with the element that represents its concrete syntax. Additionally, the concrete syntax elements have attributes that define how they should be graphically represented, an image file, for instance.

The meta-language of Microsoft DSL Tools used to represent DSLs abstract syntax, is based on a subset of the UML class diagram. The main concepts of the DSL abstract syntax (such as *Entity* and *Evaluation*) are represented as classes, and the relationships between these concepts are represented by associations, compositions, aggregations and inheritance, as defined by UML. One visual difference between the meta-language used here and a UML class diagram is the graphical representation of the association between concepts of the abstract syntax. While in a UML class diagram an association is represented by an edge between two classes, in the meta-language of Microsoft DSL Tools it is represented by a specific type of class, which itself is connected to the two elements that are being associated (see the association *Entity* and *Evaluation*, in the abstract syntax).

According to the abstract syntax of our DSL (Figure 3), a DSCHOLAR model that represents a specific evaluation process comprises one *Entity* and many *Evaluation*s.

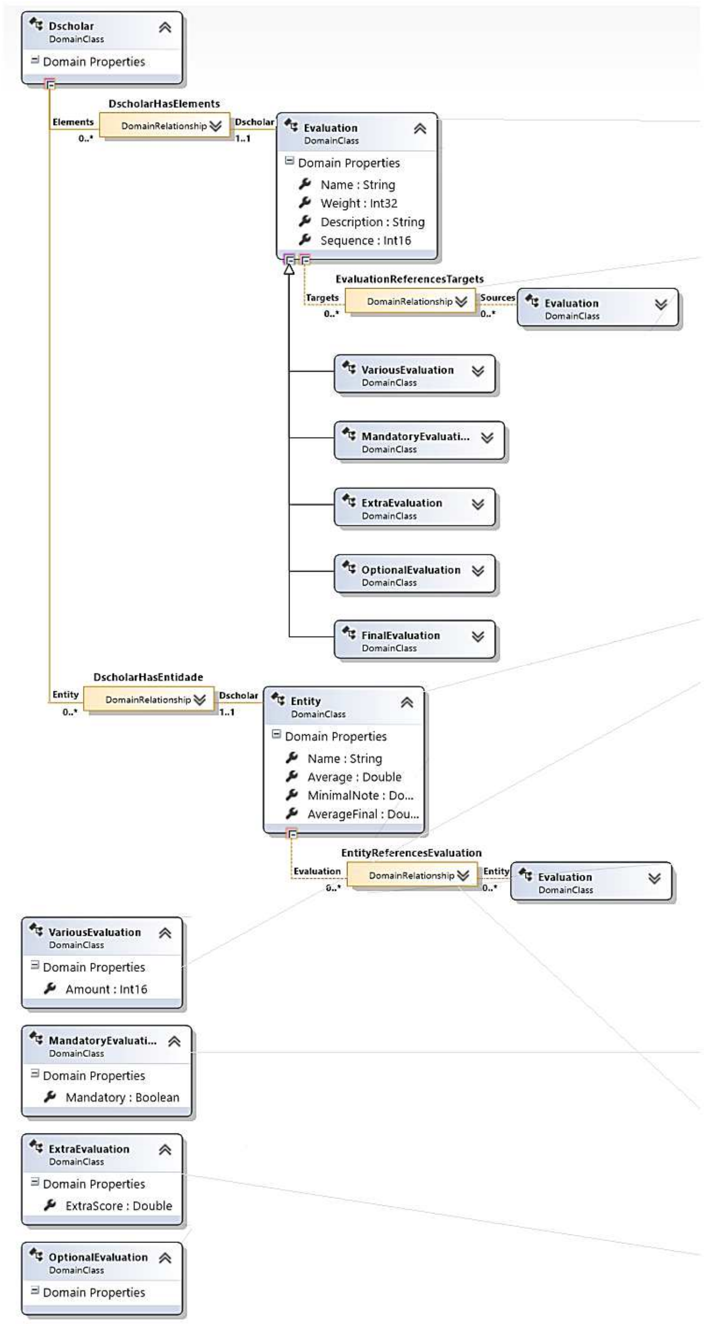

Figure 3 - Abstract Syntax of DSCHOLAR

An *Entity* represents the area to which the same evaluation process applies, such as a university or department of a university. An *Evaluation* represents the specification of an evaluation that is applied by the *Entity*, in each academic period, typically a semester. Each *Evaluation* has four attributes: *name*, *weight* (the relative weight of one evaluation relative to the others), *description* and *sequence* (the moment at which each evaluation should be applied, relative to the others). The relationship between an *Entity* and their corresponding *Evaluation* is defined through the association *EntityReferencesEvaluation*.

Figure 5 shows an example of a model defined for a private university. As can be seen, there is one *Entity*, named *Private Institution* and three *Evaluations*, named *Evaluation1*, *Evaluation2* and *Evaluation3*. In this example, all the courses at the *Private Institution* adopt the same evaluation process.

Evaluation is a general concept, specialized by four other concepts: *Mandatory Evaluation,* which must applied; *Optional Evaluation,* which is part of the evaluation process but that may be applied at teachers discretion; *Various Evaluations***,** when teachers may freely define a number of evaluations not predefined by the evaluation process; and *Extra Evaluation*, which is a special evaluation whose grade is to be added to that of another evaluation grade.

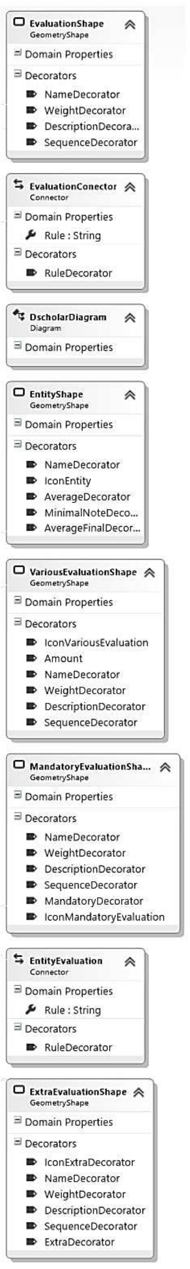

Figure 4 - Concrete Syntax of DSCHOLAR

In the model depicted in Figure 5, there are three evaluations, namely *Evaluation 1, Evaluation 2* and *Evaluation 3*, and their respective relative weight (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. The third 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 Evaluation* is already defined for an *Entity*, each one of these *Evaluations* are represented by an instance of an *Evaluation* modeling element in the respective model. Otherwise, each teacher can define the number of optional *Evaluation Specifications* so that the model will have only one instance of *VariousEvaluations* and an attribute *quantity* is used to define the upper bound of this quantity. An example of this can be seen in Section 5.2.

In the abstract syntax, there is a composition relationship between an *Evaluation* and itself. This means that the grade of a student may be a composition of grades of sub-evaluations, each with its weight. An example is shown in Figure 5 where the grade of *Evaluation 2* is a composition of *Test Evaluation*, *Arthe Evaluation* and *AIC Evaluation*, each with its respective weight. The *Arthe Evaluation* is an extra one (with a different color and small icon), which means that this grade will be added to the grade calculated by the weighted mean of the other sub-evaluations of *Evaluation 2*. For example: if the weighted mean of *Test Evaluation* and *AIC Evaluation* is 8 for a student, and the grade of *Arthe Evaluation* is 1 for the same student, her final grade in *Evaluation 2* will be 9.

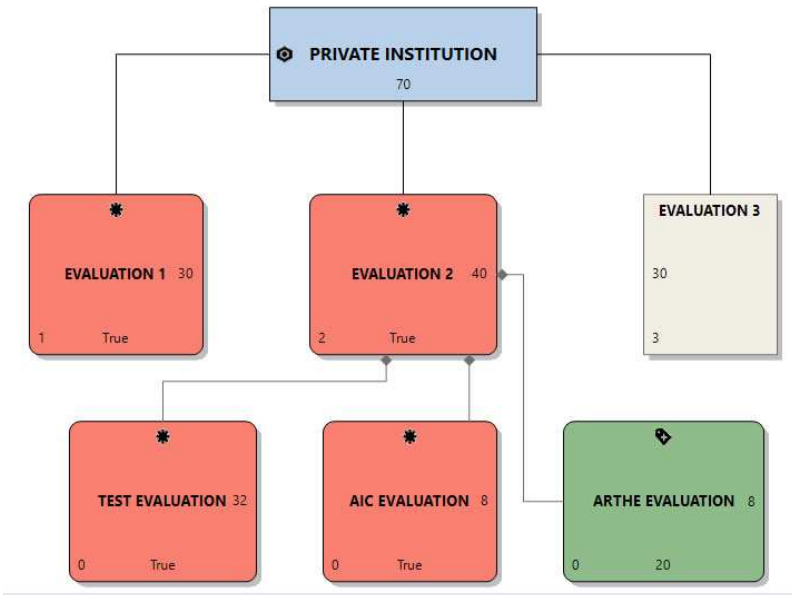

Figure 5 - Evaluation Process Model for a private University

5. DSL Evaluation

To evaluate the proposed DSL we defined and conducted a case study. It consisted of the specification of different evaluation processes at several universities in Brazil. To assist validation, we followed the metamodel design method proposed by [Magalhães, et al., 2015] and the guidelines for software engineering experimentation presented in

[Wohlin, et al., 2014]. We also used GQM [Solingen, et al., 2002] to summarize our goals (Figure 6).

The questions underlying the validation are: Q1. Do the language constructors sufficiently specify all the necessities of a university evaluation processes? Q2. Is it necessary to add new constructors in the DSL to enable the specification of different evaluation processes scenarios?

We aimed to evaluate the expressiveness of our DSL in modeling evaluation processes used at different universities. The largest four universities in Salvador regarding student quantity were selected. We detail here the evaluation performed at two of these: a private institution, which offers more than 40 courses to at least 11000 students; and a public institution with more than 300 courses and over 25000 students.

Analyze the expressiveness of the DSL for student evaluation process domain

For the purpose of specifying evaluation processes in different scenarios (different universities)

With respect to metamodel coverage

From the perspective of a university teacher staff

In the context of graduate courses

Figure 6 - Evaluation Goal

Data collection was done using two different methods: direct method, through the application of a questionnaire during the execution of the study; and independent method, through the analysis of the documentation produced by the participants, i.e. a model, written in our DSL, which represents the university evaluation process. A member of the teaching staff at each university was selected to participate in the study.

5.1 Study Preparation and Data Collection

The study was performed separately for each university. The staff member was asked to develop the model of the academic evaluation process used at their university using our DSL. To do this, they used the formal university documentation describing the academic evaluation process and their own experience as a member of the teaching staff at the university. All the participants had been teaching for more than five years at the university. Therefore, a good comprehension of the academic evaluation process was expected.

The metric used to evaluate the study was *DSL coverage*, which was measured considering two indicators: *#UC* (used constructors). It measures the number of DSL constructors used in a model, collected from the model produced by the teaching staff; and *#MC* (missing concepts), which measures the number of concepts present in the university academic evaluation process that could not be modeled by our DSL, collected in the questionnaire answered by participants. *#UC* is important to identify how many constructers as well as which of them have been validated through the study. *#MC* is important to improve the DSL. The goal is that after some validations, the *#MC* becomes zero, indicating that the metamodel covers the definition of many kinds of evaluation processes.

5.2 Study Execution

According to [Magalhães, et al., 2015] metamodel validation can be done iteratively, instantiating different models, until developers observe that the number of necessary modifications in the metamodel decreases considerably. Following this, we performed the validation between October and December 2017 iteratively instantiating a model for each selected university, collecting data and using the results to improve the metamodel before validating another university.

In each validation, we first trained the participants in DSL usage (e.g. explained the metamodel constructors and showed them how to use the modeling environment) and then asked participants to develop the model of the evaluation process used at their university. They developed the model alone using a computer in our research laboratory. The models produced for the public university and the private university can be seen in Figure 7 and Figure 5, respectively. The description of how the evaluation process is used at these universities work is briefly described in Section 4.1.

Figure 7 shows the resulting model for the public university, named *Public Institution*. At this university, there are three mandatory evaluations, e.g. *Evaluation1*, *Evaluation2*, *Evaluation3,* one *Various Evaluation* and the *Final Evaluation*. The weight of each evaluation is expressed as an attribute of the box. If there is a fixed weight, it must be defined (e.g. *Final evaluation*, with *weight*=30). Otherwise, the weight is represented as "0" indicating that the teaching staff will be responsible for this definition (e.g. *Evaluation1*, with *weight*=0).

As a result of this validation, we identified the need to include this concept of *various evaluation* (using our metric *#MC*=1). This was included in our DSL and the model was recreated before proceeding to the next university.

The model produced for the private university can be seen in Figure 5. In this validation, we identified the need to represent an evaluation as a composition of other evaluations, e.g. *Evaluation2* is a composition of *Test Evaluation*, the *Arthe Evaluation* and *AIC Evaluation* (again *#MC*=1). The metamodel was therefore modified again to add this concept and then we proceeded to another validation.

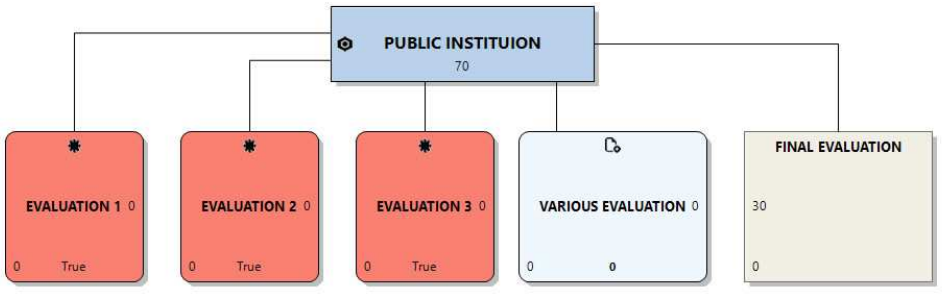

Figure 7 - Model of the evaluation process at a public higher education institution

5.3 Study Data Analysis

As we iteratively validated and modified the DSL during this study, at the end of the fourth validation, we observed that all the concepts defined in our DSL were used in the models produced, i.e. #UC=100%. Therefore, we can say that it covers the necessary constructors to instantiate the models (related to Q1). Moreover, the missing concepts identified during the process were included and are now part of the language (related to Q2).

We know that the study results are limited and do not provide statistical evidence to support general conclusions. However, we believe that it can be considered an initial step in planning future case studies. The validation reached its goal, i.e. the DSL has enough expressiveness to specify evaluation processes at different universities.

The DSCHOLAR evaluation already performed does not imply that the DSL will not need to evolve, to achieve a broader audience, where the evaluation process may significantly differ from what we found until now.

If so, to minimize the costs involved, we are designing a component-based, low coupling software solution, so that ideally no collateral effects will appear if a student evaluation process changes, meaning that the DSL generated code will have to be updated, and even if the DSL itself needs to be updated.

7. Conclusions

This paper presented the DSL DSCHOLAR as part of the Student Information System (SIS) product line. DSCHOLAR enables the use of the MDD approach to developing variable components for the student evaluation process domain.

Using DSCHOLAR, models can be graphically developed to express the variability in evaluation processes at different universities. These models can be used as the main artifacts to automatically generate code improving flexibility and productivity in software development. Therefore, implementation of changes in evaluation process can be modeled directly by the academic stuff, the domain specialist.

The student evaluation process was the variation point selected for our study for economic reasons, because the costs involved in “manually” changing this particular functionality on software products deployed before the MDD solution creation frequently made clients opt for not evolving their implementations when their process changed, which is not rare.

When that happened, the evaluation processes supported by the tools differed from the academic process in place, generating significant extra work for teachers and others involved.

So, by automating the evaluation process modeling and implementation, we are not only increasing productivity and reducing the cost of software development but also reducing the effort performed by those (typically, teachers) who use the solution.

However, quantitative measurements of quality and efficiency improvements of our solution will only be viable after the second phase of the project– the development of transformations and integration between automated and manually generated code – is concluded.

From the research point of view, we believe that the real contribution of our work is the use of MDD integrated with SPL for the educational domain – a domain for which MDD is not widely applied. Besides, this is an academic project integrated with an industrial one. The solution developed here will be incorporated into the SIS development process of a software house.

The project also aims to demonstrate the practicality and efficiency of using not one large DSL for the whole domain, but instead of many small ones for each specific variation point of the software family.

DSCHOLAR was evaluated in a case study and shown to be expressive in modeling different university scenarios. We are currently planning a controlled experiment to generate code from the models produced in the case study and integrate it with manually generated components. The next step involves a case study to validate the MDD solution as a whole for this specific variation point of the SPL.

References

Al-Hawari, F. Alufeishat, A. Alshawabkeh, M. Barham, H. and Habahbeh, M. (2017) "The Software Engineering of a Three-Tier Web-Based Student Information System (MyGJU)," Computer Applications in Engineering Education, vol. 25, no. 2, pp. 242-263.

Braga, R. V. Castelo Branco, K. R. Trindade Junior, O. Masiero, P. C. Neris, L. O. and Becker, M. (2011) "The ProLiCES Approach to Develop Product Lines for Safety-Critical Embedded Systems and its Application to the Unmanned Aerial Vehicles Domain," in CLEI, Quito.

Brambilla, M. Cabot, J. and Wimmer, M. (2012) "Model-Driven Software Engineering in Practice", Morgan & Claypool.

Bunse, C. Gross, H. G. and Peper, C. (2007) "Applying a Model-Based Approach for Embedded System Development," Delft University of Technology, Delft.

Chen, K. Sztipanovits J. and Neema, S. (2005) "Toward a Semantics Anchoring Infrastructure for Domain-Specific Modeling Languages," EMSOFT, pp. 19-22.

Clements, P. and Northrop, L. (2001) "Software Product Lines: Practices and Patterns", Addison-Wesley Professional.

Czarnecki, K. and Eisenecker, U. W. (2000) "Generative Programming: Methods, Tools, and Applications", Addison-Wesley Professional.

Deursen, A. Klint A. and Visser, J. (2006) "Domain-Specific Languages: An Annotated Bibliography," ACM SIGPLAN Notices.

Durelli R. S. and Durelli, V. H. S. (2012) "F2MoC: A Preliminary Product Line DSL for Mobile Robots," in VIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2012) Trilhas Técnicas .

Eclipse Foundation, "Eclipse Modeling Project," [Online]. Available: https://www.eclipse.org/modeling/.

Greenfield, J. Short, K. Cook, S. and Kent, S. (2004) "Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools", Wiley.

Ishida, Y. (2007) "Challenge for the SPL Approach in Enterprise Software Development," NRI Information Technology Report, vol. 8.

Istoan, P. A. (2014) “Methodology for the derivation of product behaviour in a Software Product Line”, Rennes.

Ivanova, V. Sedov, B. and Sheynin, Y. (2014) "Domain-specific languages for embedded systems portable software development," in 16th Conference of Open Innovations Association (FRUCT16), Oulu.

Kelly, S. and Tolvanen, J.P. (2007) “Domain-Specific Modeling: Enabling Full Code Generation”, Wiley-IEEE Computer Society Pr, p. 500.

Magalhães, A. P. Maciel, R. S. P. and Andrade, A. M. (2015) "Towards a Metamodel Design Methodology: Experiences from a model transformation metamodel design.," 27th International Conference on Software Engineering and Knowledge Engineering, pp. 625-630.

Microsoft Corporation, (2016) "Overview of Domain-Specific Language Tools," Microsoft, [Online]. Available: https://docs.microsoft.com/pt-br/visualstudio/modeling/overview-of-domain-specific-language-tools. [Accessed april 2018].

Microsoft Corporation (2016), "Overview of Domain-Specific Language Tools".

Molnár, Z. Balasubramanian, D. and Lédeczi, Á. (2007) "An Introduction to the Generic Modeling Environment," in Model-Driven Development Tool Implementers Forum.

Object Management Group (OMG) (2014), "ABOUT THE OBJECT CONSTRAINT LANGUAGE SPECIFICATION VERSION 2.4," [Online]. Available: https://www.omg.org/spec/OCL/About-OCL/.

OMG (2014), "Model Driven Architecture," [Online]. Available: http://www.omg.org/mda/specs.htm.

OMG (2017), "Meta Object Facility," [Online]. Available: http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf.

Polzer, A. and Kowalewski, S. (2009)"Applying software product line techniques in model-based embedded systems engineering," in ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software.

Schmidt, D. (2006) "Model-Driven Engineering.," IEEE Computer Magazine.

SEI (2012), "Software Engineering Institute," [Online]. Available: http://www.sei.cmu.edu/productlines/frame_report/PL.essential.act.htm.

Solingen, R. Basili, V. Caldiera, G. and Rombach, H. D. (2002) “Goal Question Metric” (GQM) Approach, John Wiley & Sons. Inc.

Stahl, T. and Volter, M. (2010) “Model-Driven Software Development.”, Wiley.

Tang S. and Hanneghan, M. (2011) "State of the Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning," JILR Journal.

Tokumoto, S. (2010) "Product Line Development using Multiple Domain Specific Languages in Embedded Systems," in MoDELS 2010 ACES-MB Workshop Proceedings, Oslo.

Tolvanen, J.P. (2016) "MetaEdit+ for collaborative language engineering and language use (tool demo)," in roceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam.

Wohlin, C. and Aurum, A. (2014) "Towards a decision-making structure for selecting a research design in empirical software Engineering," Empirical Software Engineering - Springer.

Zhu, M. (2014) “Model-Driven Game Development Addressing Architectural Diversity and Game Engine-Integration.